

# Create and publish a Hetzner account

---

- Sign up at <https://accounts.hetzner.com/signUp> using an account name of your choice.
- Optionally: Activate 2-factor authentication.
- You may validate your account by ID card or similar. No payment required!
- Publish your Hetzner account's username (e.g. the registration e-mail ) to your SDI course's group at <https://learn.mi.hdm-stuttgart.de>.
- Upon confirmation by your lecturer a personal Hetzner project space `sdi_gxy` (e.g. `sdi_g01` corresponding to group 1) should be accessible after login.

# Related exercises

---

Exercise 3: Your first server

# Current server security flaws

---

- No updates, just (likely) outdated installation image
- Password based logins being notoriously prone to attacks.

Solution: Use public/private key based ssh login.

- There is no firewall yet restricting network access. Insecurely configured supplementary software components e.g. database servers may lead to disaster.

Two choices:

- Cloud provider level centralized firewall.
- Host local firewall, e.g. Ufw.

## Preliminary: Create an **ssh** key pair

---

```
sdi user@martin-pc-dachboden: ~$ ssh-keygen -t ed25519 ❶  
Generating public/private ed25519 key pair.  
Enter file in which to save the key (/home/sdi user/. ssh/i d_ed25519):  
Created directory '/home/sdi user/. ssh'.  
Enter passphrase (empty for no passphrase): ❷  
Enter same passphrase again:  
Your identification has been saved in /home/sdi user/. ssh/i d_ed25519 ❸  
Your public key has been saved in /home/sdi user/. ssh/i d_ed25519. pub ❹
```

# Related exercises

---

Exercise 4: Improve your server's security!

# Cleaning up!

---

## Caution

This is about **\$\$\$ MONEY \$\$\$**

- Delete your server including the IPv4 address.
- You may optionally delete your firewall.

# What's it all about?

---

Quote:

“Terraform is an infrastructure as code tool that lets you build, change, and version cloud and on-prem resources safely and efficiently.”

# Terraform resources

---

- Why Terraform?
- Install Terraform.



# Hetzner API token

---

- Access your cloud project using the Hetzner GUI interface.
- Go to Security --> API Tokens --> Generate API token
- Provide a name and hit Generate API token.
- Copy the generated token's value and store it in a secure location e.g. a password manager.

## **Caution**

The Hetzner GUI blocks future access to the token.

# Minimal Terraform configuration

---

```
# Define Hetzner cloud provider
terraform {
  required_providers {
    hcloud = {
      source = "hetznercloud/hcloud"
    }
  }
  required_version = ">= 0.13"
}

# Configure the Hetzner Cloud API token
provider "hcloud" {
  token = "your_api_token_goes_here"
}

# Create a server
resource "hcloud_server" "helloServer" {
  name      = "hello"
  image     = "debian-12"
  server_type = "CX22"
}
```

# Terraform init

---

```
$ terraform init
```

```
Initializing the backend...
```

```
Initializing provider plugins...
```

- Finding latest version of hetznercloud/hcloud...
- Installing hetznercloud/hcloud v1.46.1...
- Installed hetznercloud/hcloud v1.46.1 (signed by a Hashi Corp partner, key ID 5219EACB3A77198B)

```
...
```

```
Terraform has created a lock file .terraform.lock.hcl to record the provider selections it made above. ...
```

# Terraform plan

---

```
$ terraform plan
```

```
Terraform used the selected providers to generate the following execution plan. Resource actions ...  
+ create
```

```
Terraform will perform the following actions:
```

```
# hcloud_server.helloServer will be created  
+ resource "hcloud_server" "helloServer" {  
  + allow_deprecated_images = false  
  + backup_window           = (known after apply)  
  ...  
}
```

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

# Terraform apply

---

```
$ terraform 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
```

```
cloud_server.helloServer: Creating...
```

```
cloud_server.helloServer: Still creating... [10s elapsed]
```

```
cloud_server.helloServer: Creation complete after 14s [id=45822789]
```

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

# Credentials by E-Mail

---

Your server "hello" was created!

You can access your server with the following credentials:

IPv4 **128.140.108.60**

IPv6 2a01:4f8:1c1c:8e3a::/64

User root

Password rJ3pNvJXbqMp3XNTvFdq

You will be prompted to change your password on your first login.

To improve security, we recommend that you add an SSH key when creating a server.

# Problems: #

---

- Firewall blocks **ssh** server access:

```
$ ssh root@128.140.108.60
```

```
ssh: connect to host 128.140.108.60 port 22: Connection refused
```

Access by Vnc console login only

- IP and (initial) credentials by email #

Solution:

1. Add firewall inbound **ssh** access rule.
2. Configure **ssh** public key login.

# ssh access, firewall

---

```
resource "hcloud_firewall" "sshFw" {
  name = "ssh-firewall"
  rule {
    direction = "in"
    protocol   = "tcp"
    port       = "22"
    source_ips = ["0.0.0.0/0", "::/0"]
  }
}

...

resource "hcloud_server" "helloServer" {
  ...
  firewall_ids = [hcloud_firewall.sshFw.id]
}
```



## ssh access, public key

---

```
resource "hcloud_ssh_key" "loginUser" {
  name          = "goik@hdmstuttgart.de"
  public_key    = file("~/ssh/id_ed25519.pub")
}

...

resource "hcloud_server" "helloServer" {
  ...
  ssh_keys      = [hcloud_ssh_key.loginUser.id]
}
```

Note: Use the Hetzner Web GUI for removing any conflicting manually installed ssh keys beforehand.

# Apply **ssh** key access

---

```
$ terraform apply
```

```
# hcloud_firewall.sshFw will be created
+ resource "hcloud_firewall" "sshFw" {
    ...
# hcloud_server.helloServer will be created
+ resource "hcloud_server" "helloServer" {
    ...
# hcloud_ssh_key.goi k will be created
+ resource "hcloud_ssh_key" "logi nUser" {
    ...
```

```
Plan: 3 to add, 0 to change, 0 to destroy.
```

```
Apply complete! Resources: 3 added, 0 changed, 0 destroyed.
```

# Output data details #1/2

---

See terraform output documentation:

File output s. tf	Result
<pre>output "hello_ip_addr" {   value      = hcloud_server.helloServer.ipv4_address   description = "The server's IPv4 address" }</pre>	<pre>\$ terraform output hello_datacenter = "nbg1-dc3" hello_ip_addr    = "159.69.152.37"</pre>
<pre>output "hello_datacenter" {   value      = hcloud_server.helloServer.datacenter   description = "The server's datacenter" }</pre>	<pre>\$ terraform output hello_ip_addr "159.69.152.37"</pre>

## Output data details #2/2

File outputs.tf

```
output "hello_ip_addr" {
  value      = hcloud_server.helloServer.ipv4_address
  description = "The server's IPv4 address"
}

output "hello_datacenter" {
  value      = hcloud_server.helloServer.datacenter
  description = "The server's datacenter"
}
```

terraform output - json

```
{
  "hello_datacenter": {
    "sensitive": false,
    "type": "string",
    "value": "nbg1-dc3"
  },
  "hello_ip_addr": {
    "sensitive": false,
    "type": "string",
    "value": "159.69.152.37"
  }
}
```

## Problem 2: VCS and visible provider API token #

---

**Versioned** file `main.tf`:

```
...  
provider "hcloud" { token = "xdaGfz9Lmw08SVkg ... " }  
...
```

Solution:

- Declare a variable `hcloud_token` in a `variables.tf` file
- Add a non-versioned file `secrets.auto.tfvars`.
- Optional: Provide a versioned `secrets.auto.tfvars.template` documenting file

# Solution by variable

---

Declaring `hcl_oud_token` in variables.tf:

```
variable "hcl_oud_token" { # See secret.auto.tfvars
  nullable = false
  sensitive = true
}
```

Using `hcl_oud_token` in main.tf:

```
provider "hcl_oud" { token = var.hcl_oud_token }
```

Defining `hcl_oud_token`'s value in secrets.auto.tfvars:

```
hcl_oud_token="xdaGfz9Lmw08SWkg ... "
```

Template file secrets.auto.tfvars.template:

```
hcl_oud_token="your_api_token_goes_here"
```

## Solution by file

---

```
# Configure the Hetzner Cloud API token
provider "hcloud" {
  token = file("../provider.token.key")
}
```

Content of file provider.token.key:

```
xdaGfz9Lm08SWkg ...
```

Content of file provider.token.key.template:

```
your_api_token_goes_here
```

## Related exercises

---

Exercise 5: Incrementally creating a base system



# Introduction and reference

---

- Cloud Stack Conference talk.
- Cloud-init documentation
- Validation
- Debugging

# In a nutshell

---

- Distribution image containing pre-installed Cloud Init
- Script configurable installation options

# Configuration options

---

- Individual CRUD file operations
- Supplying ssh user and host keys.
- Adding users
- ...
- Installing packages
- System Upgrade + reboot
- Arbitrary command execution

# Terraform interface to Cloud Init

---

```
resource "cloud_server" "web" {  
  name = var.server_name  
  ...  
  user_data = file("user_data.yml")  
}
```

## »hello, world ...« user Data. yml file

---

```
#cloud-config
```

```
packages:
```

- nginx

```
runcmd:
```

- systemctl enable nginx

- rm /var/www/html/\*

- >

```
echo "I'm Nginx @ $(dig -4 TXT +short o-o.nyaddr.1.google.com @ns1.google.com)
```

```
created $(date -u)" >> /var/www/html/index.html
```

# Using and debugging template files

---

```
resource "local_file" "user_data" {
  content      = templatefile("tpl/userData.yml", {
    loginUser  = "devops"
  })
  filename    = "gen/userData.yml"
}
```

```
resource "cloud_server" "helloServer" {
  ...
  user_data = local_file.user_data.content
}
```

# Template files demo

Terraform main.tf	Cloud-init <b>tpl/userData.yml</b>	Cloud-init <i>gen/userData.yml</i>
<pre>... resource "cloud_server" "my" { ...   user_data = templatefile(     "tpl/userData.yml", {       loginUser = "devops"     }) } ...</pre>	<pre>write_files: - content:       AllowUsers \${loginUser}     PasswordAuthentication no ... users: - name: \${loginUser}   groups: sudo ...</pre>	<pre>write_files: - content:       AllowUsers devops     PasswordAuthentication no ... users: - name: devops   groups: sudo ...</pre>

# Validation

---

- **# cloud-init schema --system --annotate**

```
...
apt: # E1
- debconf_selections: openssh-server openssh-server/password-authentication boolean
  false openssh-server openssh-server/permit-root-login boolean false
...
# Errors: -----
# E1: [{'debconf_selections': 'openssh-server ....boolean false'}] is not of type 'object'
```

- **# cloud-init schema --config-file /var/lib/cloud/instance/user-data.txt**

```
Valid cloud-config: /var/lib/cloud/instance/user-data.txt
```



# Watch out for your enemies!

---

```
root@hello: ~# journalctl -f
```

```
May 06 04: 41: 20 hello cloud-init[898]: Cloud-init v. 22.4.2 finished at Mon, 06 May 2024 04: 41: 20 +0000. DataSource
```

```
...
```

```
May 06 04: 46: 16 hello sshd[927]: Invalid user abc from 43.163.218.130 port 33408
```

```
May 06 04: 46: 17 hello sshd[927]: Received disconnect from 43.163.218.130 port 33408: 11: Bye Bye [preauth]
```

```
May 06 04: 46: 17 hello sshd[927]: Disconnected from invalid user abc 43.163.218.130 port 33408 [preauth]
```

```
...
```

```
May 06 04: 50: 54 hello sshd[930]: fatal: Timeout before authentication for 27.128.243.225 port 59866
```

```
...
```

```
May 06 04: 52: 45 hello sshd[933]: Invalid user cos from 43.163.218.130 port 59776
```

```
...
```

```
May 06 04: 53: 04 hello sshd[935]: Invalid user admin from 194.169.175.35 port 51128
```

```
May 06 04: 53: 49 hello sshd[937]: User root from 43.163.218.130 not allowed because not listed in AllowUsers
```

```
May 06 04: 53: 49 hello sshd[937]: Disconnected from invalid user root 43.163.218.130 port 50592 [preauth]
```

# Related exercises

---

Exercise 6: Working on Cloud-init

# Problem: Duplicate known\_hosts entry on re-creating server

---

Problem of repeated terraform apply:

```
$ ssh root@128.140.108.60
```

```
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
```

```
@      WARNING: REMOTE HOST IDENTIFICATION HAS CHANGED!      @
```

```
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
```

```
IT IS POSSIBLE THAT SOMEONE IS DOING SOMETHING NASTY!
```

```
Someone could be eavesdropping on you right now (man-in-the-middle attack)!
```

## Solution: Generating known\_hosts ...

---

```
resource "local_file" "known_hosts" {
  content      = "${hcloud_server.helloServer.ipv4_address} ...
                ... ${tls_private_key.host.public_key_openssh}"
  filename    = "gen/known_hosts"
  file_permission = "644"
}
```

## ... and ssh wrapper

main.tf

```
resource "local_file" "ssh_script" {  
  content = templatefile("tpl/ssh.sh", {  
    ip=cloud_server.hello.ipv4_address  
  })  
  filename      = "bin/ssh"  
  file_permission = "700"  
  depends_on    = [local_file.known_hosts]  
}
```

tpl/ssh.sh

```
#!/usr/bin/env bash  
  
GEN_DIR=$(dirname "$0")/../../gen  
  
ssh -o UserKnownHostsFile=\  
    "$GEN_DIR/known_hosts" devops@${ip} "$@"
```

## Related exercises

---

Exercise 7: Solving ~/.ssh/known\_hosts quirk

# Failsafe console login

---

```
#cloud-config
```

```
chpasswd:
```

```
  list: |
```

```
    root: bingo9wingo
```

```
  expire: False
```

# Avoiding Yaml parsing issues

---

```
#cloud- config
```

```
write_files:
```

```
- content:
```

```
    ${base64encode(private_key.pem)}
```

```
    encoding: base64
```

```
    path: /etc/nginx/snippets/cert/private.pem
```

## Note

Congrats to paultyng



# A volume: The easy way

---

```
resource "hcloud_server" "helloServer" {  
  server_type = "CX22"  
  ...  
}
```

```
resource "hcloud_volume" "volume01" {  
  name      = "volume1"  
  size      = 10  
  server_id = hcloud_server.helloServer.id  
  auto_mount = true  
  format    = "xfs"  
}
```

```
df  
...  
/mnt/HC_Volume_100723816
```

# Volume details

---

```
output "volume_id" {
  value=cloud_volume.volume01.id
  description = "The volume's id"
}
```

```
terraform apply
```

```
...
hello_ip_addr="37.27.22.189"
volume_id="100723816"
```

```
# ls /dev/disk/by-id/*100723816
/dev/disk/by-id/scsi-OHC_Volume_100723816
```

```
Desired /etc/fstab:
```

```
/dev/disk/by-id/scsi-OHC_Volume_100723816
    /volume01 xfs discard, noatime, defaults 0 0
```

## Related exercises

---

Exercise 8: Auto mounting a volume

# Providing a mount point's name

---

```
resource "hcloud_server" "helloServer" {  
  ...  
  user_data = templatefile("tpl/userData.yml", {  
    ...  
    volume_id = hcloud_volume.volume_id  
  })  
}  
resource "hcloud_volume" "volume01" {  
  server_id = hcloud_server.helloServer.id  
  ...  
}
```

Problem: Cyclic dependency  
helloServer <- -> volume01

## Solution: Independent resource creation

```
main.tf
```

```
resource "hcloud_volume" "vol 01" {
  size = 10
}
resource "hcloud_server" "hello" {
  user_data = templatefile(
    "userdata.yml.tpl", {
      # No cycle
      volume_id=hcloud_volume.vol 01.id
    })
}
resource "hcloud_volume_attachment"
"main" {
  volume_id=hcloud_volume.vol 01.id
  server_id=hcloud_server.hello.id
}
```

```
userdata.yml.tpl
```

```
echo
`/bin/ls /dev/disk/by-id/*${volume_id}`
/vol 01 xfs discard,noatime,defaults 0 0
>> /etc/fstab
```

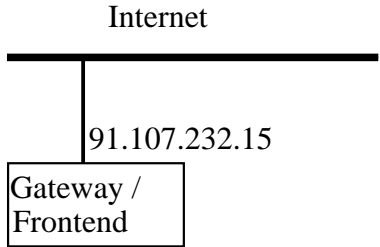
# Related exercises

---

Exercise 9: Mount point's name specification

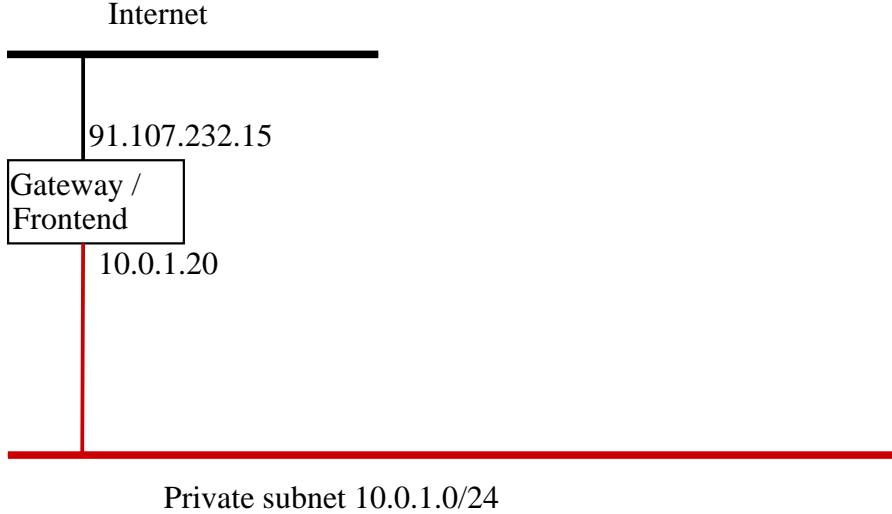
# Private subnet overview

---



# Private subnet overview

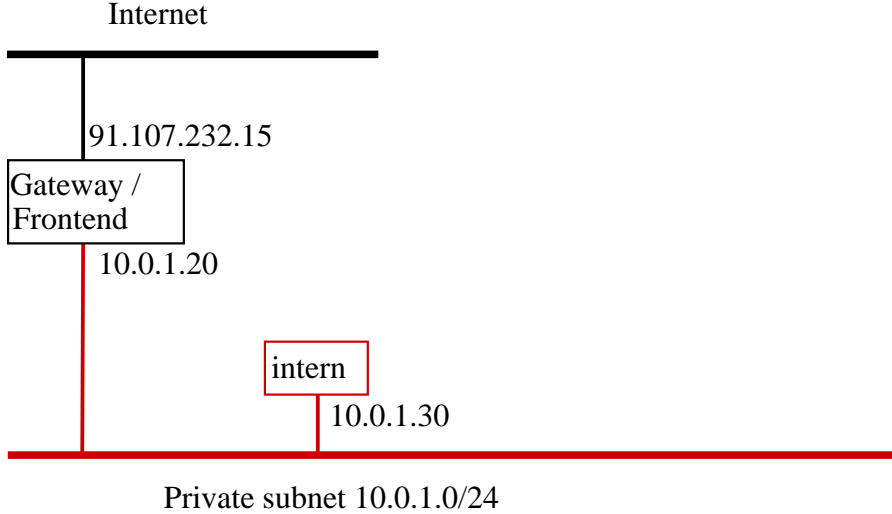
---





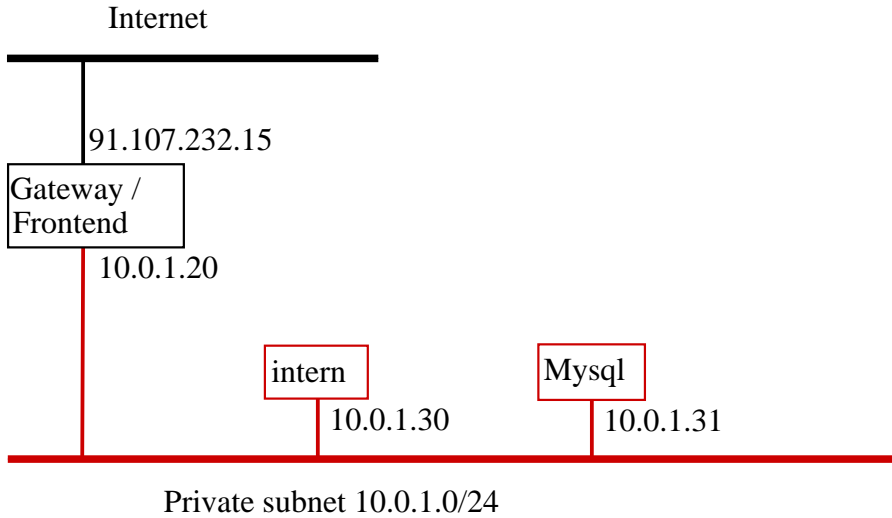
# Private subnet overview

---



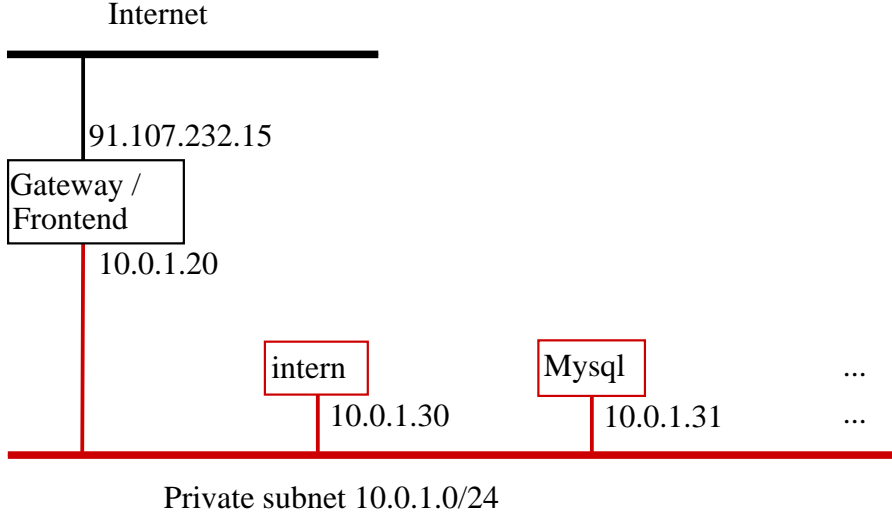
# Private subnet overview

---



# Private subnet overview

---



# Terraform subnet creation

---

```
resource "hcloud_network" "pNet" {
  name      = "Private network"
  ip_range = "10.0.0.0/8"
}

resource "hcloud_network_subnet" "pSubnet" {
  network_id = hcloud_network.pNet.id
  type       = "cloud"
  network_zone = "eu-central"
  ip_range   = "10.0.1.0/24"
}

resource "hcloud_network_route" "gateway" {
  network_id = hcloud_network.pNet.id
  destination = "0.0.0.0/0"
  gateway    = "10.0.1.20"
}
```

# Gateway host

---

```
resource "hcloud_server" "web" {  
  . . . .  
  network {  
    network_id = hcloud_network.pNet.id  
    ip         = "10.0.1.20"  
  }  
}
```

# intern host

---

```
resource "hcl_oud_server" "web" {  
  . . .  
  public_net {  
    ipv4_enabled = false  
    ipv6_enabled = false  
  }  
  network {  
    network_id = hcl_oud_network.pNet.id  
    ip         = "10.0.1.20"  
  }  
}
```

# Related exercises

---

Exercise 10: Creating a subnet

# Lack of internet access

---

- Host “intern” does not have Internet access.
- Consequences:
  1. No package updates.
  2. No package installs
  3. ...



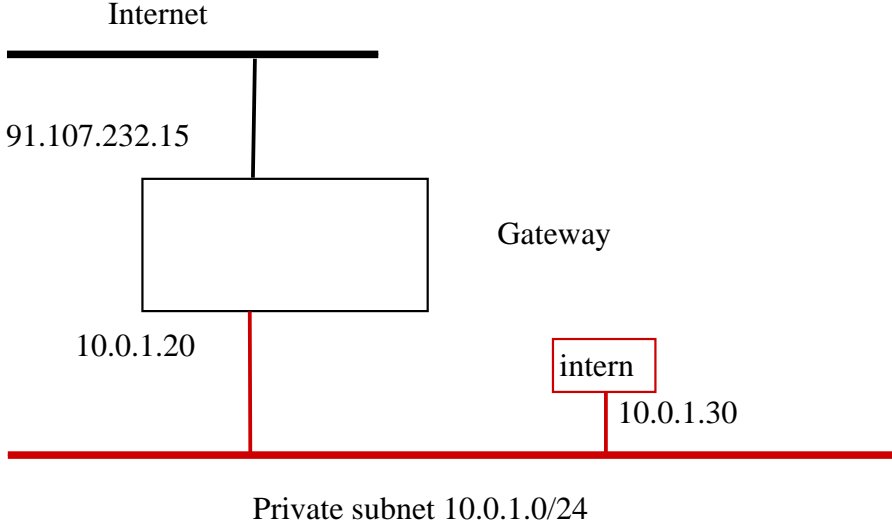
# Possible solutions

---

1. • Allow IP forwarding on gateway host
  - Configure NAT enabling gateway host as router
2. Use an application level gateway:
  - a. apt-cacher-ng
  - b. Squid

# http proxy apt-cacher-ng

---



[http proxy apt-cacher-ng](http://proxy.apt-cacher-ng)

---

# Cloud-init problem

---

- Problem: apt-cacher-ng installation requires time for service to become available.
- Consequence: Package installs on host “**intern**” must be deferred.
- Problem: No standard Terraform “service ready” dependency management hook.

# Service ready query script

---

```
#!/bin/bash
echo "Waiting for apt-cacher-ng to launch on port 3142 ..."

while ! nc -z ${frontendPrivateNetIp} 3142;
do
    sleep 8 # wait for 8 second before polling again
    echo apt-cacher-ng not yet ready ...
done

echo "apt-cacher-ng service ready"
```

# Terraform "service ready" dependency hook

---

```
resource "null_resource" "waitForProxy" {
  connection {
    type      = "ssh"
    user      = "devops"
    host_key  = ... public_key_openssh
    agent     = "true"
    host      = ... web.ipv4_address
  }
  provisioner "remote-exec" {
    inline=[ "/usr/bin/waitForAptProxy" ]
  }
}
```

```
resource "hcloud_server" "intern" {
  ...
  depends_on = [
    hcloud_network_subnet.pSubnet
    , null_resource.waitForProxy
  ]
}
```

## Related exercises

---

Exercise 11: Adding an application level gateway

# Subdomain per group

---

- Dedicated lecture related DNS server ns1.sdi.hdm-stuttgart.cloud.
- One subdomain per group e.g. **g03.sdi.hdm-stuttgart.cloud** corresponding to **group 3**.
- Zone edits require a subdomain specific hmac secret key being provided as dnsupdate.sec file in your personal group entry below the SDI course:

```
hmac-sha512: g03. key: I5sDDS3L1BU . .
```

## Note

The per zone secrets are being created using tsig-keygen. Value appearing here do not reflect production settings.

- Edits become globally visible. Mind the TTL setting: A higher value means you'll have to wait longer until updates become visible.



# Key file location

---

Key file available in your working group below 113475 Software defined Infrastructure.

# Querying DNS by zone transfer

---

```
$ export HMAC=hnac-sha512:g03.key:YXVSh3l...
$ dig @ns1.sdi.hdmstuttgart.cloud -y $HMAC -t AXFR g03.sdi.hdmstuttgart.cloud
...
g03.sdi.hdmstuttgart.cloud. 10 IN SOA ns1.g03.sdi.hdmstuttgart.cloud. goi.k.hdmstuttgart.de. 2024051551
g03.sdi.hdmstuttgart.cloud. 10 IN NS ns1.g03.sdi.hdmstuttgart.cloud.
g03.sdi.hdmstuttgart.cloud. 10 IN TXT "Hello Nerds, how are you going? :-)"
ns1.g03.sdi.hdmstuttgart.cloud. 10 IN A 195.201.113.223
g03.sdi.hdmstuttgart.cloud. 10 IN SOA ns1.g03.sdi.hdmstuttgart.cloud. goi.k.hdmstuttgart.de. 2024051551
...
```

# Creating an A record

---

```
export HMAC=hmactool sha512: g03. key: YXW5eh3l...
```

```
$ nsupdate -y $HMAC
```

```
> server ns1.sdi.hdm-stuttgart.cloud
```

```
> update add www.g03.sdi.hdm-stuttgart.cloud 10 A 141.62.75.114
```

```
> send
```

```
> quit
```

```
$ dig +noall +answer @ns1.sdi.hdm-stuttgart.cloud www.g03.sdi.hdm-stuttgart.cloud
```

```
www.g03.sdi.hdm-stuttgart.cloud. 10 IN A 141.62.75.114
```

```
$ dig +noall +answer @8.8.8.8 www.g03.sdi.hdm-stuttgart.cloud
```

```
www.g03.sdi.hdm-stuttgart.cloud. 10 IN A 141.62.75.114
```

## Modify by delete/create

---

```
$ nsupdate -y $HMAC
> server ns1.sdi.hdm.stuttgart.cloud
> update delete www.g03.sdi.hdm.stuttgart.cloud. 10 IN A 141.62.75.114
> send
> quit
>
$ dig +noall +answer @8.8.8.8 www.g03.sdi.hdm.stuttgart.cloud
$
```

### Note

Examples at DNS Updates with nsupdate

# Bind server ns 1. sdi . hdm st ut t gart . cl oud

---

- Providing DNS info for sdi.hdm-stuttgart.cloud and sub-zones:
  - g01.sdi.hdm-stuttgart.cloud
  - g02.sdi.hdm-stuttgart.cloud
  - ...
- Remote API for per-zone editing

# DNS provider

---

```
provider "dns" {
  update {
    server          = "ns1.sdi.hdm.stuttgart.cloud"
    key_name        = "gok.key."
    key_algorithm  = "hmac-sha512"
    key_secret      = file("../dnsupdateoken.key")
  }
}
```

## Defining an "A" record

---

```
resource "dns_a_record_set" "helloRecord" {
  zone = "${var.dnsSubnetName}." # The dot matters!
  name = hcloud_server.helloServer.name
  addresses = [hcloud_server.helloServer.ipv4_address]
  ttl = 10
}
```

## Related exercises

---

Exercise 12: Creating a host with corresponding DNS entry



# Understanding web certificates

---

[https://www.youtube.com/watch?v=T4Df5\\_cjAs](https://www.youtube.com/watch?v=T4Df5_cjAs)

# Certificate trust level

---

Domain Validated	Fully automated process solely based on DNS / infrastructure challenges.
Organization Validated	Checking organization in question.
Extended Validation	Additional checks i.e. telephone based verification.

# Certificates by Terraform

---

```
provider "acme" {
  server_url = "https://acme-staging-v02.api.letsencrypt.org/directory"
}
resource "tls_private_key" "private_key" { algorithm = "RSA" }
resource "acme_registration" "reg" {
  account_key_pem = tls_private_key.private_key.private_key_pem
  email_address   = "nobody@example.com"
}
resource "acme_certificate" "certificate" {
  ...
  dns_challenge { ... }
}
```

# dns\_challenge provider

---

```
resource "acme_certificate" "certificate" {  
  ...  
  dns_challenge {  
    provider = "route53"  
  }  
}
```

acme DNS provider list:

- acme-dns
- alidns
- ...
- rfc2136
- ...
- zonomi

# rfc2136 provider configuration

---

```
dns_challenge {
  provider = "rfc2136"

  config = {
    RFC2136_NAMESERVER      = "ns1.sdi.hdm.stuttgart.cloud"
    RFC2136_TSIG_ALGORITHM = "hmac-sha512"
    RFC2136_TSIG_KEY        = "goik.key."
    RFC2136_TSIG_SECRET     = file("../dnsupdate.oken.key")
  }
}
```

# Bind server logfile

---

```
... updating zone 'goi.k.sdi.hdmstuttgart.cloud/IN':
  deleting rrsset at '_acme-challenge.goi.k.sdi.hdmstuttgart.cloud' TXT
... updating zone 'goi.k.sdi.hdmstuttgart.cloud/IN':
  adding an RR at '_acme-challenge.goi.k.sdi.hdmstuttgart.cloud' TXT
    "GtJZJZjCZLV6GsQDODCFnY37TmM Ri y8Hw9MeDGhkQ"
... deleting rrsset at ... TXT
... adding an RR ... TXT "eJckW2F43nsf27bzVQjcrTGp_VFeCj2qTVM5Uodg-4"
... deleting an RR at _acme-challenge.goi.k.sdi.hdmstuttgart.cloud TXT
... updating zone ... deleting an RR ... TXT
```

## Related exercises

---

Exercise 13: Creating a web certificate

Exercise 14: Testing your web certificate

Exercise 15: Combining certificate generation and server creation

# Terraform module Documentation

---

- Reuse Configuration with Modules
- What are Terraform Modules and How to Use Them?



## Example: Creating bin/ssh and gen/known\_hosts

main.tf	Generated files
<pre> resource "tls_private_key" "host" {   algorithm = "ED25519" }  resource "hcloud_ssh_key" "loginUser" {   name      = "devops"   public_key = file("~/ssh/id_ed25519.pub") }  resource "hcloud_server" "server" {   name = "www" ... }  resource "local_file" "ssh_script" {   content = templatefile("tpl/ssh.sh", {     ip                = hcloud_server.helloServer.ipv4_address     devopsUsername    = hcloud_ssh_key.loginUser.name   })   filename      = "bin/ssh"   depends_on    = [local_file.known_hosts] } </pre>	<ul style="list-style-type: none"> <li>• Template file <i>tpl/ssh.sh</i></li> <li>• <i>gen/known_hosts</i>  <pre> 94.130.229.221 ssh-ed25519 AAAAC3NzaC1lZDI1N... www.g03.sdi.hdmstuttgart.cloud ssh-ed25519 ... </pre> </li> <li>• <b>bin/ssh</b>  <pre> #!/usr/bin/env bash  GEN_DIR=\$(dirname "\$0")/../gen  ssh -o UserKnownHostsFile="\$GEN_DIR/known_hosts" \     devops@www.g03.sdi.hdmstuttgart.cloud "\$@" </pre> </li> </ul>

# Local file generation by module

main.tf	Using a module
<pre>resource "local_file" "known_hosts" {   ... }  resource "local_file" "ssh_script" {   ... }  resource "local_file" "scp_script" {   ... }</pre>	<pre>module "local_files" {   source = "../modules/local_files"    ipv4      = hcloud_server.webServer.ipv4_address   dnsZone   = var.dnsZone   hostNames = ["www", "cloud"]   loginUser = hcloud_ssh_key.loginUser.name   hostKey   = tls_private_key.host.public_key_openssh }</pre>

# Module implementation

---

```
variable "ipv4" {  
  description = "The server's IPV4 address e.g. '141.62.1.5'"  
  type        = string  
}
```

```
variable "hostNames" {  
  description = "Set of unique local host names e.g. [\"www\", \"cloud\"]"  
  type        = list  
}
```

```
...  
resource "local_file" "known_hosts" {...
```

# Careful: local vs. parent context

---

Switching between parent and child module context by `$(path.module)`:

```
resource "local_file" "ssh_script" {
  content = templatefile("${path.module}/tpl/ssh.sh", {
    serverFqdn      = "${var.ostNames}. ${var.dnsZone}"
    devopsUsername = var.loginUser
  })
  filename      = "bin/ssh"
  file_permission = "755"
  depends_on    = [local_file.known_hosts]
}
```

# Related exercises

---

Exercise 16: A module for local file generation

# Loop documentation

---

- Terraform tips & tricks: loops, if-statements, and gotchas
- Strings and Templates
- The count Meta-Argument

# Using count

Defining 10 server

```
resource "hcloud_server" "server" {  
  count      = 10  
  name      = "www-${count.index}"  
  user_data = local_file.user_data[count.index].content  
  ...  
}
```

10 corresponding A-records

```
resource "dns_a_record_set" "dnsRecordSet" {  
  count      = 10  
  zone      = "g03.sdi.hdm.stuttgart.cloud."  
  name      = hcloud_server.server[count.index].name  
  addresses = [hcloud_server.server[count.index].ipv4_address]  
}
```

## Related exercises

---

Exercise 17: Creating a fixed number of servers