HCL
Struktur

./terraform

./main.tf

./terraform

./main.tf
./outputs.tf
./variables.tf

./terraform

./main.tf
./outputs.tf
./variables.tf
./ec2.tf
./subnet.tf
./vpc.tf
…

./terraform

./main.tf
./outputs.tf
./variables.tf
./modules

./us7
./us7/ec2.tf
./us7/subnet.tf
./us7variables.tf
./us7/…

./terraform

./main.tf
./outputs.tf
./variables.tf
./modules

./vm/ec2.tf
./vm/variable.tf
…

./net/vpc.tf
./net/subnet.tf
./net/variable.tf
…

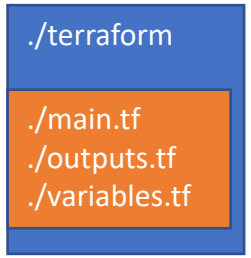
plain

modules

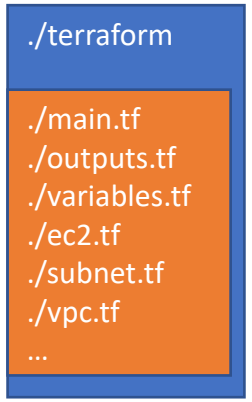
./terraform

./main.tf

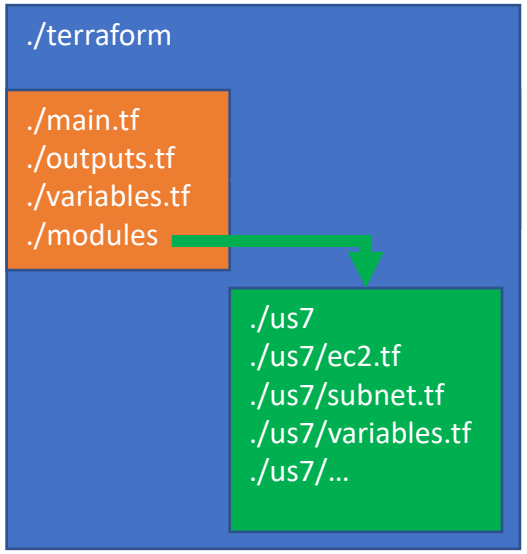
```
variable "aws_cli_profile" {
  description = "aws profile"
  default     = "devops"
}

…

# aws
provider "aws" {
  profile = var.aws_cli_profile
  region  = var.aws_region
}

# vpc
resource "aws_vpc" "vpc_devops" {
  cidr_block           = var.cidr_vpc
  enable_dns_support   = true
  enable_dns_hostnames = true
  tags = {
    Name = join("_", [var.namespace, "vpc"])
  }
}
…

output "vpc" {
  description = "vpc id"
  value       = aws_vpc.vpc_devops.id
}

output "vpc-info" {
  value = join(" : ", [aws_vpc.vpc_devops.id, aws_vpc.vpc_devops.tags_all["Name"]])
}
```



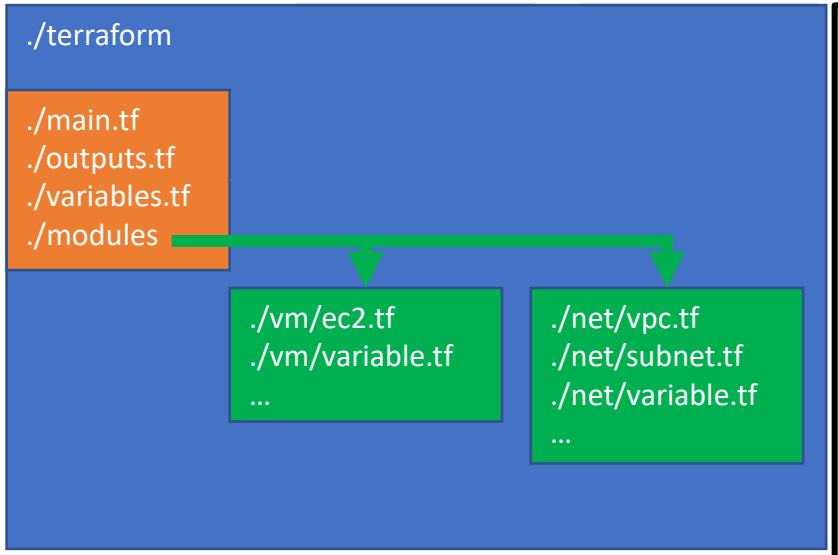

```
# aws
provider "aws" {
  profile = var.aws_cli_profile
  region  = var.aws_region
}

# vpc
resource "aws_vpc" "vpc_devops" {
  cidr_block = var.cidr_vpc
  enable_dns_support   = true
  enable_dns_hostnames = true
  tags = {
    Name = join("_",[var.namespace, "vpc"])
  }
}


# public subnet
resource "aws_subnet" "subnet_public_devops" {
  vpc_id = aws_vpc.vpc_devops.id
  cidr_block = var.cidr_subnet
  map_public_ip_on_launch = "true"
  availability_zone = var.availability_zone
  tags = {
    Name = join("_",[var.namespace, "subnet"])
  }
}
```




```
terraform {
  required_version = ">= 0.12"
  backend "local" {
    path = "state/terraform.tfstate"
  }
}

#aws
provider "aws" {
  profile = var.profile
  region  = var.region
}
```




```
# version/local backend
terraform {
  required_version = ">= 0.12"
  backend "local" {
    path = "state/terraform.tfstate"
  }
}

# aws client
provider "aws" {
  profile = var.profile
  region  = var.region
}

# module call
module "stack" {
  source = "./modules/stack"

  vpc_cidr = var.vpc_cidr

  subnet_cidrs = var.subnet_cidrs
  av_zones = var.av_zones

  namespace = var.namespace

  ssh_credentials = var.ssh_credentials

  ec2 = var.ec2
}
```




```
# version/local backend
terraform {
  required_version = ">= 0.12"
  backend "local" {
    path = "state/terraform.tfstate"
  }
}

# aws client
provider "aws" {
  profile = var.profile
  region  = var.region
}

# module call
module „net" {
  source = "./modules/net"
  vpc_cidr = „128.0.0.0/16"
  subnet_cidrs = ["128.0.1.0/24", "128.0.2.0/24", "128.0.3.0/24"]
  av_zones = ["eu-central-1a", "eu-central-1b", "eu-central-1c"]
}

module „vm1" {
  source = "./modules/vm"
…
}

module „vm2" {
  source = "./modules/vm"
…
}
```

./outputs.tf:

```
output "vpc-info" {
    value = join(" : ",[module.stack.vpc_id, module.stack.vpc_cidr_block, module.stack.vpc_tags_all["Name"]])
}

output "subnets" {
    value = module.stack.subnet_cidr_blocks
}

output "pub_ec2_public_ips" {
    value = module.stack.ec2_public_ips
}

output "elb_fqdn" {
    value = module.stack.elb_fqdn
}

output "ec2_public_fqdns" {
    value = module.stack.ec2_public_fqdns
}

resource "local_file" "inventory" {
  content = templatefile(var.ansible["ansible_inv_template"],
    {
      group_name = var.namespace
      public_ips = module.stack.ec2_public_ips
      public_fqdns = module.stack.ec2_public_fqdns
    }
  )
  filename = var.ansible["ansible_inv"]
}
```

- Addressierung via module.<modulname>.<resource>.<attribut>

HCL Loop

count

```
# ec2
resource "aws_instance" "pub" {
  ami           = var.pub_instance_ami
  instance_type = var.pub_instance_type
  subnet_id = aws_subnet.pub.id
  vpc_security_group_ids = [aws_security_group.pub.id]
  key_name = aws_key_pair.global.key_name

  count = var.pub_instance_count

  tags = {
    Name = join("_",[var.team, "pub_ec2", count.index ])
  }
```

- Meta-Arguments / Loops
  - count
    - Loop über Resource
  - for_each
    - Loop über Resource mit Inline Block
  - for
    - Loop über Lists/Maps

- Keine nested Loops …
- Geht aber auch ohne …

```
# ec2
resource "aws_instance" "pub" {
  ami           = var.pub_instance_ami
  instance_type = var.pub_instance_type
  subnet_id = aws_subnet.pub.id
  vpc_security_group_ids = [aws_security_group.pub.id]
  key_name = aws_key_pair.global.key_name



  count = var.pub_instance_count



  tags = {
    Name = join("_",[var.team, "pub_ec2", count.index])
  }
```

- Set count to 3
- Loop über Resource
  - 3 Durchläufe
    - 3 VM

- Pro Loop wird der Schleifenindex incrementiert
  - Zugriff via index-Methode des count-Objektes
- 0
  - 1
    - 2

- var.pub_instance_count = 3
  - count = 3
    - count.index
      - Schleifenindex (beginnend mit 0)

```
# ec2
resource "aws_instance" "dev" {
  ami           = var.instance_ami
  instance_type = var.instance_type


  count = length(var.subnet_cidrs) * var.instance_count


  subnet_id = element(aws_subnet.dev.*.id,count.index)


  vpc_security_group_ids = [aws_security_group.dev.id]
  key_name = aws_key_pair.dev.key_name
  user_data = file(var.cloud_init_script)
  tags = {
    Name = join("_",[var.namespace, "ec2", count.index, element(var.av_zones,count.index) ])
  }
}
```

- Set count to 3
  - 3 (anzahl subnetze) * 1 (anzahl instanzen pro subnet) = 3
- Loop über Resource
  - 3 Durchläufe
    - 1 VM pro Subnet

- Indizierter Zugriff auf aws_subnet.dev.*.id via count.index
  - aws_subnet.dev.0.id
  - aws_subnet.dev.1.id
  - aws_subnet.dev.2.id

```
variable "subnet_cidrs" {
  description = "pub_subnet_cidrs"
  type = list
  default = ["128.0.1.0/24", "128.0.2.0/24", "128.0.3.0/24"]
}

variable "instance_count" {
  type = number
  description = "count of instances to build per subnet"
  default = 10
}
```

```
# terraform state list -state=state/terraform.tfstate
local_file.inventory
module.stack.aws_cloudwatch_metric_alarm.cpuutilization[0]
module.stack.aws_cloudwatch_metric_alarm.cpuutilization[1]
module.stack.aws_cloudwatch_metric_alarm.cpuutilization[2]
module.stack.aws_cloudwatch_metric_alarm.statuscheckfailed[0]
module.stack.aws_cloudwatch_metric_alarm.statuscheckfailed[1]
module.stack.aws_cloudwatch_metric_alarm.statuscheckfailed[2]
module.stack.aws_elb.dev
module.stack.aws_instance.dev[0]
module.stack.aws_instance.dev[1]
module.stack.aws_instance.dev[2]
module.stack.aws_internet_gateway.dev
module.stack.aws_key_pair.dev
module.stack.aws_route_table.dev
module.stack.aws_route_table_association.dev[0]
module.stack.aws_route_table_association.dev[1]
module.stack.aws_route_table_association.dev[2]
module.stack.aws_security_group.dev
```

subnet_id = element(aws_subnet.dev[*].id,count.index)

module.stack.aws_subnet.dev[0]
module.stack.aws_subnet.dev[1]
module.stack.aws_subnet.dev[2]

```
module.stack.aws_vpc.dev
```

HCL
Variables
Maps

variables.tf:

```
# ec2
variable "ec2" {
  description = "ec2 attributes"
  type = map
  default = {
    "instance_ami"  = "ami-0a02ee601d742e89f"
    "instance_type"  = "t2.nano"
    "instance_count" = 1
    "ebs_device" = "/dev/sdb"
    "ebs_vol_size" = 1
    "ebs_vol_type" = "gp2"
    "cloud_init_file" = "./files/cloud_init/cloud_init_ansible_user.yml"
  }
}
```

main.tf:

```
# module call
module "stack" {
  source = "./modules/stack"

…

  ec2 = var.ec2
}
```

./modules/stack/ec2.tf:

```
# ec2
resource "aws_instance" "dev" {
  ami           = var.ec2["instance_ami"]
  instance_type = var.ec2["instance_type"]

  …
}
```