Untangling Infrastructure Code Through Refactoring

DevOpsDays Chicago

@nellshamrell

Who Am I?

Nell Shamrell-Harrington

- Software Engineer at Chef
- Former O'Fallon, IL resident (**Southern IL ftw!**)
- @nellshamrell
- nshamrell@chef.io

Why should I **refactor**
my **infrastructure code?**

Why Refactor?

- Add a feature

Why Refactor?

- Add a feature
- Fix a bug

Why Refactor?

- Add a feature
- Fix a bug
- Improve the design

Why Refactor?

- Add a feature
- Fix a bug
- Improve the design
- Optimize resource usage

Let’s go through
an **example!**

Refactoring a **Terraform** config

supermarket-terraform

Today, we will refactor supermarket-terraform

http://github.com/nellshamrell/supermarket-terraform

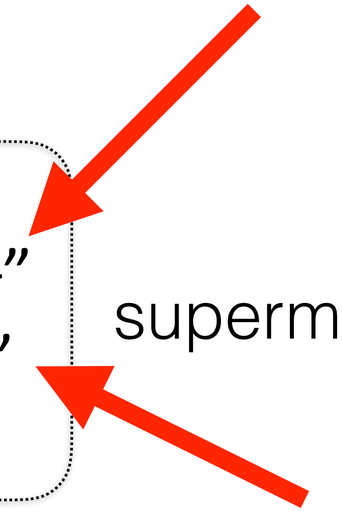
supermarket-terraform

```
provider “aws” {
  access_key = “${var.access_key}”
  secret_key = “${var.secret_key}”
}
```

supermarket-cluster.tf

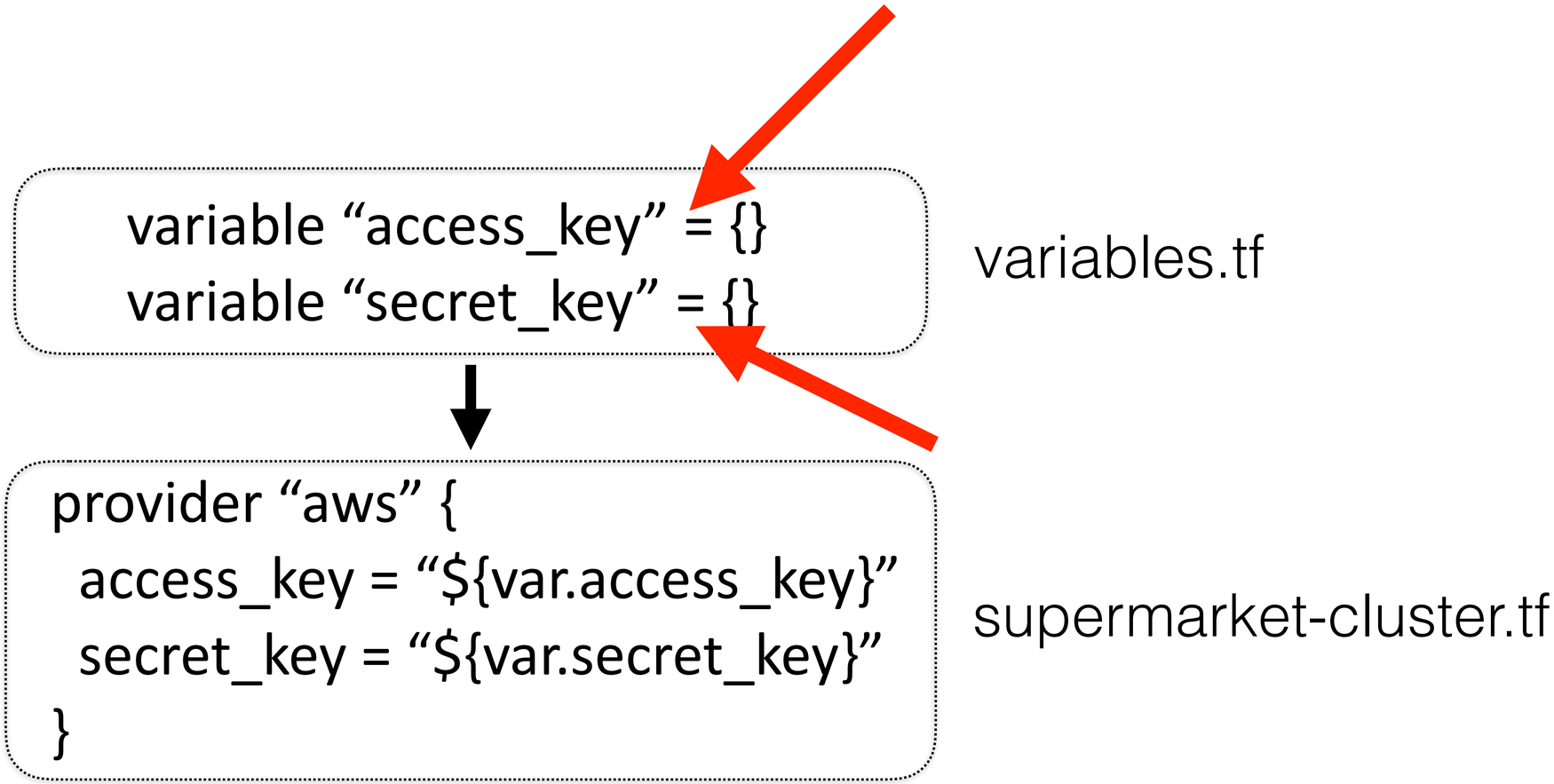
supermarket-terraform

```
provider “aws” {
  access_key = “${var.access_key}”
  secret_key = “${var.secret_key}”
}
```

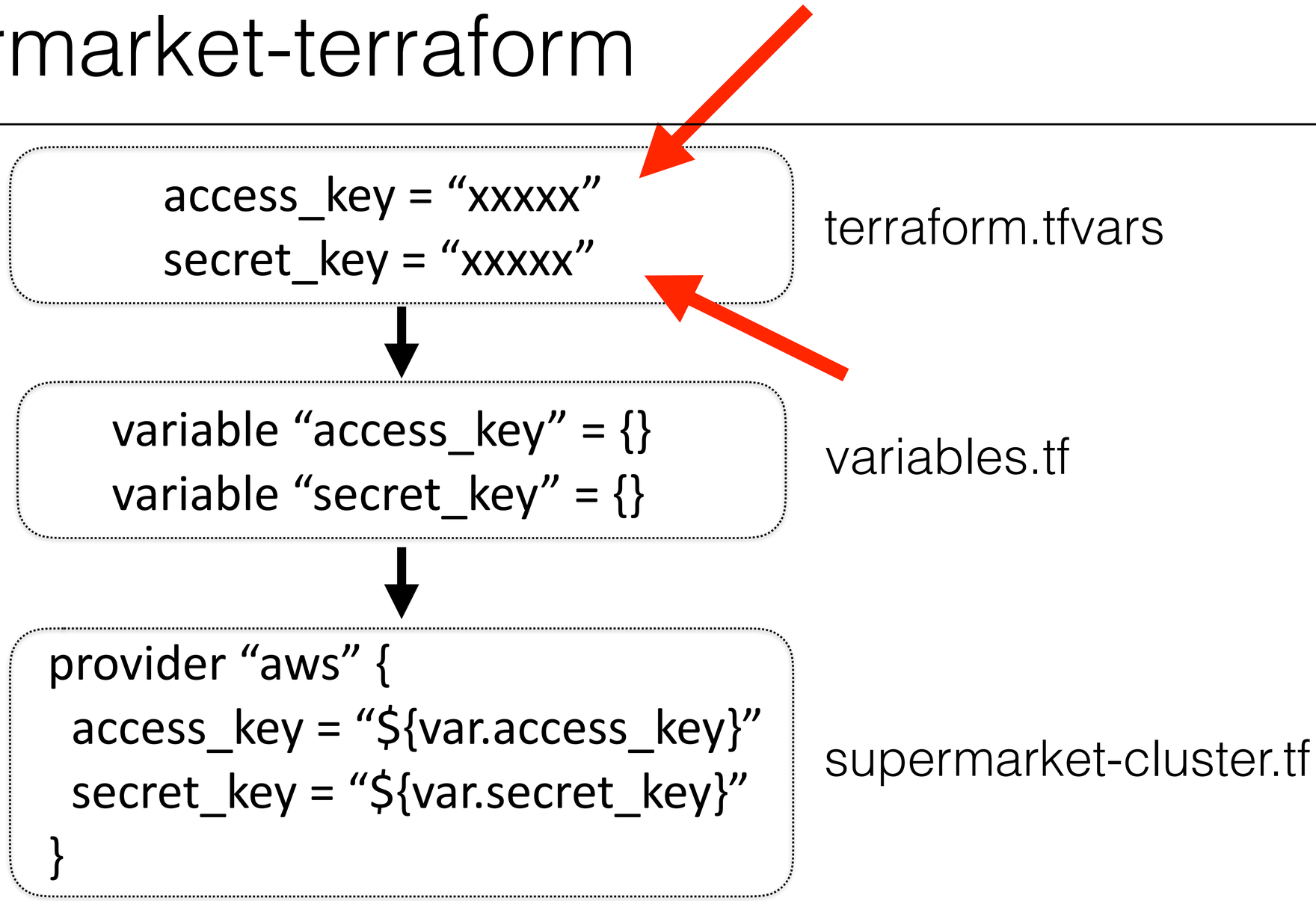


supermarket-cluster.tf

supermarket-terraform



supermarket-terraform



Wait…should you use
a credentials file?

You can…but for the
sake of the example,
let's supply them inline

supermarket-terraform

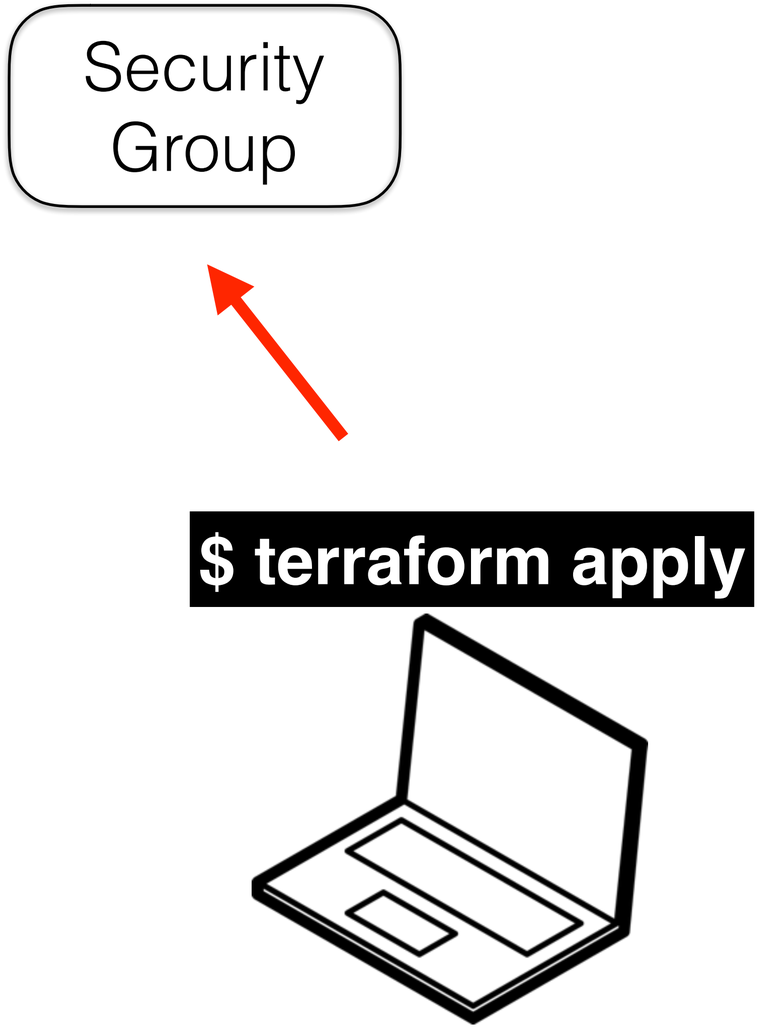
```
provider “aws” {
  access_key = “${var.access_key}”
  secret_key = “${var.secret_key}”
}
```

supermarket-cluster.tf

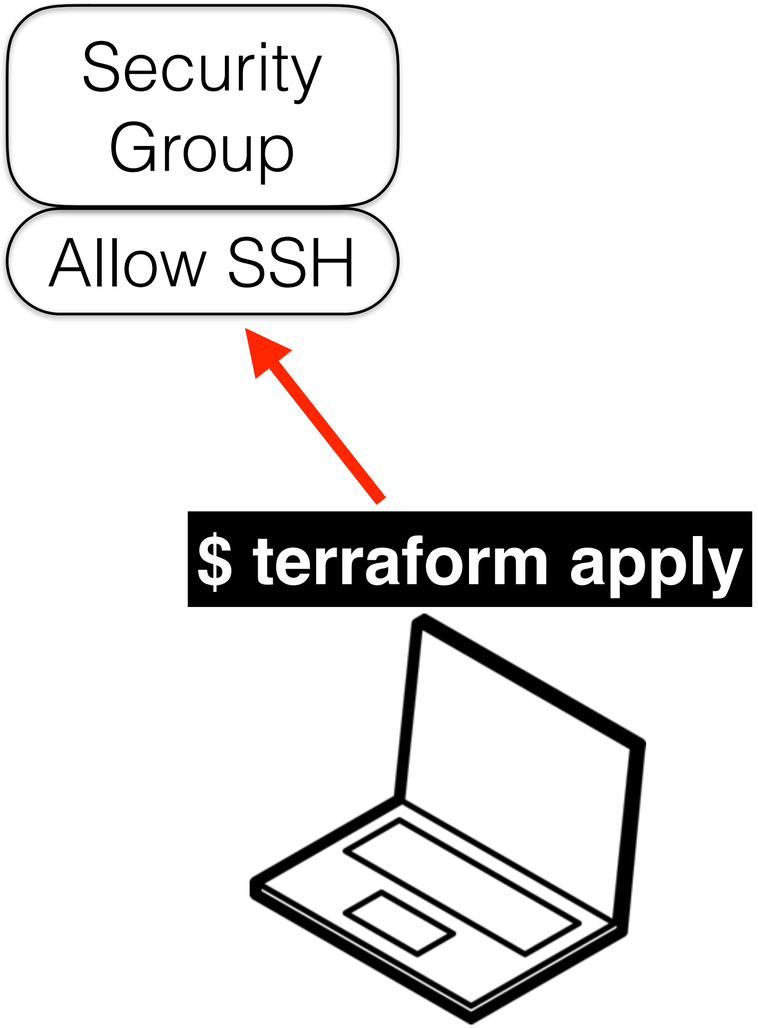
supermarket-cluster.tf

$ terraform apply

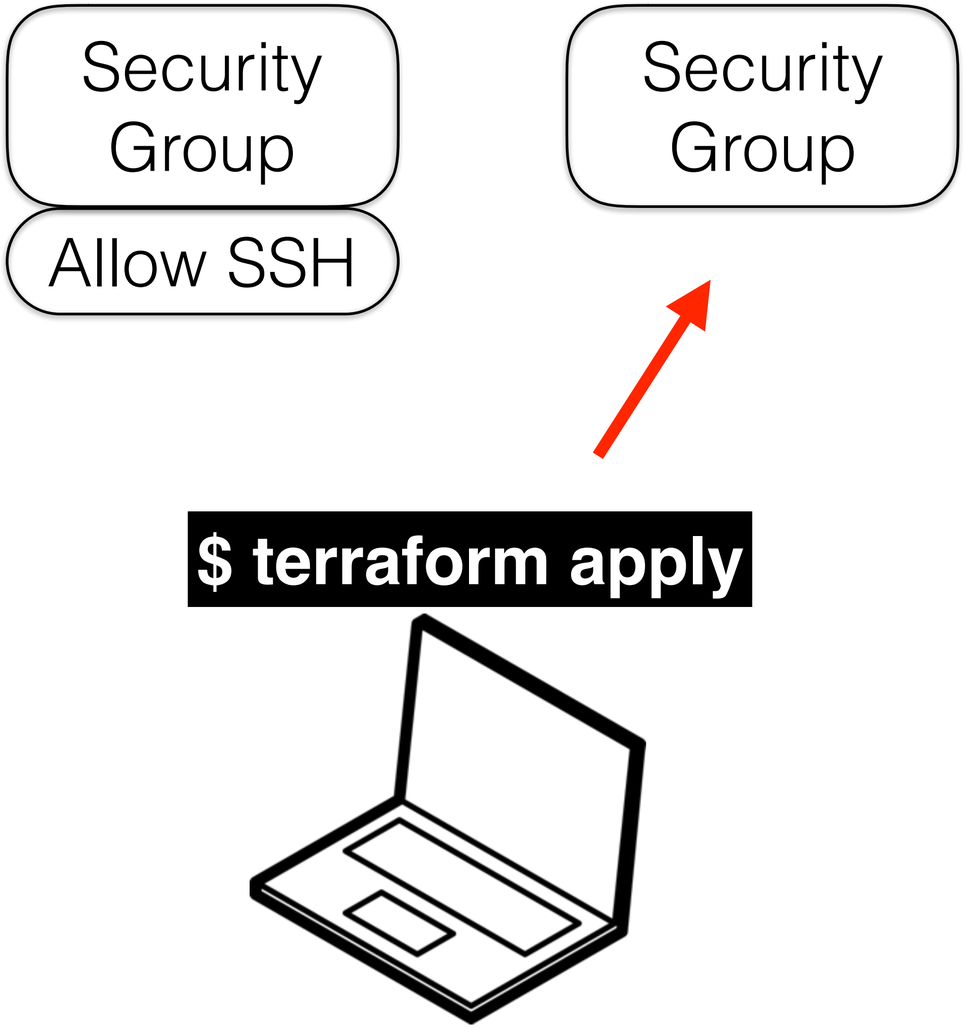
supermarket-cluster.tf



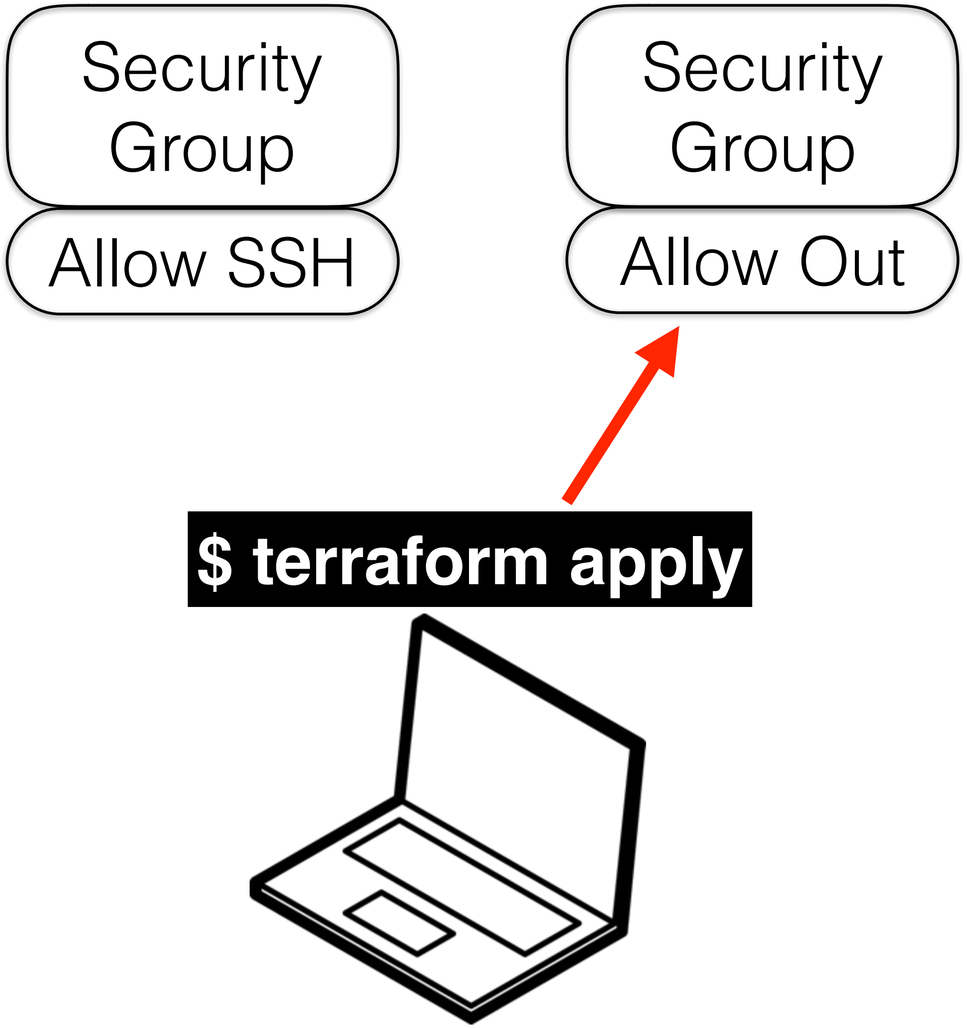
supermarket-cluster.tf



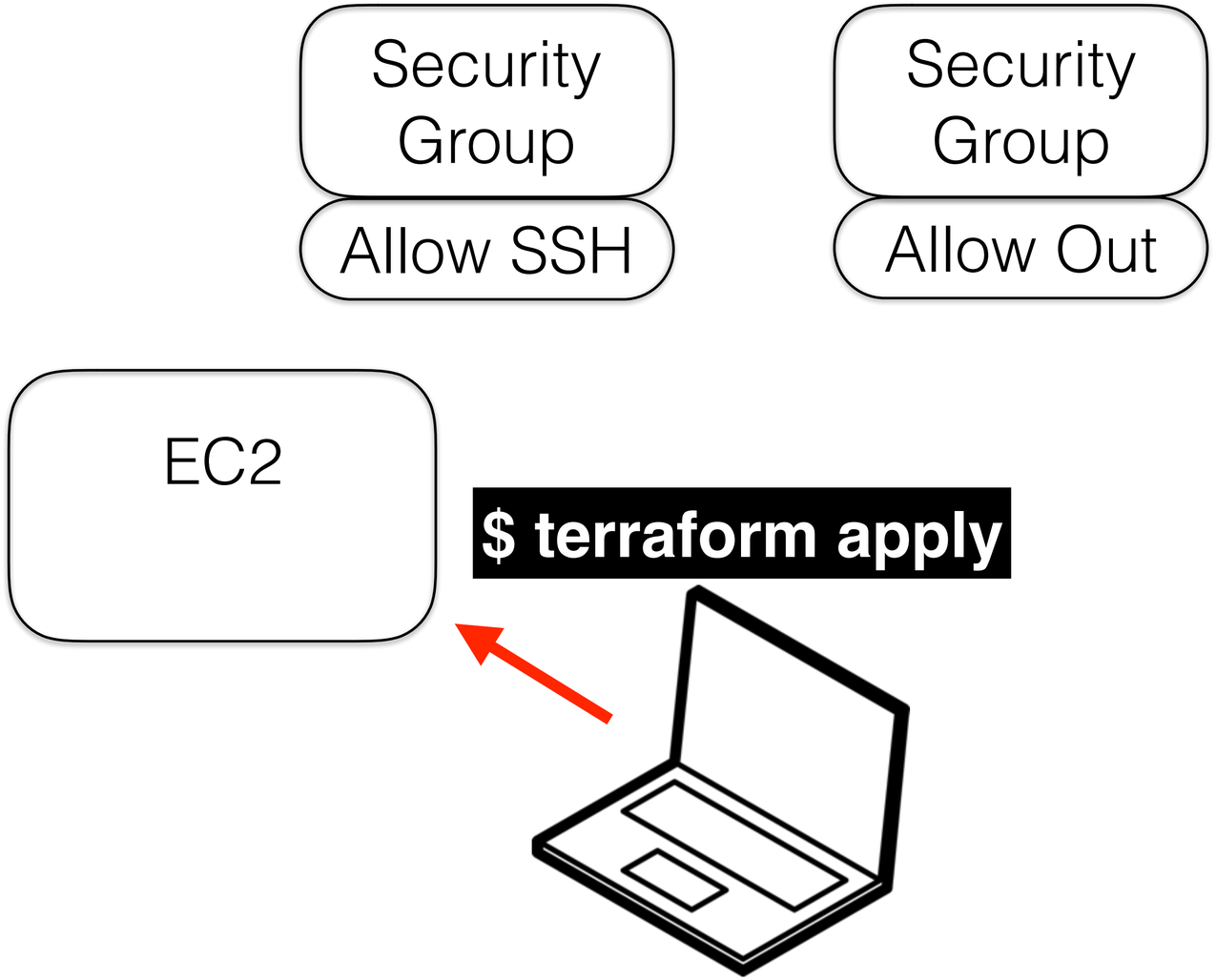
supermarket-cluster.tf



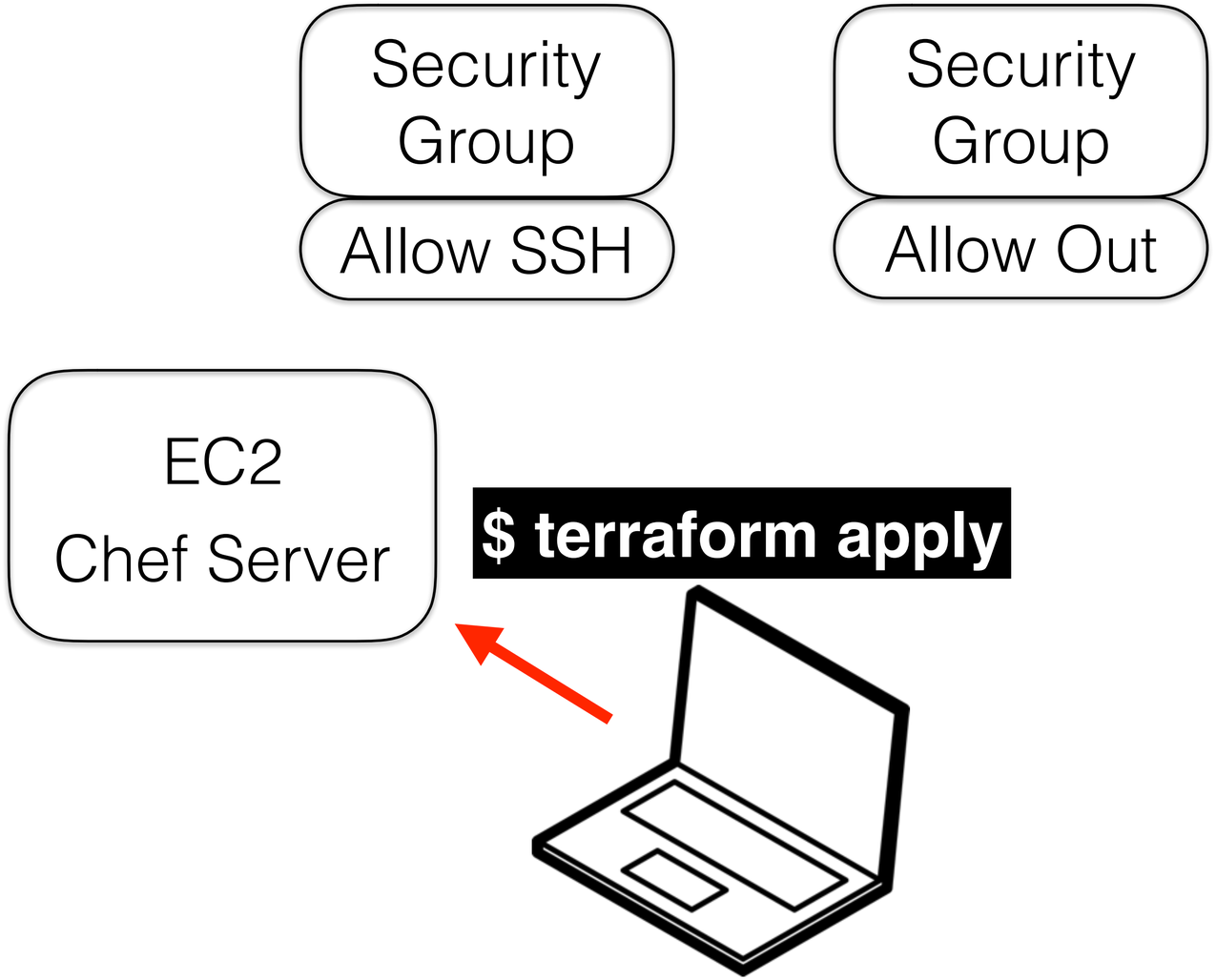
supermarket-cluster.tf



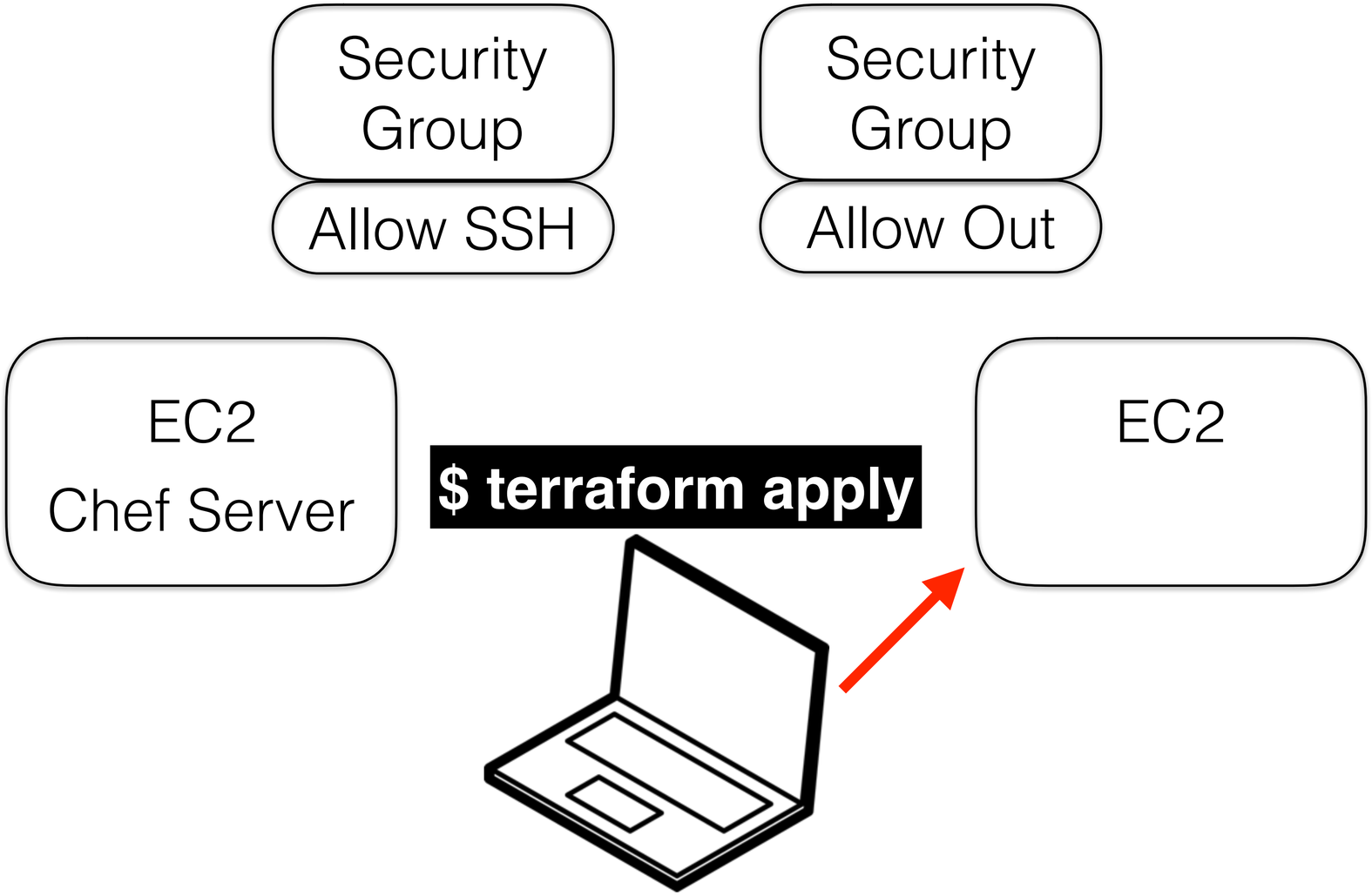
supermarket-cluster.tf



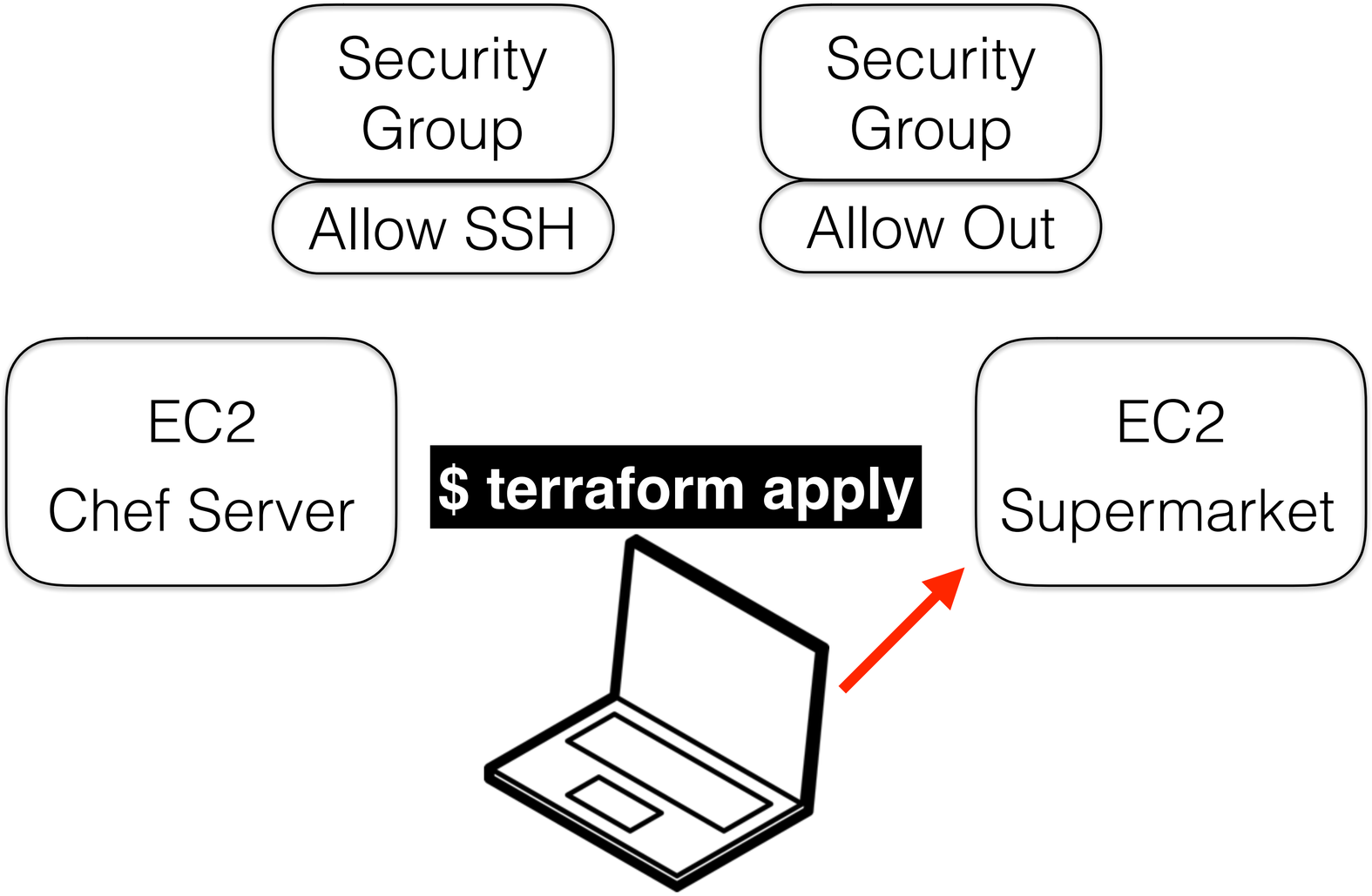
supermarket-cluster.tf



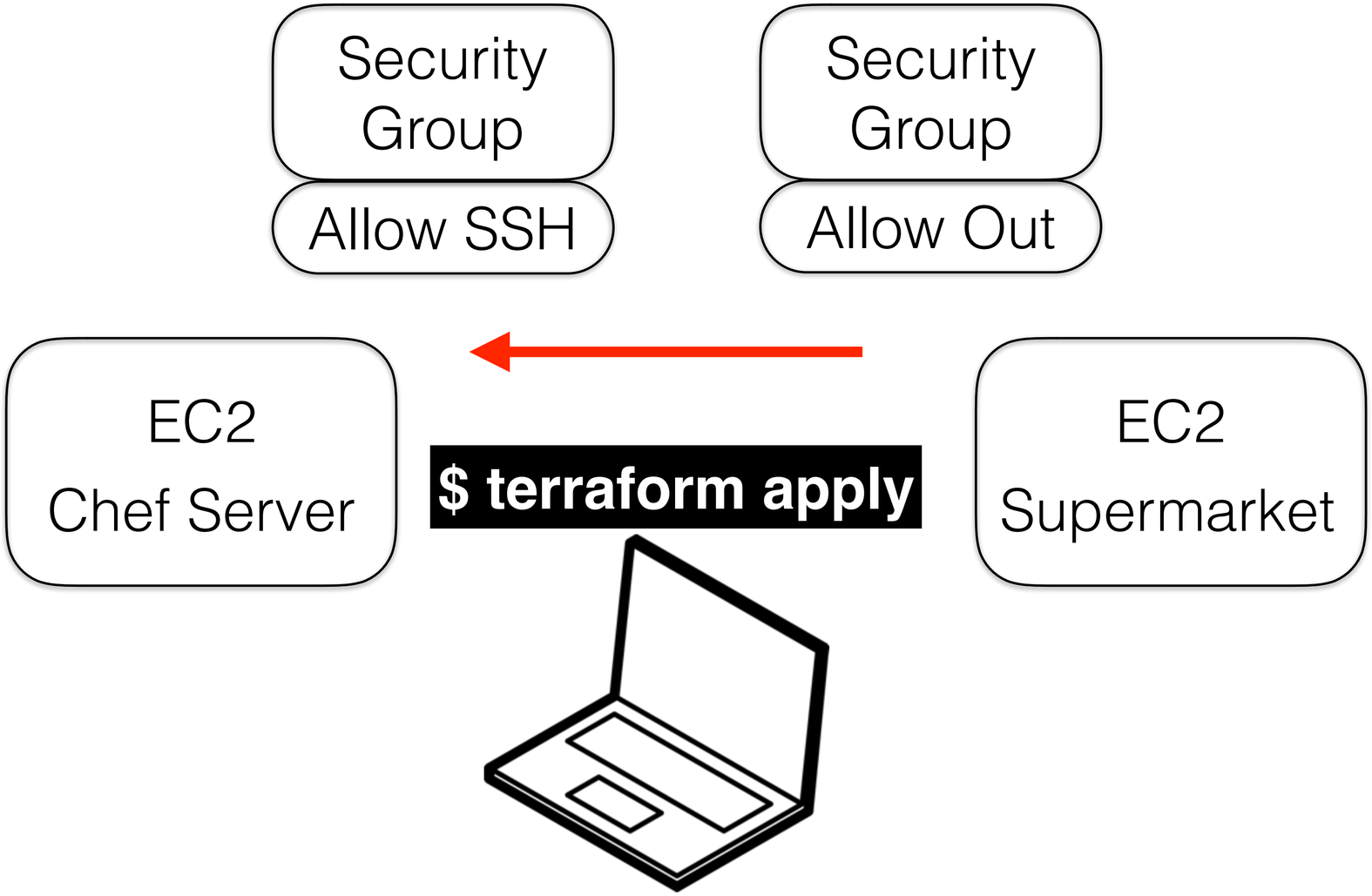
supermarket-cluster.tf



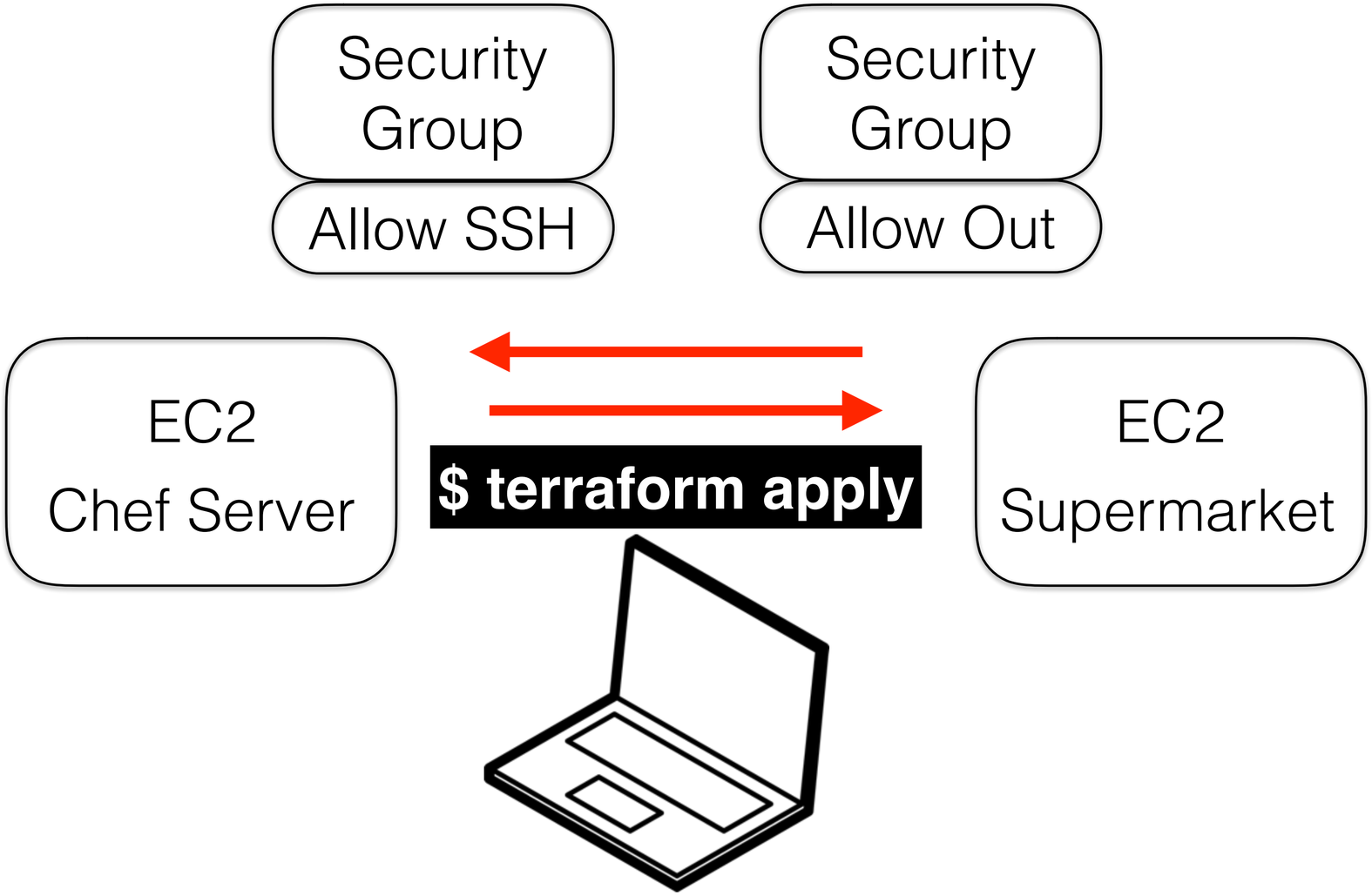
supermarket-cluster.tf



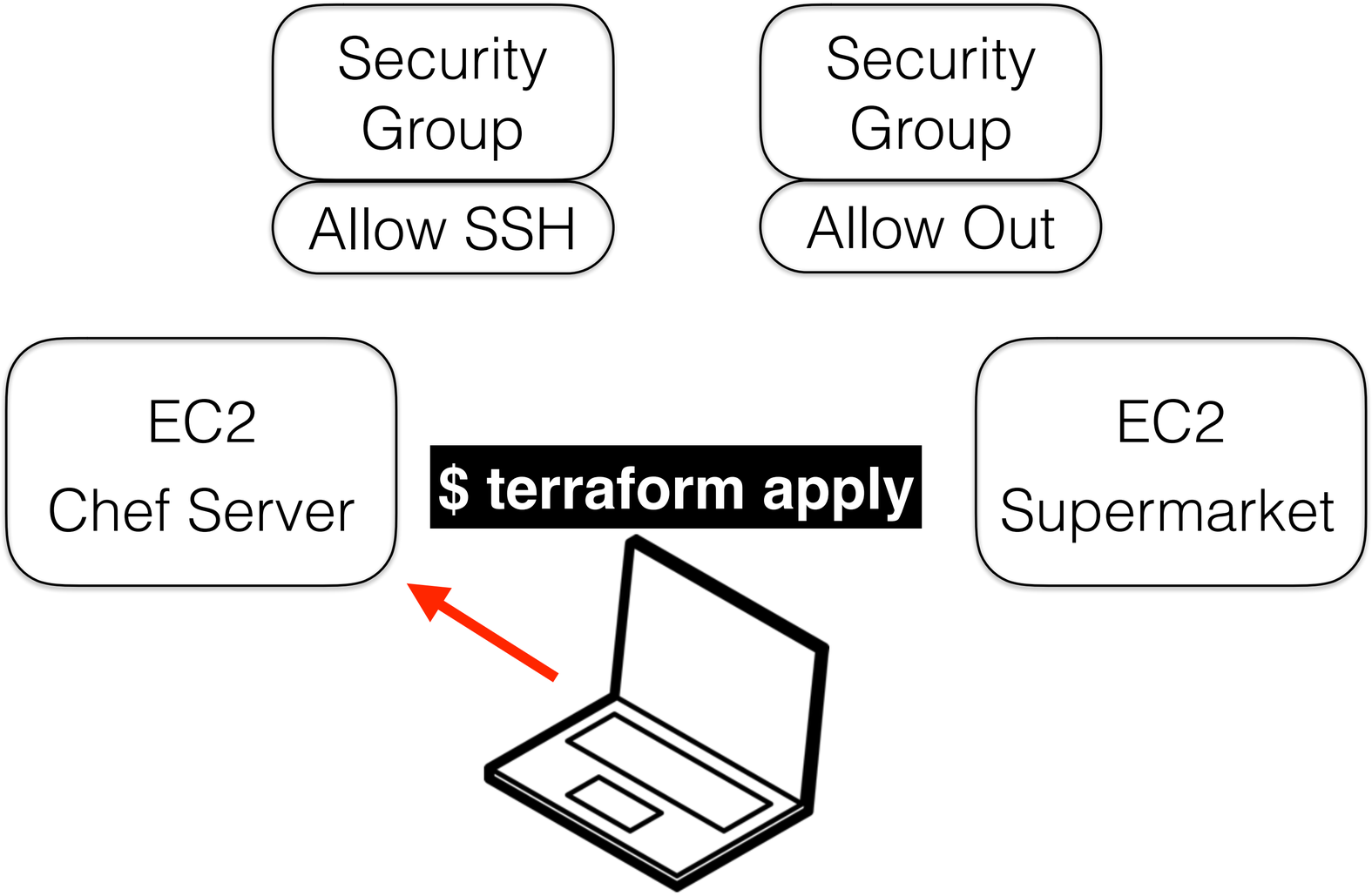
supermarket-cluster.tf



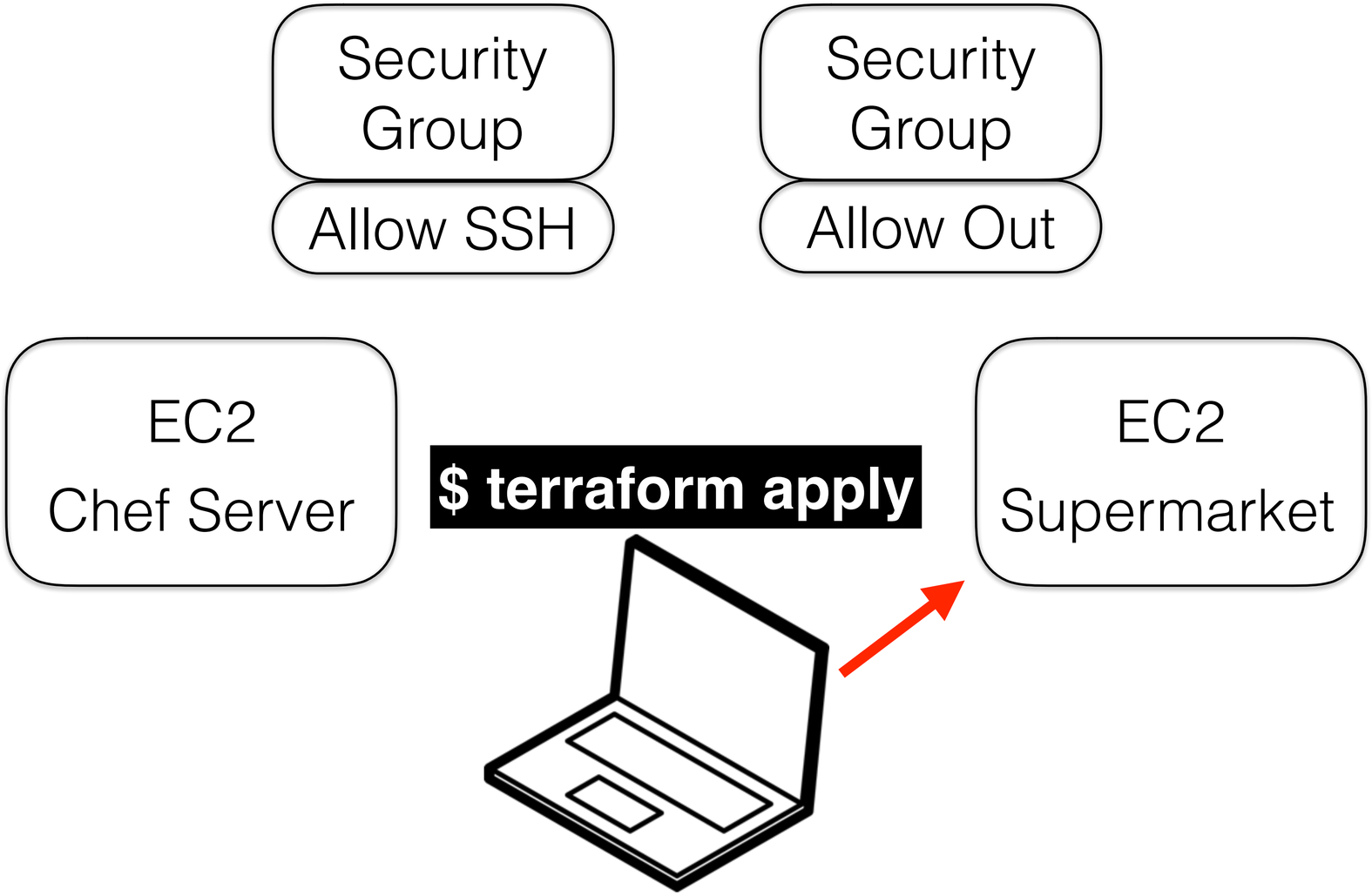
supermarket-cluster.tf



supermarket-cluster.tf



supermarket-cluster.tf



Why Refactor?

Hypothetical:
We are using **too many**
AWS Security Groups

Why Refactor?

We need this config
to create only
one security group

How to Refactor?

Two Approaches

Source: Working Effectively with Legacy Code

How to Refactor?

Two Approaches

- Edit and Pray

Source: Working Effectively with Legacy Code

How to Refactor?

Two Approaches

- Edit and Pray
- Cover and Modify

Source: Working Effectively with Legacy Code

Confidence in code
without tests is
false confidence

What **code** is **intended to do**
is much **less** important
than what it **actually does**

Do I **have** to **add tests**
for the **entire thing?**

No.

The **point** of adding tests is
to not **make** things worse

And to start making
the code **better**
here and **now**

How can we **test Terraform?**

How Can We Test Terraform?

- **Test Kitchen** (http://kitchen.ci)

How Can We Test Terraform?

- Test Kitchen (http://kitchen.ci)

- **Kitchen Terraform** (http://github.com/newcontext/kitchen-terraform)

.kitchen.yml

```
driver:
  name: terraform

provisioner:
  name: terraform
  variable_files: terraform.tfvars

transport:
  name: ssh
  ssh_key: ~/path/to/your/aws/key

platforms:
  - name: ubuntu

suites
  - name: default
```

.kitchen.yml

```
driver:
  name: terraform

provisioner:
  name: terraform
  variable_files: terraform.tfvars

transport:
  name: ssh
  ssh_key: ~/path/to/your/aws/key

platforms:
  - name: ubuntu

suites
  - name: default
```

.kitchen.yml

```
driver:
  name: terraform

provisioner:
  name: terraform
  variable_files: terraform.tfvars

transport:
  name: ssh
  ssh_key: ~/path/to/your/aws/key

platforms:
  - name: ubuntu

suites
  - name: default
```

.kitchen.yml

```
driver:
  name: terraform

provisioner:
  name: terraform
  variable_files: terraform.tfvars

transport:
  name: ssh
  ssh_key: ~/path/to/your/aws/key

platforms:
  - name: ubuntu

suites
  - name: default
```

.kitchen.yml

```
driver:
  name: terraform

provisioner:
  name: terraform
  variable_files: terraform.tfvars

transport:
  name: ssh
  ssh_key: ~/path/to/your/aws/key

platforms:
  - name: ubuntu

suites
  - name: default
```

.kitchen.yml

```
driver:
  name: terraform

provisioner:
  name: terraform
  variable_files: terraform.tfvars

transport:
  name: ssh
  ssh_key: ~/path/to/your/aws/key

platforms:
  - name: ubuntu

suites
  - name: default
```

Test Kitchen Boilerplate

How Can We Test Terraform?

- Test Kitchen (http://kitchen.ci)
- Kitchen Terraform (http://github.com/newcontext/kitchen-terraform)
- **Inspec** (http://chef.io/inspec)

Now that we have
our **tools**…

Let’s start from
a **clean** slate

supermarket-cluster.tf

```
#provider “aws” {
#  access_key = “${var.access_key}”
#  secret_key = “${var.secret_key}”
#  region = “${var.region}”
#}

#resource “aws_security_group” “allow-ssh” {
#  name = “${var.user_name}-allow-ssh”
#  tags = {
#    Name = “${var.user_name} Allow All SSH”
#  }
…
```

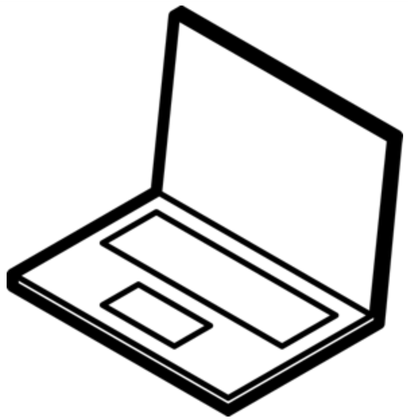
supermarket-cluster.tf

$ terraform apply

supermarket-cluster.tf

Nothing happens!

$ terraform apply



First, we need the **provider**

supermarket-cluster.tf

```
provider “aws” {
  access_key = “${var.access_key}”
  secret_key = “${var.secret_key}”
  region = “${var.region}”
}

#resource “aws_security_group” “allow-ssh” {
#  name = “${var.user_name}-allow-ssh”
#  tags = {
#    Name = “${var.user_name} Allow All SSH”
#  }
…
```

We also need **actual**
AWS instances

Including **both** our
Chef Server…

supermarket-cluster.tf

```
...
#resource “aws_instance” “chef_server” {
#  ami = “${var.ami}”
#  instance_type = “${var.instance_type}”
#  key_name = “${var.key_name}”
#  tags {
#    Name = “dev-chef-server”
#  }
#  security_groups = [“${aws_security_group.allow-ssh.name}”,
#                     “${aws_security_group.allow-out.name}”]
#  (...)
#}
...
```

This is the Chef Server

supermarket-cluster.tf

```
…
resource “aws_instance” “chef_server” {
  ami = “${var.ami}”
  instance_type = “${var.instance_type}”
  key_name = “${var.key_name}”
  tags {
    Name = “dev-chef-server”
  }
  security_groups = [“${aws_security_group.allow-ssh.name}”,
                     “${aws_security_group.allow-out.name}”]
  (…)
}
…
```

This is the Chef Server

supermarket-cluster.tf

$ terraform apply

supermarket-cluster.tf

EC2
Chef Server

$ terraform apply

And our **supermarket server**

supermarket-cluster.tf

```
...
#resource “aws_instance” “supermarket_server” {
#  ami = “${var.ami}”
#  instance_type = “${var.instance_type}”
#  key_name = “${var.key_name}”
#  tags {
#    Name = “dev-supermarket-server”
#  }
#  security_groups = [“${aws_security_group.allow-ssh.name}”,
#                     “${aws_security_group.allow-out.name}”]
#  (...)
#}
...
```

This is the Supermarket Server

supermarket-cluster.tf

```
…
resource “aws_instance” “supermarket_server” {
  ami = “${var.ami}”
  instance_type = “${var.instance_type}”
  key_name = “${var.key_name}”
  tags {
    Name = “dev-supermarket-server”
  }
  security_groups = [“${aws_security_group.allow-ssh.name}”,
                     “${aws_security_group.allow-out.name}”]
  (…)
}
…
```

This is the Supermarket Server

supermarket-cluster.tf

We also need at least
one security group

supermarket-cluster.tf

```
#resource “aws_security_group” “allow-ssh” {
#  name = “${var.user_name}-allow-ssh”
#  tags = {
#    Name = “${var.user_name} Allow all ssh”
#  }
#}

#resource “aws_security_group_rule” “allow-ssh” {
#  type = “ingress”
#  from_port = 22
#  to_port = 22
  …
#}
```

supermarket-cluster.tf

```
resource “aws_security_group” “allow-ssh” {
  name = “${var.user_name}-allow-ssh”
  tags = {
    Name = “${var.user_name} Allow all ssh”
  }
}

#resource “aws_security_group_rule” “allow-ssh” {
#  type = “ingress”
#  from_port = 22
#  to_port = 22
  …
#}
```

supermarket-cluster.tf

Security Group

EC2 Chef Server

$ terraform apply

EC2 Supermarket

kitchen-terraform
needs to be able to
ssh into our **instances**

We need a **security group rule**

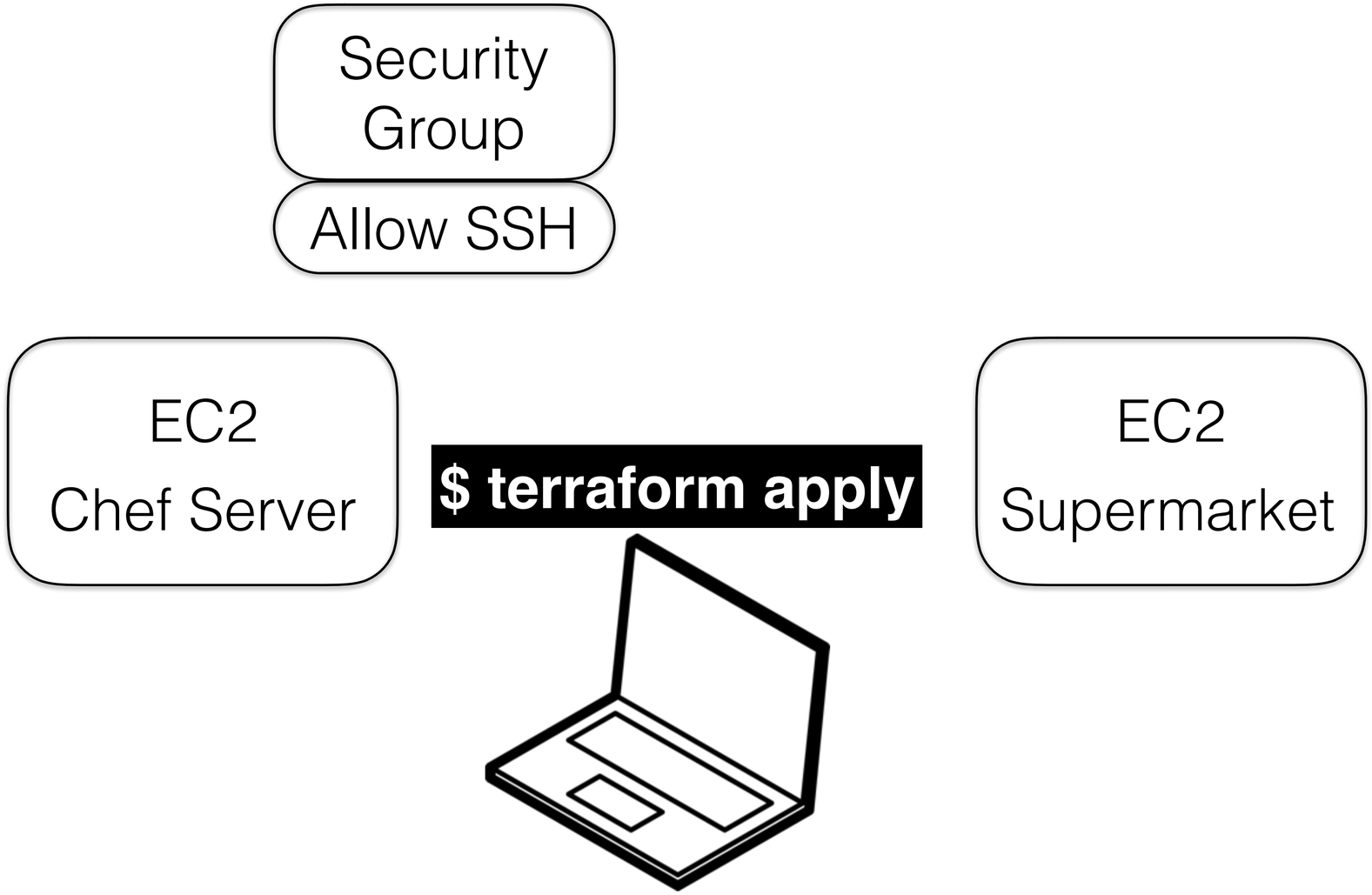
supermarket-cluster.tf

```
resource “aws_security_group” “allow-ssh” {
  name = “${var.user_name}-allow-ssh”
  tags = {
    Name = “${var.user_name} Allow all ssh”
  }
}

#resource “aws_security_group_rule” “allow-ssh” {
# type = “ingress”
# from_port = 22
# to_port = 22
  …
#}
```

supermarket-cluster.tf

```
resource “aws_security_group” “allow-ssh” {
  name = “${var.user_name}-allow-ssh”
  tags = {
    Name = “${var.user_name} Allow all ssh”
  }
}

resource “aws_security_group_rule” “allow-ssh” {
 type = “ingress”
 from_port = 22
 to_port = 22
  …
}
```

supermarket-cluster.tf



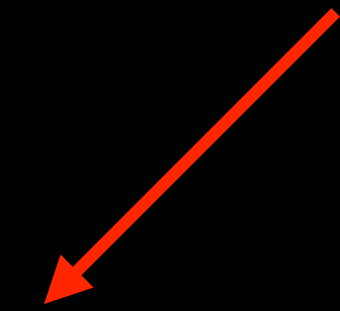
Now, let’s create
our **test cluster**

$ kitchen converge

$ kitchen converge

Like running terraform apply

```
$ kitchen converge
(...)
>>>>>>> ------Exception-------
>>>>>>> Class: Kitchen::ActionFailed
>>>>>>> Message: 1 actions failed.
>>>>>>>     Converge failed on instance
<default-ubuntu>.
>>>>>>> Please see .kitchen/logs/kitchen.log for
more details
```



.kitchen/logs/default-ubuntu.log

---- Begin output of terraform validate /root/supermarket-terraform-2 ----
STDOUT:
STDERR: ^[[31mError validating: 2 error(s) occurred:

* resource 'aws_instance.chef_server' config: **unknown resource**
* **'aws_security_group.allow-egress'** referenced in
* variable aws_security_group.allow-egress.name
* resource 'aws_instance.supermarket_server' config:
* unknown resource 'aws_security_group.allow-egress'
* referenced in variable aws_security_group.allow-egress.
* name

We need our EC2 resources
to reference only **one**
security group

Let’s look at the **Chef Server**

supermarket-cluster.tf

```
…
resource “aws_instance” “chef_server” {
  ami = “${var.ami}”
  instance_type = “${var.instance_type}”
  key_name = “${var.key_name}”
  tags {
    Name = “dev-chef-server”
  }
  security_groups = [“${aws_security_group.allow-ssh.name}”,
                     “${aws_security_group.allow-out.name}”]
  (…)
}
…
```

This is the Chef Server

supermarket-cluster.tf

```
…
resource “aws_instance” “chef_server” {
  ami = “${var.ami}”
  instance_type = “${var.instance_type}”
  key_name = “${var.key_name}”
  tags {
    Name = “dev-chef-server”
  }
  security_groups = [“${aws_security_group.allow-ssh.name}”]
  (…)
}
…
```

This is the Chef Server

And the **Supermarket Server**

supermarket-cluster.tf

```
…
resource “aws_instance” “supermarket_server” {
  ami = “${var.ami}”
  instance_type = “${var.instance_type}”
  key_name = “${var.key_name}”
  tags {
    Name = “dev-supermarket-server”
  }
  security_groups = [“${aws_security_group.allow-ssh.name}”,
                     “${aws_security_group.allow-out.name}”]
  (…)
}
…
```

This is the Supermarket Server

supermarket-cluster.tf

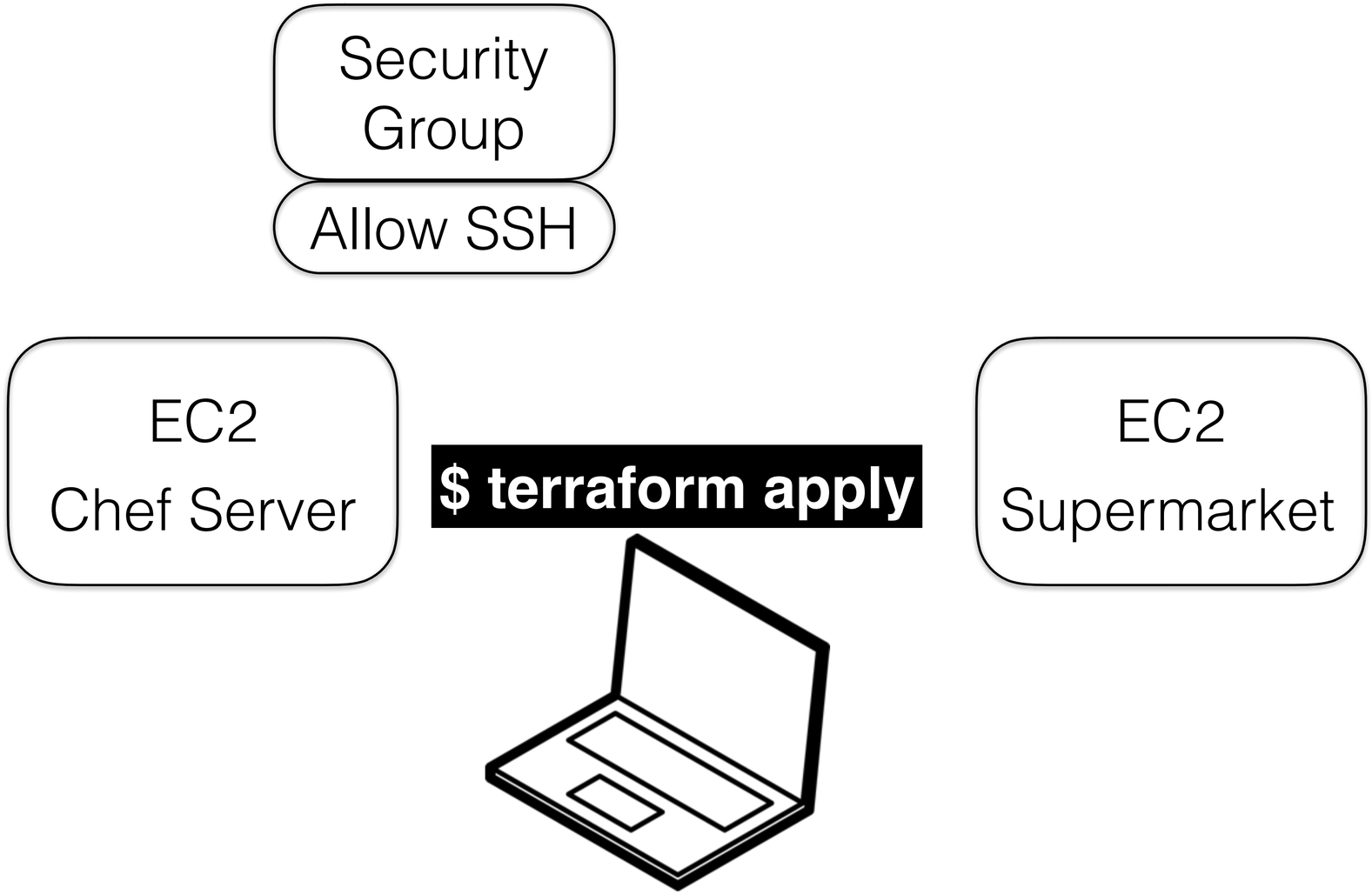
```
…
resource “aws_instance” “supermarket_server” {
  ami = “${var.ami}”
  instance_type = “${var.instance_type}”
  key_name = “${var.key_name}”
  tags {
    Name = “dev-supermarket-server”
  }
  security_groups = [“${aws_security_group.allow-ssh.name}”

  (…)
}
…
```

This is the Supermarket Server

$ kitchen converge

```
$ kitchen converge
Apply complete! Resources: 2 added,
0 changed, 0 destroyed.
(…)
Finished converging <default-ubuntu>
(0m7.10s).
```

supermarket-cluster.tf

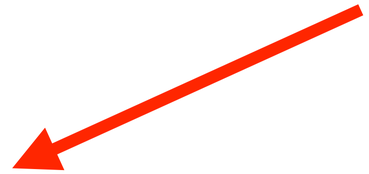


Now let’s write
some **tests**

First, let’s define
a **test group**

.kitchen.yml

```
driver:
  name: terraform
  (…)

verifier:
  name: terraform
  format: doc
  groups:
  - name: default
    tests:
      - security_groups
    hostnames: aws_hostnames
    username: ubuntu
```

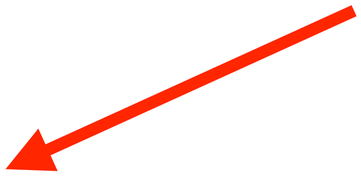


.kitchen.yml

```
driver:
  name: terraform
  (…)

verifier:
  name: terraform
  format: doc
  groups:
  - name: default
    tests:
      - security_groups
    hostnames: aws_hostnames
    username: ubuntu
```

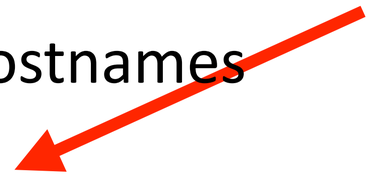
.kitchen.yml

```
driver:
  name: terraform
  (…)

verifier:
  name: terraform
  format: doc
  groups:
  - name: default
    tests:
      - security_groups
    hostnames: aws_hostnames
    username: ubuntu
```



.kitchen.yml

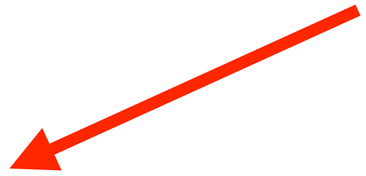
```
driver:
  name: terraform
  (…)

verifier:
  name: terraform
  format: doc
  groups:
  - name: default
    tests:
      - security_groups
    hostnames: aws_hostnames
    username: ubuntu
```



.kitchen.yml

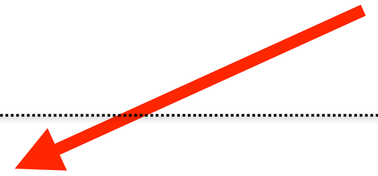
```
driver:
  name: terraform
  (…)

verifier:
  name: terraform
  format: doc
  groups:
  - name: default
    tests:
      - security_groups
    hostnames: aws_hostnames
    username: ubuntu
```

Output variable



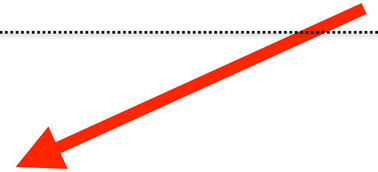
We need to **create**
that **output variable**

outputs.tf



```
output “aws_hostnames” {


}
```

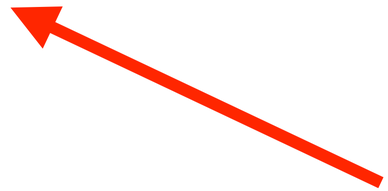
outputs.tf



```
output “aws_hostnames” {
  value = “${aws_instance.chef_server.public_dns},
            ${aws_instance.supermarket_server.public_dns}”
}
```

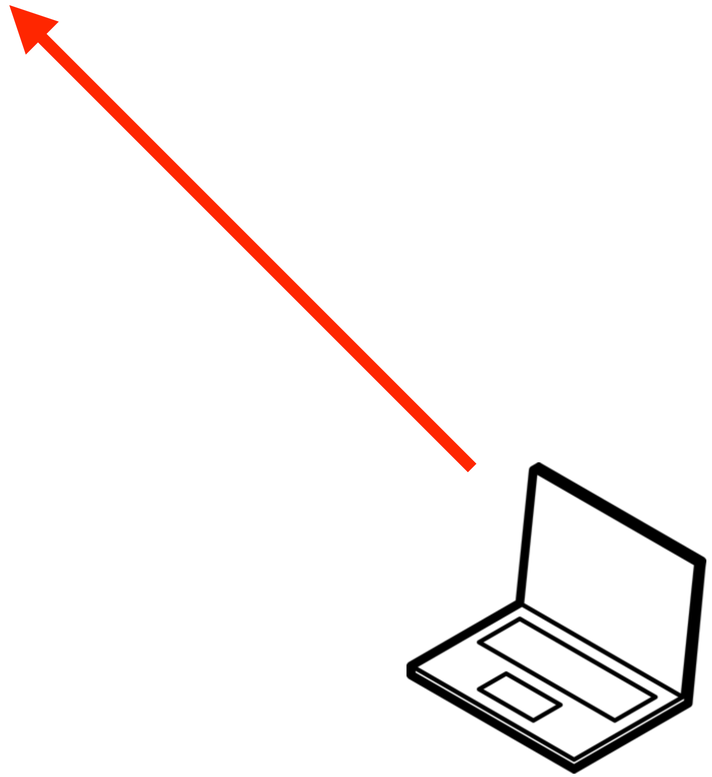
outputs.tf

```
output “aws_hostnames” {
  value = “${aws_instance.chef_server.public_dns},
            ${aws_instance.supermarket_server.public_dns}”
}
```



Using Outputs

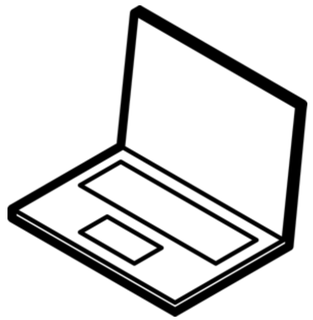
Spins up AWS resources



Using Outputs

Spins up AWS resources

Captures public_dns of EC2 instances in **aws_hostnames**

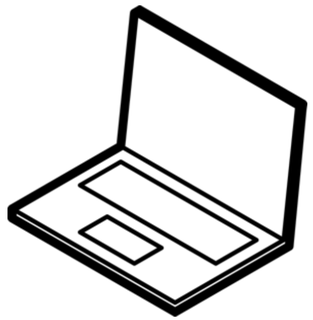


Using Outputs

Spins up AWS resources

↓

Captures public_dns of EC2 instances in **aws_hostnames** → Passes to kitchen-terraform



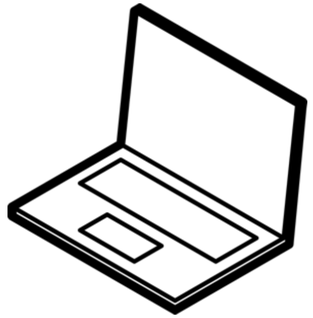
Using Outputs

Spins up AWS resources

Captures public_dns of EC2 instances in **aws_hostnames**

Passes to kitchen-terraform

kitchen-terraform uses **aws_hostnames** to ssh into the instances



Using Outputs

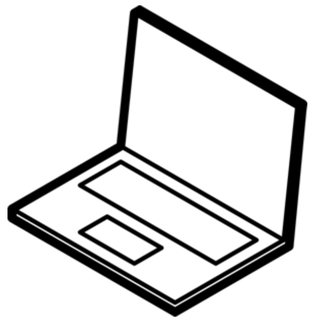
Spins up AWS resources

Captures public_dns of EC2 instances in **aws_hostnames**

Passes to kitchen-terraform

kitchen-terraform uses **aws_hostnames** to ssh into the instances

Runs Tests



$ kitchen destroy

```
$ kitchen destroy
$ kitchen converge
```

supermarket-cluster.tf

```
#resource “aws_security_group” “allow-egress” {
#  name = “${var.user_name}-allow-egress”
#  tags = {
#    Name = “${var.user_name} Allow connections out”
#  }
#}

#resource “aws_security_group_rule” “allow-out” {
# type = “egress”
# from_port = 0
# to_port = 65535
# cidr_blocks = ["0.0.0.0/0"]
  …
```

Let’s write a **test**

security_groups_spec.rb

describe command(‘**ping -c 1 google.com**’) do

end

security_groups_spec.rb

```
describe command('ping -c 1 google.com') do
  its('stdout') { should match /1 packets transmitted, 1 received/ }
end
```

$ kitchen verify

```
$ kitchen verify
Failure/Error:
expected "PING google.com (216.58.218.238)
56(84) bytes of data.\n\n--- google.com
ping statistics —\n
1 packets transmitted, 0 received,
to match /1 packets transmitted, 1 received/
 Diff:
          @@ -1,2 +1,5 @@
          -/1 packets transmitted, 1 received/
```

Good! We have a **failure!**

Now let’s make it **pass**

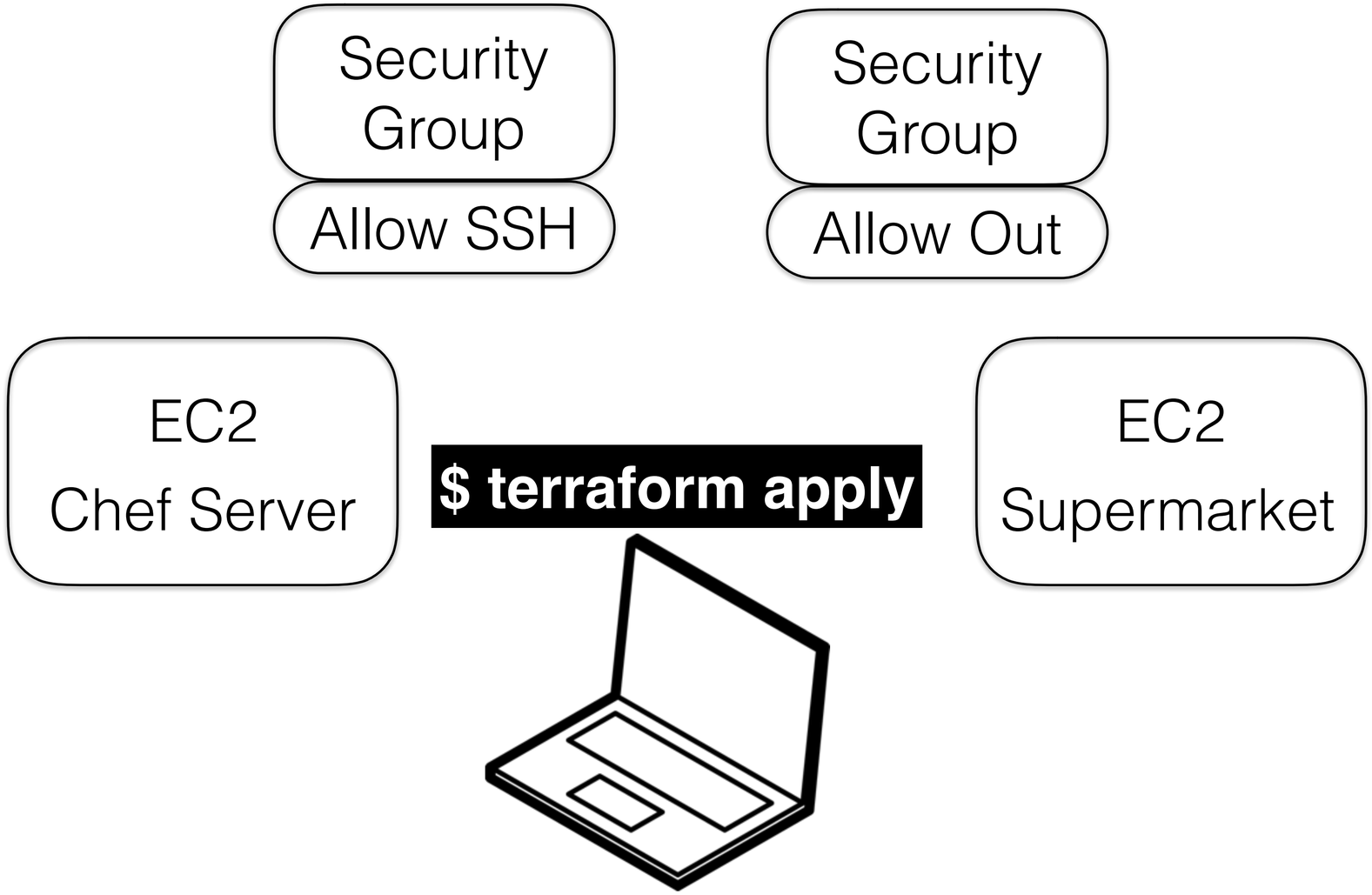
supermarket-cluster.tf

```
#resource “aws_security_group” “allow-egress” {
#  name = “${var.user_name}-allow-egress”
#  tags = {
#    Name = “${var.user_name} Allow connections out”
#  }
#}

#resource “aws_security_group_rule” “allow-out” {
# type = “egress”
# from_port = 0
# to_port = 65535
# cidr_blocks = ["0.0.0.0/0"]
  …
```

supermarket-cluster.tf

```
resource “aws_security_group” “allow-egress” {
  name = “${var.user_name}-allow-egress”
  tags = {
    Name = “${var.user_name} Allow connections out”
  }
}

resource “aws_security_group_rule” “allow-out” {
 type = “egress”
 from_port = 0
 to_port = 65535
 cidr_blocks = ["0.0.0.0/0"]
  …
```

supermarket-cluster.tf



Now let’s call this
security group from our
Chef Server

supermarket-cluster.tf

```
…
resource “aws_instance” “chef_server” {
  ami = “${var.ami}”
  instance_type = “${var.instance_type}”
  key_name = “${var.key_name}”
  tags {
    Name = “dev-chef-server”
  }
  security_groups = [“${aws_security_group.allow-ssh.name}”]
  (…)
}
…
```

This is the Chef Server

supermarket-cluster.tf

```
…
resource “aws_instance” “chef_server” {
  ami = “${var.ami}”
  instance_type = “${var.instance_type}”
  key_name = “${var.key_name}”
  tags {
    Name = “dev-chef-server”
  }
    security_groups = [“${aws_security_group.allow-ssh.name}”,
                       “${aws_security_group.allow-out.name}”]
  (…)
}
…
```

This is the Chef Server

And the **Supermarket Server**

supermarket-cluster.tf

```
…
resource “aws_instance” “supermarket_server” {
  ami = “${var.ami}”
  instance_type = “${var.instance_type}”
  key_name = “${var.key_name}”
  tags {
    Name = “dev-supermarket-server”
  }
  security_groups = [“${aws_security_group.allow-ssh.name}”]

  (…)
}
…
```

This is the Supermarket Server

supermarket-cluster.tf

```
…
resource “aws_instance” “supermarket_server” {
  ami = “${var.ami}”
  instance_type = “${var.instance_type}”
  key_name = “${var.key_name}”
  tags {
    Name = “dev-supermarket-server”
  }
  security_groups = [“${aws_security_group.allow-ssh.name}”,
                     “${aws_security_group.allow-out.name}”]
  (…)
}
…
```

This is the Supermarket Server

```
$ kitchen destroy
$ kitchen converge
```

```
$ kitchen destroy
$ kitchen converge
$ kitchen verify
```

```
$ kitchen verify
Command ping -c 1 google.com
  stdout
    should match /1 packets transmitted,
                          1 received/
1 example, 0 failures
```

It **passes!**
Now let's **make a change**

Let's **condense** the
two security groups
into **one** security group

supermarket-cluster.tf

```
resource “aws_security_group” “allow-egress” {
  name = “${var.user_name}-allow-egress”
  tags = {
    Name = “${var.user_name} Allow connections out”
  }
}

resource “aws_security_group_rule” “allow-out” {
 type = “egress”
 from_port = 0
 to_port = 65535
 cidr_blocks = ["0.0.0.0/0"]
  …
```

supermarket-cluster.tf

```
resource “aws_security_group_rule” “allow-out” {
 type = “egress”
 from_port = 0
 to_port = 65535
 cidr_blocks = ["0.0.0.0/0"]
 security_group_id “${aws_security_group.allow-egress.id}”
}
```

supermarket-cluster.tf

```
resource “aws_security_group_rule” “allow-out” {
 type = “egress”
 from_port = 0
 to_port = 65535
 cidr_blocks = ["0.0.0.0/0"]
 security_group_id “${aws_security_group.allow-ssh.id}”
}
```

Now our **instances**
should only use the
one security group

supermarket-cluster.tf

```
…
resource “aws_instance” “chef_server” {
  ami = “${var.ami}”
  instance_type = “${var.instance_type}”
  key_name = “${var.key_name}”
  tags {
    Name = “dev-chef-server”
  }
  security_groups = [“${aws_security_group.allow-ssh.name}”,
                     “${aws_security_group.allow-out.name}”]
  (…)
}
…
```

This is the Chef Server

supermarket-cluster.tf

```
…
resource “aws_instance” “chef_server” {
  ami = “${var.ami}”
  instance_type = “${var.instance_type}”
  key_name = “${var.key_name}”
  tags {
    Name = “dev-chef-server”
  }
  security_groups = [“${aws_security_group.allow-ssh.name}”]

  (…)
}
…
```

This is the Chef Server

supermarket-cluster.tf

```
...
resource “aws_instance” “supermarket_server” {
  ami = “${var.ami}”
  instance_type = “${var.instance_type}”
  key_name = “${var.key_name}”
  tags {
    Name = “dev-supermarket-server”
  }
  security_groups = [“${aws_security_group.allow-ssh.name}”,
                     “${aws_security_group.allow-out.name}” ]
 (...)
}
...
```

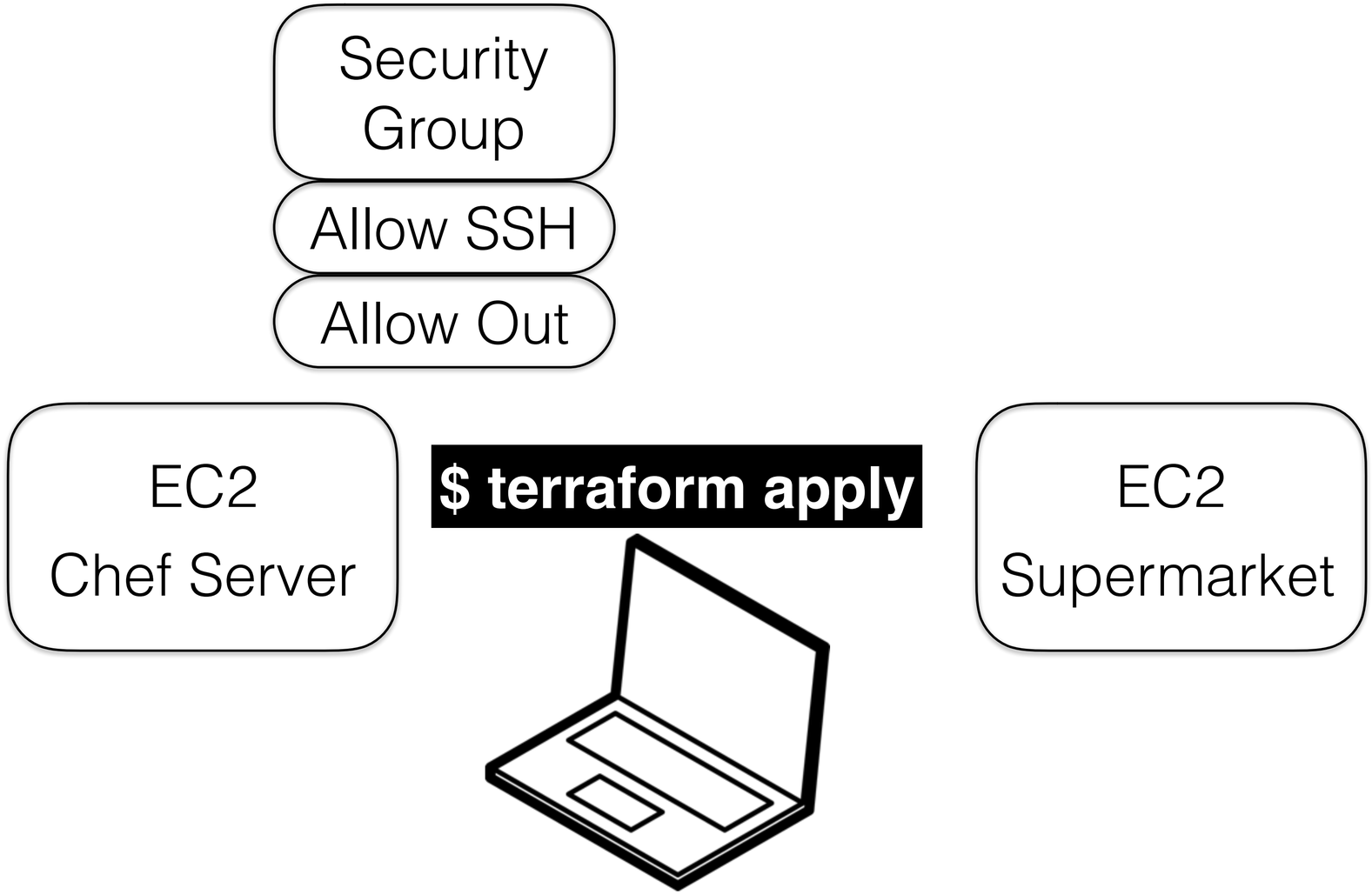
This is the Supermarket Server

supermarket-cluster.tf

```
...
resource “aws_instance” “supermarket_server” {
  ami = “${var.ami}”
  instance_type = “${var.instance_type}”
  key_name = “${var.key_name}”
  tags {
    Name = “dev-supermarket-server”
  }
  security_groups = [“${aws_security_group.allow-ssh.name}”]
  (...)
}
...
```

This is the Supermarket Server

supermarket-cluster.tf



```
$ kitchen destroy
$ kitchen converge
```

```
$ kitchen destroy
$ kitchen converge
$ kitchen verify
```

```
$ kitchen verify
Command ping -c 1 google.com
  stdout
    should match /1 packets transmitted,
                          1 received/

1 example, 0 failures
```

Now let’s **improve** the design

By moving the security group code into a **module**

Why Move into a Module?

- Self-contained package

Source: Terraform Docs

Why Move into a Module?

- Self-contained package
- Reusable component

Source: Terraform Docs

Why Move into a Module?

- Self-contained package
- Reusable component
- Improve organization

Source: Terraform Docs

$ kitchen destroy

```
$ mkdir security_groups
```

security-groups/main.tf

security-groups/main.tf

```
resource “aws_security_group” “allow-ssh” {
 name = “${var.user_name}-allow-ssh”
 tags = {
   Name = “${var.user_name} Allow All SSH”
 }

resource “aws_security_group_rule” “allow-ssh” {
 type = “ingress”
 from_port = 22
 to_port = 22
  …
}
```

Now we need to
connect to the **module**
from main config

First, we need to
know what **variables**
the **module needs**
passed to it

security-groups/main.tf

```
resource “aws_security_group” “allow-ssh” {
 name = “${var.user_name}-allow-ssh”
 tags = {
   Name = “${var.user_name} Allow All SSH”
 }

resource “aws_security_group_rule” “allow-ssh” {
 type = “ingress”
 from_port = 22
  …
  security_group_id = “${aws_security_group.allow-ssh.id}”
}
```

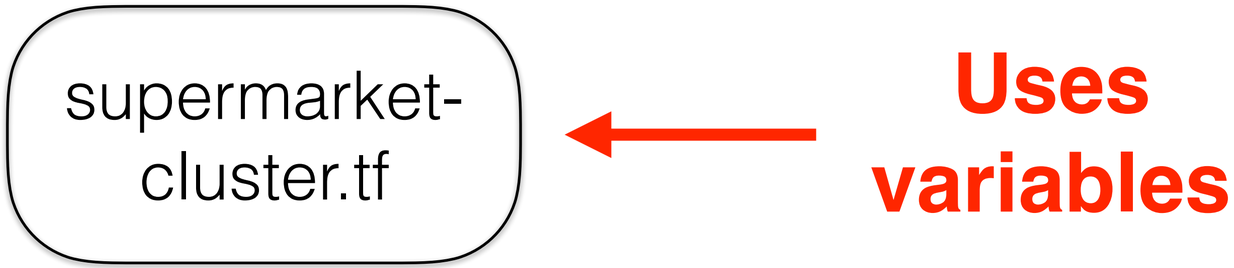
Needs to be passed to the module

security-groups/main.tf

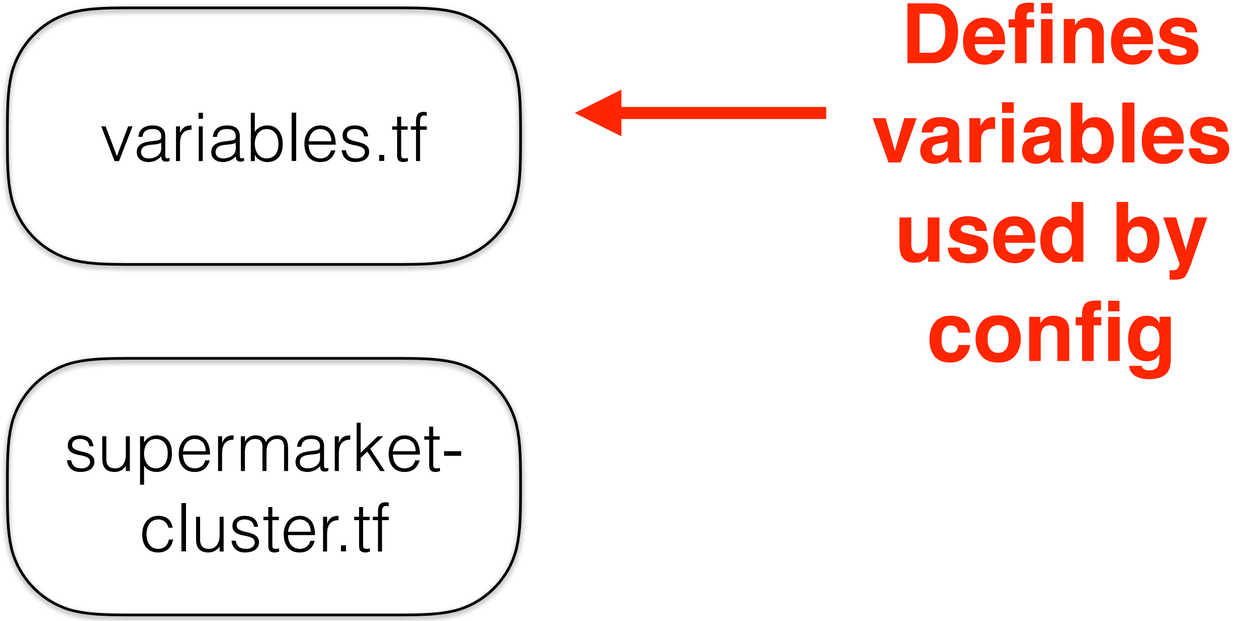
```
resource “aws_security_group” “allow-ssh” {
 name = “${var.user_name}-allow-ssh”
 tags = {
   Name = “${var.user_name} Allow All SSH”
 }


resource “aws_security_group_rule” “allow-ssh” {
 type = “ingress”
 from_port = 22
  …
  security_group_id = “${aws_security_group.allow-ssh.id}”
}
```

Does not need to be passed in

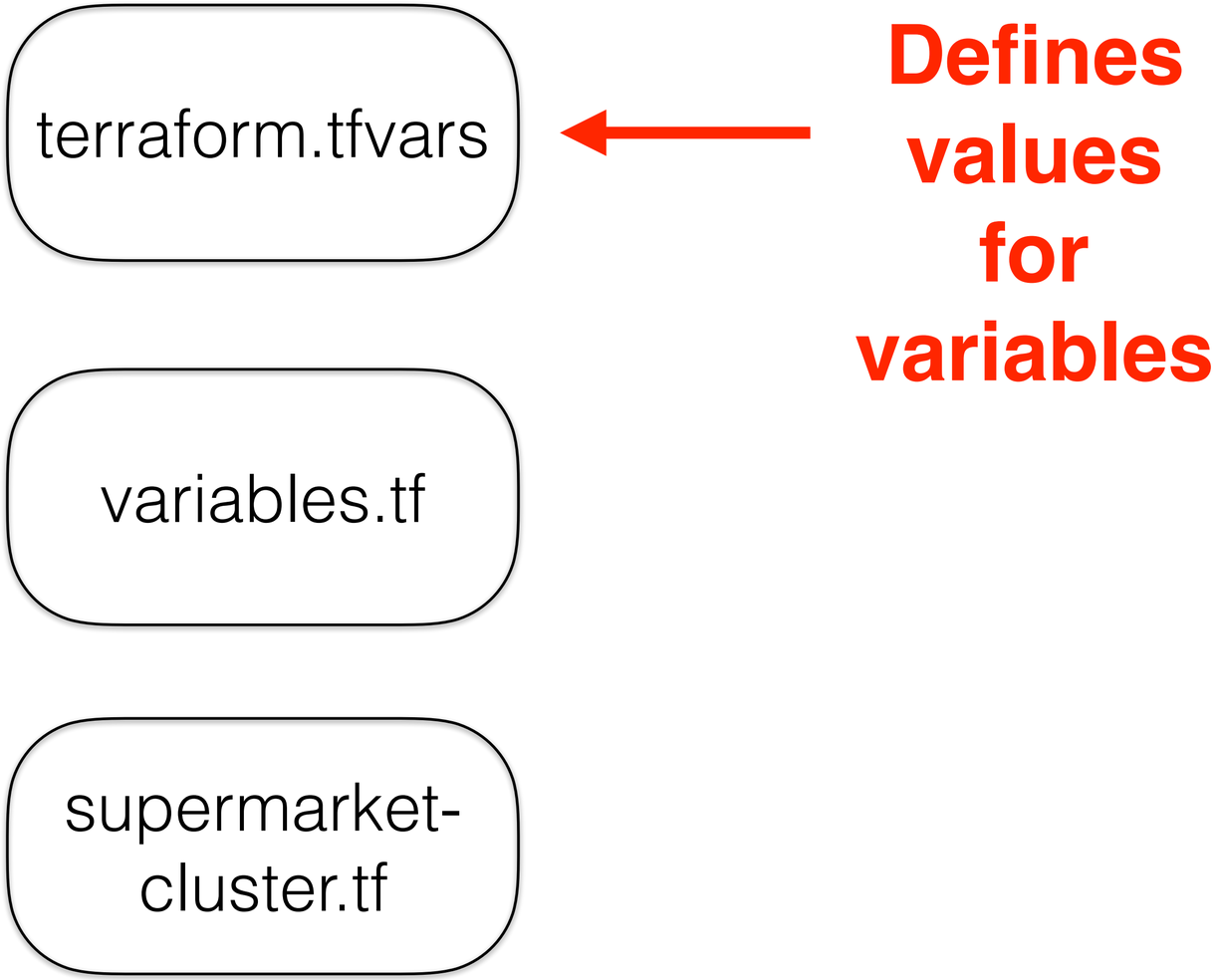
Using the Module



Using the Module



Using the Module

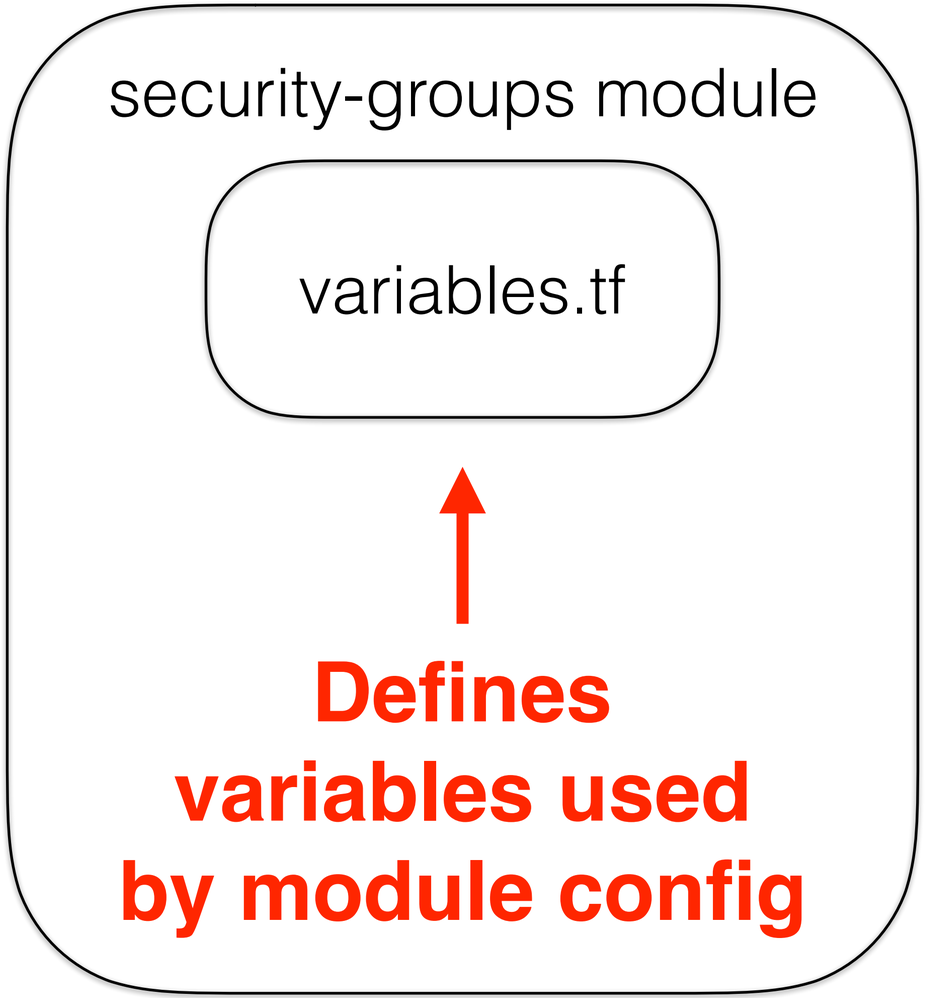


Using the Module

terraform.tfvars

variables.tf

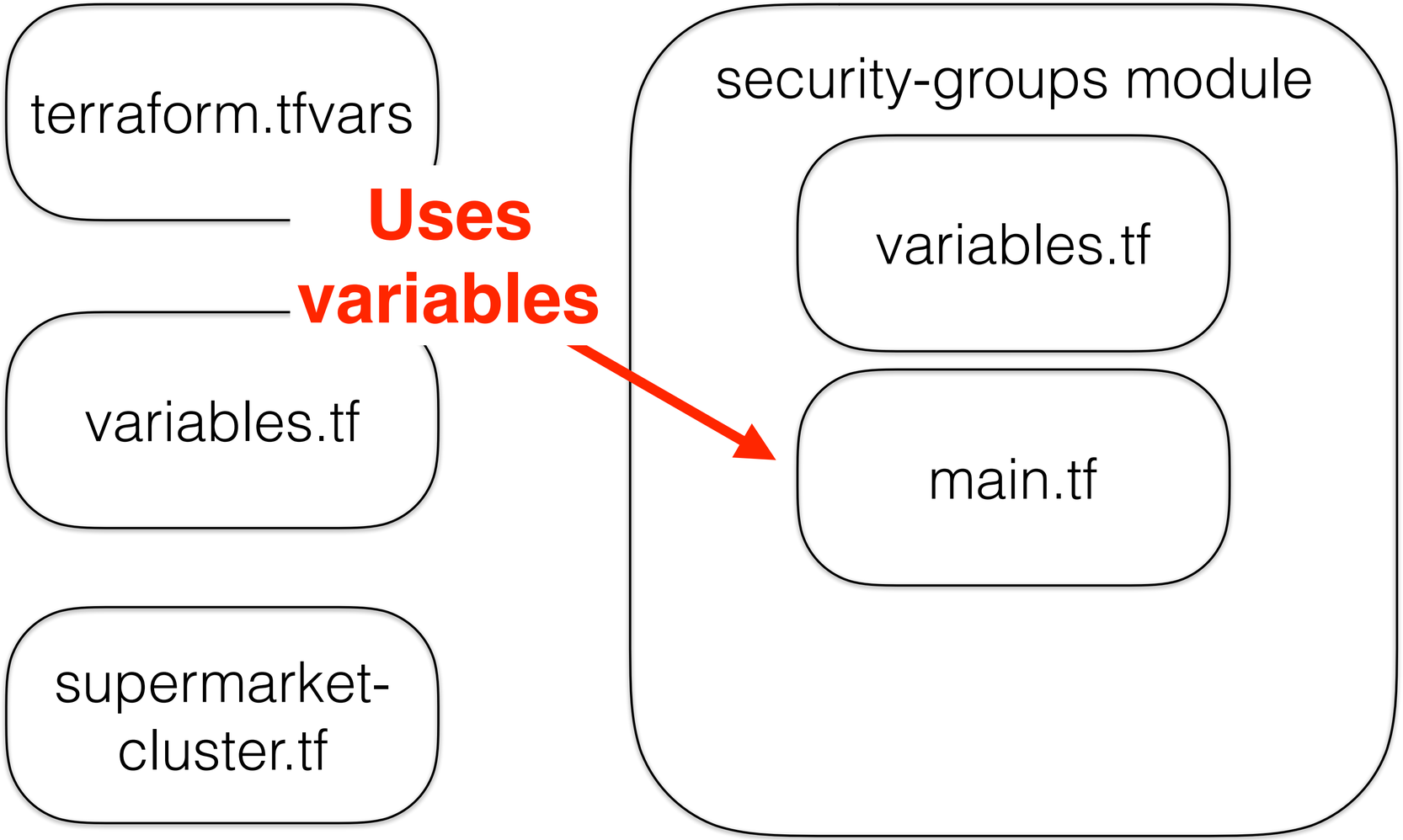
supermarket-cluster.tf



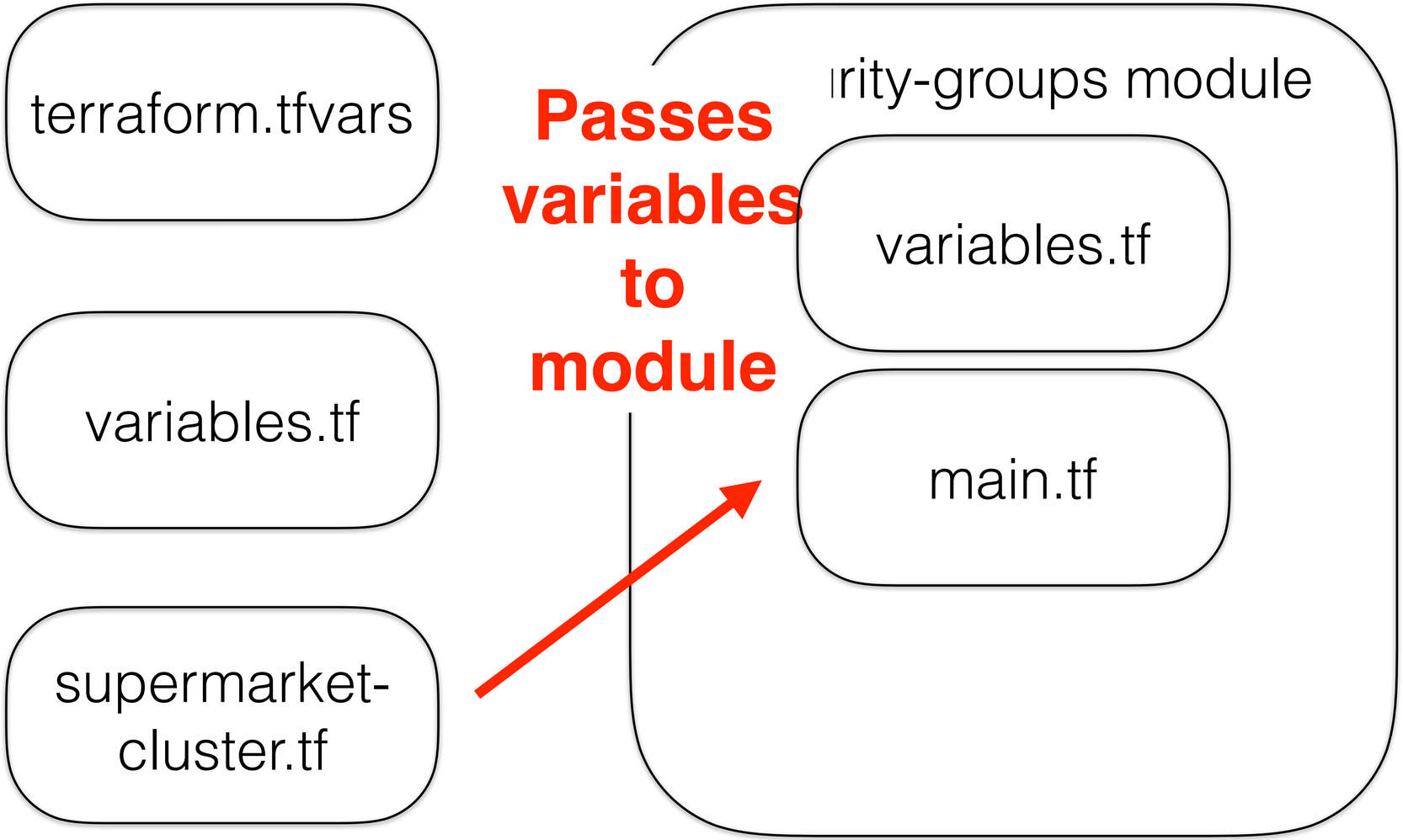
security-groups/variables.tf

variable “user_name” {}

Using the Module

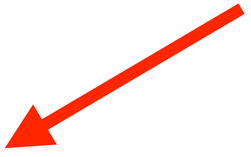


Using the Module



supermarket-cluster.tf

```
module “security-groups” {
  source = “./security-groups”
  user_name = “${var.user_name}”
}
```



supermarket-cluster.tf

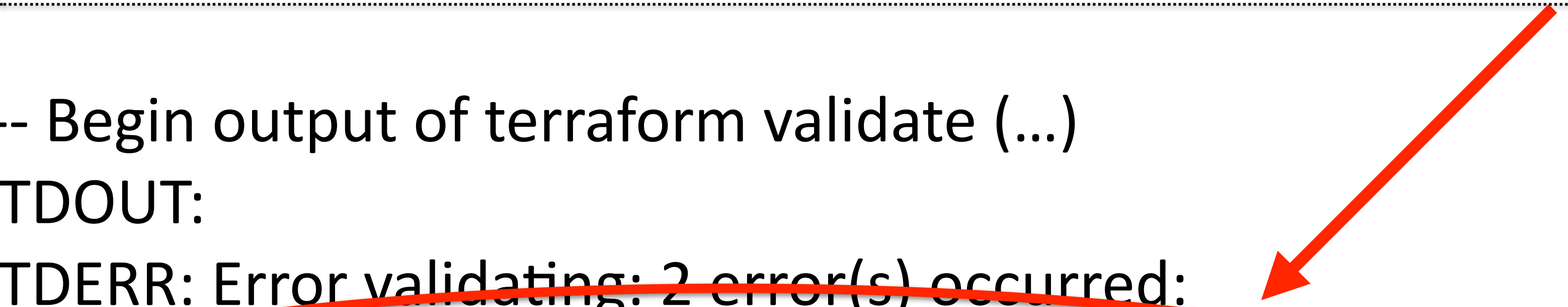
```
module “security-groups” {
  source = “./security-groups”
  user_name = “${var.user_name}”
}
```

$ kitchen converge

```
$ kitchen converge
-----> Starting Kitchen (v1.10.2)
-----> Converging <default-ubuntu>…
(…)
>>>>>>> ------Exception-------
>>>>>>> Class: Kitchen::ActionFailed
Message: 1 actions failed.
>>>>>>>     Converge failed on instance
<default-ubuntu>.  Please see
.kitchen/logs/default-ubuntu.log for more details
```

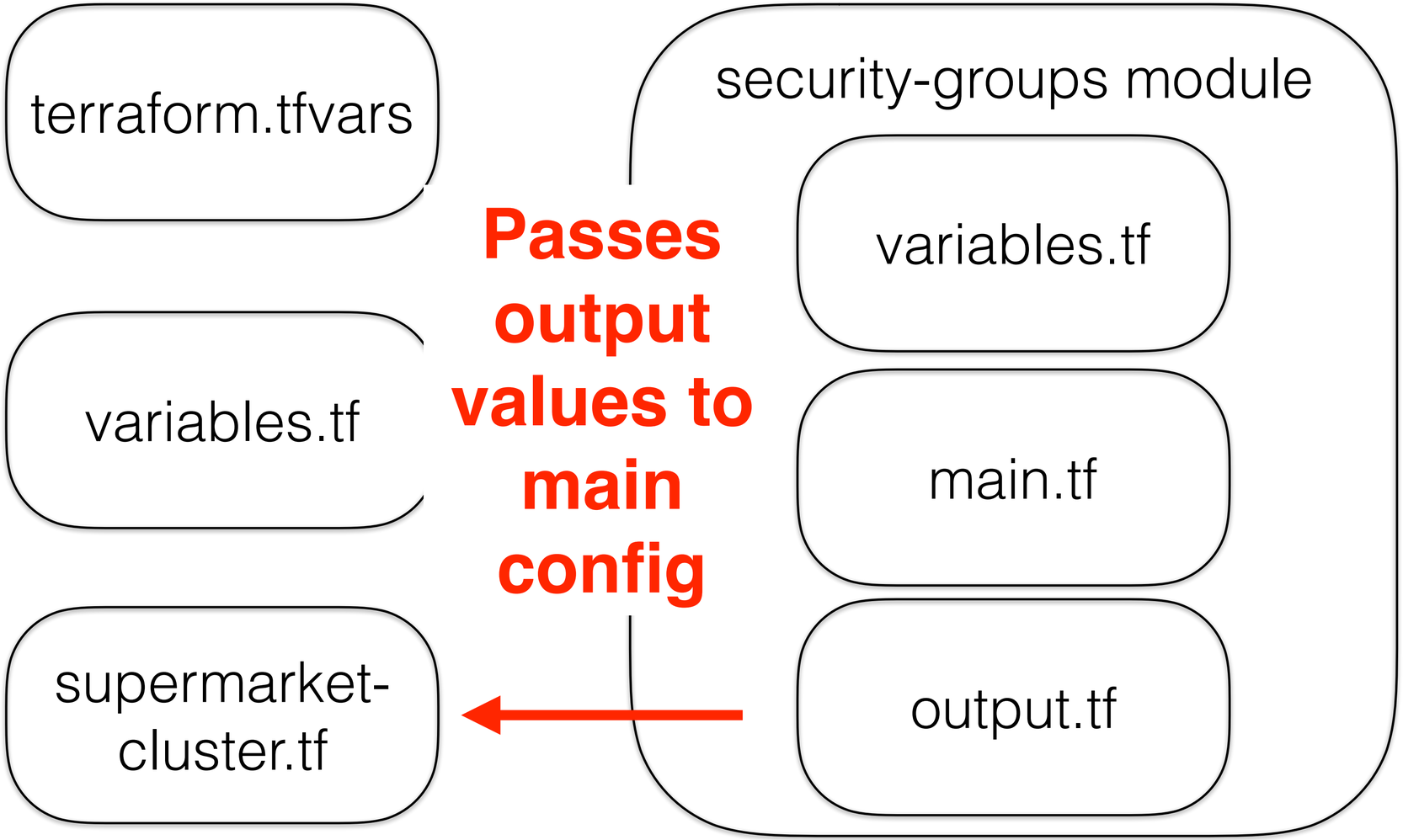
```
$ kitchen converge
-----> Starting Kitchen (v1.10.2)
-----> Converging <default-ubuntu>…
(…)
>>>>>>> ------Exception-------
>>>>>>> Class: Kitchen::ActionFailed
Message: 1 actions failed.
>>>>>>>     Converge failed on instance
<default-ubuntu>.  Please see
.kitchen/logs/default-ubuntu.log for more details
```

.kitchen/logs/default-ubuntu.log



--- Begin output of terraform validate (...)
STDOUT:
STDERR: Error validating: 2 error(s) occurred:
* resource 'aws_instance.supermarket_server' config:
* **unknown resource 'aws_security_group.allow-ssh'**
* referenced in variable aws_security_group.allow-ssh.name
* resource 'aws_instance.chef_server' config:
* unknown resource 'aws_security_group.allow-ssh'
* referenced in variable aws_security_group.allow-ssh.name

Workflow Into The Module



security-groups/output.tf

```
output “sg-name” {

}
```

security-groups/output.tf

```
output “sg-name” {
  value = “${aws_security_group.allow-ssh.name}”
}
```

Now, we need to **use** this **output** in our supermarket-cluster config

supermarket-cluster.tf

```
…
resource “aws_instance” “chef_server” {
  ami = “${var.ami}”
  instance_type = “${var.instance_type}”
  key_name = “${var.key_name}”
  tags {
    Name = “dev-chef-server”
  }
  security_groups = [“${aws_security_group.allow-ssh.name}”]

  (…)
}
…
```

This is the Chef Server

supermarket-cluster.tf

```
…
resource “aws_instance” “chef_server” {
  ami = “${var.ami}”
  instance_type = “${var.instance_type}”
  key_name = “${var.key_name}”
  tags {
    Name = “dev-chef-server”
  }
  security_groups = [“${module.security-groups.sg-name}”]

  (…)
}
…
```

This is the Chef Server

supermarket-cluster.tf

```
…
resource “aws_instance” “supermarket_server” {
  ami = “${var.ami}”
  instance_type = “${var.instance_type}”
  key_name = “${var.key_name}”
  tags {
    Name = “dev-supermarket-server”
  }
  security_groups = [“${aws_security_group.allow-ssh-name}”]

  (…)
}
…
```

This is the Supermarket Server

supermarket-cluster.tf

```
…
resource “aws_instance” “supermarket_server” {
  ami = “${var.ami}”
  instance_type = “${var.instance_type}”
  key_name = “${var.key_name}”
  tags {
    Name = “dev-supermarket-server”
  }
  security_groups = [“${module.security-groups.sg-name}”]

  (…)
}
…
```

This is the Supermarket Server

$ kitchen converge

```
$ kitchen converge
$ kitchen verify
```

```
$ kitchen verify
Command ping -c 1 google.com
  stdout
    should match /1 packets transmitted,
                          1 received/

1 example, 0 failures
```

So we have improved our
resource usage

And the code's
organization and
design

With **minimal** risk

Infrastructure code
must be **maintained**
and **refactored** just like
application code

Even more so
because **infrastructure code** involves so many **moving pieces**

When **refactoring**,
always cover with tests

And **refactor** one **small piece** at a time

Thank you

Who Am I?

Nell Shamrell-Harrington

- Software Engineer at Chef
- Former O'Fallon, IL resident (**Southern IL ftw!**)
- @nellshamrell
- nshamrell@chef.io

Who Am I?

Nell Shamrell-Harrington

- Software Engineer at Chef
- Former O'Fallon, IL resident (**Southern IL ftw!**)
- @nellshamrell
- nshamrell@chef.io

Any Questions?