



Automate All The Things!



Automate all the Things!

<https://consensus.enterprises>

Colan Schwartz ([colan](#) on d.o)

Christopher Gervais ([ergonlogic](#) on d.o)

Dan Friedman ([llamech](#) on d.o)



Drupal GovCon is brought to you by Drupal4Gov, a non-profit organization dedicated to connecting and serving government employees committed to open source technology. Drupal is a registered trademark of Dries Buytaert



- ★ Veteran open source programmers and sysadmins
- ★ Specializing in **Drupal**™  and  **Ægir**
- ★ Experts in end-to-end application lifecycle
- ★ Focus on social enterprises, non-profits, and public sector

Some of our Partnerships



What we'll discuss

A brief history of cloud computing

How did we get into this mess?

How do Terraform & Ansible support an *infrastructure-as-code* strategy?

Providers, resources and provisioners; oh my!

Principles and Practices of *infrastructure-as-code*

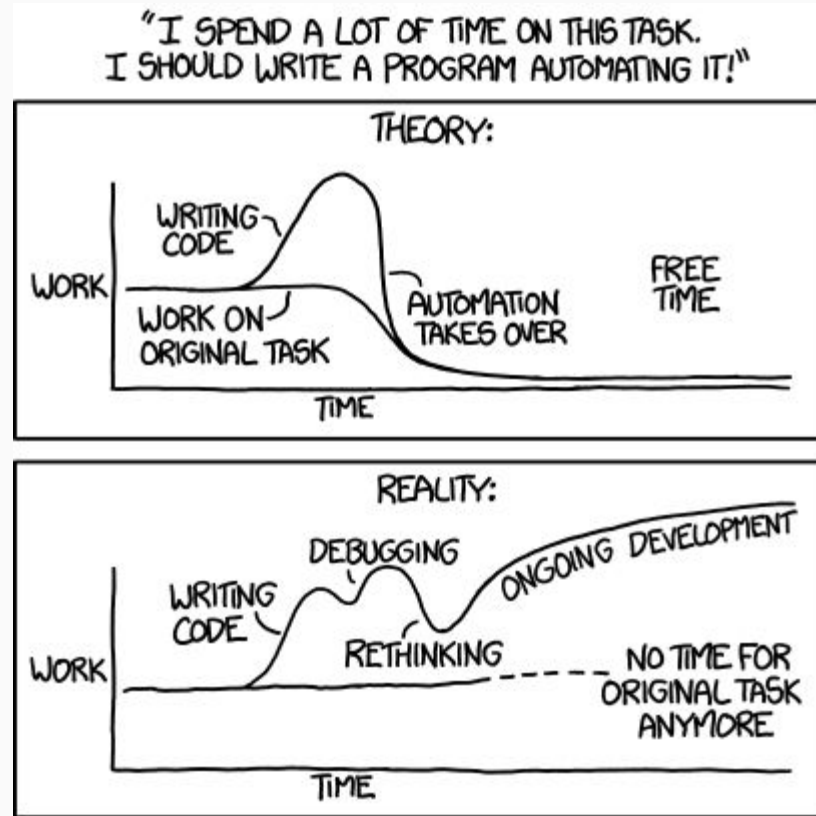
Why should we care?

Putting it all together

Demo time!

XKCD

... because, somehow, a webcomic provides the most succinct descriptions of the reality of automation.



You can never have too much XKCD!

HOW LONG CAN YOU WORK ON MAKING A ROUTINE TASK MORE EFFICIENT BEFORE YOU'RE SPENDING MORE TIME THAN YOU SAVE?
(ACROSS FIVE YEARS)

		HOW OFTEN YOU DO THE TASK					
		50/DAY	5/DAY	DAILY	WEEKLY	MONTHLY	YEARLY
HOW MUCH TIME YOU SHAVE OFF	1 SECOND	1 DAY	2 HOURS	30 MINUTES	4 MINUTES	1 MINUTE	5 SECONDS
	5 SECONDS	5 DAYS	12 HOURS	2 HOURS	21 MINUTES	5 MINUTES	25 SECONDS
	30 SECONDS	4 WEEKS	3 DAYS	12 HOURS	2 HOURS	30 MINUTES	2 MINUTES
	1 MINUTE	8 WEEKS	6 DAYS	1 DAY	4 HOURS	1 HOUR	5 MINUTES
	5 MINUTES	9 MONTHS	4 WEEKS	6 DAYS	21 HOURS	5 HOURS	25 MINUTES
	30 MINUTES		6 MONTHS	5 WEEKS	5 DAYS	1 DAY	2 HOURS
	1 HOUR		10 MONTHS	2 MONTHS	10 DAYS	2 DAYS	5 HOURS
	6 HOURS				2 MONTHS	2 WEEKS	1 DAY
	1 DAY					8 WEEKS	5 DAYS

A Brief history of Cloud Computing

Automate All the Things!

A brief history of cloud computing

- **Time-sharing**
(government/academic)



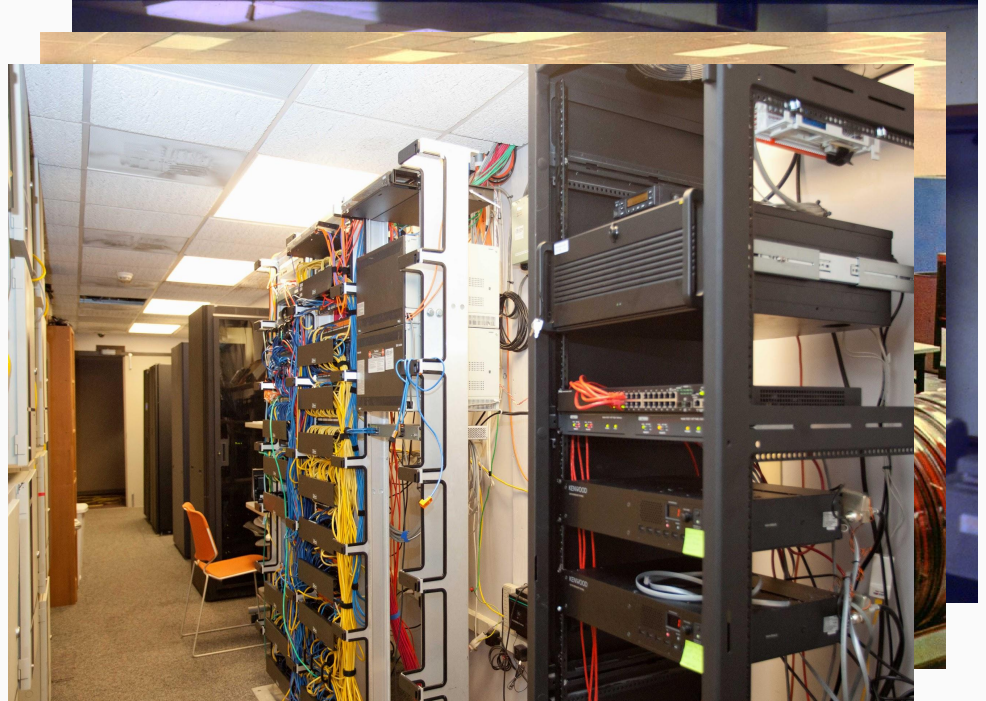
A brief history of cloud computing

- Time-sharing
(government/academic)
- **Mainframes**
(centralized/institutional)



A brief history of cloud computing

- Time-sharing
(government/academic)
- Mainframes
(centralized/institutional)
- **Server rooms**
(distributed/on-premise)



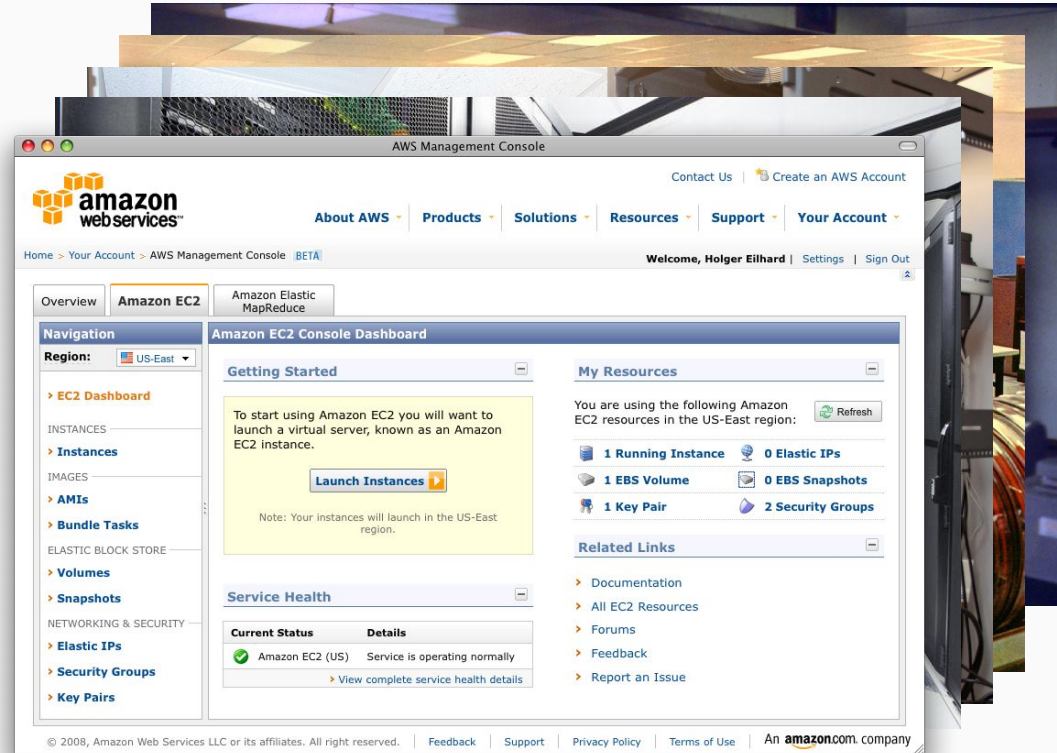
A brief history of cloud computing

- Time-sharing
(government/academic)
- Mainframes
(centralized/institutional)
- Server rooms
(distributed/on-premise)
- **Datacenters**
(co-location/hosted)



A brief history of cloud computing

- Time-sharing
(government/academic)
- Mainframes
(centralized/institutional)
- Server rooms
(distributed/on-premise)
- Datacenters
(co-location/hosted)
- Cloud
(utility computing)



The era of cloud computing

Benefits	Challenges
Scalability	Controlling costs

The era of cloud computing

Benefits	Challenges
Scalability	Controlling costs
Flexibility	Increased complexity

The era of cloud computing

Benefits	Challenges
Scalability	Controlling costs
Flexibility	Increased complexity
Automation	Scarce expertise

Principles and Practices of *infrastructure-as-code*

Automate All the Things!

Infrastructure-as-code Practices

- **Define resources in code**

(avoid snowflake servers)

```
1
2
3 - name: Create Linode VMs.
4   # Only create VMs that aren't already in our
5   # that can be changed after creation are hand
6   # Ref.: http://docs.ansible.com/ansible/linod
7   linode:
8     name: "{{ item.key }}"
9     plan: "{{ item.value.plan | default('1') }}"
10    datacenter: "{{ item.value.datacenter | def
11    distribution: "{{ item.value.distro | defau
12    ssh_pub_key: "{{ lookup('file', '~/.ssh/id_
13    wait: yes
14    state: "{{ item.value.state | default(linod
15    when: cached_linodes[item.key] is not defined
16    with_dict: "{{ cloud.linode }}"
17
18
```


Infrastructure-as-code Practices

- Define resources in code
(avoid snowflake servers)
- **Keep documentation inline**
(self-documented systems)

```
1
2
3 - name: Create Linode VMs.
4   # Only create VMs that aren't already in our
5   # that can be changed after creation are handled
6   # Ref.: http://docs.ansible.com/ansible/linode
7   linode:
8     name: "{{ item.key }}"
9     plan: "{{ item.value.plan | default('1') }}"
10    datacenter: "{{ item.value.datacenter | default('us-east-1') }}"
11    distribution: "{{ item.value.distro | default('ubuntu1404') }}"
12    ssh_pub_key: "{{ lookup('file', '~/.ssh/id_rsa.pub') }}"
13    wait: yes
14    state: "{{ item.value.state | default('present') }}"
15  when: cached_linodes[item.key] is not defined
16  with_dict: "{{ cloud.linode }}"
17
18
```

Infrastructure-as-code Practices

- Define resources in code
(avoid snowflake servers)
- Keep documentation inline
(self-documented systems)
- **Version-control everything**
(audit trail and reproducible builds)

```
commit 8d93818e4b6a4ade45cb9d2447d939754b6b11e4
Author: Christopher Gervais <chris@ergonlogic.c
Date: Tue Aug 8 21:18:49 2017 -0400

    Use a variable to set default Linode plan (nod

commit 78456f2b573aa59b54ba87e0cc72b00f1d1b5a2a
Author: Christopher Gervais <chris@ergonlogic.c
Date: Tue Aug 8 21:17:07 2017 -0400

    Change name key to be more descriptive.

commit 6c519e6e908567fe1d5d39d10edbe9a7e13e25e8
Author: Christopher Gervais <chris@ergonlogic.c
Date: Tue Aug 8 21:15:20 2017 -0400

    Initial commit.
```

Infrastructure-as-code Practices

- Define resources in code
(avoid snowflake servers)
- Keep documentation inline
(self-documented systems)
- Version-control everything
(audit trail and reproducible builds)
- **Make small changes**
(easier rollbacks)

```
commit 8d93818e4b6a4ade45cb9d2447d939754b6b11e4
Author: Christopher Gervais <chris@ergonlogic.com>
Date: Tue Aug 8 21:18:49 2017 -0400

    Use a variable to set default Linode plan (VM size)

diff --git a/create.yml b/create.yml
index 4147637..55e7192 100644
--- a/create.yml
+++ b/create.yml
@@ -6,7 +6,7 @@
 # Ref.: http://docs.ansible.com/ansible/linode_module.html
 linode:
   name: "{{ item.name }}"
-  plan: "{{ item.value.plan | default('1') }}"
+  plan: "{{ item.value.plan | default(linode_plan) }}"
   datacenter: "{{ item.value.datacenter | default('us-east') }}"
   distribution: "{{ item.value.distro | default('ubuntu12.04') }}"
   ssh_pub_key: "{{ lookup('file', '~/.ssh/id_rsa.pub') | quote }}"
```

Infrastructure-as-code Practices

- Define resources in code
(avoid snowflake servers)
- Keep documentation inline
(self-documented systems)
- Version-control everything
(audit trail and reproducible builds)
- Make small changes
(easier rollbacks)
- **Test continuously**
(fail early)

```
commit 8d93818e4b6a4ade45cb9d2447d939754b6b11e4
commit 8d93818e4b6a4ade45cb9d2447d939754b6b11e4
TASK [Check inventory.] *****
ok: [localhost]

TASK [Assert that neither 'test1' nor 'test2' appear in th
ok: [localhost] => {
  "changed": false,
  "msg": "All assertions passed"
}

TASK [Query Linode API.] *****
ok: [localhost]

TASK [Assert that neither 'test1' nor 'test2' appear in th
ok: [localhost] => {
  "changed": false,
  "msg": "All assertions passed"
}
```

How do Terraform and Ansible support an *infrastructure-as-code* strategy?

Automate All the Things!

How does Terraform support an *infrastructure-as-code* strategy?

Terraform allows us to define infrastructure components in files written in [Terraform language](#) syntax.

These files can, in turn, be committed into version control, and thus handled as software.

```
resource "openstack_compute_instance_v2" "gateway" {  
  count = 1  
  name = "gateway${count.index}"  
  image_id      = var.compute_image_id  
  flavor_id     = var.compute_flavor_id_smallest  
  key_pair      = "my-keypair"  
  security_groups = ["vpn"]  
  network {  
    name = "Main network"  
  }  
  depends_on = [  
    openstack_networking_secgroup_v2.vpn,  
  ]  
}
```

Components

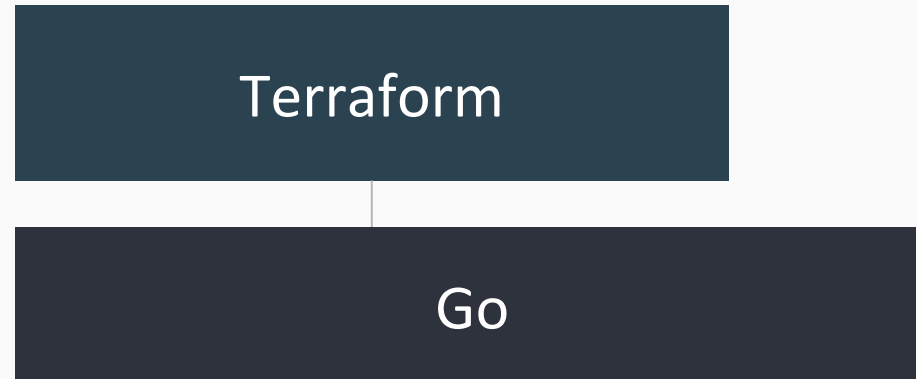
Custom *infrastructure-as-code* configuration depends on Terraform providers and resources, which in turn depend on Go, which Terraform is written in.



Terraform

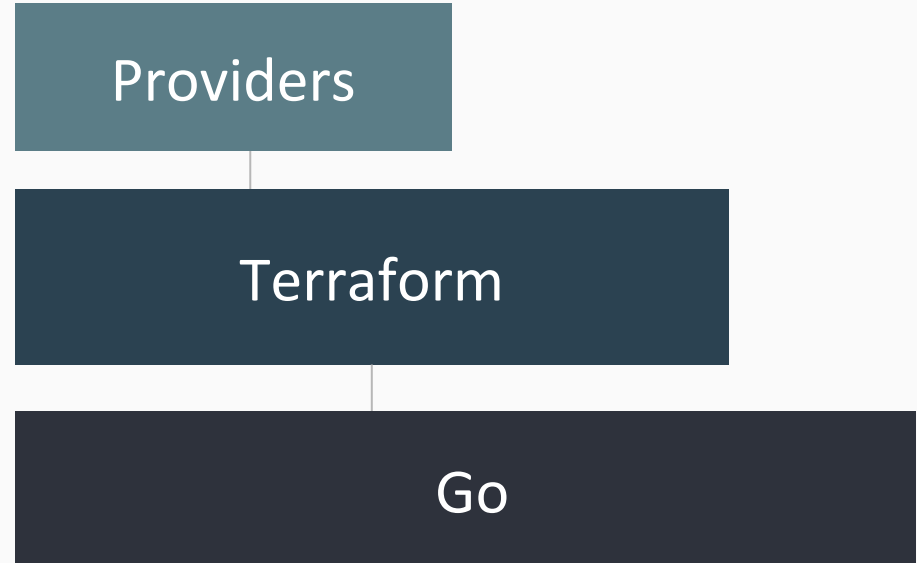
Components

Custom *infrastructure-as-code* configuration depends on Terraform providers and resources, which in turn depend on Go, which Terraform is written in.



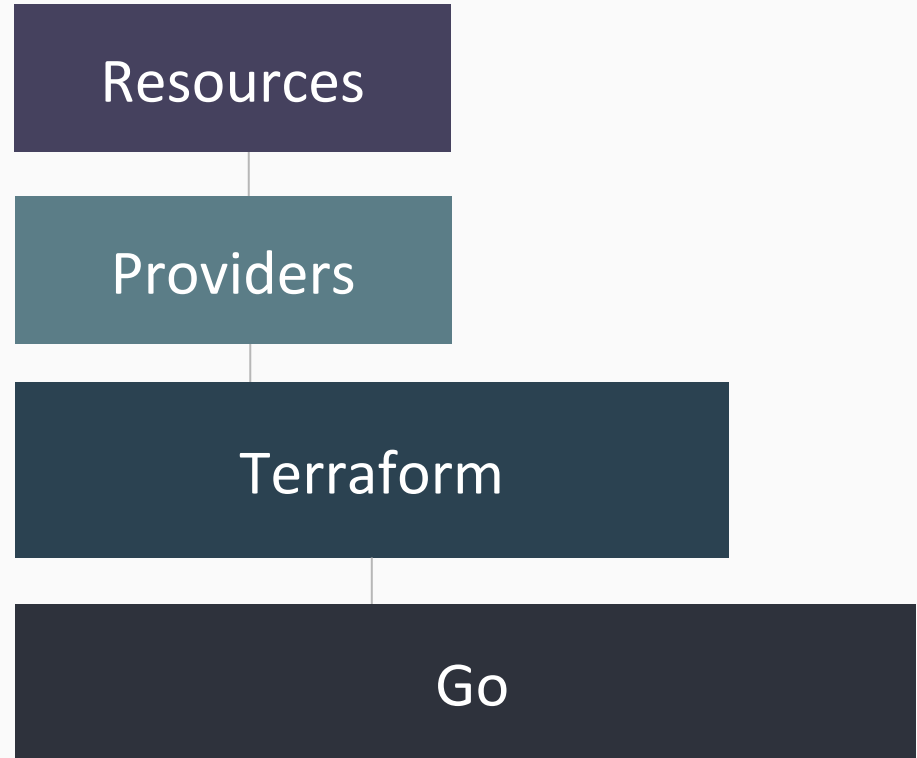
Components

Custom *infrastructure-as-code* configuration depends on Terraform providers and resources, which in turn depend on Go, which Terraform is written in.



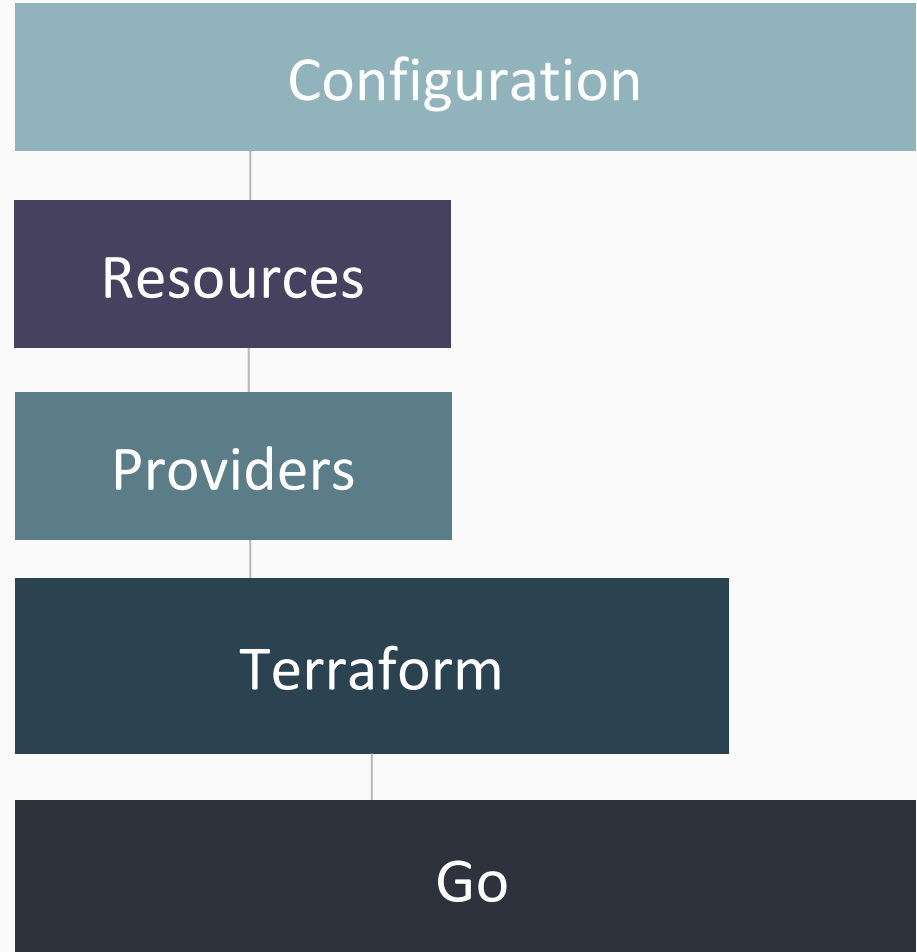
Components

Custom *infrastructure-as-code* configuration depends on Terraform providers and resources, which in turn depend on Go, which Terraform is written in.



Components

Custom *infrastructure-as-code* configuration depends on Terraform providers and resources, which in turn depend on Go, which Terraform is written in.



Terraform Providers

A provider is responsible for understanding API interactions and exposing resources (things you want to create). Examples of providers are [OpenStack](#), AWS, Azure, GCP, Digital Ocean, etc.

```
provider "openstack" {  
  # Use the BC data centre's resources generally, as they're cheaper.  
  auth_url = var.compute_auth_url  
  region = var.compute_region_name  
  # Authentication variables taken from environment after running your  
  # OpenStack RC file, which authenticates a bash shell session to access the  
  # APIs. Download it from https://dash.ca-bc-1.clouda.ca/project/api_access/.  
}
```

297

That's how many different cloud providers Terraform supports out-of-the-box. For details, see [the list of providers](#) in the Terraform Registry.

Terraform Resources

A resource is a thing you want to create at your cloud provider. Examples of resources are security groups, VMs, and networking elements such as routers and floating IP addresses.

```
resource "openstack_compute_instance_v2" "gateway" {  
  count = 1  
  name = "gateway${count.index}"  
  image_id      = var.compute_image_id  
  flavor_id     = var.compute_flavor_id_smallest  
  key_pair      = "my-keypair"  
  security_groups = ["vpn"]  
  network {  
    name = "Main network"  
  }  
  depends_on = [  
    openstack_networking_secgroup_v2.vpn,  
  ]  
}
```

Terraform Variables

Just like in other types of code, [variables](#) can be used.

Format: **var.VARIABLE_ID**

```
variable "compute_image_id" {  
  # `openstack image list` -> Ubuntu 18.04 x86_64 (05/31/2018)  
  default = "d0daa92c-d1cd-4bbb-a2b5-0c28fb4dfe85"  
}  
  
variable "compute_flavor_id_smallest" {  
  # `openstack flavor list` -> 512 MB / 1 VCPU  
  default = "0e10af5c-231b-40d7-b786-b5eeb170dea1"  
}
```

Security Groups

Good cloud providers will have firewall rules allowing access to resources, which act as an allow list preventing insecure access (e.g. to your DB servers).

- Rule groups can be created
- Rules can be added to each group

```
resource "openstack_networking_secgroup_v2" "web" {
  name          = "web"
  description   = "Web servers"
}

resource "openstack_networking_secgroup_rule_v2" "http_connections" {
  direction      = "ingress"
  ethertype      = "IPv4"
  protocol       = "tcp"
  port_range_min = 80
  port_range_max = 80
  remote_ip_prefix = "0.0.0.0/0"
  security_group_id = openstack_networking_secgroup_v2.web.id
}

resource "openstack_networking_secgroup_rule_v2" "https_connections" {
  direction      = "ingress"
  ethertype      = "IPv4"
  protocol       = "tcp"
  port_range_min = 443
  port_range_max = 443
  remote_ip_prefix = "0.0.0.0/0"
  security_group_id = openstack_networking_secgroup_v2.web.id
}
```

Defining VMs/servers

Using everything we've learned so far, it's possible to configure virtual machines (VMs) to be whatever server types are required.

```
resource "openstack_compute_instance_v2" "gateway" {  
  count = 1  
  name = "gateway${count.index}"  
  image_id      = var.compute_image_id  
  flavor_id     = var.compute_flavor_id_smallest  
  key_pair      = "my-keypair"  
  security_groups = ["vpn"]  
  network {  
    name = "Main network"  
  }  
  depends_on = [  
    openstack_networking_secgroup_v2.vpn,  
  ]  
}
```

Configuring VMs from within Terraform

While infrastructure automation can be handled by Terraform, VM configuration can be handled by Ansible.

They can work together in several ways:

- Terraform can run Ansible playbooks. ← *Today's demo*
- Ansible can run Terraform code.
- Terraform can provide a dynamic inventory of VMs that can be used independently by Ansible.

Configuring VMs from within Terraform

```
resource "null_resource" "provision_wireguard_on_gateway" {
  depends_on = [openstack_compute_floatingip_associate_v2.public_ip_gateway0]
  connection {
    type = "ssh"
    user = "ubuntu"
    host = openstack_networking_floatingip_v2.floating_ip_gateway0.address
  }

  # Run the Ansible role to install the app.
  provisioner "local-exec" {
    command = "export ANSIBLE_STDOUT_CALLBACK=debug; ansible-playbook -vv --inventory ${openstack_networking_floatingip_v2.floating_ip_gateway0.address}, --extra-vars 'skip_host_key_validation=true' vpn-server.yml"
    working_dir = "../playbooks/hosts"
  }
}
```

Applying defined infrastructure (& deleting it)

terraform apply commits defined infrastructure by making changes necessary to actualize the desired state.

terraform destroy removes all defined infrastructure *managed by Terraform*.

```
Plan: 24 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: █
```


Saving State

In order keep track of IDs and other metadata, Terraform saves state. Traditionally, this was stored locally, but for enterprise/team settings, remote [backends](#) can now be used.

```
terraform {  
  backend "swift" {  
    container          = "terraform-state"  
    archive_container = "terraform-state-archive"  
    # Use the NS data centre's object store (as that's the only one).  
    auth_url = "https://keystone.ca-ns-1.clouda.ca:8443/v2.0"  
    region_name = "regionOne"  
  }  
}
```

How does Ansible support an *infrastructure-as-code* strategy?

Ansible allows us to define configuration in a simple YAML syntax.

These files can, in turn, be committed into version control, and thus handled as software.

```
- name: Create a Linode server.  
  linode:  
    name: linode-test1  
    plan: 1  
    datacenter: 2  
    distribution: 99  
    password: 'secureRootPassword'  
    private_ip: yes  
    ssh_pub_key: 'ssh-rsa qwerty'  
    swap: 768  
    wait: yes  
    wait_timeout: 600  
    state: present
```

Components

Applying configuration depends on Ansible, roles and modules, which in turn depend on various Python libraries.



Ansible

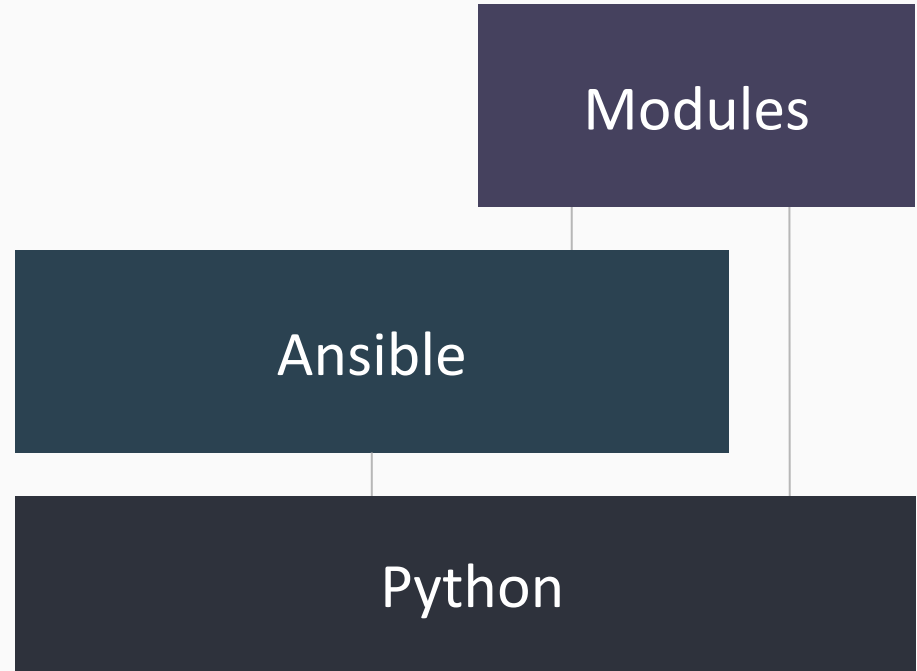
Components

Applying configuration depends on Ansible, roles and modules, which in turn depend on various Python libraries.



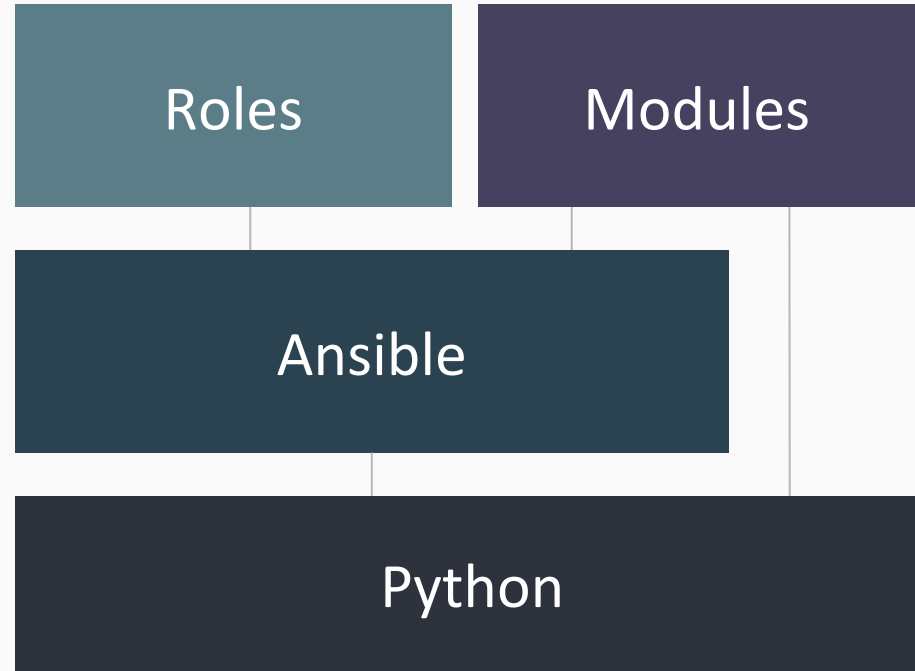
Components

Applying configuration depends on Ansible, roles and modules, which in turn depend on various Python libraries.



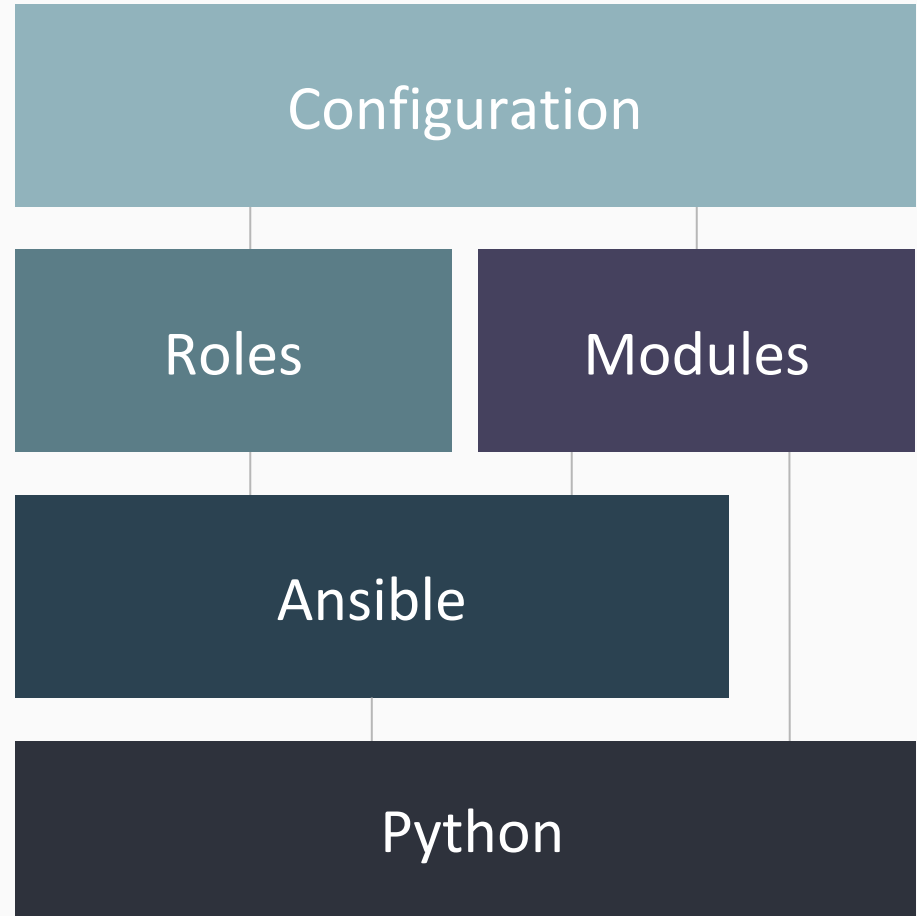
Components

Applying configuration depends on Ansible, roles and modules, which in turn depend on various Python libraries.



Components

Applying configuration depends on Ansible, roles and modules, which in turn depend on various Python libraries.



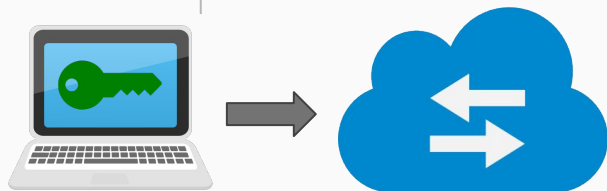
Authentication and Authorization



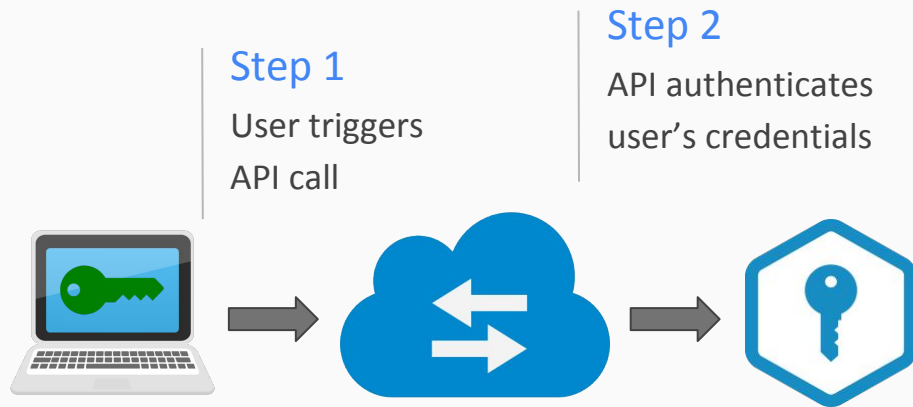
Authentication and Authorization

Step 1

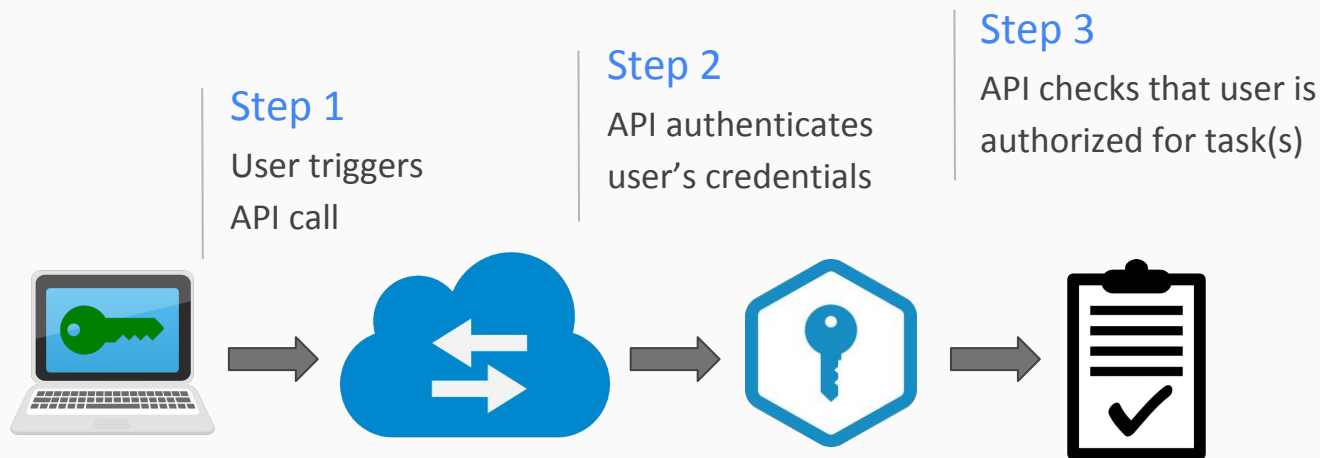
User triggers
API call



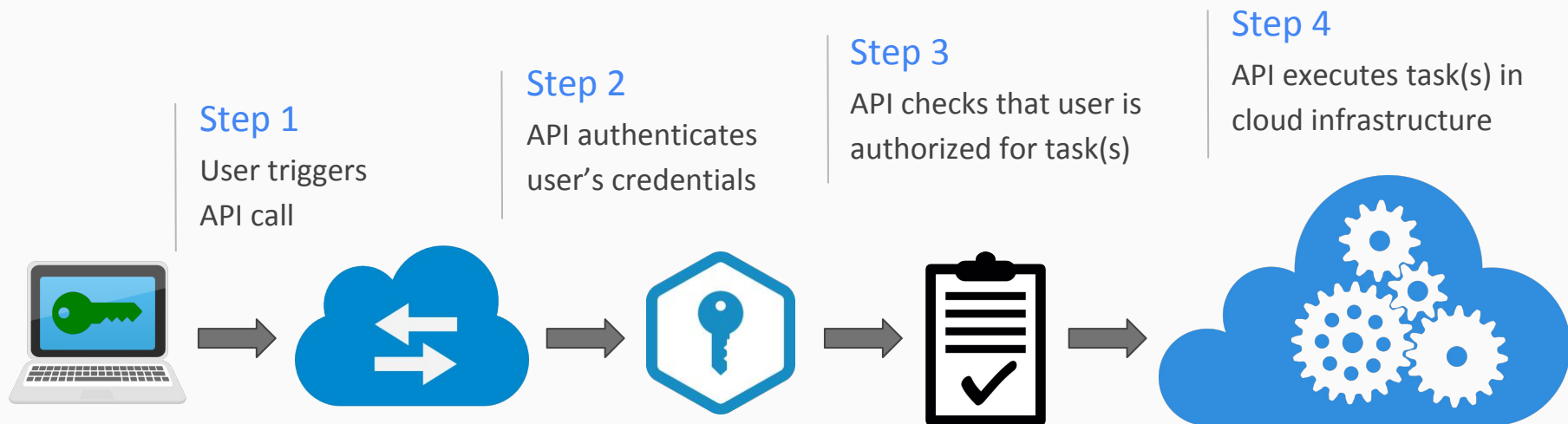
Authentication and Authorization



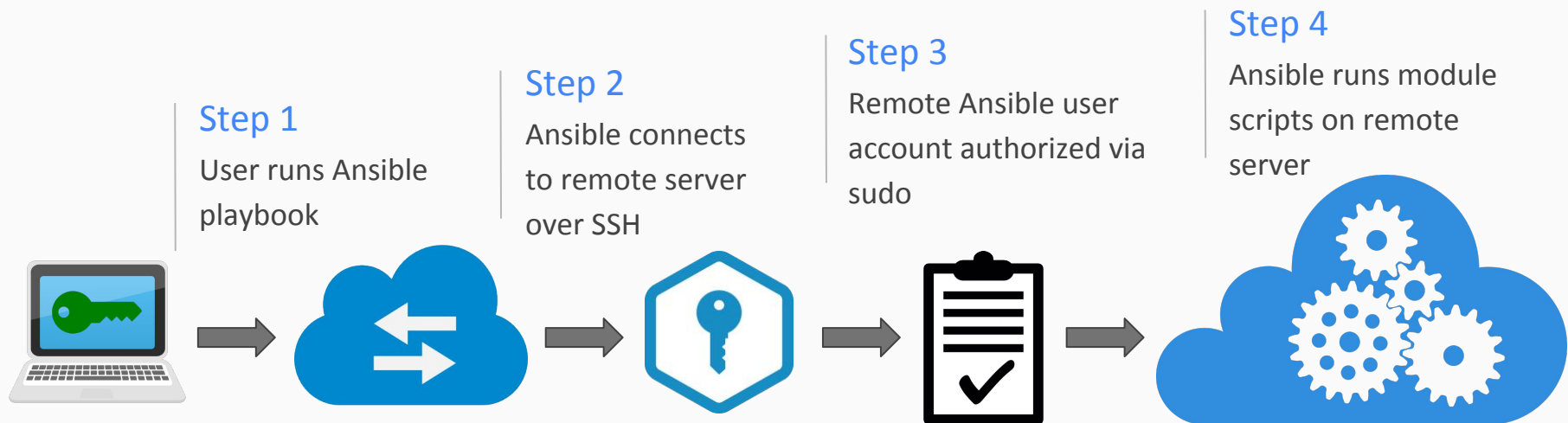
Authentication and Authorization



Authentication and Authorization (Terraform)



Authentication and Authorization (Ansible)



Putting It All Together

Automate All the Things!

Questions?



<https://consensus.enterprises>

Colan Schwartz (colan on d.o)

Christopher Gervais (ergonlogic on d.o)

Dan Friedman (llamech on d.o)



Drupal GovCon is brought to you by Drupal4Gov, a non-profit organization dedicated to connecting and serving government employees committed to open source technology. Drupal is a registered trademark of Dries Buytaert