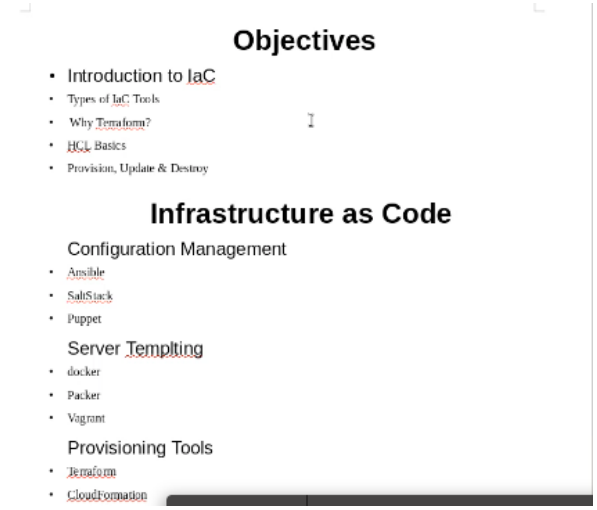
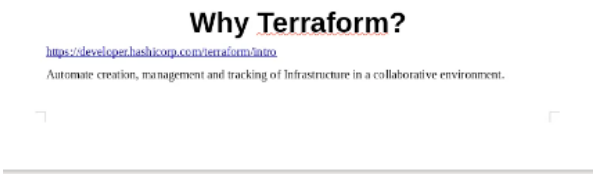
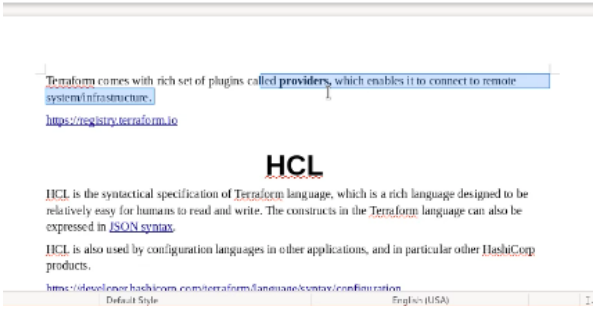
Terraform & Ansible

https://itnext.io/how-to-use-terraform-to-create-a-small-scale-cloud-infrastructure-abf54fabc9dd#73cd

So, before Infrastructure as Code(Terraform). All this creation of infrastructure was done manually by browsing & clicking. While terraform automates this thing by having HCL scripts for the same tasks.







How does Terraform work?

Terraform creates and manages resources on cloud platforms and other services through their application programming interfaces (APIs). Providers enable Terraform to work with virtually any platform or service with an accessible API.

HashiCorp and the Terraform community have already written **thousands of providers** to manage many different types of resources and services. You can find all publicly available providers on the Terraform Registry, including Amazon Web Services (AWS), Azure, Google Cloud Platform (GCP), Kubernetes, Helm, GitHub, Splunk, DataDog, and many more.

The core Terraform workflow consists of three stages:

- **Write:** You define resources, which may be across multiple cloud providers and services. For example, you might create a configuration to deploy an application on virtual machines in a Virtual Private Cloud (VPC) network with security groups and a load balancer.
- **Plan:** Terraform creates an execution plan describing the infrastructure it will create, update, or destroy based on the

```
resource "local_file" "games" {
  filename     = "/tmp/favorite-games"
  content  = "FIFA 21 and IPL"
}
file.tf (END)
```

```
root@test:~                                    guest@test:~/Terraform/file
[guest@test file]$ terraform plan

Terraform used the selected providers to generate the following execution plan. Resource actions are indicated with the following
symbols:
  + create

Terraform will perform the following actions:

  # local_file.games will be created
  + resource "local_file" "games" {
      + content              = "FIFA 21"
      + directory_permission = "0777"
      + file_permission      = "0777"
      + filename             = "/tmp/favorite-games"
      + id                   = (known after apply)
    }

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


Note: You didn't use the -out option to save this plan, so Terraform can't guarantee to take exactly these actions if you run
"terraform apply" now.
[guest@test file]$ cat file.tf
resource "local_file" "games" {
  filename     = "/tmp/favorite-games"
  content  = "FIFA 21"
}
[guest@test file]$
```

```
Terraform will perform the following actions:

  # local_file.games will be created
  + resource "local_file" "games" {
      + content              = "FIFA 21"
      + directory_permission = "0777"
      + file_permission      = "0777"
      + filename             = "/tmp/favorite-games"
      + id                   = (known after 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

local_file.games: Creating...
local_file.games: Creation complete after 0s [id=f68b901eb16aff12e9458bdb656a7df8d3425d4c]

Apply complete! Resources: 1 added, 0 changed, 0 destroyed.
[guest@test file]$
```

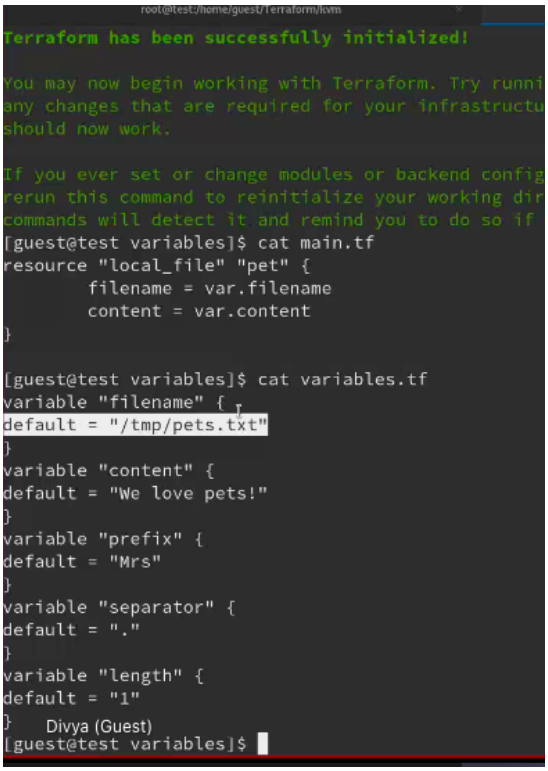
`terraform init` installs providers

`terraform plan`

`terraform apply`

`terraform destroy`

Name of the file: variables.tf



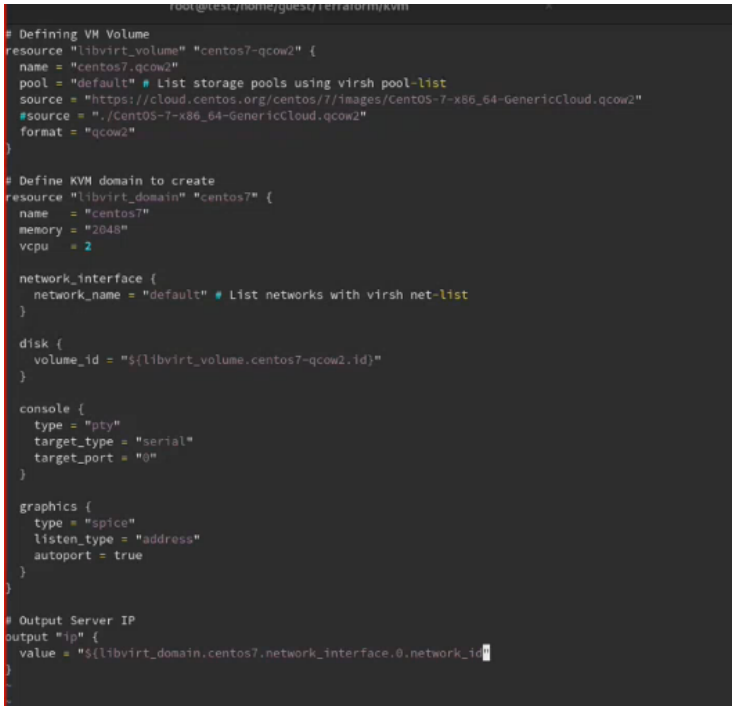
A resource in Terraform code is written as follows:

```
resource“<resource_type>” “<resource_name>” {
}
```

The “*<resource_name>*” is a user defined name. This name must be unique in the scope of your code. Lets say its called, `“ node1 ”`
, then this would mean that no other resource of type `“ null_resource ”`
with name `“ node1 ”`
may exist.

```
# add provisioners here
 provisioner“local-exec”{
 command = “echo >> ${path.module}/node1.txt”
 }provisioner“local-exec”{
 command = “rm ${path.module}/node1.txt”
 when = destroy
 }
```

Now two local provisioners are added. These, as described before, are functions used to execute commands on the host machine. The `‘provisioner’` functions used here are of type `“local-exec”`**.** It expects a command which it will run on an interpreter, on default being `‘/bin/sh’`
on ubuntu OS. In the case of the example, the first function runs a command to create a new text file called ‘*node1.txt*’ in the directory of our ‘*test.tf*’ file and the second function runs a command to remove a file called ‘*node1.txt*’ in the directory its called from. Note that the second function is only called when Terraform destroy is called. This is configured with `‘when = destroy’`



`terraform init` installs providers in .terraform folder

Filename	Purpose
main.tf	Main configuration file containing resource definition
variables.tf	Contains variable declarations
outputs.tf	Contains outputs from resources
provider.tf	Contains Provider definition

Variable Type

- **string**
- **number**
- **bool**
- **any**
- **list**
- **map**
- **set**
- **object**
- **tuple**

The name of file containing variables is not necessarily variables.tf, it can be anything. but its best practice to name it like that.

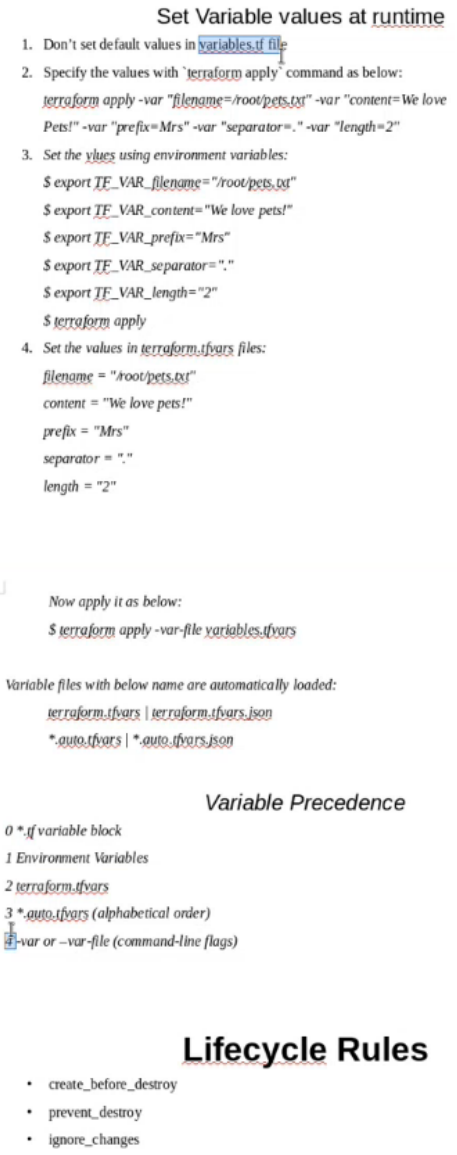
If we run terraform apply and the variables defined don't have a default value, then it asks for values from the user, similar to scanf.

List

```
variable "prefix" {
default = ["Mr", "Mrs", "Sir"]
type = list  0    1    2
}
```

Map

```
variable file-content {
  type = map
  default = {
    "statement1" = "We love pets!"
    "statement2" = "We love animals!"
  }
}
```



so command-line flags has the highest precedence, it will over ride others.

Lifecycle in Terraform

usually when we change some config in .tf file, for example changing content in `local_file` kind of resource, then it destroys the previous resource and creates a new one:

```
kartik@kartik-MacBookAir:~/Desktop/Devops/terraform$ terraform apply
local_file.games: Refreshing state... [id=10a06af598b93b35b86f92bdb417accf889272
e7]

Terraform used the selected providers to generate the following execution plan.
Resource actions are indicated with the following symbols:
-/+ destroy and then create replacement

Terraform will perform the following actions:

  # local_file.games must be replaced
-/+ resource "local_file" "games" {
      ~ content              = "this is created from terraform" -> "this is crea
ted" # forces replacement
      ~ id                   = "10a06af598b93b35b86f92bdb417accf889272e7" -> (kn
own after apply)
        # (3 unchanged attributes hidden)
    }

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

but if we specify lifecycle method of prevent_destroy as shown: then, it prevents the terraform apply after making change to content. as show:

```
kartik@kartik-MacBookAir:~/Desktop/Devops/terraform$ cat main.tf
resource "local_file" games{
filename ="/tmp/favourite-games"
content = "this is created in terraform"
lifecycle{
        prevent_destroy =true
}
}
kartik@kartik-MacBookAir:~/Desktop/Devops/terraform$ terraform apply
local_file.games: Refreshing state... [id=5a573460be834d5389aeccfc68ce75f8e8c459
a1]

  Error: Instance cannot be destroyed

    on main.tf line 1:
     1: resource "local_file" games{

  Resource local_file.games has lifecycle.prevent_destroy set, but the plan
  calls for this resource to be destroyed. To avoid this error and continue
  with the plan, either disable lifecycle.prevent_destroy or reduce the scope
  of the plan using the -target flag.

kartik@kartik-MacBookAir:~/Desktop/Devops/terraform$
```

Now, imagine if this was an VM resource, then you have to prevent any changes that can destroy VM, since destroying VM means bringing your application down, hence by using this lifecycle method, this can be prevented.

the other two lifecycle methods are:

`create_before_destroy and ignore_changes`

Data source:

the purpose of data block is to read from the resource and create a tfstate file which can be used in some other terraform script, especially the content block for binary file like in above image.

Assume you created a resource without using terraform, but you need its data in some other resource object in terraform script then you will use data object in terraform to fetch that info and use it using `data.type.varname`

terraform state list & terraform state show <resource>

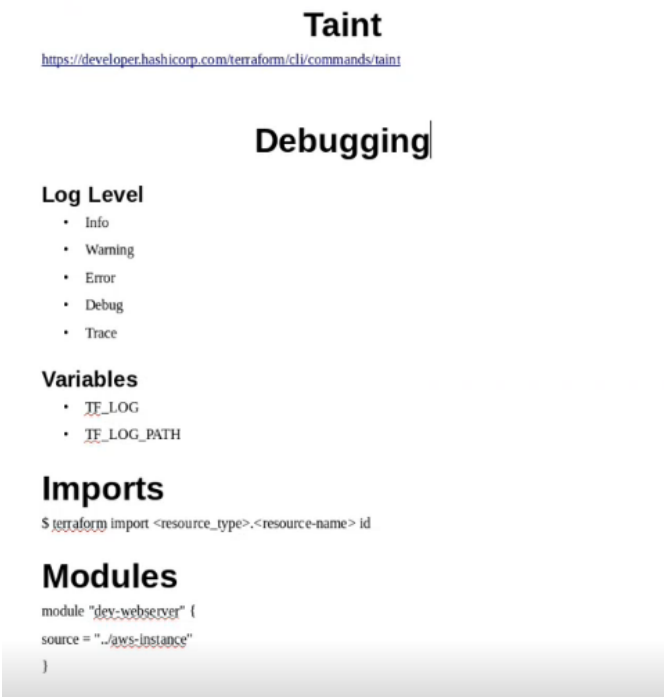
```
kartik@kartik-MacBookAir$ terraform state list
local_file.games
kartik@kartik-MacBookAir$ terraform state show local_file.games
# local_file.games:
resource "local_file" "games" {
    content              = "this is created"
    directory_permission = "0777"
    file_permission      = "0777"
    filename             = "/tmp/favourite-games"
    id                   = "5a573460be834d5389aeccfc68ce75f8e8c459a1"
}
kartik@kartik-MacBookAir:~/Desktop/Devops/terraform$
```

Provisioner

It is used to perform some action, after creating a resource

```
resource "aws_instance" "web" {
  # ...

  provisioner "local-exec" {
    command = "echo The server's IP address is ${self.private_ip}"
  }
}
```



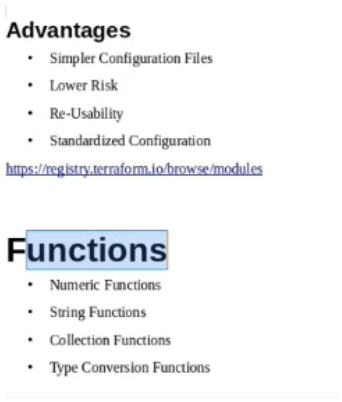
turn on logging as:, set it to any logging level, trace has max amount of information printed.

TF_LOG_PATH is for redirecting logs to a file.

```
export TF_LOG="Trace"
```

terraform import is used to import an resource which was manually created and now it is to be managed by terraform, it creates the tfstate file for resource containing all metadata.

Modules are used to extend a already created terraform object into a new project, for code reusablity.



Numeric Functions

- Max
- min
- ceil
- floor

String Functions

- Split
- lower
- upper
- title
- substr
- join

Collection Functions

- Length

Operators & Conditional Expressions

- Numeric Operators → +, -, *, /
- Equality Operator → ==, !=
- Comparison Operator → >, >=, <, <=
- Logical Operator → &&, ||, !

condition ? true_val : false_val

Workspace

$ terraform workspace new <workspace-name>

$ terraform workspace list

Provision VM on KVM

https://computingforgeeks.com/how-to-provision-vms-on-kvm-with-terraform/

count and foreach

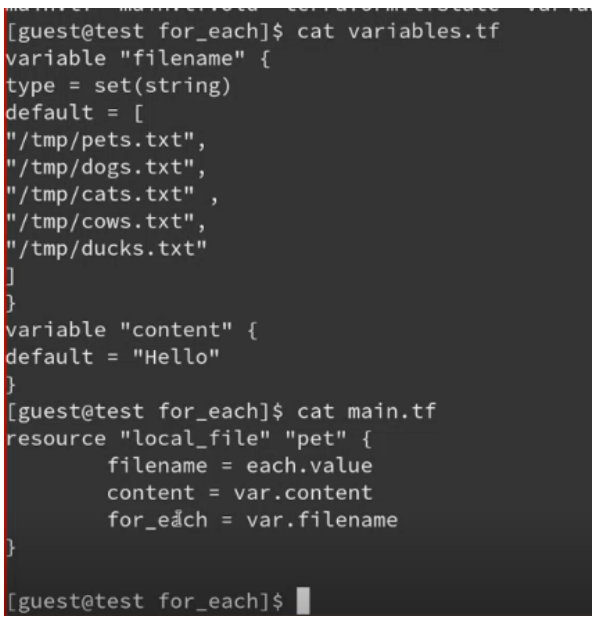
```
[guest@test Terraform]$ cd count/
[guest@test count]$ ls
main.tf  main.tf.old  variables.tf
[guest@test count]$ cat main.tf
resource "local_file" "pet" {
        filename = var.filename
        content = var.content
        count = 2
}

[guest@test count]$ cat variables.tf
variable "filename" {
default = "/tmp/pets.txt"
}
variable "content" {
default = "Hello"
}
[guest@test count]$ terraform ini
```

count =2 will create 2 terraform objects pointing to the same physical object, which is a file /tmp/pets.txt here. Note: only 1 file is created.

here is the tfstate file:

```
"instances": [
  {
    "index_key": 0,
    "schema_version": 0,
    "attributes": {
      "content": "Hello",
      "content_base64": null,
      "directory_permission": "0777",
      "file_permission": "0777",
      "filename": "/tmp/pets.txt",
      "id": "f7ff9e8b7bb2e09b70935a5d785e0cc5d9d0abf0",
      "sensitive_content": null,
      "source": null
    },
    "sensitive_attributes": [],
    "private": "bnVsbA=="
  },
  {
    "index_key": 1,
    "schema_version": 0,
    "attributes": {
      "content": "Hello",
      "content_base64": null,
      "directory_permission": "0777",
      "file_permission": "0777",
      "filename": "/tmp/pets.txt",
      "id": "f7ff9e8b7bb2e09b70935a5d785e0cc5d9d0abf0",
      "sensitive_content": null,
      "source": null
    },
    "sensitive_attributes": [],
```

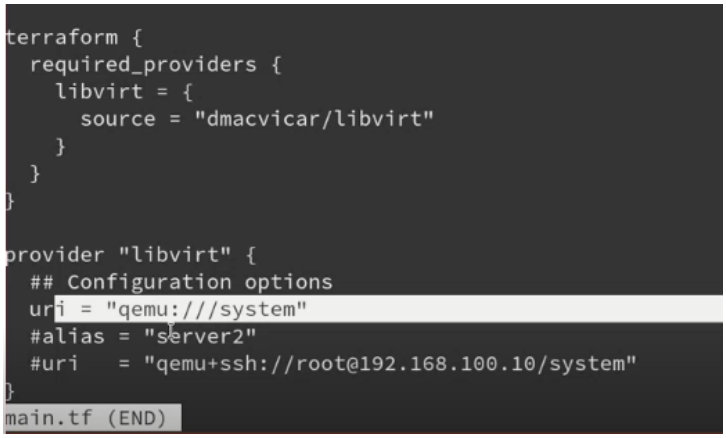



```
[guest@test for_each]$ cat variables.tf
variable "filename" {
type = set(string)
default = [
"/tmp/pets.txt",
"/tmp/dogs.txt",
"/tmp/cats.txt" ,
"/tmp/cows.txt",
"/tmp/ducks.txt"
]
}
variable "content" {
default = "Hello"
}
[guest@test for_each]$ cat main.tf
resource "local_file" "pet" {
        filename = each.value
        content = var.content
        for_each = var.filename
}

[guest@test for_each]$
```

If you manually delete a resource created by terraform then on terraform apply it recreates the file, whe

creating VM from terraform




```
terraform {
  required_providers {
    libvirt = {
      source  = "dmacvicar/libvirt"
      version = "0.7.0"
    }
  }
}

provider "libvirt" {
  uri = "qemu:///system"
}

resource "libvirt_domain" "ubuntu-domain" {
  name   = "ubuntu-22.04-domain"
  memory = "512"

  network_interface {
    network_name = "default"
  }

  disk {
    volume_id = libvirt_volume.ubuntu-server.id
  }
}

resource "libvirt_volume" "ubuntu-server" {
  name   = "ubuntu-22.04"
  source = "https://cloud-images.ubuntu.com/jammy/current/jammy-server-cloudimg-amd64-disk-kvm.img"
}

resource "libvirt_domain" "fedora-domain" {
  name   = "fedora-36-domain"
  memory = "512"

  network_interface {
    network_name = "default"
  }

  disk {
    volume_id = libvirt_volume.fedora-server.id
  }
}

resource "libvirt_volume" "fedora-server" {
  name   = "fedora-36"
  source = "https://download.fedoraproject.org/pub/fedora/linux/releases/36/Cloud/x86_64/images/Fedora-Cloud-Base-36-1.5.x86_64.qcow2"
}
```

for_each is used with set only, so typecast using toset() as follows

```
# Defining VM Volume
resource "libvirt_volume" "devops-qcow" {
  name = each.value
  pool = "default" # List storage pools using virsh pool-list
  source = "/var/lib/libvirt/images/fedora-template.qcow2"
  format = "qcow2"
  for_each = var.volume
}


# Define KVM domain to create
resource "libvirt_domain" "devops-VMs" {
  name   = each.value
  memory = each.value == "jenkins" ? "2048" : "6144"
  vcpu   = each.value == "jenkins" ? 2 : 4

  network_interface {
    network_name = "default" # List networks with virsh net-list
  }

  disk {
    volume_id = "${libvirt_volume.devops-qcow[format("%s.qcow2",each.value)].id}"
  }
  for_each = toset(var.domain)
}
```

Ansible

less /etc/ansible/hosts

/etc/ansible/ansible.cfg

```
# Defining VM Volume
resource "libvirt_volume" "web1-qcow" {
  name = "server1.qcow2"
  pool = "default" # List storage pools using virsh pool-list
  source = "/var/lib/libvirt/images/web1.qcow2"
  #source = "./CentOS-7-x86_64-GenericCloud.qcow2"
  format = "qcow2"
}

resource "libvirt_volume" "server2-qcow" {
  name = "server2.qcow2"
  pool = "default" # List storage pools using virsh pool-list
  source = "/var/lib/libvirt/images/server2.qcow2"
  #source = "./CentOS-7-x86_64-GenericCloud.qcow2"
  format = "qcow2"
}


# Define KVM domain to create
resource "libvirt_domain" "server1" {
  name   = "server1"
  memory = "2048"
  vcpu   = 2

  network_interface {
libvirt.tf
```

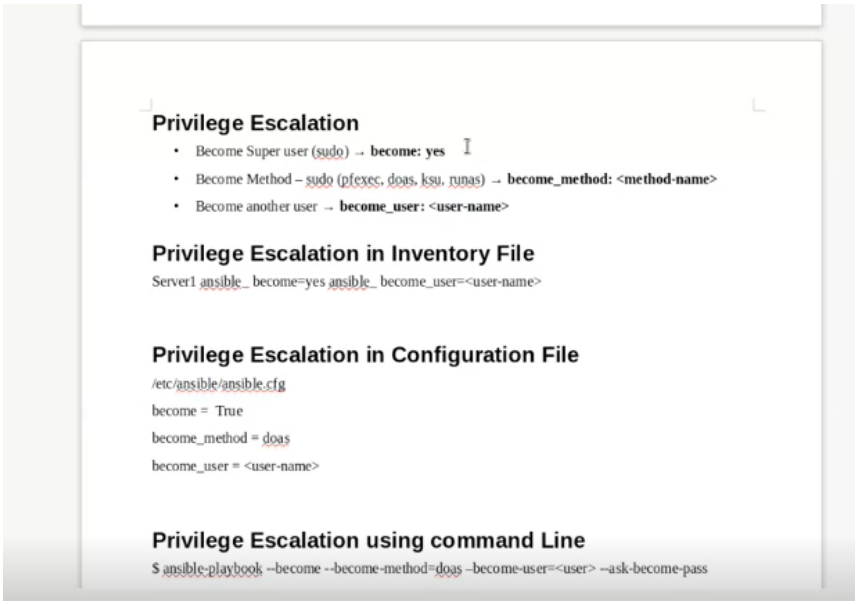
```
server1 ansible_host=192.168.122.193 ansible_ssh_pass=redhat ansible_user=root
server2 ansible_host=192.168.122.156 ansible_ssh_pass=redhat ansible_user=root
inventory (END)
```

```
- hosts: all

  tasks:
    - name: Display inventory hostname
      command: echo "{{inventory_hostname}}"
    - name: Change the hostname to our standard
      ansible.builtin.hostname:
        name="{{inventory_hostname}}"

    - name: Unconditionally reboot the machine with all defaults
      ansible.builtin.reboot:

    - name: Waiting for server to come back up
      ansible.builtin.wait_for_connection:
```



Privilege Escalation

- Become Super user (sudo) → **become: yes**
- Become Method – sudo (pfexec, doas, ksu, runas) → **become_method: <method-name>**
- Become another user → **become_user: <user-name>**

Privilege Escalation in Inventory File

Server1 ansible_ become=yes ansible_ become_user=<user-name>

Privilege Escalation in Configuration File

/etc/ansible/ansible.cfg

become = True

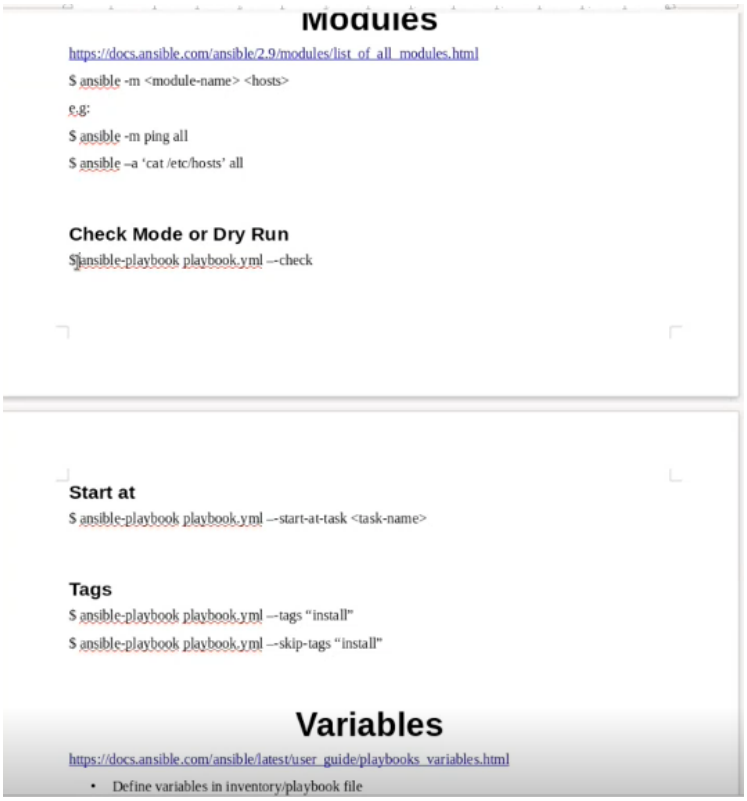
become_method = doas

become_user = <user-name>

Privilege Escalation using command Line

$ ansible-playbook --become --become-method=doas --become-user=<user> --ask-become-pass

```
- name: Print hello message
  hosts: all
  tasks:
    - debug:
        msg: Hello from Ansible!
hello.yaml (END)
```

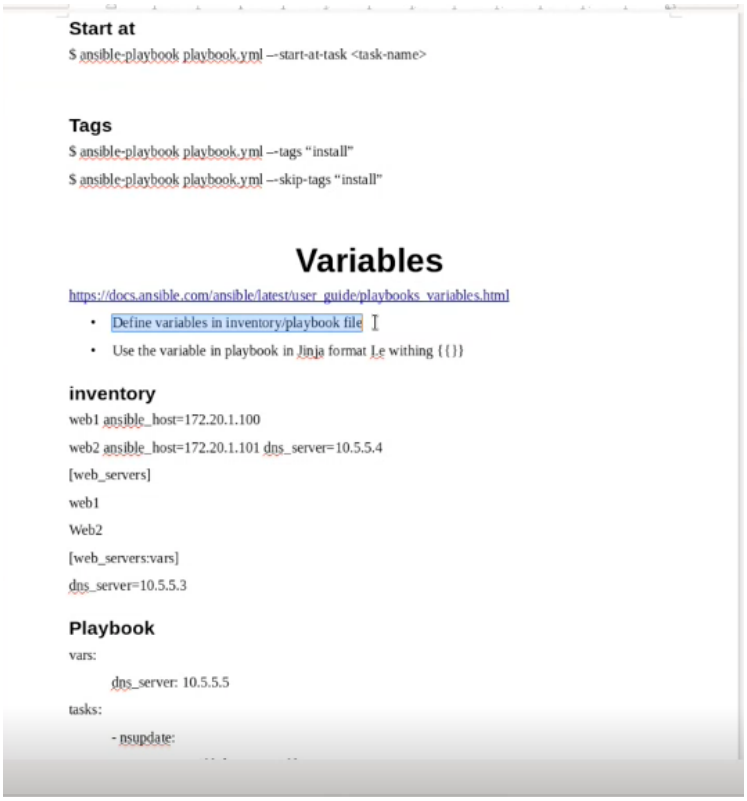



```
hosts: all
tasks:
  - name: Copy index.html to remote servers
    copy:
      src: server.j2
      dest: /tmp/server
copy.yml (END)
```

```
---
  - name: Install httpd
    tags: install and start
    hosts: all
    tasks:
    - yum:
        name: httpd
        state: installed
      tags: install
    - service:
        name: httpd
        state: started
      tags: start httpd service
(END)
```

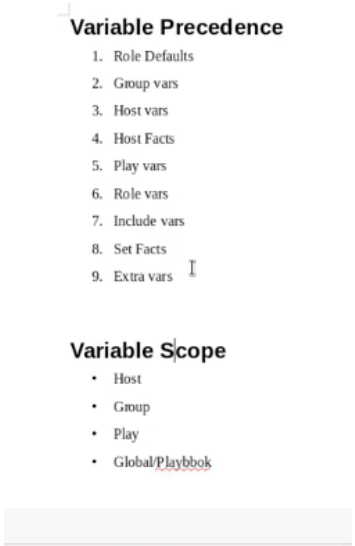
`ansible-playbook playbook.yml --check` validates the yaml file and checks for err

you can use tags to just run a specific task instead of all, or even skip some tasks.



In inventory file, we first create two hosts web1 and web2, provide default variable called ansible_host, and dns_server var for web2.

then we create a group called web_servers, add web1, web2 to it. then we define variables for the group using `[web_servers.vars]` now this new variable dns_server=10.5.5.3 is effective for whole group, but web2 will have the 10.5.5.4 dns_server only . we will see variable precedence further.



register variable:

```
---
- name: Check /etc/hosts file
  hosts: all
  tasks:
  - shell: cat /etc/hosts
    register: result
  - debug:
      var: result
register.yml (END)
```

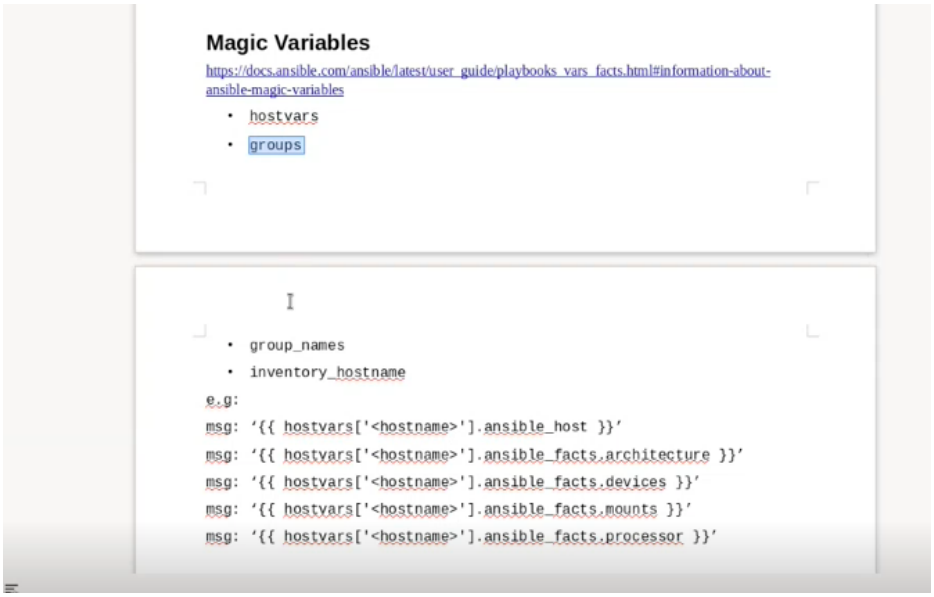
saving the output of shell : cat /etc/hosts to variable called result. and printing it in the debug task. Here “shell” and “debug” are different modules.

you can set inventory as env variable using this:

```
[guest@test sample]$ env | grep ANSIBLE
ANSIBLE_INVENTORY=/home/guest/Ansible/hostname/inventory
[guest@test sample]$ ansible-playbook
```

So to capture output of a command we use register variables.

Magic variables are the builtin variables defined.



Conditionals

- when

Operators

- Or
- and

Loops

- Loop keyword to iterate over a list

Blocks

- Groups tasks

Error Handling

- Rescue block for action to be taken in case of failure
- always block to be executed at the end irrespective of task status

```
---
- name: Install NGINX
  hosts: all
  tasks:
  - name: Install NGINX on Debian
    apt:
      name: nginx
      state: present
    when: ansible_os_family == “Debian”
  - name: Install NGINX on Redhat
    yum:
      name: nginx
      state: present
    when: ansible_os_family == “RedHat”
conditional.yml (END)
```

```
- name: Check status of a service and email if its down
  hosts: all
  tasks:
  - command: service httpd status
      register: result
  - mail:
      to: admin@company.com
      subject: Service Alert
      body: Httpd Service is down
    when: result.stdout.find('down') != -1
cond-reg.yml (END)
```

adding hostname also on which the service is down using magic variables.

```
root@test:/home/guest
- name: Check status of a service and email if its down
  hosts: all
  tasks:
  - command: service httpd status
      register: result
  - mail:
      to: admin@company.com
      subject: Service Alert
      body: Httpd Service is down {{inventory_hostname}}
    when: result.stdout.find('down') != -1
~
~
~
~
```

loop

```
---
- name: Install Softwares
  hosts: all
  vars:
    packages:
      - name: nginx
        required: True
      - name: mysql
        required : True
      - name: apache
        required : False
  tasks:
  - name: Install “{{ item.name }}” on Debian
    apt:
      name: “{{ item.name }}”
      state: present
    when: item.required == True
    loop: "{{ packages }}"
loop.yml (END)
```

blocks to group similar kind of tasks

```
-
  hosts: server1
  tasks:
    - block:
      - name: Install MySQL
        yum: name=mysql-server state=present
      - name: Start MySQL Service
        service: name=mysql-server state=started
      become_user: db-user
      when: ansible_facts['distribution'] == 'CentOS'
    - block:
      - name: Install Nginx
        yum: name=nginx state=present
      - name: Start Nginx Service
        service: name=nginx state=started
      become_user: web-user
      when: ansible_facts['distribution'] == 'CentOS'
block.yml (END)
```

`become_user` keyword is used to do a task as that user, it is used to escalate priviliges.

rescue block: it is executed if only of the task failed.

always: it will execute whether passed or failed.

```
-
  hosts: server1
  tasks:
  - block:
    - name: Install MySQL
      yum: name=mysql-server state=present
    - name: Start MySQL Service
      service: name=mysql-server state=started
    become_user: db-user
    when: ansible_facts['distribution'] == 'CentOS'
    rescue:
      - mail:
          to: admin@company.com
          subject: Installation Failed
          body: DB Install Failed at {{ ansible_failed_task.name }}
    always:
      - mail:
          to: admin@company.com
          subject: Installation Status
          body: DB Install Status - {{ ansible_failed_result }}
error.yml (END)
```

Task failure

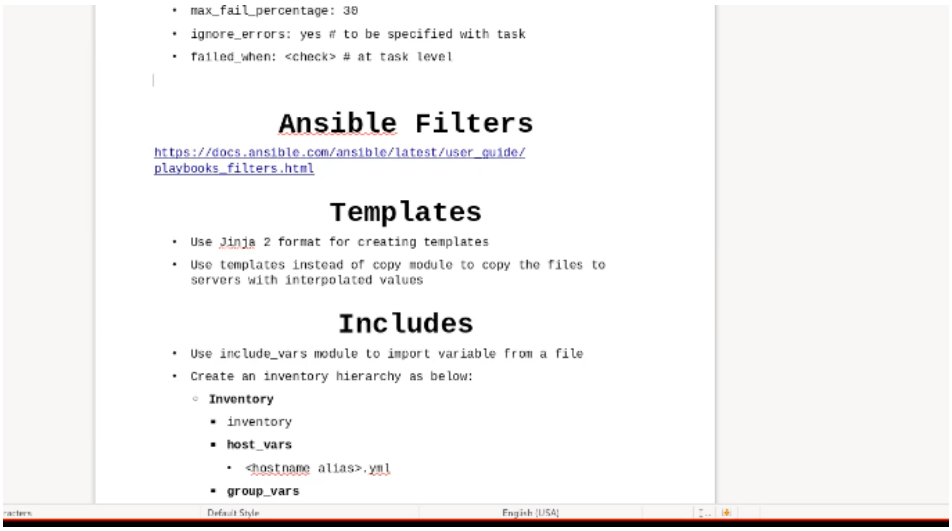
- any_errors_fatal: true
- max_fail_percentage: 30
- ignore_errors: yes # to be specified with task
- failed_when: <check> # at task level

Ansible Filters

https://docs.ansible.com/ansible/latest/user_guide/

- any_errors_fatal:true means it aborts the playbook, if any error occurs in a task, whereas by default all the tasks are performed sequentially, even if one task fails.
- max_fail_percentage: if you have n number of servers, for 1 task if it fails for 30% of servers then do not proceed further and abort, else continue.
- ignore_errors: obvious
- failed_when: it searches for a particular keyword in output of command, and marks it failed if that keyword is found. example:

```
- name: Deploy web application
  hosts: all
  any_errors_fatal: true
  tasks:
    - name: Install dependencies
      << code hidden >>
    - name: Install MySQL Database
      << code hidden >>
    - name: Start MySQL Service
      << code hidden >>
    - name: Install Python Flask Dependencies
      << code hidden >>
    - name: Run web-server
      << code hidden >>
    - command: cat /var/log/server.log
      register: command_output
      failed_when: 'ERROR' in command_output.stdout
    - mail:
        to: admin@company.com
        subject: Server Configured
        body: Web server has been configured
fail.yml (END)
```



ansible filters

Manipulating strings

To add quotes for shell usage:

```
- name: Run a shell command
  ansible.builtin.shell: echo {{ string_value | quote }}
```

To concatenate a list into a string:

```
{{ list | join(" ") }}
```

To split a string into a list:

```
{{ csv_string | split(",") }}
```

New in version 2.11.

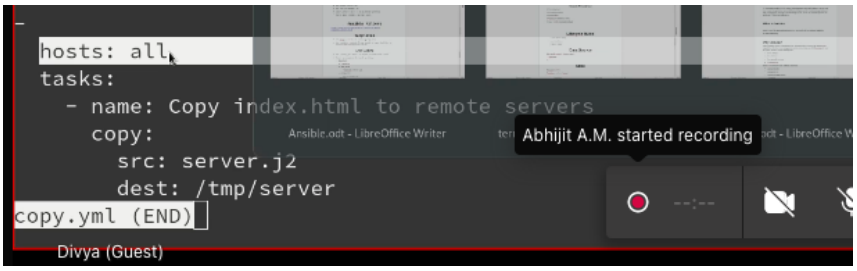
To work with Base64 encoded strings:

```
{{ encoded | b64decode }}
{{ decoded | string | b64encode }}
```

As of version 2.6, you can define the type of encoding to use, the default is `utf-8`:

```
{{ encoded | b64decode(encoding='utf-16-le') }}
{{ decoded | string | b64encode(encoding='utf-16-le') }}
```

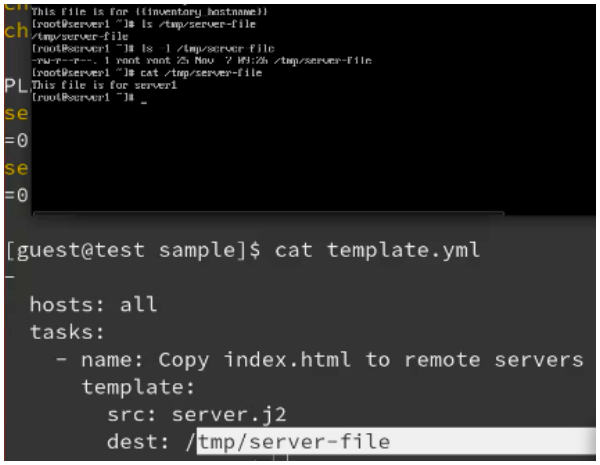
templates:



If you run above file, it just copies as text, the inventory_hostname is not replaced while copying. whereas if you use template module then it will replace the magic variable in this case.

```
-
  hosts: all
  tasks:
    - name: Copy index.html to remote servers
      template:
        src: server.j2
        dest: /tmp/server-file
template.yml (END)
```

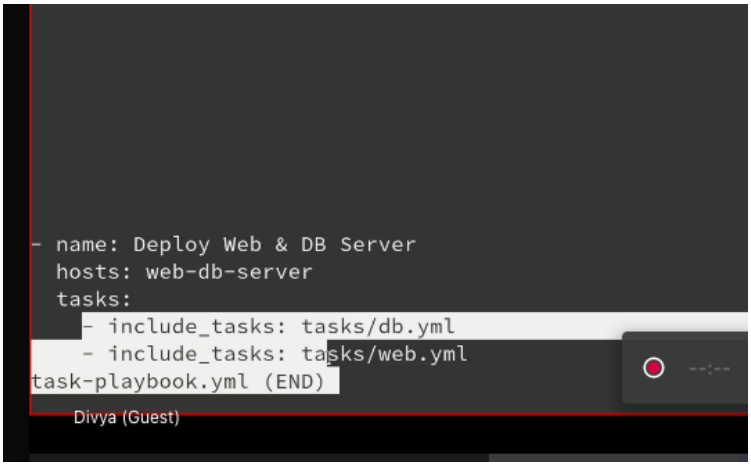
Divya (Guest)

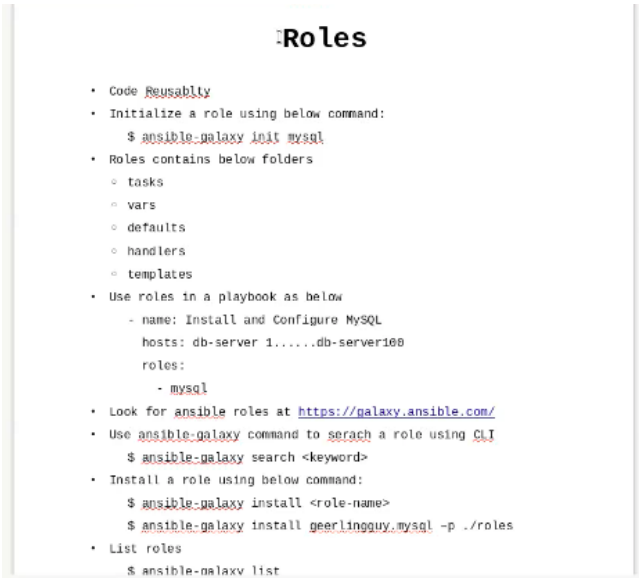


see it says, `this file is for server1`

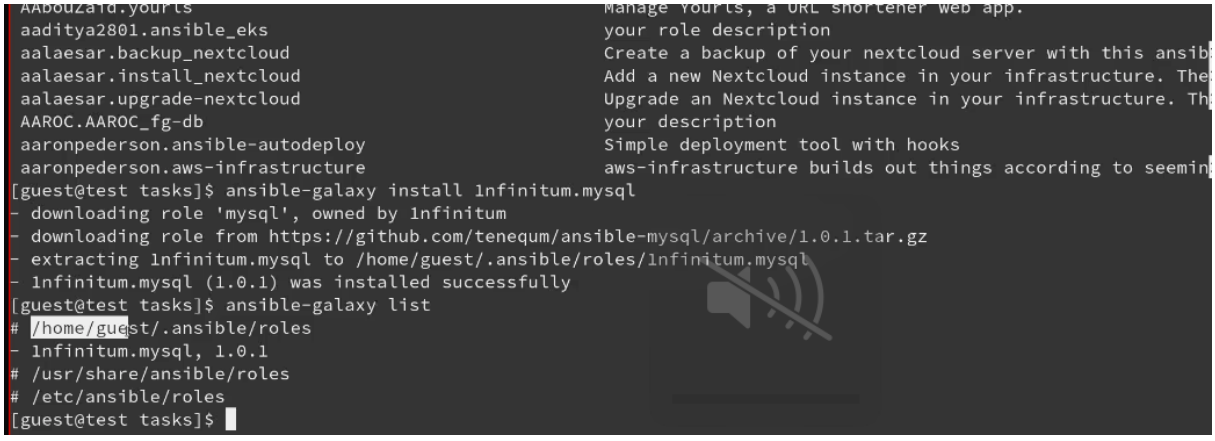
```
- name: Deploy Web & DB Server
  hosts: all
  tasks:
  - include_vars:
      file: /home/guest/Ansible/info.yml
      name: email_data
  - mail:
      to: {{email_data.admin_email }}
      subject: Service Alert
      body: Httpd Service is down
include.yml (END)
```

```
admin_email: admin@company.com
 ESC
```





roles



`/etc/ansible/ansible.cfg` has the file containing path for roles

```
[guest@test 1nfinitum.mysql]$ cd ~
/home/guest/.ansible/roles/1nfinitum.mysql/tasks
[guest@test tasks]$ ls
main.yml
[guest@test tasks]$ cd ..
[guest@test 1nfinitum.mysql]$ cd ~/Ansible/sample/
[guest@test sample]$ ls
archive.yml      cron.yml    filesystem.yml  lineinfile.yml  ping-register.yml  storage.yml        test-role
block.yml        date.yml    file.yml        loop.yml        ping.yml           tags.yml           user.yml
conditional.yml  error.yml   firewall.yml    magic.yml       register.yml       task-playbook.yml  variable.yml
cond-reg.yml     facts.yaml  hello.yaml      no-facts.yml    server.j2          tasks              yum.yml
copy.yml         fail.yml    include.yml     operator.yml    service.yml        template.yml
[guest@test sample]$ grep -i role /etc/ansible/ansible.cfg
# additional paths to search for roles in, colon separated
#roles_path    = /etc/ansible/roles
# by default, variables from roles will be visible in the global variable
# tasks and handlers within the role will see the variables there
#private_role_vars = yes
[guest@test sample]$ export ANSIBLE_ROLES_PATH=/home/guest/role
```

Press ⌘+Shift+M to unmute your microphone.

Strategy

- Linear (Default)
- Free
  - strategy: free
- Batch
  - serial: 3
  - forks value in .cfg file

default behavior is it executes 1st task for all 10 servers then 2nd tasks for all 10 and so on. this is called linear strategy

- strategy: free → it doesn't need to wait for task1 to finish on all 10 servers, the servers run tasks independently, they are free from each other.
- strategy: serial → forks:5 by default. it means it will try to execute task on 5 servers, then on next 5 servers. number of forks should be set according to capacity of machine that is running ansible.