

Jenkins

| | |
|-------------------------------|-----------|
| Jenkins | 1 |
| Terraform Architecture | 4 |
| Ansible | 10 |
| Links | 13 |

Complete Jenkins Pipeline Tutorial | Jenkinsfile explained

Required fields of Jenkinsfile

```
Y dev / Jenkinsfile
1 pipeline {
2     agent any
3     stages {
4         stage("build") {
5             steps {
6                 ...
7             }
8         }
9     }
10 }
11 }
12 }
13 }
14 }
15 }
16 node {
17     // groovy script
18 }
```

- "pipeline" must be top-level
- "agent" - where to execute
- "stages" - where the "work" happens
 - "stage" and "steps"

3:51 / 35:05 • Basic Structure of Jenkinsfile >

Complete Jenkins Pipeline Tutorial | Jenkinsfile explained [/dev/Jenkinsfile](#)

GitLab Projects Groups More [Edit](#)

techworld-js-docker-demo-app

Project overview Repository Files Commits Branches Tags Contributors Graph Compare Charts Locked Files Issues 0 Merge Requests 0 CI / CD

Y dev Jenkinsfile

```
7 stage("build") {  
8     steps {  
9         echo 'building the application...'  
10    }  
11}  
12  
13 stage("test") {  
14    steps {  
15        echo 'testing the application...'  
16    }  
17}  
18  
19 stage("deploy") {  
20    steps {  
21        echo 'deploying the application...'  
22    }  
23}  
24  
25 post {  
26    always {  
27        //  
28    }  
29    failure {  
30        //  
31    }  
32}  
33  
34}  
35  
36}  
37
```

Commit message [Update Jenkinsfile](#)

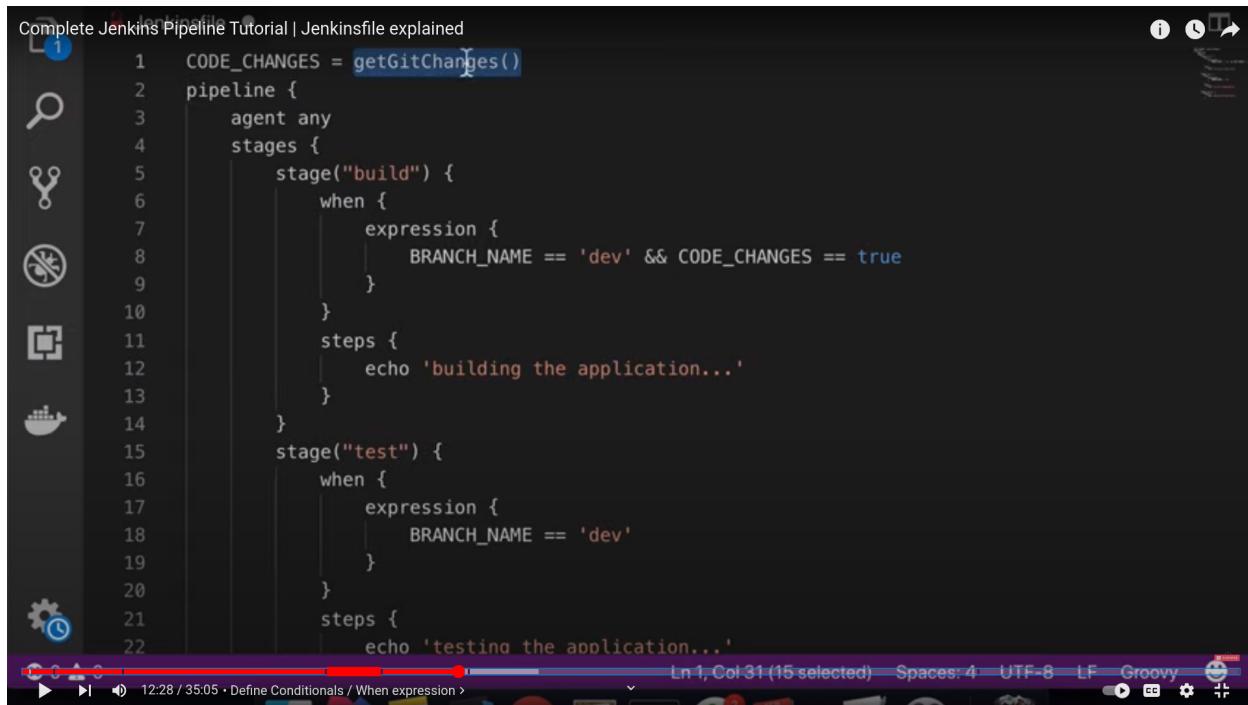
Execute some logic **AFTER** all stages executed

Conditions:

- always
- success
- failure

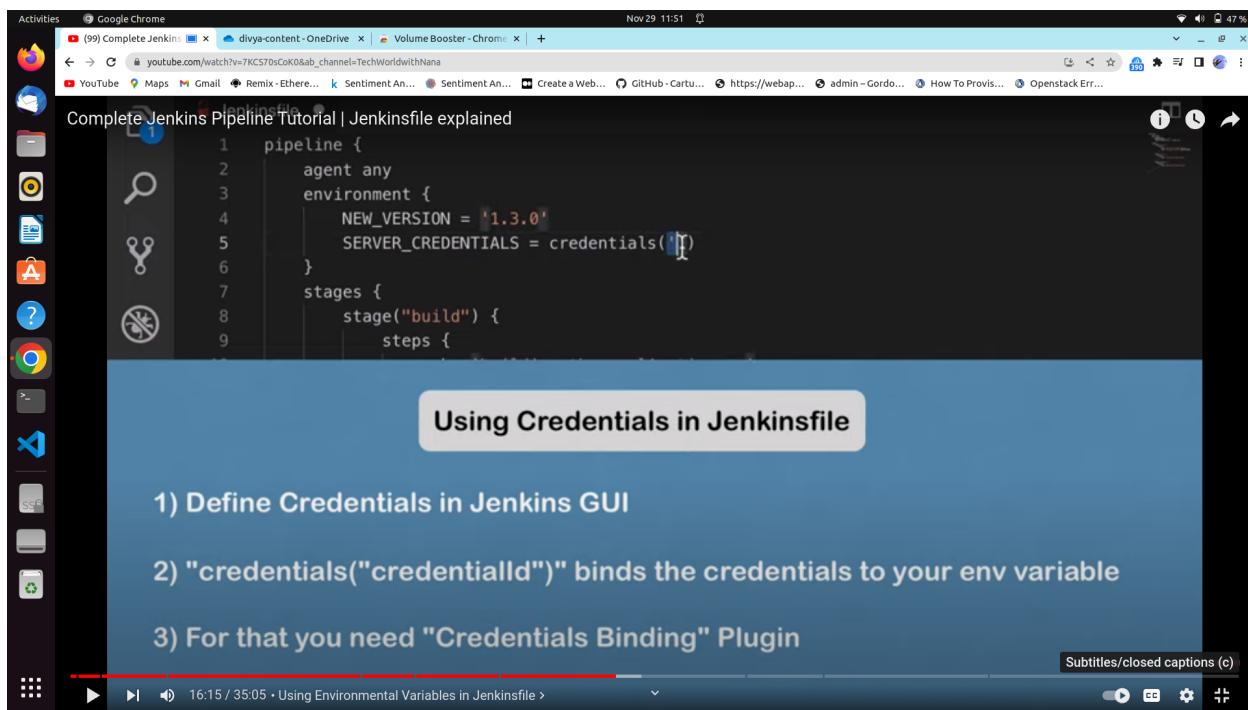
Build Status or Build Status Changes

9:57 / 35:05 • Post Build Actions In Jenkinsfile



```
Complete Jenkins Pipeline Tutorial | Jenkinsfile explained
1   CODE_CHANGES = getGitChanges()
2   pipeline {
3       agent any
4       stages {
5           stage("build") {
6               when {
7                   expression {
8                       BRANCH_NAME == 'dev' && CODE_CHANGES == true
9                   }
10              }
11              steps {
12                  echo 'building the application...'
13              }
14          }
15          stage("test") {
16              when {
17                  expression {
18                      BRANCH_NAME == 'dev'
19                  }
20              }
21              steps {
22                  echo 'testing the application...'
23              }
24          }
25      }
26  }
```

Ln 1, Col 31 (15 selected) Spaces: 4 UTF-8 LF Groovy



Complete Jenkins Pipeline Tutorial | Jenkinsfile explained

```
1   pipeline {
2       agent any
3       environment {
4           NEW_VERSION = '1.3.0'
5           SERVER_CREDENTIALS = credentials()
6       }
7       stages {
8           stage("build") {
9               steps {
```

Using Credentials in Jenkinsfile

- 1) Define Credentials in Jenkins GUI
- 2) "credentials("credentialId")" binds the credentials to your env variable
- 3) For that you need "Credentials Binding" Plugin

Subtitles/closed captions (c)

Activities Google Chrome Nov 29 12:00 46%

(99) Complete Jenkinsfile - OneDrive Volume Booster - Chrome

YouTube Maps Gmail Remix - Ether... Sentiment An... Sentiment An... Create a Web... GitHub - Cartu... https://webap... admin - Gordo... How To Provis... Openstack Err...

Complete Jenkins Pipeline Tutorial | Jenkinsfile explained

```

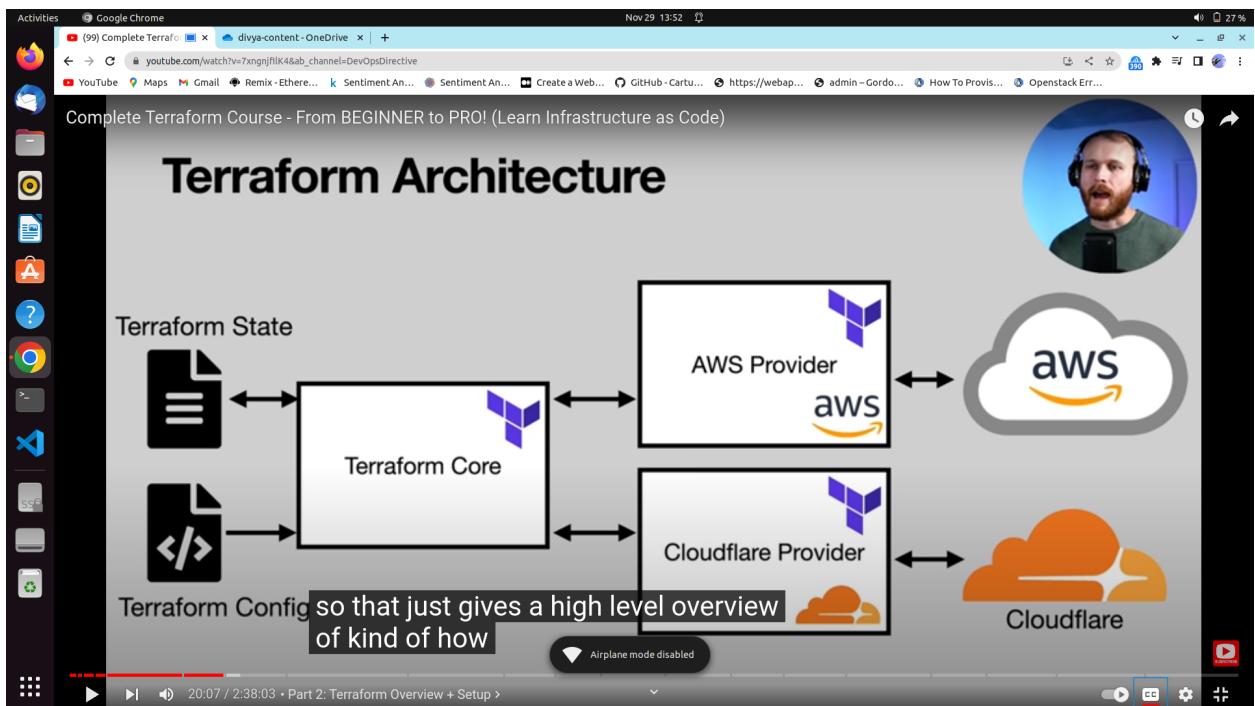
1 Jenkinsfile
2 pipeline {
3     agent any
4     environment {
5         NEW_VERSION = '1.3.0'
6         SERVER_CREDENTIALS = credentials('server-credentials')
7     }
8     stages {
9         stage("build") {
10            steps {
11                echo "building the application..."
12                echo "building version ${NEW_VERSION}"
13            }
14        }
15        stage("test") {
16            steps {
17                echo 'testing the application...'
18            }
19        }
20        stage("deploy") {
21            steps {
22                echo 'deploying the application...'
23                echo "deploying with ${SERVER_CREDENTIALS}"
24            }
25        }
26    }
27 }

```

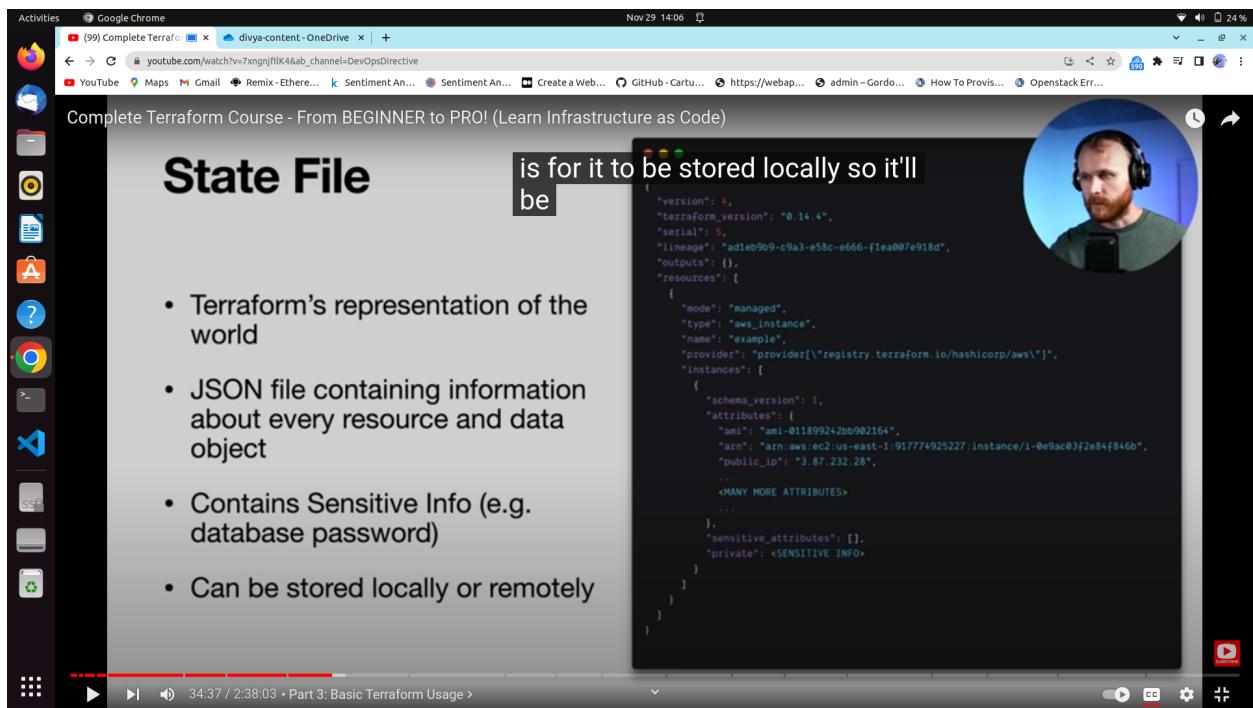
Ln 5, Col 27 (18 selected) Spaces: 4 UTF-8 LF Groovy

17:30 / 35:05 • Using Environmental Variables in Jenkinsfile >

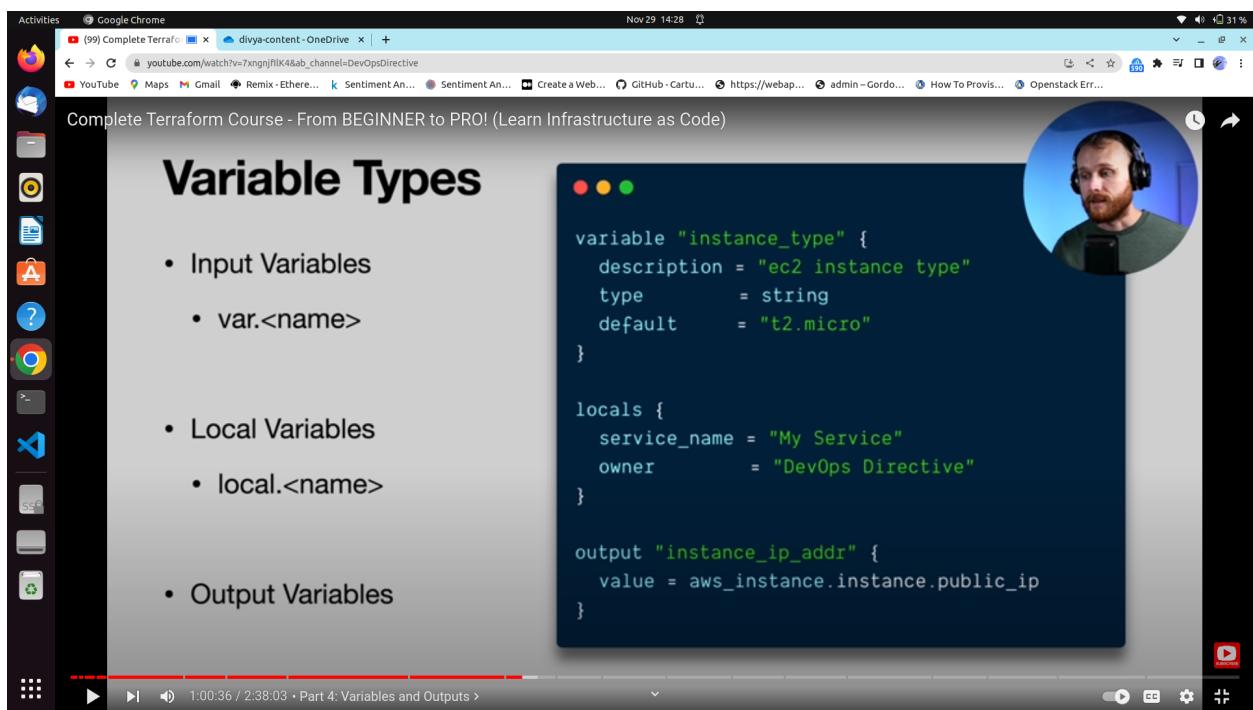
Terraform Architecture



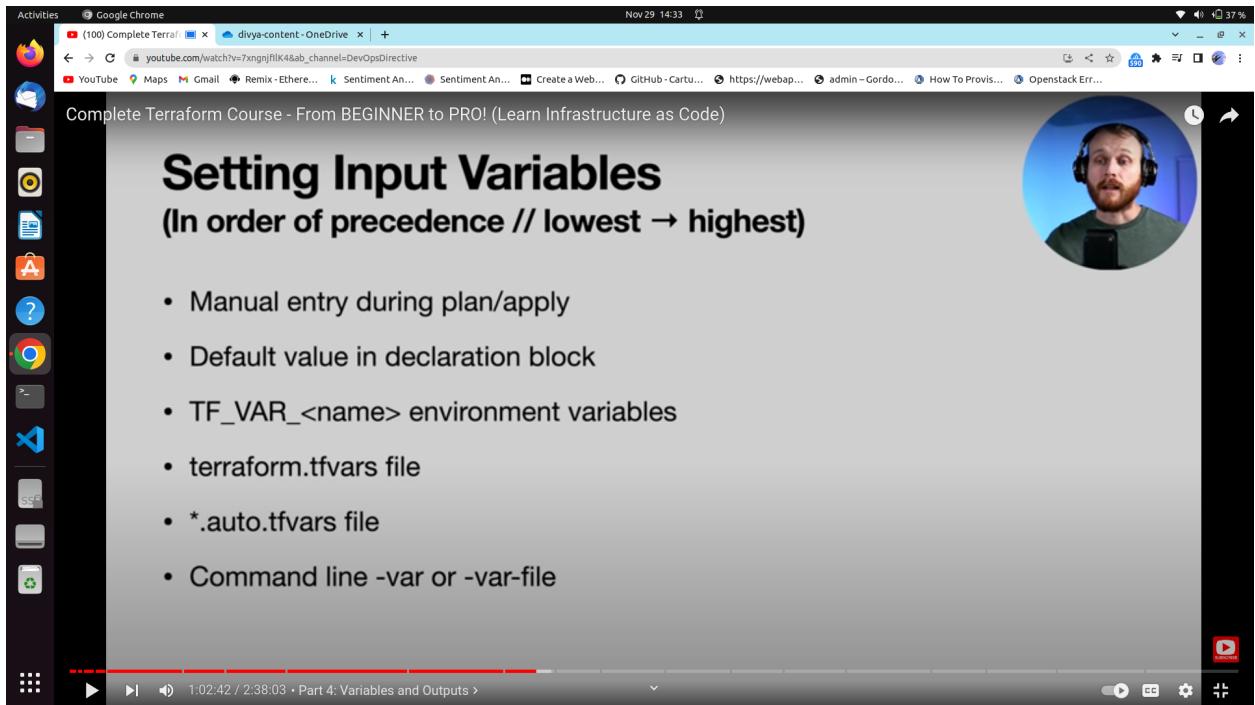
State file



variables



Input variables



A screenshot of a code editor (VS Code) showing a Terraform configuration file named "variables.tf". The code defines two variables:

```
variable "db_user" {
    description = "username for database"
    type        = string
    default     = "foo"
}

variable "db_pass" {
    description = "password for database"
    type        = string
    sensitive   = true
}
```

The code editor has an Explorer sidebar showing other files like ".terraform.lock.hcl", "architecture.png", "main.tf", "README.md", and "outputs.tf". A terminal tab at the bottom shows the command "terraform apply -var=db_user=myuser -var=db_pass=SOMETHINGSUPERSECURE".

Depends_on

Activities Google Chrome Nov 29 14:56 64%

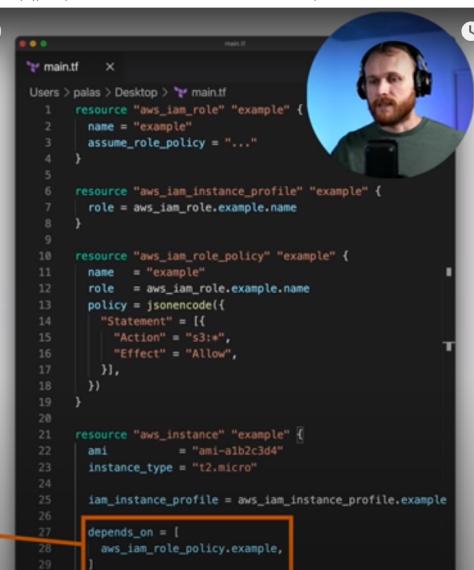
YouTube Maps Gmail Remix - Ethereum Sentiment Analysis Create a Web... GitHub - Cart... https://webap... admin - Gordo... How To Provis... Openstack Err...

Complete Terraform Course - From BEGINNER to PRO! (Learn Infrastructure as Code)

Meta-Arguments

depends_on

- Terraform automatically generates dependency graph based on references
- If two resources depend on each other (but not each others data), `depends_on` specifies that dependency to enforce ordering
- For example, if software on the instance needs access to S3, trying to create the `aws_instance` would fail if attempting to create it before the `aws_iam_role_policy`



1:14:29 / 2:38:03 • Part 5: Additional Language Features >

Count

Activities Google Chrome Nov 29 14:56 65%

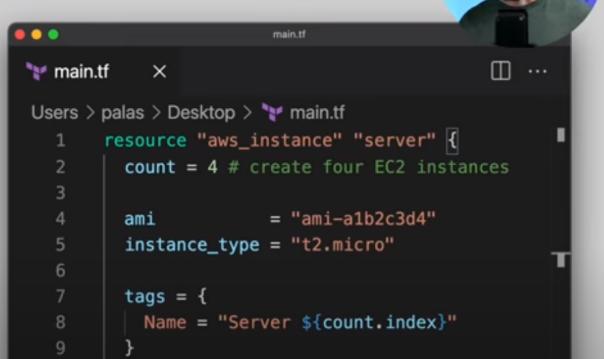
YouTube Maps Gmail Remix - Ethereum Sentiment Analysis Create a Web... GitHub - Cart... https://webap... admin - Gordo... How To Provis... Openstack Err...

Complete Terraform Course - From BEGINNER to PRO! (Learn Infrastructure as Code)

Meta-Arguments

Count

- Allows for creation of multiple resources/modules from a single block
- Useful when the multiple necessary resources are nearly identical



1:15:07 / 2:38:03 • Part 5: Additional Language Features >

for_each

Activities Google Chrome Nov 29 14:56 65%

YouTube Maps Gmail Remix - Ethereum Sentiment Analysis Create a Web... GitHub - Cart... https://webap... admin - Gordo... How To Provis... Openstack Err...

Complete Terraform Course - From BEGINNER to PRO! (Learn Infrastructure as Code)

Meta-Arguments

for_each

- Allows for creation of multiple resources/modules from a single block
- Allows more control to customize each resource than `count`

main.tf

```
locals {  
    subnet_ids = toset([  
        "subnet-abcdef",  
        "subnet-012345",  
    ])  
}  
  
resource "aws_instance" "server" {  
    for_each = local.subnet_ids  
  
    ami          = "ami-a1b2c3d4"  
    instance_type = "t2.micro"  
    subnet_id   = each.key  
  
    tags = {  
        Name = "Server ${each.key}"  
    }  
}
```



Lifecycle

Activities Google Chrome Nov 29 14:56 65%

YouTube Maps Gmail Remix - Ethereum Sentiment Analysis Create a Web... GitHub - Cart... https://webap... admin - Gordo... How To Provis... Openstack Err...

Complete Terraform Course - From BEGINNER to PRO! (Learn Infrastructure as Code)

Meta-Arguments

Lifecycle

- A set of meta arguments to control terraform behavior for specific resources
- `create_before_destroy` can help with zero downtime deployments
- `ignore_changes` prevents Terraform from trying to revert metadata being set elsewhere
- `prevent_destroy` causes Terraform to reject any plan which would destroy this resource

main.tf

```
resource "aws_instance" "server" {  
    ami          = "ami-a1b2c3d4"  
    instance_type = "t2.micro"  
  
    lifecycle {  
        create_before_destroy = true  
        ignore_changes = [  
            # Some resources have metadata  
            # modified automatically outside  
            # of Terraform  
            tags  
        ]  
    }  
}
```



Modules

Activities Google Chrome Nov 29 15:07 76%

YouTube Maps Gmail Remix - Ethereum Sentiment Analysis Create a Webpage GitHub - Cartographer https://webapp... admin - Gordon How To Provision VMs OpenStack Error

What is a Module?

Modules are containers for multiple resources that are used together. A module consists of a collection of .tf and/or .tf.json files kept together in a directory.

Modules are the main way to package and reuse resource configurations with Terraform.

think third parties might find them useful



Data block

Data Sources | Terraform Tutorial | #10

main.tf - course - Visual Studio Code

```
provider "aws" {
  version = "~> 2.65"
  region = "us-west-2"
}

data "aws_vpc" "tuts" {
  filter {
    name = "tag:Name"
    values = ["Tuts"]
  }
}

output "foo" {
  value = data.aws_vpc.tuts
}

resource "aws_subnet" "web" {
  vpc_id = data.aws_vpc.tuts.id
  cidr_block = "10.0.0.0/16"
}

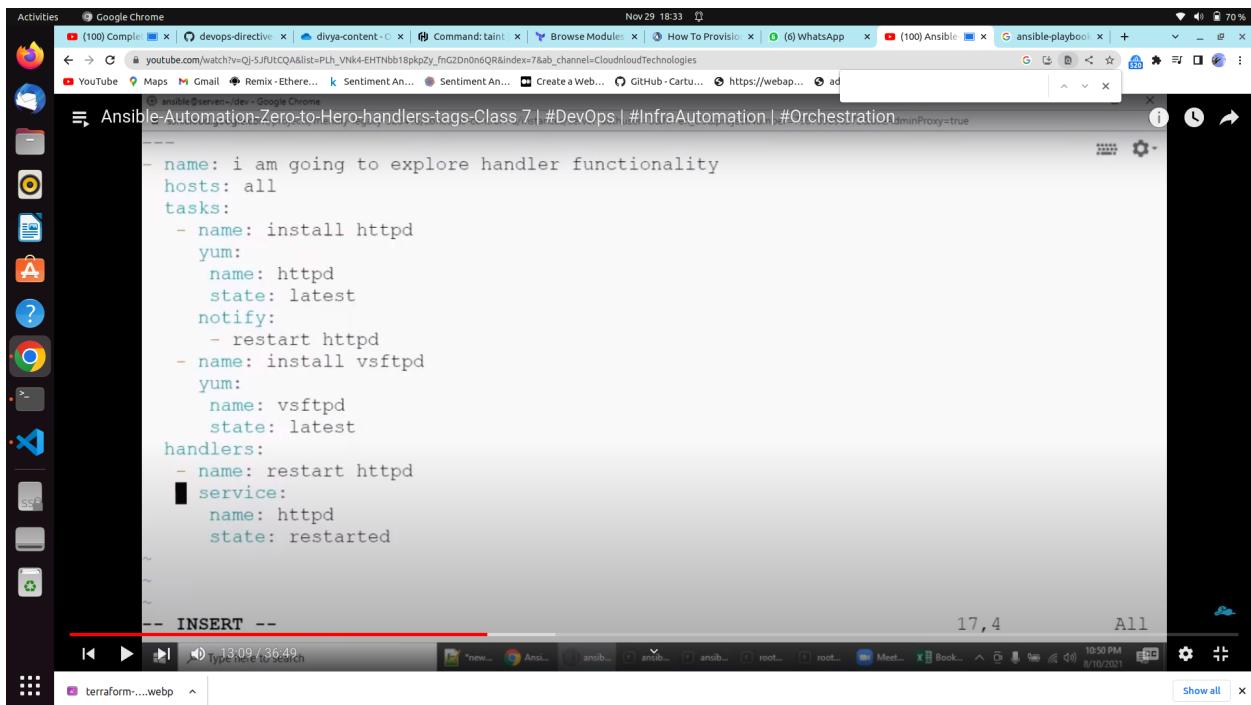
resource "aws_instance" "web" {
  ami           = "ami-0c1e33e1a2930c9d"
  instance_type = "t2.micro"
  subnet_id     = aws_subnet.web.id
  tags = [
    { "Name": "Tuts" }
  ]
}
```

main_route_table_id = "rtb-0cale33e1a2930c9d"
owner_id = "530219893428"
state = "available"
tags = {
 "Name" = "Tuts"
}

focus@will-UX330UAR:~/tuts/terraform/course\$

Ansible

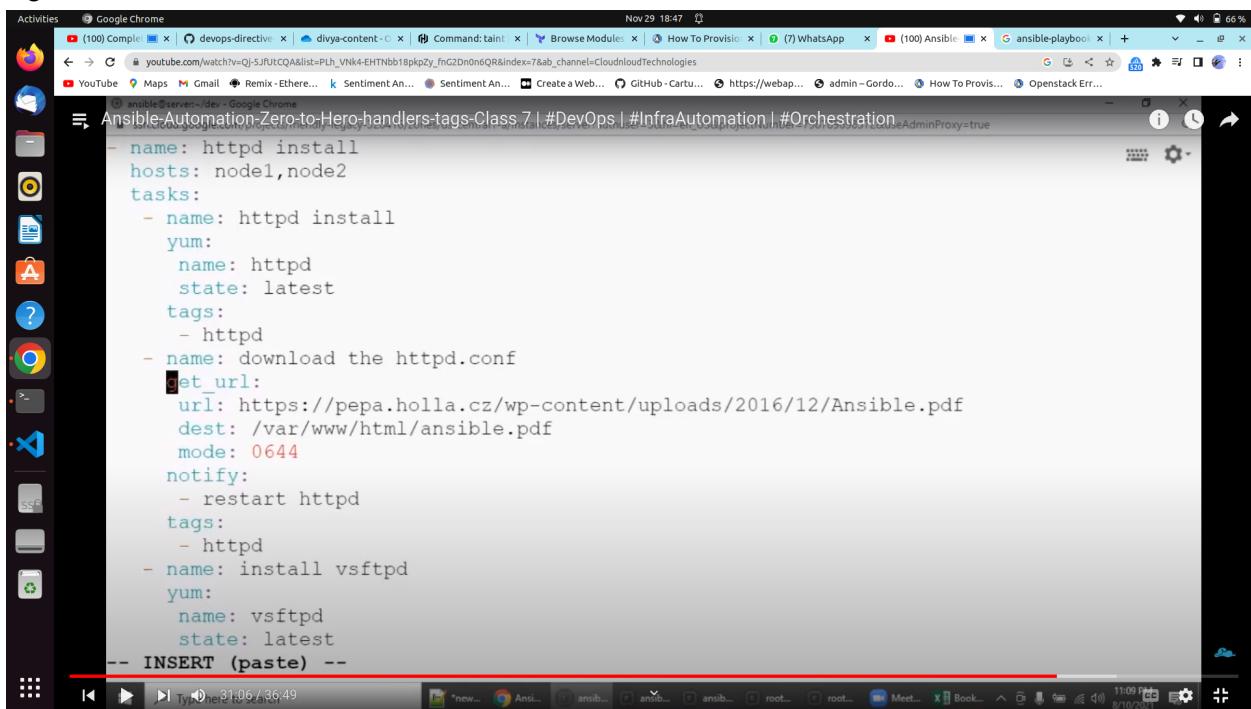
handler



```
Ansible-Automation-Zero-to-Hero-handlers-tags-Class 7 | #DevOps | #InfraAutomation | #Orchestration
-----
- name: i am going to explore handler functionality
  hosts: all
  tasks:
    - name: install httpd
      yum:
        name: httpd
        state: latest
      notify:
        - restart httpd
    - name: install vsftpd
      yum:
        name: vsftpd
        state: latest
    handlers:
      - name: restart httpd
        service:
          name: httpd
          state: restarted

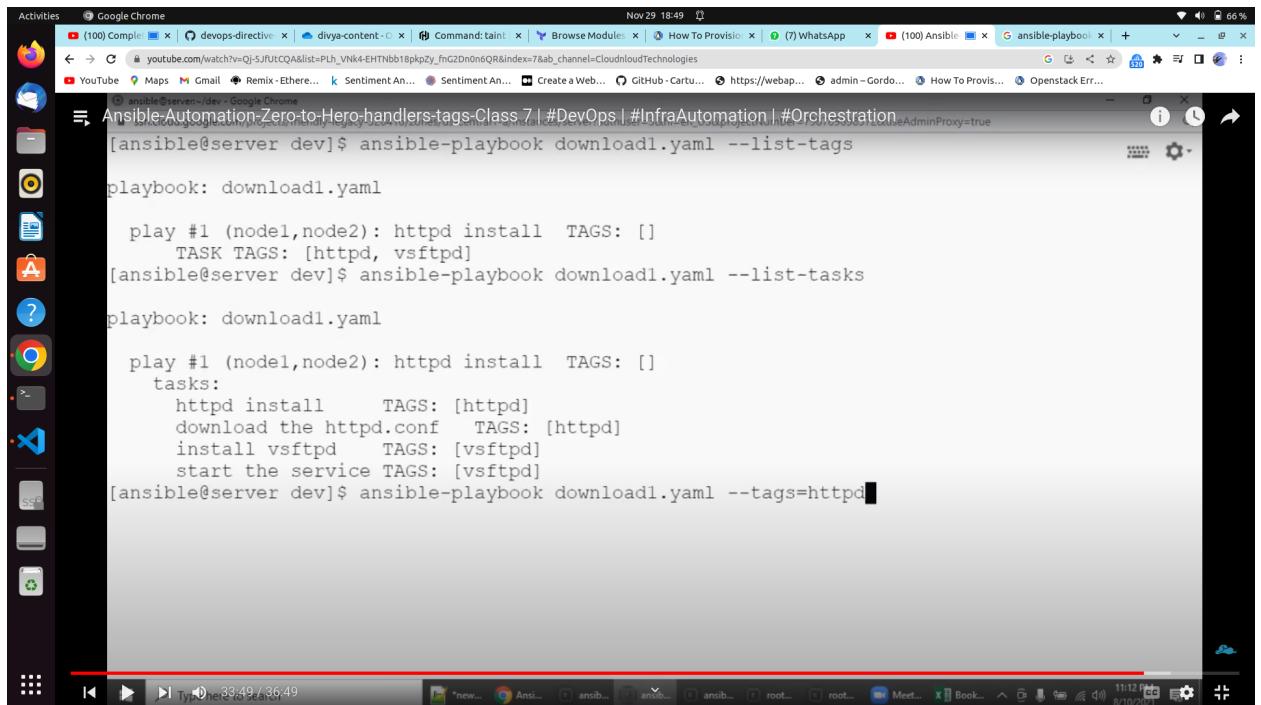
-- INSERT --
```

tags



```
Ansible-Automation-Zero-to-Hero-handlers-tags-Class 7 | #DevOps | #InfraAutomation | #Orchestration
-----
- name: httpd install
  hosts: node1,node2
  tasks:
    - name: httpd install
      yum:
        name: httpd
        state: latest
      tags:
        - httpd
    - name: download the httpd.conf
      get_url:
        url: https://pepa.holla.cz/wp-content/uploads/2016/12/Ansible.pdf
        dest: /var/www/html/ansible.pdf
        mode: 0644
      notify:
        - restart httpd
      tags:
        - httpd
    - name: install vsftpd
      yum:
        name: vsftpd
        state: latest
-- INSERT (paste) --
```

Tags command



The screenshot shows a terminal window titled "Ansible-Automation-Zero-to-Hero-handlers-tags-Class 7 | #DevOps | #InfraAutomation | #Orchestration". The terminal is running on a server named "ansible@server dev". The user has run several Ansible commands:

```
[ansible@server dev]$ ansible-playbook download1.yaml --list-tags
playbook: download1.yaml

play #1 (node1,node2): httpd install  TAGS: []
    TASK TAGS: [httpd, vsftpd]
[ansible@server dev]$ ansible-playbook download1.yaml --list-tasks
playbook: download1.yaml

play #1 (node1,node2): httpd install  TAGS: []
  tasks:
    httpd install  TAGS: [httpd]
    download the httpd.conf  TAGS: [httpd]
    install vsftpd  TAGS: [vsftpd]
    start the service TAGS: [vsftpd]
[ansible@server dev]$ ansible-playbook download1.yaml --tags=httpd
```

groups

A screenshot of a Linux desktop environment. On the left is a vertical dock with icons for various applications like a terminal, file manager, and system tools. In the center, a terminal window is open, displaying Ansible playbook code. The code defines two inventory groups: 'node1 --> Dev Group' and 'node2 --> Prod Group'. It then installs httpd on all hosts in the 'prod' group and restarts the service. A red line highlights the line '- name: restart web service'. At the bottom of the terminal, there's a status bar showing 'TyDher 5:46 / 18:38'. Above the terminal, the desktop environment shows several other browser tabs, including YouTube, Gmail, and GitHub, indicating a multi-tasking session.

```
12 ansible all -m ping
13
14 this above ping command should return with ping / pong green color.
15
16 ****
17 Inventory Grouping
18
19 node1 --> Dev Group
20
21 node2 --> Prod Group
22
23 Install httpd in production group of servers only and restart service also in production servers only
24
25 vim httpd.yaml
26
27 - name: install httpd
28   hosts: prod
29   tasks:
30     - name: install httpd
31       yum:
32         name: httpd
33         state: latest
34     - name: restart web service
35       service:
36         name: httpd
37         state: restarted
```

When inventory_hostname

A screenshot of a Linux desktop environment, similar to the one above. The terminal window shows the same Ansible playbook code, but with a significant modification. Line 132 now includes a condition 'when: inventory_hostname in groups ['prod']', which restricts the task to execute only on hosts in the 'prod' group. A red line highlights this conditional statement. The rest of the playbook remains the same, defining inventory groups and installing httpd. The desktop environment and browser tabs are identical to the first screenshot.

```
116
117 node1 --> Dev Group
118
119 node2 --> Prod Group
120
121 Install all the plays into all the servers which are listed out in the inventory file.But only one play (or) task must be executed in prod group
122
123 vim httpd.yaml
124
125 - name: install httpd
126   hosts: all
127   tasks:
128     - name: install httpd
129       yum:
130         name: httpd
131         state: latest
132       when: inventory_hostname in groups ['prod']
133     - name: restart web service
134       service:
135         name: httpd
136         state: restarted
137
138
139 ****
140
141 Inventory Grouping
```

Links

<https://github.com/ginigangadharan/30-Days-of-Ansible-Bootcamp>

https://github.com/cloudnloud/Ansible_Automation