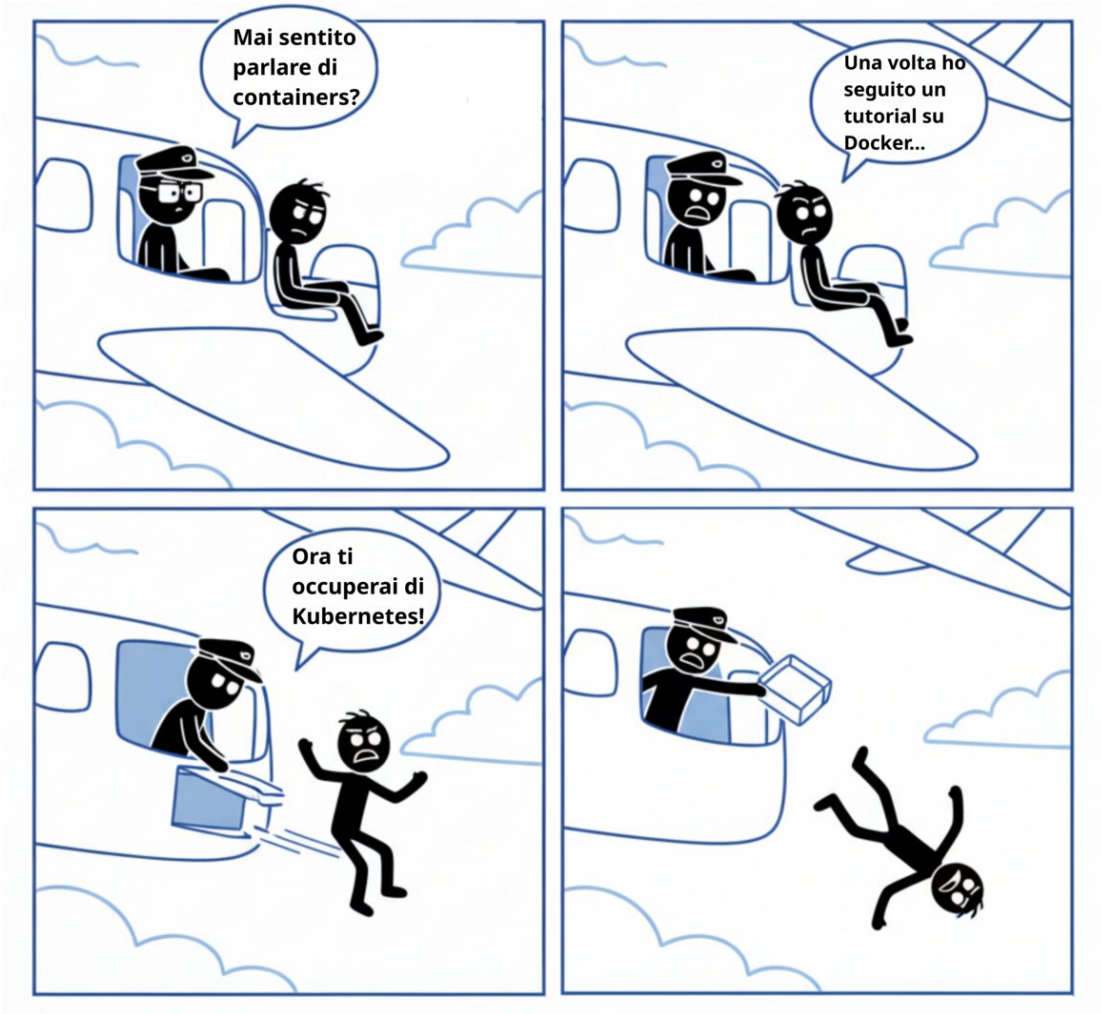
CaC: "*Configuration as Code*" o "*Cose a Caso*"?

Isac Pasianotto

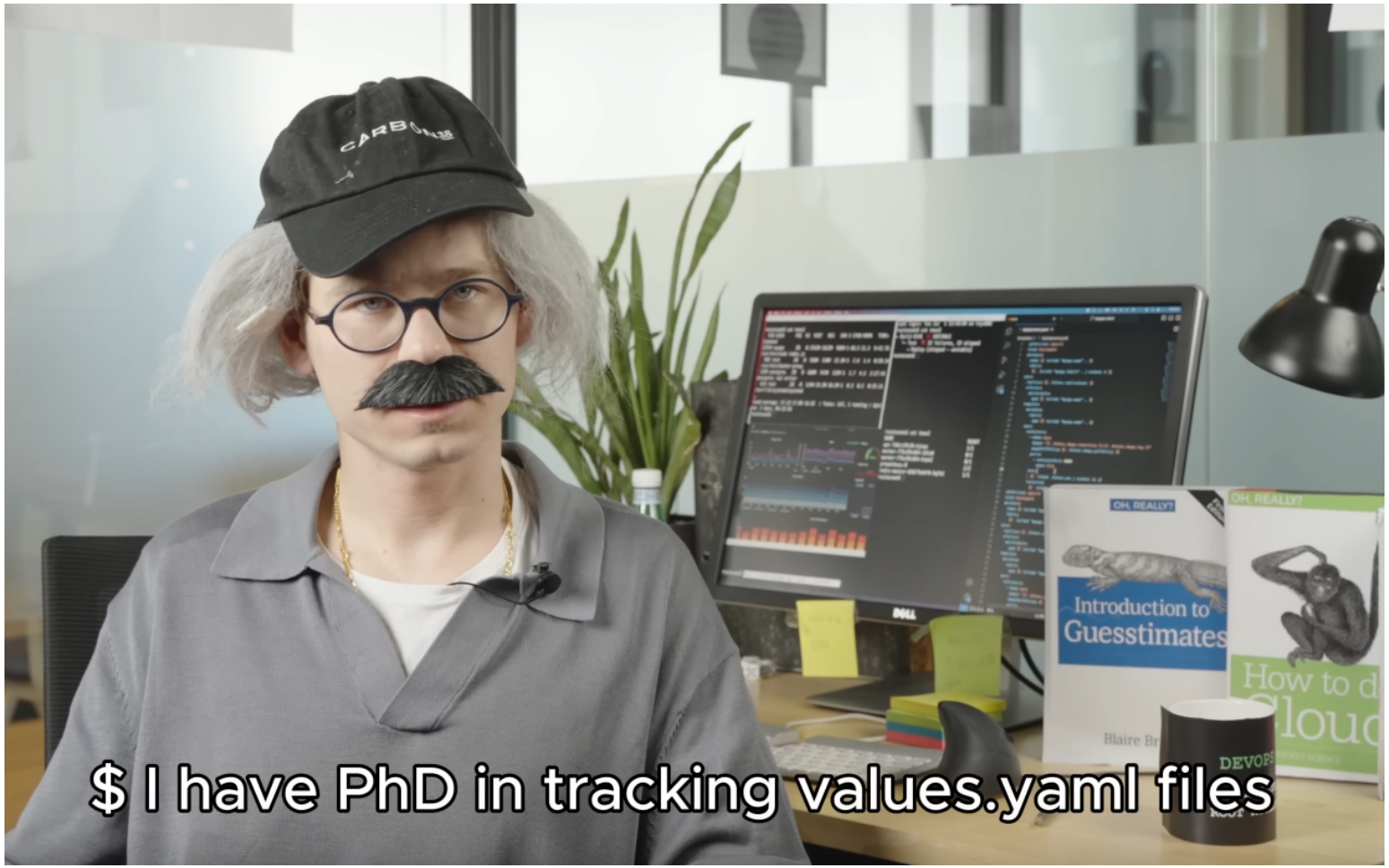
Linux Day 2025 – 25 Ottobre 2025

$ whoami



Il tirocinio... dove tutto è iniziato!

$ whoami, pt. 2



Reference: Kai Lentit - Interview with Cloud Architect in 2025

Agenda

1. About me
2. Il viaggio che ci ha portati fin qui
3. CaC: Configuration as Code
4. *STENCIL*: il mio parco giochi personale... tutto in *CaC*
5. Infrastruttura as Code con OpenTofu
6. Provisioning con Ansible
7. Orchestrazione di servizi con Kubernetes
8. Conclusioni

"Il progresso per amore del progresso" – D. Umbridge

Un tempo, per configurare un server bastava SSH, vim, e un po' di fiducia.
Poi è arrivato il cloud, e abbiamo deciso che scrivere file YAML fosse meglio.
Ora abbiamo `main.tf`, `kickstart.ks`, `cloud-init.yaml`, `inventory.yml`, `ansible.cfg`, `Chart.yaml`, e tanti, tanti, ma tanti `values.yaml`

L'era pre-cloud: i PET server

Un *PET* server è una macchina a cui ti affezioni: la curi, la nomini, la configuri a mano e la tieni in vita con amore e SSH.

- Configurazioni manuali, spesso via SSH.
- Documentazione? Prega che chi l'ha fatto prima di te l'abbia fatta bene (e che non sia andato in pensione).
- Essendo implementato a mano pezzo per pezzo, sai esattamente cosa c'è dentro e perché qualcosa funziona/non funziona.

ssh $\rightarrow$ `vim /etc/httpd.conf` $\rightarrow$ reboot $\rightarrow$ prega.

PET server: le criticità

- Ogni server è unico: ~~difficile~~ impossibile replicare l'ambiente.
- La scalabilità è un miraggio: ogni nuova macchina richiede tempo e attenzione.
- La manutenzione è onerosa: aggiornamenti e patching manuali.
- Se si rompe qualcosa è un disastro... Si ripara!

Il triste approccio moderno: cattle servers

I cattle (lett. “bestiame”) servers sono server generici, intercambiabili e replicabili.

- Non funziona? Elimina e ricrea da zero.
- Server identici, uno clone dell’altro.
- Creati e distrutti automaticamente secondo necessità, con tools di automazione.

La parola chiave è

Il punto di non ritorno

Tra il 2006 e il 2010 appaiono i primi strumenti che incarnano l'idea:

- *Puppet* (2005) e *Chef* (2009): Linguaggi dichiarativi per descrivere lo stato del server
- *Ansible* (2012) Soluzione che utilizza YAML come Domani Specific Language (DSL).
- *Terraform* (2014) Estende il concetto di configurazione riproducibile anche all'infrastruttura.

Come garantire la riproducibilità?

Configurazione dichiarativa!

Agenda

1. About me
2. Il viaggio che ci ha portati fin qui
3. CaC: Configuration as Code
4. *STENCIL*: il mio parco giochi personale... tutto in *CaC*
5. Infrastruttura as Code con OpenTofu
6. Provisioning con Ansible
7. Orchestrazione di servizi con Kubernetes
8. Conclusioni

Cos'è *Configuration as Code* (CaC)?

Definizione

Configuration as Code è un approccio alla gestione e alla configurazione dei sistemi informatici in cui le configurazioni sono definite e gestite tramite file di testo scritti in linguaggi di markup (o di programmazione).

Vantaggi

- **Approccio dichiarativo**: Indichi lo stato desiderato, non le istruzioni per ottenerlo.
- **Versioning**: Essendo una collezione di file, è facile integrarli con git o altri version source control
- **Documentazione**: Il codice è la documentazione! L'approccio *CaC* obbliga a lasciare una traccia scritta di cosa è stato fatto.

L'altra faccia della medaglia...

- **Approccio dichiarativo**: Indica quello che vuoi e qualcuno (forse) lo farà accadere:
  - Devi sapere cosa vuoi davvero.
  - Difficile capire *cosa sta accadendo sotto la scocca*.
- **Versioning**: Molti branch, molte configurazioni diverse e incompatibili... Ma quale sta girando adesso?
- **Documentazione**. Molto spesso il codice è l'unica documentazione.

La teoria è bellissima...

- Tutto è dichiarativo.
- Tutto è riproducibile.
- Tutto è versionato.
- Tutto è automatizzato.

La teoria è bellissima...

- Tutto è dichiarativo.
- Tutto è riproducibile.
- Tutto è versionato.
- Tutto è automatizzato.

... finché non arriva il primo bug in produzione.

Lezioni che ho imparato alla scuola delle legnate sui denti

1. Automatizzare non significa semplificare.
2. La complessità non scompare, si sposta: La configurazione come codice è potente, ma va capita.
3. Non c'è limite al livello di astrazione che puoi raggiungere...
4. a volte quei limiti dovrebbero esserci.

Ma soprattutto...

5. È più facile comprendere una regexp in klingon che un messaggio d'errore, ammesso che esista.


L'intera
infrastruttura
Una modifica insignificante
imgflip.com

Agenda

1. About me
2. Il viaggio che ci ha portati fin qui
3. CaC: Configuration as Code
4. *STENCIL*: il mio parco giochi personale... tutto in *CaC*
5. Infrastruttura as Code con OpenTofu
6. Provisioning con Ansible
7. Orchestrazione di servizi con Kubernetes
8. Conclusioni

STENCIL: il mio parco giochi personale... tutto in *CaC*

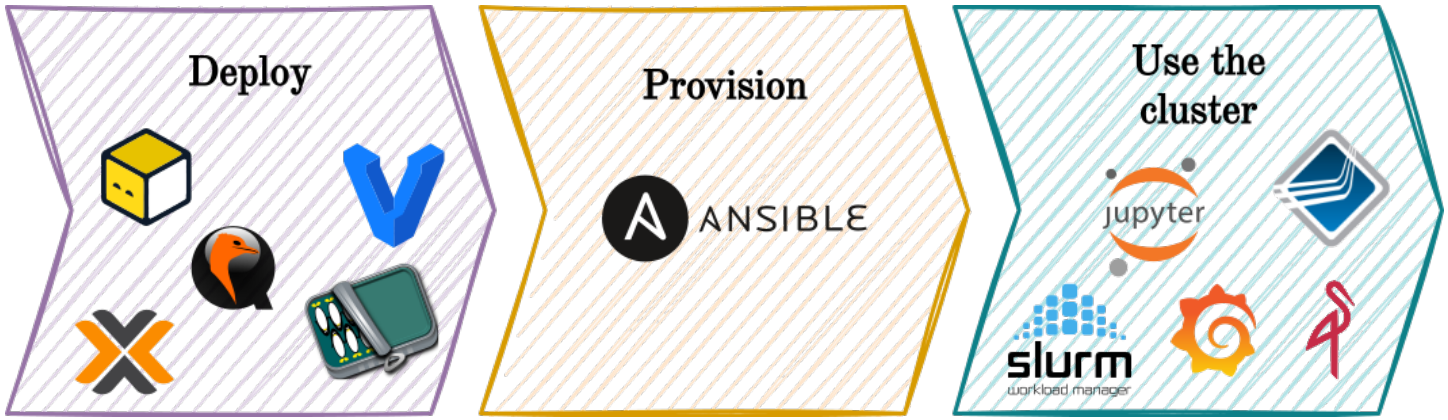
gitlab.com/projectstencil

- Progetto che sto portando avanti nel corso del mio PhD.
- Permette di creare interi cluster virtualizzati in modo completamente automatizzato e riproducibile.
- Usato come infrastruttura di testing nel laboratorio di ricerca dove lavoro.
- **Tutto** è gestito in pieno spirito *CaC*.

STENCIL: perchè ve ne parlo?

Questo esempio può aiutare a capire come l'adesione al paradigma *"as Code"* possa essere applicata a vari livelli:

- **Infrastruttura**: (*IaC*: Infrastructure as Code): OpenTofu.
- **Provisioning**: Configurazione dei server completamente dichiarativa e grazie ad Ansible.
- **Servizi**: Orchestrazione dei container per i servizi con Kubernetes.



Agenda

1. About me
2. Il viaggio che ci ha portati fin qui
3. CaC: Configuration as Code
4. *STENCIL*: il mio parco giochi personale... tutto in *CaC*
5. Infrastruttura as Code con OpenTofu
6. Provisioning con Ansible
7. Orchestrazione di servizi con Kubernetes
8. Conclusioni

OpenTofu

OpenTofu: che cos'è?

OpenTofu è uno strumento open-source per la gestione dell'infrastruttura come codice (*IaC*). Permette di definire, fornire e gestire l'infrastruttura IT attraverso file di configurazione dichiarativi.

- Rilasciato ad Agosto 2023 come fork di Terraform dopo che HashiCorp ne ha modificato la licenza da Mozzilla Public License (MPL) a Business Source License (BUSL).
- Compatibile con praticamente tutti i provider di Terraform.
- Comunità attiva e in crescita, con contributi regolari e supporto per nuovi provider.
- Basato sulla `HCL`: HashiCorp Configuration Language.

Concetti di base

- **Provider**: Plugin che permettono a OpenTofu di interagire con vari servizi (non solo) cloud e piattaforme, come per es: AWS, Azure, Libvirt, Proxmox, ...
- **Resources**: Componenti dell'infrastruttura che vogliamo creare e gestire, come VM, reti, dischi, storage ...
- **State**: File che tiene traccia dello stato attuale dell'infrastruttura gestita da OpenTofu.
- **Variables**: Permettono di rendere i file di configurazione più flessibili e riutilizzabili, oltre che adattabili a diversi ambienti.

OpenTofu: un esempio pratico

```
# Configurazione del provider
provider "libvirt" {
  uri = "qemu:///system"
}
```

```
# Definizione della network
resource "libvirt_network" "virtnet" {
  name = "virt-net"
  mode = "nat"
  addresses = ["192.168.132.0/24"]
  dhcp {
    enabled = false
  }
}
```

```
# Disco per lo storage
resource "libvirt_volume" "vm_disk" {
  count = var.vm_count
  name = "vm_disk_${count.index}.qcow2"
  size = var.settings.disk_size
  # ...
}
```

```
# Disco di boot
resource "libvirt_cloudinit_disk" "cidsk"
  count = var.vm_count
  name = "ci_disk_${count.index}.iso"
  user_data = templatefile(
    "clouinit_template",
    {
      hostname = "vm-${count.index}"
      ssh_authorized_keys = var.ssh_key
      # ...
    }
  )
}
```

OpenTofuL un esempio pratico – pt.2

Ora abbiamo tutti gli ingredienti necessari per spawnare `var.vm_count` VMs:

```
resource "libvirt_domain" "vm" {
  count = var.vm_count
  name = "vm-${count.index}"
  memory = var.settings.memory
  vcpu = var.settings.vcpus
  arch = "x86_64"

  disk {
    volume_id = libvirt_volume.vm_disk[count.index].id
  }

  cloudinit = libvirt_cloudinit_disk.cidsk[count.index].id
  network_interface {
    network_name = "virt-net"
    hostname = "vm-${count.index}"
    address = "192.168.132.${100 + count.index}"
  }
}
```

Agenda

1. About me
2. Il viaggio che ci ha portati fin qui
3. CaC: Configuration as Code
4. *STENCIL*: il mio parco giochi personale... tutto in *CaC*
5. Infrastruttura as Code con OpenTofu
6. Provisioning con Ansible
7. Orchestrazione di servizi con Kubernetes
8. Conclusioni

Ansible: che cos'è?

Ansible è uno strumento open-source per l'automazione IT che consente di gestire la configurazione, il provisioning e l'orchestrazione dei sistemi in modo semplice ed efficiente.

- Sviluppato da Michael DeHaan e rilasciato nel 2012, acquisito da Red Hat nel 2015.
- Comunità molto attiva: ampia disponibilità di ruoli e moduli su Ansible Galaxy, forum e contributi open-source.

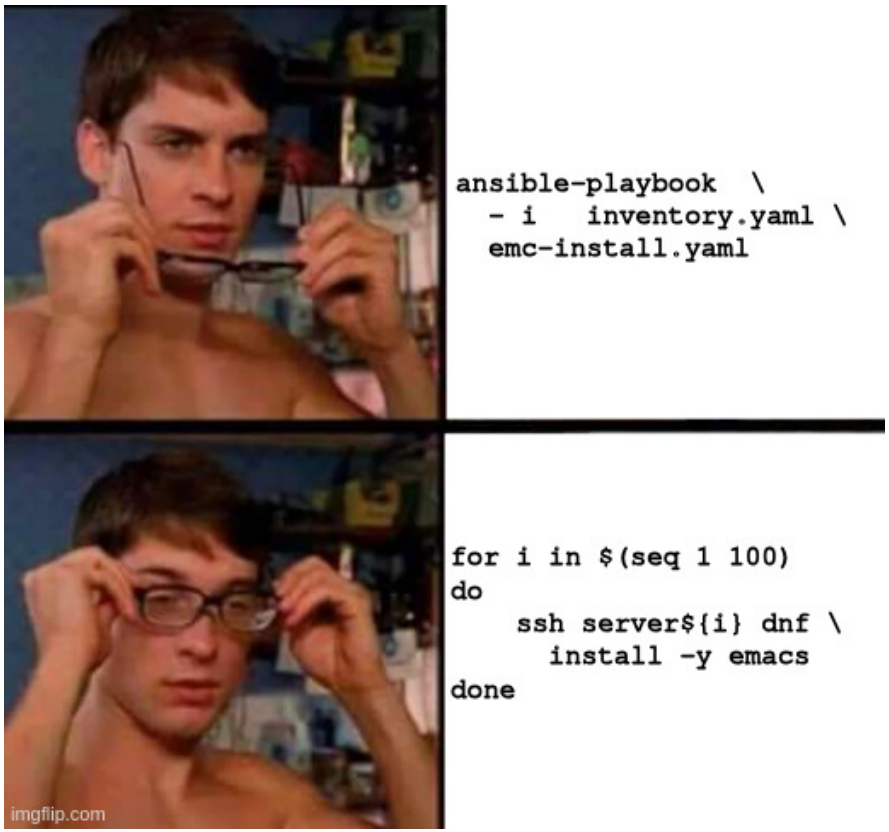
Ansible: concetti di base

- Non richiede agenti sui nodi gestiti, utilizza `ssh` per comunicare con i server da remoto.
- **Playbook**: File YAML che definiscono le attività da eseguire sui nodi gestiti.
- **Inventory**: Elenco dei nodi (host) che Ansible gestisce, organizzati in gruppi.
- **Roles**: Strutture organizzative per raggruppare playbook, task, variabili e file correlati.
- **Tasks**: Singole azioni definite all'interno di un playbook.

Vantaggi di Ansible?

Il mio commento la prima volta che l'ho usato

Detta così sembra un "ssh con degli step extra"



Vantaggi di Ansible:

In realtà usare dei ruoli e playbook trovati su Ansible galaxy può avere certi vantaggi:

- (Probabilmente) sono stati scritti da qualcuno che sa cosa sta facendo, più di te.
- **Idempotenza**: Eseguire lo stesso playbook più volte non cambia il risultato dopo la prima esecuzione.
- ***CaC***: Si dichiara lo stato desiderato, non serve sapere (anche se sarebbe utile) come fare a raggiungerlo.
- **Integrazione con altri strumenti DevOps**: Ansible si integra bene con CI/CD, Kubernetes, Ceph e altri strumenti.

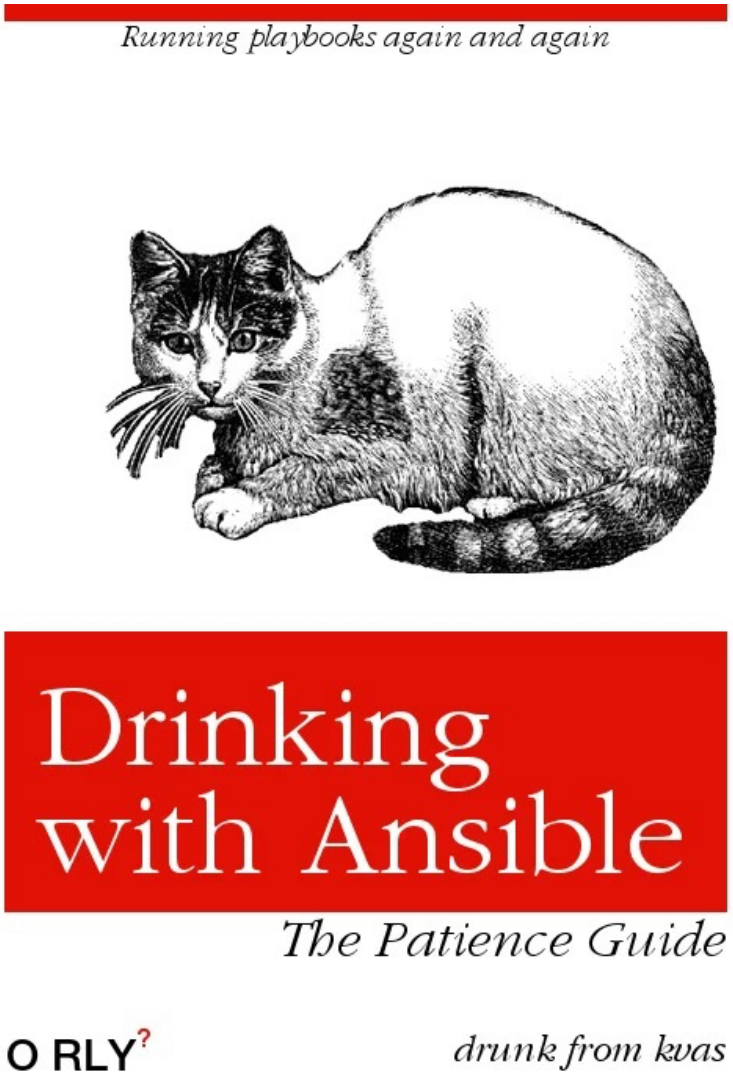
Ansible: nella realtà

```yaml
hosts: my_server_group
become: yes
tasks:
  - name: Installazione di alcuni pacchetti essenziali
    ansible.builtin.shell:
      cmd: dnf install -y emacs cowsay sl
    ignore_errors: yes
```

Per faovre, non fate mai una cosa del genere.

Disclaimer!

Addentrarsi nella tana del Bianconiglio di Ansible può avere effetti collaterali.



Ansible: un esempio pratico

```yaml
hosts: all
become: true
tasks:
  - name: Installa nginx
    ansible.builtin.yum:
      name: nginx
      state: present

  - name: Copia la config di nginx
    ansible.posix.synchronize:
      src: files/nginx-config/
      dest: /etc/nginx/
      delete: false
  - name: Fix permessi
    ansible.builtin.file:
      path: /etc/nginx/
      owner: root
      group: root
      mode: '0644'
      recurse: true
```

```yaml
  - name: Abilita e avvia nginx
    ansible.builtin.service:
      name: nginx
      state: started
      enabled: true

  - name: Verifica che sia tutto ok
    ansible.builtin.uri:
      url: http://miowebserver.it
      status_code: 200
      timeout: 5
```

Agenda

1. About me
2. Il viaggio che ci ha portati fin qui
3. CaC: Configuration as Code
4. *STENCIL*: il mio parco giochi personale... tutto in *CaC*
5. Infrastruttura as Code con OpenTofu
6. Provisioning con Ansible
7. Orchestrazione di servizi con Kubernetes
8. Conclusioni

kubernetes

Kubernetes: che cos'è?

Kubernetes (K8s) è un software open-source per l'automazione del deployment, scalabilità, e gestione di applicativi in **containers**.

- Rilasciato da Google nel 2014, ora gestito dalla Cloud Native Computing Foundation (CNCF).
- Espone delle API che sono diventate lo standard de facto per l'orchestrazione dei container anche in ambienti on-premise da parte di tutti i principali vendor (AWS, Azure, Google Cloud, IBM . . .).
- Basato su un'**architettura a microservizi**, con componenti modulari che possono essere scalati o sostituiti indipendentemente.

Kubernetes: concetti di base

- **Pod**: Unità base di esecuzione in Kubernetes, che può contenere uno o più container. È la più piccola entità che può essere distribuita e gestita.
- **Deployment**, **StatefulSet**, **DaemonSet**: Risorse che gestiscono il ciclo di vita dei pod, garantendo che il numero desiderato di repliche sia sempre in esecuzione.
- **kubectl**: Strumento da linea di comando per interagire con il cluster Kubernetes, spesso utilizzato passandogli in input file YAML che descrivono lo stato desiderato del cluster.
- **Manifests**: File YAML che descrivono lo stato desiderato delle risorse nel cluster Kubernetes.

Ancora una volta, tutto è *CaC*! Descrivi lo stato desiderato, e qualcuno[1] (auspicabilmente) lo farà accadere.

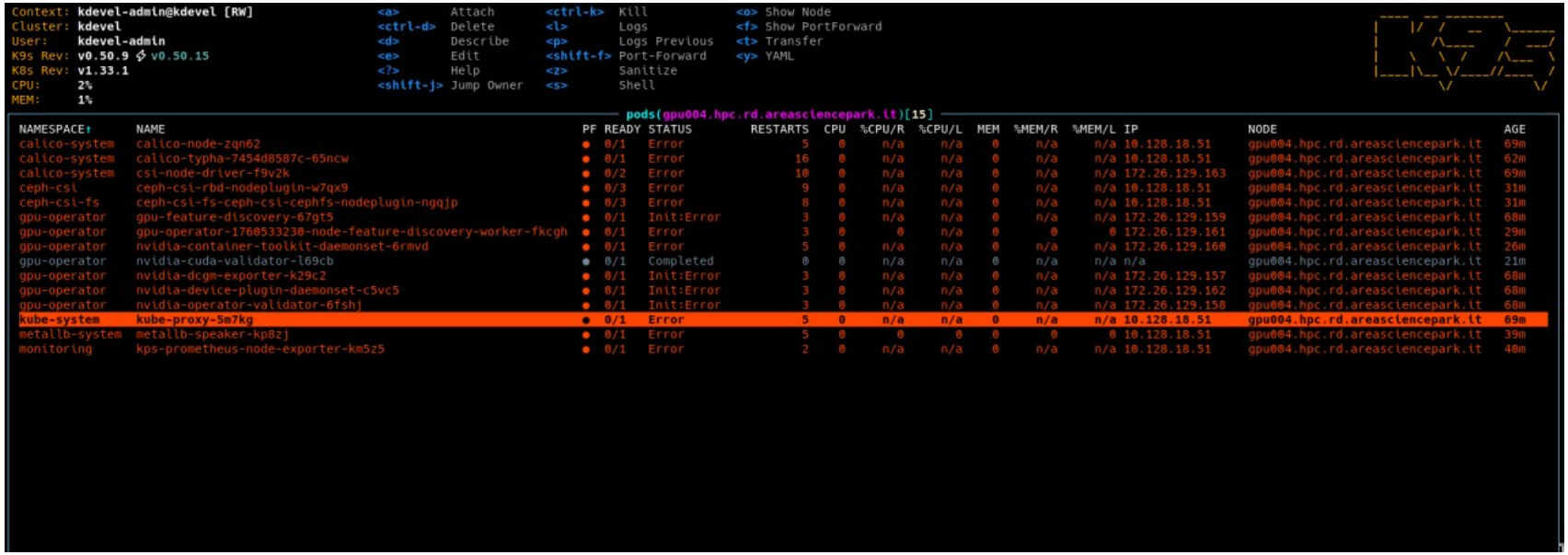
[1] Kubernetes è così modulare che certi microservices possono essere gestiti da `operators` esterni.

Kubernetes: le principali fonti di mal di testa

- **CNI-plugins**: *(Container Network Interface)* attaccano ad ogni pod un'interfaccia di rete virtuale e poi *"Modificano la configurazione di rete del nodo"* per far si che *"in qualche modo"* pod tra nodi diversi possano comunicare tra di loro.
- **CoreDNS**: "It's not the DNS, There's no way it's the DNS ... it was the DNS!".
- **CSI-plugins**: *(Container Storage Interface)* permettono ai pod di montare volumi di storage esterni. Spesso richiedono configurazioni specifiche e possono essere fonte di problemi di performance o compatibilità. Alcuni richiedono invocazioni a Cthulhu per funzionare correttamente.

Kubernetes: le principali fonti di mal di testa, pt. 2

- **Operators**: micro-servizi che estendono le funzionalità di k8s. Nascono per semplificare la gestione di applicazioni complesse. Spesso utilizzarli fa si che si impari ad usare l'operator stesso piuttosto che padroneggiare il servizio che si vuole gestire.
- **Helm Charts**: Dovrebbe essere un "package manager" per k8s. Se utilizzi i valori di default funziona, ma appena provi ad adeguarlo alle tue esigenze ti ritrovi a dover leggere ∼8000 righe di YAML per (non) capire cosa stai facendo.
- **Debugging**: Tanti microservizi = tanti container. Tanti container = tanto tempo perso solo per capire *dove* cercare i log (ammesso che non ci sia un default `debug=false` da qualche parte).
- **GPU Nvidia**: . . . non ne parliamo nemmeno . . .



Il server in produzione la scorsa settimana dopo aver aggiornato una delle 573 variabili in `values.yaml`

Era moralmente necessario inserire questa slide in questa presentazione.

Kubernetes: esempio pratico

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.14.2
        ports:
        - containerPort: 80
```

```
$ kubectl apply -f file.yaml
```

```yaml
apiVersion: v1
kind: Service
metadata:
  name: nginx-service
spec:
  selector:
    app: nginx
  ports:
  - protocol: TCP
    port: 80
    targetPort: 80
  type: ClusterIP
```

```yaml
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: nginx-ingress
spec:
  rules:
  - host: mynginx.example.com
    http:
      paths:
      - path: /
        pathType: Prefix
        backend:
          service:
            name: nginx-service
            port: 80
```

Se vi sentite così . . . è perfettamente normale!

Dopotutto, Kubernetes è stato progettato da Google per gestire i propri problemi. Voi non siete Google!

Però Kubernetes a volte regala gioie . . .

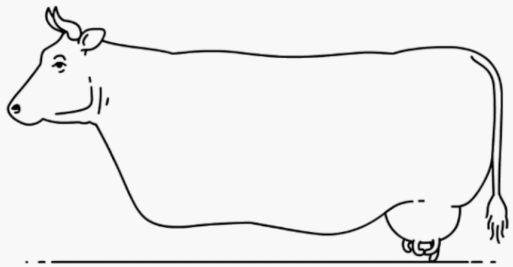
```
$ kubectl get ingress -n nextclouds

NAME                      CLASS          HOSTS         ADDRESS   PORTS      AGE
xxx-nextcloud             nginx-public   xxx.xxx.xx              80, 443    6d11h
cm-acme-http-solver-ksd   nginx-public   xxx.xxx.xx              80         6d11h
```

Non esiste nessun indirizzo IP associato all'Ingress, ma (non sappiamo come) tutto funziona lo stesso!

```
$ curl -i https://xxx.xxx.xx/login

HTTP/2 200
content-type: text/html; charset=UTF-8
[ ... ]
<!DOCTYPE html>
[ ... ]
```



Prima regola: Se funziona, *Non Toccare!*

Agenda

1. About me
2. Il viaggio che ci ha portati fin qui
3. CaC: Configuration as Code
4. *STENCIL*: il mio parco giochi personale... tutto in *CaC*
5. Infrastruttura as Code con OpenTofu
6. Provisioning con Ansible
7. Orchestrazione di servizi con Kubernetes
8. Conclusioni

Conclusioni

Cosa abbiamo imparato

- *Configuration as Code* è un approccio potente per gestire infrastrutture e servizi in modo riproducibile e scalabile.
- Strumenti come OpenTofu, Ansible e Kubernetes facilitano l'implementazione di questo paradigma, ma richiedono una buona comprensione per evitare problemi complessi.
- La chiave del successo risiede nell'imparare cosa questi strumenti fanno sotto la scocca, e non solo come usarli.
- Adottare tecnologie moderne solo perché sono di moda ti garantisce un debito tecnico enorme.

Il *CaC* è ovunque

Abbiamo visto come il paradigma *"as Code"* possa estendersi a quasi ogni ambito dell'informatica: dall'infrastruttura al deployment, fino alla configurazione dei sistemi.
Tuttavia, capire *dove fermarsi* è una questione di equilibrio e di vostro buon senso ...

Il *CaC* è ovunque

. . . perché, a quanto pare, non c'è mai fine alla malvagità umana.

Grazie per l'attenzione

Addio, e grazie per tutto il pesce!

Domande?

Licenza d'uso di questo documento