FR

Guide IaC

Terraform, Ansible, GitOps pour PME

Fini le clic-ops — pipeline IaC complet, cas concret BOTUM

Terraform · **Ansible** · **GitOps**

Mars 2026

BOTUM INC. www.botum.ca contact@botum.ca | blog.botum.ca

Série Cloud Journey — Billet B14

Votre infra cloud a ete construite a la main. Un clic ici, un portail Azure la, une VM creee en console AWS un vendredi soir sous pression. Six mois plus tard, personne ne sait vraiment ce qui tourne, pourquoi, ni comment le reproduire en cas de sinistre. C'est le "clic-ops" — et c'est la premiere source de dette technique dans les organisations IT que nous accompagnons chez BOTUM.

L'Infrastructure as Code (IaC) est la reponse structurelle a ce probleme. Pas une tendance, pas un luxe reserve aux grandes entreprises — un prerequis pour toute PME qui veut maitriser son infra cloud en 2025.

L'IaC en vrai : declarer l'etat desire

La distinction fondamentale de l'IaC est philosophique avant d'etre technique : vous declarez ce que vous voulez, pas comment y arriver. L'approche declarative est idempotente : appliquer deux fois le meme code donne le meme resultat, sans effets de bord.

Votre infra devient du code versionne dans Git : historique complet, revues de code, rollback instantane, audit trail. Tout ce que vous faites pour votre code applicatif, vous le faites maintenant pour votre infrastructure.

Terraform : provisionnement cloud

Terraform (HashiCorp, open-source) est le standard de facto pour provisionner des ressources cloud : VMs, VNets, bases de donnees, load balancers, DNS, IAM. Il parle a l'API de 3000+ providers (Azure, AWS, GCP, Cloudflare, GitHub...).

```
# main.tf
terraform {
  required_providers {
    azurerm = {
      source  = "hashicorp/azurerm"
      version = "~> 3.90"
    }
  }
  backend "azurerm" {
    resource_group_name  = "rg-tfstate"
    storage_account_name = "botumtfstate"
    container_name       = "tfstate"
    key                  = "prod.terraform.tfstate"
  }
}

resource "azurerm_resource_group" "main" {
  name     = "rg-botum-prod"
  location = "canadacentral"
}

resource "azurerm_linux_virtual_machine" "app" {
  name                = "vm-app-prod-01"
  resource_group_name = azurerm_resource_group.main.name
  size                = "Standard_B2s"
  admin_username      = "azureuser"
  os_disk {
    storage_account_type = "Premium_LRS"
    disk_size_gb         = 128
  }
}
```

Points cles :

- State file : doit etre stocke en backend remote (Azure Blob, S3) — jamais en local, jamais dans Git. C'est la principale source de catastrophes chez les debutants.
- Modules : regroupez les ressources en modules reutilisables (VM, reseau, DB) entre projets.
- Plan avant Apply : terraform plan montre exactement ce qui va changer — soumis a approbation humaine avant apply.

Ansible : configuration et deploiement

Terraform cree l'infra. Ansible la configure. Une fois votre VM Azure creee, Ansible installe Nginx, copie les configs, cree les comptes utilisateurs — de facon idempotente.

```yaml
# playbooks/web-server.yml
- name: Configure web server
  hosts: web_servers
  become: yes
  tasks:
    - name: Install nginx
      ansible.builtin.apt:
        name: nginx
        state: present
    - name: Deploy nginx config
      ansible.builtin.template:
        src: templates/nginx.conf.j2
        dest: /etc/nginx/sites-available/botum
      notify: Reload nginx
    - name: Create app user
      ansible.builtin.user:
        name: botum
        system: yes
  handlers:
    - name: Reload nginx
      ansible.builtin.service:
        name: nginx
        state: reloaded
```

Forces Ansible : idempotence native, agentless (SSH), roles reutilisables, inventory dynamique Azure/AWS.

GitOps : Git comme source de verite

GitOps unifie tout : toute modification d'infra passe par une Pull Request — ticket de change auditable, avec discussion, approbation et historique. Le pipeline GitHub Actions : plan auto sur PR, apply sur merge avec approbation.

```yaml
# .github/workflows/terraform.yml
on:
  pull_request:
    paths: ['terraform/**']
  push:
    branches: [main]
    paths: ['terraform/**']

jobs:
  plan:
    if: github.event_name == 'pull_request'
    steps:
      - uses: hashicorp/setup-terraform@v3
      - run: terraform init && terraform plan
        env:
          ARM_CLIENT_ID: ${{ secrets.ARM_CLIENT_ID }}
          ARM_CLIENT_SECRET: ${{ secrets.ARM_CLIENT_SECRET }}

  apply:
    if: github.ref == 'refs/heads/main'
    environment: production  # Approbation manuelle
    steps:
      - run: terraform apply -auto-approve
```

Terraform vs Ansible : qui fait quoi

Critere	Terraform	Ansible
Role principal	Provisionnement (creer/detruire)	Configuration (installer, deployer)
Cible	APIs cloud (Azure, AWS, DNS)	Serveurs existants (OS, packages, fichiers)
Approche	Declarative (HCL)	Declarative + YAML
Etat	State file (gestion explicite)	Stateless (lu sur serveur)
Idempotence	Oui (natif)	Oui (si bien ecrit)
Apprentissage	Moyenne (HCL, state)	Faible (YAML lisible)
PME use case	VMs, VNets, DBs, LBs dans le cloud	Nginx, Postgres, deploi apps

Regle simple : Terraform pour tout ce qui est dans le portail cloud. Ansible pour tout ce qui se fait en SSH sur un serveur.

Pour les PME : commencer simple

Une structure de repo unique suffit pour demarrer. La valeur de l'IaC explose quand vous pouvez creer staging en 10 min depuis les modules prod eprouves.

```
infra-repo/
├── terraform/
│   ├── environments/
│   │   ├── dev/
│   │   ├── staging/
│   │   └── prod/
│   └── modules/
│       ├── vm/
│       └── network/
├── ansible/
│   ├── inventory/
│   ├── playbooks/
│   └── roles/
└── .github/
    └── workflows/
        ├── terraform.yml
        └── ansible.yml
```

Les pieges qui font mal

- State file en local ou dans Git : contient des secrets en clair. Backend remote chiffre obligatoire (Azure Blob CMK, S3 SSE).
- Secrets en dur dans HCL/YAML : injectez depuis Azure Key Vault, AWS Secrets Manager ou HashiCorp Vault via CI/CD.
- Versions Terraform non epinglees : sans required_version, une MaJ auto peut casser le pipeline du jour au lendemain.
- Pas de separation des environnements : un state file dev/staging/prod unique est une bombe. Directories et backends distincts obligatoires.

Cas concret BOTUM : 3 environnements, 10 min de deploiement

Client SaaS B2B, 35 employes, stack Azure. Situation initiale : infra creee a la main. Aucune doc a jour. Nouveau DevOps bloque 3 semaines pour "comprendre ce qui tourne". Reprise apres incident : 4h minimum.

Migration IaC avec BOTUM — 6 semaines :

- Semaines 1-2 : audit infra, terraform import, creation modules VM/reseau/DB
- Semaines 3-4 : structuration Ansible (roles nginx, node.js, postgresql, monitoring)
- Semaines 5-6 : pipeline GitHub Actions, migration staging puis prod

Resultats :

Metrique	Avant	Apres IaC
Creation env staging	4 heures	**11 minutes**
Onboarding nouveau DevOps	3 semaines	**2 jours**
Incidents config manuelle	Reguliers	**0**
Audit trail modifications	Absent	**Git complet**

Conclusion

L'IaC n'est plus une pratique "avancee". C'est le minimum pour : reproductibilite, tracabilite, resilience, conformite. Commencez simple : un repo, Terraform pour le cloud, Ansible pour les VMs, GitHub Actions pour le pipeline.