

TP9 - Vérification des contraintes

Cette documentation démontre comment l'infrastructure respecte toutes les contraintes imposées, avec une justification par le code pour chaque point, assurant ainsi une solution robuste, sécurisée et hautement disponible pour l'application Nextcloud.

Contrainte	Description	Conformité
Haute Disponibilité Réseau	Multi-AZ, ALB, NAT Gateway	Réalisée
Nomenclature des Ressources	Utilisation de <code>\${local.name}</code>	Réalisée
Authentification SSH EC2	Clés SSH via AWS Key Pair	Réalisée
Accessibilité SSH Bastion	Accès SSH limité via IP whitelistS	Réalisée
Accessibilité SSH Nextcloud	Accès SSH via Bastion uniquement	Réalisée
Utilisation EC2 Instance Connect	Pas de credentials persistants	Réalisée
EFS - Sécurité des Fichiers	Chiffrement AES-256	Réalisée
EFS - Accessibilité	Accès limité via SG	Réalisée
EFS - Haute Disponibilité	Déploiement multi-AZ	Réalisée
RDS - Accessibilité	Accès limité via SG	Réalisée
RDS - Haute Disponibilité	RDS Multi-AZ	Réalisée
Application NextCloud -Accessibilité	ALB avec SG restrictif	Réalisée
Application NextCloud - Haute Disponibilité	ASG avec ALB	Réalisée
Application NextCloud - Élasticité	ASG - Scaling automatique basé sur charge	Réalisée
Application NextCloud - Scalabilité	ASG avec min_size/max_size dynamiques	Réalisée
Authentification des instances EC2 avec l'API AWS	IAM Roles sans credentials persistants	Réalisée
Permissions entre Nextcloud & le bucket S3	Rôle IAM avec permissions minimales	Réalisée

1. Contraintes liées à la haute disponibilité réseau

Solution implémentée:

- Déploiement dans 3 zones de disponibilité différentes
- Sous-réseaux publics et privés dans chaque AZ
- Utilisation des foreach

Exemple pour les subnets privés :

```
# Pour chaque sous-réseau privé, on définit le CIDR et l'AZ dans lequel il se trouve
resource "aws_subnet" "private" {
  for_each      = local.private_subnet
  vpc_id        = aws_vpc.main.id
  cidr_block    = each.value.cidr
  availability_zone = each.value.az
  tags = {
    Name = "${local.name}-private-${each.value.az}"
  }
}
```

2. Contraintes de nomenclature des ressources

Solution implémentée:

- Utilisation systématique de la variable `${local.name}` ou `${local.user}` comme préfixe
- Convention de nommage (tags) cohérente pour tous les composants

Justification par le code:

```
resource "aws_security_group" "bastion_sg" {
  name = "${local.name}-bastion-sg"
  # ...
}
```

3. Contraintes liées à l'authentification SSH sur les instances EC2

Solution implémentée:

- Utilisation de clés SSH via AWS Key Pair
- Désactivation de l'authentification par mot de passe

Justification par le code:

```
resource "aws_key_pair" "nextcloud" {
  key_name     = "${local.name}-ssh-key"
  public_key   = file("~/ssh/id_rsa.pub")
}
```

4. Contraintes liées à l'accessibilité (SSH) du bastion

Solution implémentée:

- Bastion dans un sous-réseau public
- Accès SSH limité à des IPs spécifiques

Exemple, autorisation depuis l'IP ynov:

```
# Autoriser le SSH depuis YNOV
resource "aws_vpc_security_group_ingress_rule" "allow_ssh_from_ynov_to_bastion" {
  security_group_id = aws_security_group.bastion_sg.id

  # YNOV IP
  cidr_ipv4      = "195.7.117.146/32"
  from_port      = 22
  ip_protocol    = "tcp"
  to_port        = 22

  tags = {
    Name = "Allow SSH from YNOV"
  }
}
```

5. Contraintes liées à l'accessibilité (SSH) des instances Nextcloud

Solution implémentée:

- Instances dans des sous-réseaux privés
- Accès SSH uniquement via le bastion (security group)

Autorisation depuis le SG du bastion:

```
# Autoriser le trafic SSH depuis le bastion
resource "aws_vpc_security_group_ingress_rule" "allow_ssh_from_bastion" {
  security_group_id = aws_security_group.nextcloud_sg.id

  # Autoriser le trafic SSH depuis le security group du bastion
  referenced_security_group_id = aws_security_group.bastion_sg.id
  from_port                    = 22
  ip_protocol                  = "tcp"
}
```

```

to_port                = 22

tags = {
    Name = "Allow SSH from Bastion"
}
}

```

6. Contraintes liées à l'utilisation du service EC2 instance connect

Solution implémentée:

- On interdit l'utilisation d'EC2 instant connect au niveau de l'ACL

Au niveau de l'ACL:

```

# Règle 100: Bloquer le trafic SSH depuis la plage d'adresses 13.48.4.200/30 (EC
ingress {
    rule_no    = 100
    action     = "deny"
    protocol   = "tcp"
    cidr_block = "13.48.4.200/30"
    from_port  = 22
    to_port    = 22
}

```

7. Contraintes liées à la sécurité des fichiers stockés sur le système de fichier partagé

Solution implémentée:

- EFS avec chiffrement activé `encrypted = true`
- Accès restreint via security groups

Justification par le code:

```

# Créer un EFS pour Nextcloud
resource "aws_efs_file_system" "nextcloud_efs" {
    creation_token = "nextcloud-efs-token"
    encrypted      = true
    performance_mode = "generalPurpose"
    tags = {
        Name = "${local.name}-nextcloud-efs"
    }
}

```

8. Contraintes liées à l'accessibilité du système de fichier

partagé

Solution implémentée:

- Accès NFS (port 2049) limité aux instances Nextcloud

Limitation de l'accès via une règle dans le SG de l'EFS:

```
# Autoriser uniquement Nextcloud à accéder à EFS sur le port 2049
resource "aws_vpc_security_group_ingress_rule" "allow_nfs_from_nextcloud" {
  security_group_id = aws_security_group.efs_sg.id

  # Autoriser le trafic NFS/EFS depuis le security group de Nextcloud
  referenced_security_group_id = aws_security_group.nextcloud_sg.id
  from_port                    = 2049
  ip_protocol                  = "tcp"
  to_port                     = 2049

  tags = {
    Name = "Allow NFS/EFS access from Nextcloud SG"
  }
}
```

9. Contraintes liées à la haute disponibilité du système de fichier partagé

Solution implémentée:

- EFS déployé en mode multi-AZ
- Mount targets dans chaque zone de disponibilité

```
# Créer un mount target pour chaque subnet privé
resource "aws_efs_mount_target" "nextcloud_efs_targets" {
  for_each = aws_subnet.private

  file_system_id = aws_efs_file_system.nextcloud_efs.id
  subnet_id      = each.value.id
  security_groups = [aws_security_group.efs_sg.id]
}
```

10. Contraintes liées à l'accessibilité de la base de données

Solution implémentée:

- RDS dans des sous-réseaux privés
- Accès limité aux instances Nextcloud

Règle pour autoriser uniquement NextCloud à se connecter au RDS:

```
# Autoriser uniquement Nextcloud à accéder à MySQL sur le port 3306
resource "aws_vpc_security_group_ingress_rule" "allow_mysql_from_nextcloud" {
  security_group_id = aws_security_group.nextcloud_db_sg.id

  # Autoriser le trafic MySQL depuis le security group de Nextcloud
  referenced_security_group_id = aws_security_group.nextcloud_sg.id
  from_port                    = 3306
  ip_protocol                  = "tcp"
  to_port                     = 3306

  tags = {
    Name = "Allow MySQL access from Nextcloud SG"
  }
}
```

11. Contraintes liées à la haute disponibilité de la base de données

Solution implémentée:

- RDS en mode Multi-AZ `multi_az = true`
- DB Subnet Group pointe vers les différents sous réseaux privés

```
# Créer un groupe de sous-réseaux pour la base de données RDS
resource "aws_db_subnet_group" "nextcloud_rds_subnet" {
  name      = "ymontagnier-nextcloud-rds-subnet"
  subnet_ids = [for subnet in aws_subnet.private : subnet.id]
  tags = {
    Name = "${local.name}-nextcloud-rds-subnet"
  }
}
```

12. Contraintes liées à l'accessibilité de l'application Nextcloud

Solution implémentée:

- Application Load Balancer dans les sous-réseaux publics
- Accès contrôlé par whitelist d'IP

Autorisation d'accès pour une adresse IP:

```
# Autoriser le trafic HTTP depuis l'IP d'YNOV
resource "aws_vpc_security_group_ingress_rule" "allow_http_from_ynov_to_alb" {
  security_group_id = aws_security_group.nextcloud-alb-sg.id
```

```
cidr_ipv4    = "195.7.117.146/32"
from_port    = 80
ip_protocol  = "tcp"
to_port      = 80

tags = {
  Name = "Autoriser l'accès à NextCloud depuis Ynov"
}
}
```

13. Contraintes liées à la haute disponibilité de l'application

Solution implémentée:

- Launch Template & Auto Scaling Group avec distribution multi-AZ
- Load balancer avec surveillance de l'état des instances

14. Contraintes liées à l'élasticité de l'application

Solution implémentée:

- Politiques de scaling basées sur la charge CPU et les requêtes
- Déclenchement automatique des alarmes CloudWatch

15. Contraintes liées à la scalabilité de l'application

Solution implémentée:

- Auto Scaling Group avec capacité dynamique (min=1, max=5)
- Architecture permettant d'ajouter/supprimer des instances à la demande

16. Contraintes liées à la méthode d'authentification des instances EC2 auprès des API AWS

Solution implémentée:

- Utilisation de rôles IAM plutôt que de clés d'accès stockées
- Profil d'instance attaché aux instances EC2

Justification par le code:

```
resource "aws_iam_role" "nextcloud_role" {
  name = "${local.name}-nextcloud"
  assume_role_policy = jsonencode({
    Version = "2012-10-17"
    Statement = [{
```

```

    Effect = "Allow"
    Principal = { Service = "ec2.amazonaws.com" }
    Action = "sts:AssumeRole"
  }]
})
}

```

17. Contraintes liées aux permissions entre Nextcloud et le Bucket S3

Solution implémentée:

- Rôle IAM avec permissions minimales nécessaires
- Politique IAM inline limitant l'accès aux opérations essentielles

IAM Role Policy:

```

resource "aws_iam_role_policy" "nextcloud_role_policy" {
  name     = "NextcloudS3AccessPolicy"
  role     = aws_iam_role.nextcloud_role.id

  policy = jsonencode({
    Version = "2012-10-17"
    Statement = [
      {
        Effect = "Allow"
        Action = [
          "s3:ListBucket",
          "s3:ListBucketMultipartUploads"
        ]
        Resource = "arn:aws:s3:::${aws_s3_bucket.nextcloud_bucket.id}"
      },
      {
        Effect = "Allow"
        Action = [
          "s3:GetObject",
          "s3:PutObject",
          "s3:DeleteObject",
          "s3:AbortMultipartUpload",
          "s3:ListMultipartUploadParts"
        ]
        Resource = "arn:aws:s3:::${aws_s3_bucket.nextcloud_bucket.id}/*"
      }
    ]
  })
}

```

18. Contraintes liées aux permissions sur le bucket S3

Solution implémentée:

- Chiffrement côté serveur AES-256
- Bucket policy restrictive avec deny explicite pour toutes les actions non autorisées

S3 Bucket Policy:

```
resource "aws_s3_bucket_policy" "nextcloud_bucket_policy" {
  bucket = aws_s3_bucket.nextcloud_bucket.id

  policy = jsonencode({
    Version = "2012-10-17"
    Statement = [
      {
        Sid      = "AllowTerraformAdmin"
        Effect   = "Allow"
        Principal = {
          AWS = "arn:aws:iam::${data.aws_caller_identity.current.account_id}:user/"
        }
        Action = [
          "s3:ListBucket",
          "s3:GetBucketLocation",
          "s3:GetBucketPolicy"
        ]
        Resource = aws_s3_bucket.nextcloud_bucket.arn
      },
      {
        Sid      = "DenyTerraformDataAccess"
        Effect   = "Deny"
        Principal = {
          AWS = aws_iam_role.nextcloud_role.arn
        }
        Action = [
          "s3:GetObject",
          "s3:PutObject",
          "s3:DeleteObject"
        ]
        Resource = "${aws_s3_bucket.nextcloud_bucket.arn}/*"
      },
      {
        Sid      = "AllowEC2Access"
        Effect   = "Allow"
        Principal = {
          AWS = aws_iam_role.nextcloud_role.arn
        }
        Action = [
          "s3:ListBucket",
          "s3:ListBucketMultipartUploads",
          "s3:GetObject",
          "s3:PutObject",
          "s3:DeleteObject",
          "s3:AbortMultipartUpload",
          "s3:ListMultipartUploadParts"
        ]
        Resource = [
          aws_s3_bucket.nextcloud_bucket.arn,
          "${aws_s3_bucket.nextcloud_bucket.arn}/*"
        ]
      }
    ]
  })
}
```

