Kubernetes 1.33完全実装ガイド

Proxmox + Terraform + Ansible + helmfile統合構成

対象環境: 開発環境（本番環境への拡張可能）
構成: Control Plane 2台 + Worker 3台
最終更新: 2025年10月

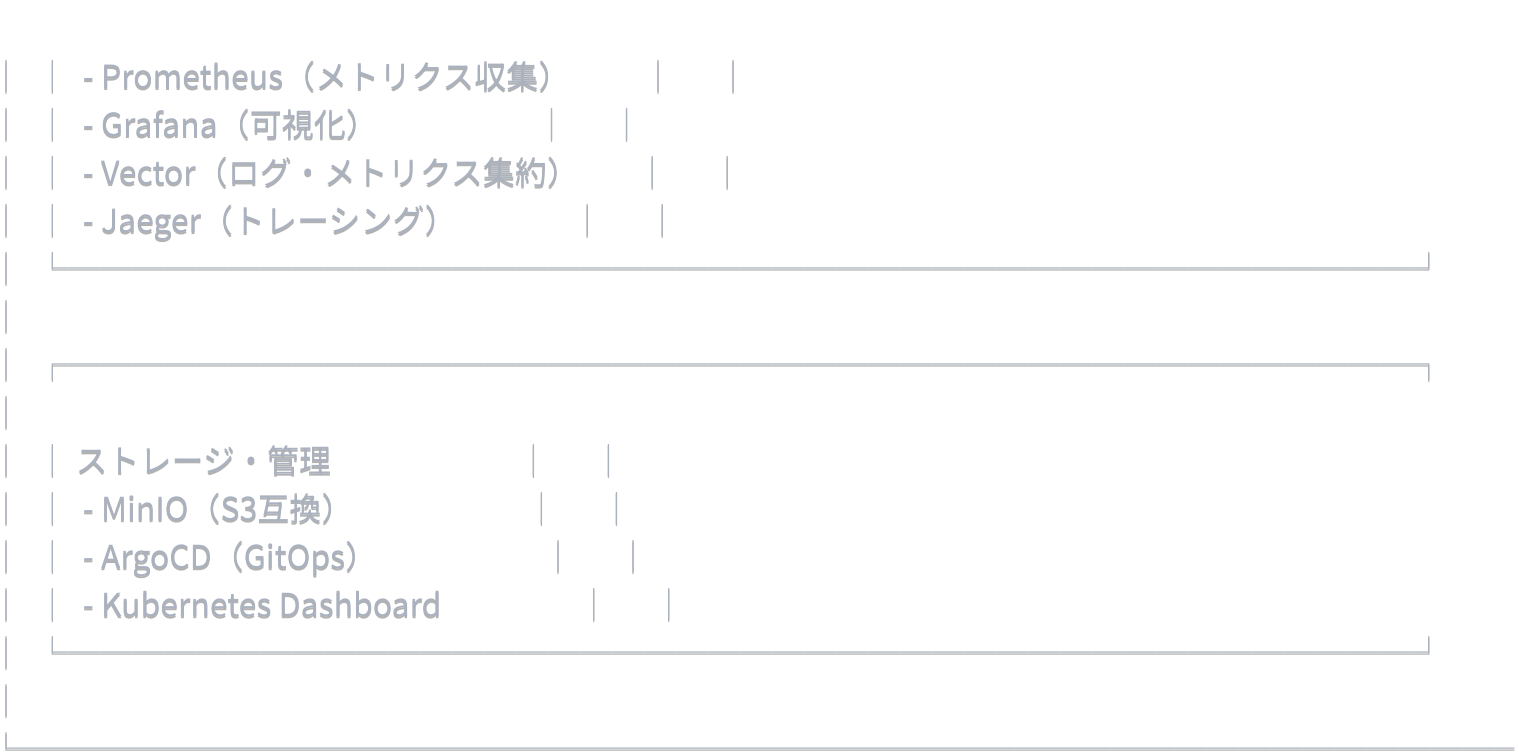
目次

---

1. アーキテクチャ概要

1.1 全体構成図


Proxmox VE
Terraform管理VM群
CP-1
4vCPU
8GB RAM
CP-2
4vCPU
8GB RAM
Control Plane
Worker-1
8vCPU
16GB RAM
Worker-2
8vCPU
16GB RAM
Worker-3
8vCPU
16GB RAM
Ansible自動構築（kubeadm）
Kubernetes 1.33クラスタ
Cilium CNI（kube-proxy完全置換）
- LB IPAM（MetalLB代替）
- Hubble可視化
- Gateway API（HTTP/HTTPS/TLS/gRPC）
Traefik Ingress
- TCP/UDPルーティング
- 全UIへのアクセス提供
監視スタック



1.2 ネットワーク設計

用途	CIDR	例
Proxmox管理ネットワーク	192.168.1.0/24	-
Control Plane	192.168.1.10-11	CP-1: .10, CP-2: .11
Worker	192.168.1.20-22	W-1: .20, W-2: .21, W-3: .22
API VIP	192.168.1.100	HAProxy/keepalived
Cilium LB IPAM Pool	192.168.100.0/24	LoadBalancer Service用
Pod Network	10.244.0.0/16	Cilium管理
Service Network	10.96.0.0/12	ClusterIP範囲

1.3 重要な設計判断

✅ 採用する技術

- **Cilium Gateway API**: HTTP/HTTPS/gRPC/TLS専用（本番実績あり）
- **Traefik Ingress**: TCP/UDP用（Gateway APIのTCPRoute/UDPRouteは代替）
- **kube-proxy完全置換**: eBPFベースの高性能化
- **LB IPAM**: MetalLB不要、Cilium統合機能を使用
- **helmfile**: 環境切り替え＋GitOps準備

❌ 使用しない技術

- **Cilium Gateway API for TCP/UDP**: 2025年10月時点で未実装（GitHub Issue #21929オープン中）
- **MetalLB**: Cilium LB IPAMで代替
- **Istio/Linkerd**: 開発環境では過剰

---

2. バージョン一覧

2.1 推奨バージョン

コンポーネント	バージョン	リリース日	サポート期限	Helmチャート
Kubernetes	1.33.5	2025年1月	2026年6月	-
Cilium	1.18.2	2025年9月	LTS	cilium/cilium:1.18.2
Traefik	3.5.0	2025年10月	-	traefik/traefik:37.1.2
kube-prometheus-stack	78.3.0	2025年10月	-	prometheus-community/kube-prometheus-stack:78.3.0
Grafana	11.x	2025年	-	(stackに含む)
Jaeger Operator	1.65.0	2025年	-	jaegertracing/jaeger-operator:2.57.0
Vector	0.46.0	2025年	-	vector/vector:0.46.0
MinIO Operator	7.1.1	2025年	-	minio-operator/operator:7.1.1
ArgoCD	2.13.x	2025年	-	argo/argo-cd:8.6.2
Kubernetes Dashboard	7.13.0	2025年	-	kubernetes-dashboard/kubernetes-dashboard:7.13.0

2.2 互換性マトリックス

コンポーネント	K8s 1.31	K8s 1.32	K8s 1.33	K8s 1.34
Cilium 1.18	✅	✅	✅	✅
Traefik 3.5	✅	✅	✅	✅
kube-prometheus-stack 78.x	✅	✅	✅	✅
ArgoCD 2.13	✅	✅	✅	✅

---

3. Proxmox + Terraform構成

3.1 ディレクトリ構造

```
infrastructure/
├── terraform/
│   ├── main.tf
│   ├── variables.tf
│   ├── outputs.tf
│   ├── terraform.tfvars
│   └── modules/
│       ├── control-plane/
│       │   ├── main.tf
│       │   └── variables.tf
│       └── worker/
│           ├── main.tf
│           └── variables.tf
├── ansible/
│   ├── inventory/
│   │   └── hosts.ini
│   ├── group_vars/
│   │   ├── all.yml
│   │   └── k8s-cluster.yml
│   ├── playbooks/
│   │   ├── 01-prerequisites.yml
│   │   ├── 02-container-runtime.yml
│   │   ├── 03-kubernetes-install.yml
│   │   └── 04-cluster-init.yml
│   └── templates/
│       └── kubeadm-config.yaml.j2
└── helmfile/
    ├── helmfile.yaml
    ├── environments/
    │   ├── dev.yaml
    │   ├── staging.yaml
    │   └── prod.yaml
    └── values/
        ├── dev/
        ├── staging/
        └── prod/
```

3.2 Terraform設定

terraform/main.tf:

hcl

```
terraform {
  required_version = ">= 1.9"
  required_providers {
    proxmox = {
      source  = "Telmate/proxmox"
      version = "~> 3.0"
    }
  }
}

provider "proxmox" {
  pm_api_url      = var.proxmox_api_url
  pm_user         = var.proxmox_user
  pm_password     = var.proxmox_password
  pm_tls_insecure = true
}

# Control Plane VMs
module "control_plane" {
  source = "./modules/control-plane"

  count       = 2
  vm_name     = "k8s-cp-${count.index + 1}"
  target_node = var.proxmox_node
  vm_id       = 200 + count.index

  cores       = 4
  sockets     = 1
  memory      = 8192
  disk_size   = "100G"

  network_bridge = "vmbr0"
  ip_address     = "192.168.1.${10 + count.index}"
  gateway        = var.gateway

  ssh_keys       = var.ssh_public_keys
  cloud_init_image = var.cloud_init_image
}

# Worker VMs
module "worker" {
  source = "./modules/worker"

  count       = 3
```

```
  vm_name     = "k8s-worker-${count.index + 1}"
  target_node  = var.proxmox_node
  vm_id       = 210 + count.index

  cores       = 8
  sockets     = 1
  memory      = 16384
  disk_size   = "200G"

  network_bridge = "vmbr0"
  ip_address   = "192.168.1.${20 + count.index}"
  gateway     = var.gateway

  ssh_keys     = var.ssh_public_keys
  cloud_init_image = var.cloud_init_image
}

# Output
output "control_plane_ips" {
 value = [for vm in module.control_plane : vm.ip_address]
}

output "worker_ips" {
 value = [for vm in module.worker : vm.ip_address]
}
```

terraform/variables.tf:

hcl

```
variable "proxmox_api_url" {
  description = "Proxmox API URL"
  type        = string
  default     = "https://proxmox.example.com:8006/api2/json"
}

variable "proxmox_user" {
  description = "Proxmox user"
  type        = string
}

variable "proxmox_password" {
  description = "Proxmox password"
  type        = string
  sensitive   = true
}

variable "proxmox_node" {
  description = "Proxmox node name"
  type        = string
  default     = "pve"
}

variable "gateway" {
  description = "Network gateway"
  type        = string
  default     = "192.168.1.1"
}

variable "ssh_public_keys" {
  description = "SSH public keys"
  type        = list(string)
}

variable "cloud_init_image" {
  description = "Cloud-init template name"
  type        = string
  default     = "ubuntu-22.04-cloud"
}
```

terraform/terraform.tfvars:

hcl

```
proxmox_api_url = "https://192.168.1.5:8006/api2/json"
proxmox_user    = "root@pam"
proxmox_node    = "pve"
gateway         = "192.168.1.1"

ssh_public_keys = [
  "ssh-rsa AAAAB3NzaC1yc2EA... user@laptop"
]

cloud_init_image = "ubuntu-22.04-cloud-init"
```

terraform/modules/control-plane/main.tf:

hcl

```
resource "proxmox_vm_qemu" "control_plane" {
  name        = var.vm_name
  target_node = var.target_node
  vmid        = var.vm_id

  clone      = var.cloud_init_image
  full_clone = true

  cores   = var.cores
  sockets = var.sockets
  memory  = var.memory

  disk {
    size    = var.disk_size
    type    = "scsi"
    storage = "local-lvm"
    iothread = 1
  }

  network {
    model  = "virtio"
    bridge = var.network_bridge
  }

  # Cloud-init
  os_type   = "cloud-init"
  ipconfig0 = "ip=${var.ip_address}/24,gw=${var.gateway}"

  sshkeys = join("\n", var.ssh_keys)

  lifecycle {
    ignore_changes = [
      network,
    ]
  }

  tags = "kubernetes,control-plane"
}

output "ip_address" {
  value = var.ip_address
}
```

3.3 Terraform実行

bash

```
# 初期化
cd terraform/
terraform init

# 計画確認
terraform plan

# 実行
terraform apply -auto-approve

# 出力確認
terraform output
```

---

4. Ansible自動構築

4.1 インベントリ

ansible/inventory/hosts.ini:

ini

```
[control_plane]
k8s-cp-1 ansible_host=192.168.1.10 ansible_user=ubuntu
k8s-cp-2 ansible_host=192.168.1.11 ansible_user=ubuntu

[workers]
k8s-worker-1 ansible_host=192.168.1.20 ansible_user=ubuntu
k8s-worker-2 ansible_host=192.168.1.21 ansible_user=ubuntu
k8s-worker-3 ansible_host=192.168.1.22 ansible_user=ubuntu

[k8s_cluster:children]
control_plane
workers

[all:vars]
ansible_python_interpreter=/usr/bin/python3
ansible_ssh_common_args='-o StrictHostKeyChecking=no'
```

4.2 Group Variables

ansible/group_vars/all.yml:

yaml

```yaml
# Kubernetes設定
kubernetes_version: "1.33.5"
kubernetes_cni: "cilium"
skip_kube_proxy: true

# Container Runtime
container_runtime: "containerd"
containerd_version: "1.7.x"

# Network設定
pod_network_cidr: "10.244.0.0/16"
service_cidr: "10.96.0.0/12"
cluster_dns: "10.96.0.10"

# Control Plane
control_plane_endpoint: "192.168.1.100:6443"  # VIPまたはDNS
api_server_cert_sans:
 - "k8s-api.example.com"
 - "192.168.1.100"
 - "192.168.1.10"
 - "192.168.1.11"

# Cilium設定
cilium_version: "1.18.2"
cilium_lb_ipam_pool: "192.168.100.0/24"
```

4.3 Playbook 1: 前提条件

ansible/playbooks/01-prerequisites.yml:

yaml

```yaml
---
- name: Kubernetes前提条件セットアップ
  hosts: k8s_cluster
  become: yes
  tasks:
    - name: システムアップデート
      apt:
        update_cache: yes
        upgrade: dist
      when: ansible_os_family == "Debian"

    - name: 必要パッケージインストール
      apt:
        name:
          - apt-transport-https
          - ca-certificates
          - curl
          - gnupg
          - lsb-release
          - socat
          - conntrack
          - ipset
        state: present

    - name: Swapの無効化
      shell: |
        swapoff -a
        sed -i '/swap/d' /etc/fstab

    - name: カーネルモジュールの永続化
      copy:
        dest: /etc/modules-load.d/k8s.conf
        content: |
          overlay
          br_netfilter

    - name: カーネルモジュールのロード
      modprobe:
        name: "{{ item }}"
        state: present
      loop:
        - overlay
        - br_netfilter
```

```yaml
- name: sysctlパラメータ設定
  copy:
    dest: /etc/sysctl.d/k8s.conf
    content: |
      net.bridge.bridge-nf-call-iptables  = 1
      net.bridge.bridge-nf-call-ip6tables = 1
      net.ipv4.ip_forward                 = 1

- name: sysctlパラメータ適用
  command: sysctl --system

- name: ファイアウォール無効化（開発環境）
  systemd:
    name: ufw
    state: stopped
    enabled: no
  ignore_errors: yes
```

4.4 Playbook 2: Container Runtime

ansible/playbooks/02-container-runtime.yml:

yaml

```yaml
---
- name: containerdインストール
  hosts: k8s_cluster
  become: yes
  tasks:
   - name: Dockerリポジトリ追加
     shell: |
      install -m 0755 -d /etc/apt/keyrings
      curl -fsSL https://download.docker.com/linux/ubuntu/gpg | gpg --dearmor -o /etc/apt/keyrings/doc
      chmod a+r /etc/apt/keyrings/docker.gpg
      echo "deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docker.gpg] https://dowr
     args:
      creates: /etc/apt/sources.list.d/docker.list

   - name: containerdインストール
     apt:
      name: containerd.io
      state: present
      update_cache: yes

   - name: containerd設定ディレクトリ作成
     file:
      path: /etc/containerd
      state: directory

   - name: containerdデフォルト設定生成
     shell: containerd config default > /etc/containerd/config.toml
     args:
      creates: /etc/containerd/config.toml

   - name: SystemdCgroup有効化
     lineinfile:
      path: /etc/containerd/config.toml
      regexp: 'SystemdCgroup = false'
      line: '            SystemdCgroup = true'
      state: present

   - name: containerd再起動
     systemd:
      name: containerd
      state: restarted
```

```
    enabled: yes
    daemon_reload: yes
```

4.5 Playbook 3: Kubernetesインストール

ansible/playbooks/03-kubernetes-install.yml:

yaml

```yaml
---
- name: Kubernetesパッケージインストール
  hosts: k8s_cluster
  become: yes
  tasks:
    - name: Kubernetesリポジトリキー追加
      shell: |
        mkdir -p /etc/apt/keyrings
        curl -fsSL https://pkgs.k8s.io/core:/stable:/v1.33/deb/Release.key | gpg --dearmor -o /etc/apt/keyrin
      args:
        creates: /etc/apt/keyrings/kubernetes-apt-keyring.gpg

    - name: Kubernetesリポジトリ追加
      copy:
        dest: /etc/apt/sources.list.d/kubernetes.list
        content: |
          deb [signed-by=/etc/apt/keyrings/kubernetes-apt-keyring.gpg] https://pkgs.k8s.io/core:/stable:/v1

    - name: Kubernetesパッケージインストール
      apt:
        name:
          - "kubelet={{ kubernetes_version }}-*"
          - "kubeadm={{ kubernetes_version }}-*"
          - "kubectl={{ kubernetes_version }}-*"
        state: present
        update_cache: yes

    - name: Kubernetesパッケージバージョン固定
      dpkg_selections:
        name: "{{ item }}"
        selection: hold
      loop:
        - kubelet
        - kubeadm
        - kubectl
```

4.6 Playbook 4: クラスタ初期化

ansible/playbooks/04-cluster-init.yml:

yaml

```yaml
---
- name: 最初のControl Plane初期化
  hosts: control_plane[0]
  become: yes
  vars:
    kubeconfig_path: "{{ ansible_env.HOME }}/.kube/config"
  tasks:
    - name: kubeadm設定ファイル生成
      template:
        src: ../templates/kubeadm-config.yaml.j2
        dest: /root/kubeadm-config.yaml

    - name: クラスタ初期化チェック
      stat:
        path: /etc/kubernetes/admin.conf
      register: k8s_init

    - name: クラスタ初期化
      command: kubeadm init --config=/root/kubeadm-config.yaml --upload-certs
      register: kubeadm_init_output
      when: not k8s_init.stat.exists

    - name: 初期化ログ保存
      copy:
        content: "{{ kubeadm_init_output.stdout }}"
        dest: /root/kubeadm-init.log
      when: kubeadm_init_output.changed

    - name: kubeconfigディレクトリ作成
      file:
        path: "{{ ansible_env.HOME }}/.kube"
        state: directory
        mode: '0755'

    - name: kubeconfigコピー
      copy:
        src: /etc/kubernetes/admin.conf
        dest: "{{ kubeconfig_path }}"
        remote_src: yes
        owner: "{{ ansible_user }}"
        group: "{{ ansible_user }}"
        mode: '0644'

    - name: Join Token取得
```

```
    command: kubeadm token create
    register: join_token

  - name: CA証明書ハッシュ取得
    shell: openssl x509 -pubkey -in /etc/kubernetes/pki/ca.crt | openssl rsa -pubin -outform der 2>/dev/n
    register: ca_cert_hash

  - name: Certificate Key取得
    shell: kubeadm init phase upload-certs --upload-certs 2>/dev/null | tail -n 1
    register: certificate_key

  - name: Join情報を変数に保存
    set_fact:
      k8s_join_token: "{{ join_token.stdout }}"
      k8s_ca_hash: "{{ ca_cert_hash.stdout }}"
      k8s_cert_key: "{{ certificate_key.stdout }}"

  - name: Helmインストール
    shell: |
      curl https://raw.githubusercontent.com/helm/helm/main/scripts/get-helm-3 | bash
    args:
      creates: /usr/local/bin/helm

  - name: Ciliumインストール
    shell: |
      helm repo add cilium https://helm.cilium.io/ || true
      helm repo update
      helm install cilium cilium/cilium \
        --version {{ cilium_version }} \
        --namespace kube-system \
        --set kubeProxyReplacement=true \
        --set k8sServiceHost={{ control_plane_endpoint.split(':')[0] }} \
        --set k8sServicePort={{ control_plane_endpoint.split(':')[1] }} \
        --set hubble.enabled=true \
        --set hubble.relay.enabled=true \
        --set hubble.ui.enabled=true \
        --wait \
        --timeout=10m
    environment:
      KUBECONFIG: "{{ kubeconfig_path }}"

- name: 2台目のControl Plane参加
  hosts: control_plane[1:]
  become: yes
```

```
  serial: 1
  tasks:
   - name: Control Plane参加済みチェック
     stat:
       path: /etc/kubernetes/kubelet.conf
     register: cp_joined

   - name: Control Planeとして参加
     command: >
       kubeadm join {{ control_plane_endpoint }}
       --token {{ hostvars[groups['control_plane'][0]]['k8s_join_token'] }}
       --discovery-token-ca-cert-hash sha256:{{ hostvars[groups['control_plane'][0]]['k8s_ca_hash'] }}
       --control-plane
       --certificate-key {{ hostvars[groups['control_plane'][0]]['k8s_cert_key'] }}
       --apiserver-advertise-address={{ ansible_default_ipv4.address }}
     when: not cp_joined.stat.exists

- name: Worker参加
  hosts: workers
  become: yes
  tasks:
   - name: Worker参加済みチェック
     stat:
       path: /etc/kubernetes/kubelet.conf
     register: worker_joined

   - name: Workerとして参加
     command: >
       kubeadm join {{ control_plane_endpoint }}
       --token {{ hostvars[groups['control_plane'][0]]['k8s_join_token'] }}
       --discovery-token-ca-cert-hash sha256:{{ hostvars[groups['control_plane'][0]]['k8s_ca_hash'] }}
     when: not worker_joined.stat.exists

- name: kubeconfig取得
  hosts: control_plane[0]
  become: yes
  tasks:
   - name: kubeconfigを取得
     fetch:
       src: /etc/kubernetes/admin.conf
       dest: "{{ playbook_dir }}/../kubeconfig"
       flat: yes
```

4.7 kubeadm設定テンプレート

ansible/templates/kubeadm-config.yaml.j2:

yaml

```
---
apiVersion: kubeadm.k8s.io/v1beta4
kind: InitConfiguration
nodeRegistration:
 name: "{{ ansible_hostname }}"
 criSocket: "unix:///var/run/containerd/containerd.sock"
 kubeletExtraArgs:
  - name: "node-ip"
    value: "{{ ansible_default_ipv4.address }}"
 taints:
  - key: "node-role.kubernetes.io/control-plane"
    effect: "NoSchedule"
localAPIEndpoint:
 advertiseAddress: "{{ ansible_default_ipv4.address }}"
 bindPort: 6443
{% if skip_kube_proxy %}
skipPhases:
 - addon/kube-proxy
{% endif %}

---
apiVersion: kubeadm.k8s.io/v1beta4
kind: ClusterConfiguration
kubernetesVersion: "v{{ kubernetes_version }}"
clusterName: "k8s-cluster"
controlPlaneEndpoint: "{{ control_plane_endpoint }}"
networking:
 serviceSubnet: "{{ service_cidr }}"
 podSubnet: "{{ pod_network_cidr }}"
 dnsDomain: "cluster.local"
{% if skip_kube_proxy %}
proxy:
 disabled: true
{% endif %}
apiServer:
 certSANs:
{% for san in api_server_cert_sans %}
  - "{{ san }}"
{% endfor %}
 extraArgs:
  - name: "authorization-mode"
    value: "Node,RBAC"
etcd:
 local:
```

```
    dataDir: "/var/lib/etcd"

---
apiVersion: kubelet.config.k8s.io/v1beta1
kind: KubeletConfiguration
cgroupDriver: systemd
serverTLSBootstrap: true
```

4.8 Ansible実行コマンド

bash

```
cd ansible/

# 全Playbook実行
ansible-playbook -i inventory/hosts.ini playbooks/01-prerequisites.yml
ansible-playbook -i inventory/hosts.ini playbooks/02-container-runtime.yml
ansible-playbook -i inventory/hosts.ini playbooks/03-kubernetes-install.yml
ansible-playbook -i inventory/hosts.ini playbooks/04-cluster-init.yml

# kubeconfigコピー
mkdir -p ~/.kube
cp kubeconfig ~/.kube/config

# 確認
kubectl get nodes
kubectl get pods -A
```

---

5. Cilium CNI設定（kube-proxy完全置換）

5.1 Cilium完全設定

helmfile/values/dev/cilium.yaml:

yaml

```yaml
# Cilium 1.18.2 開発環境設定

# クラスタ識別
cluster:
  name: dev-cluster
  id: 1

# ====================
# kube-proxy完全置換
# ====================
kubeProxyReplacement: true
k8sServiceHost: "192.168.1.100"
k8sServicePort: 6443

# eBPF機能
bpf:
  masquerade: true
  lbExternalClusterIP: true

# kube-proxy機能
nodePort:
  enabled: true
  mode: hybrid

externalIPs:
  enabled: true

hostPort:
  enabled: true

sessionAffinity: true

# ====================
# LoadBalancer IPAM（MetalLB代替）
# ====================
l2announcements:
  enabled: true

l2NeighDiscovery:
  enabled: true
  refreshPeriod: "30s"

k8sClientRateLimit:
  qps: 50
```

```yaml
  burst: 100

# ====================
# Hubble可観測性
# ====================
hubble:
  enabled: true

  relay:
    enabled: true
    replicas: 1
    resources:
      requests:
        cpu: 50m
        memory: 64Mi
      limits:
        cpu: 500m
        memory: 256Mi

  ui:
    enabled: true
    replicas: 1
    service:
      type: ClusterIP
    resources:
      requests:
        cpu: 50m
        memory: 64Mi
      limits:
        cpu: 500m
        memory: 256Mi

  metrics:
    enabled:
      - dns
      - drop
      - tcp
      - flow
      - http
    serviceMonitor:
      enabled: true
    port: 9965

# ====================
```

```yaml
# Gateway API（HTTP/HTTPS/TLS/gRPC専用）
# ====================
gatewayAPI:
  enabled: true

# ====================
# Prometheus統合
# ====================
prometheus:
  enabled: true
  port: 9962
  serviceMonitor:
    enabled: true
    interval: "30s"

# Operator
operator:
  replicas: 1  # 開発環境は1
  prometheus:
    enabled: true
    serviceMonitor:
      enabled: true

# Envoy（L7ポリシー）
envoy:
  enabled: true
  prometheus:
    enabled: true
    serviceMonitor:
      enabled: true

# ====================
# ネットワーク
# ====================
ipam:
  mode: kubernetes

tunnel: vxlan

ipv4:
  enabled: true
ipv6:
  enabled: false
```

```yaml
# ====================
# リソース管理（開発環境最適化）
# ====================
resources:
  requests:
    cpu: 100m
    memory: 256Mi
  limits:
    cpu: 2000m
    memory: 2Gi

# 全ノードデプロイ
tolerations:
  - operator: Exists

# ローリングアップデート
updateStrategy:
  type: RollingUpdate
  rollingUpdate:
    maxUnavailable: 1

rollOutCiliumPods: true

# デバッグ
debug:
  enabled: false
```

5.2 LoadBalancer IPプール

cilium-lb-ippool.yaml:

yaml

```
apiVersion: cilium.io/v2alpha1
kind: CiliumLoadBalancerIPPool
metadata:
  name: dev-pool
spec:
  cidrs:
    - cidr: "192.168.100.0/24"
---
apiVersion: cilium.io/v2alpha1
kind: CiliumL2AnnouncementPolicy
metadata:
  name: l2-policy
  namespace: kube-system
spec:
  interfaces:
    - ^ens[0-9]+
    - ^eth[0-9]+
  loadBalancerIPs: true
  externalIPs: true
  nodeSelector:
    matchExpressions:
      - key: node-role.kubernetes.io/control-plane
        operator: DoesNotExist
```

適用：

bash

```
kubectl apply -f cilium-lb-ippool.yaml
```

5.3 Cilium確認コマンド

bash

```
# Cilium状態
kubectl -n kube-system exec ds/cilium -- cilium status

# Connectivity Test
cilium connectivity test

# Hubble確認
kubectl -n kube-system get pods -l k8s-app=hubble-relay
kubectl -n kube-system get pods -l k8s-app=hubble-ui

# LoadBalancer IP確認
kubectl get svc -A | grep LoadBalancer
```

---

6. helmfile環境管理

6.1 helmfile.yaml

helmfile/helmfile.yaml:

yaml

```yaml
---
# 環境定義
environments:
  dev:
    values:
      - environments/dev.yaml
  staging:
    values:
      - environments/staging.yaml
  prod:
    values:
      - environments/prod.yaml

# リポジトリ
repositories:
  - name: cilium
    url: https://helm.cilium.io/
  - name: traefik
    url: https://traefik.github.io/charts
  - name: prometheus-community
    url: https://prometheus-community.github.io/helm-charts
  - name: grafana
    url: https://grafana.github.io/helm-charts
  - name: jaegertracing
    url: https://jaegertracing.github.io/helm-charts
  - name: vector
    url: https://helm.vector.dev
  - name: minio-operator
    url: https://operator.min.io/
  - name: argo
    url: https://argoproj.github.io/argo-helm
  - name: kubernetes-dashboard
    url: https://kubernetes.github.io/dashboard/

helmDefaults:
  atomic: true
  wait: true
  timeout: 600
  cleanupOnFail: true
  recreatePods: false

# リリース
releases:
  # 1. Cilium（既にAnsibleでインストール済みの場合はスキップ）
```

```yaml
# - name: cilium
#   namespace: kube-system
#   chart: cilium/cilium
#   version: 1.18.2
#   values:
#     - values/{{ .Environment.Name }}/cilium.yaml

# 2. Traefik
- name: traefik
  namespace: traefik
  chart: traefik/traefik
  version: 37.1.2
  createNamespace: true
  values:
    - values/{{ .Environment.Name }}/traefik.yaml

# 3. kube-prometheus-stack
- name: kube-prometheus-stack
  namespace: monitoring
  chart: prometheus-community/kube-prometheus-stack
  version: 78.3.0
  createNamespace: true
  values:
    - values/{{ .Environment.Name }}/prometheus.yaml
  needs:
    - traefik/traefik

# 4. Vector
- name: vector
  namespace: monitoring
  chart: vector/vector
  version: 0.46.0
  values:
    - values/{{ .Environment.Name }}/vector.yaml
  needs:
    - monitoring/kube-prometheus-stack

# 5. Jaeger Operator
- name: jaeger-operator
  namespace: observability
  chart: jaegertracing/jaeger-operator
  version: 2.57.0
  createNamespace: true
  values:
```

```
    - values/{{ .Environment.Name }}/jaeger.yaml

  # 6. MinIO Operator
  - name: minio-operator
    namespace: minio-operator
    chart: minio-operator/operator
    version: 7.1.1
    createNamespace: true
    set:
      - name: console.enabled
        value: "{{ .Values.minioConsoleEnabled }}"

  # 7. ArgoCD
  - name: argocd
    namespace: argocd
    chart: argo/argo-cd
    version: 8.6.2
    createNamespace: true
    values:
      - values/{{ .Environment.Name }}/argocd.yaml
    needs:
      - traefik/traefik

  # 8. Kubernetes Dashboard
  - name: kubernetes-dashboard
    namespace: kubernetes-dashboard
    chart: kubernetes-dashboard/kubernetes-dashboard
    version: 7.13.0
    createNamespace: true
    values:
      - values/{{ .Environment.Name }}/k8s-dashboard.yaml
    needs:
      - traefik/traefik
```

6.2 環境別設定

helmfile/environments/dev.yaml:

yaml

```yaml
# 開発環境設定
clusterName: dev-cluster
domain: dev.local
controlPlaneEndpoint: "192.168.1.100"

# レプリカ数（開発環境最小化）
traefikReplicas: 1
prometheusReplicas: 1
grafanaReplicas: 1

# ストレージ
prometheusRetention: 7d
prometheusStorageSize: 20Gi

# 機能フラグ
minioConsoleEnabled: true
enableServiceMonitor: true
enableIngress: true

# 認証
enableBasicAuth: false  # 開発環境では無効
```

helmfile/environments/prod.yaml:

yaml

```yaml
# 本番環境設定
clusterName: prod-cluster
domain: example.com
controlPlaneEndpoint: "10.0.1.100"

# レプリカ数（HA）
traefikReplicas: 3
prometheusReplicas: 2
grafanaReplicas: 2

# ストレージ
prometheusRetention: 30d
prometheusStorageSize: 100Gi

# 機能フラグ
minioConsoleEnabled: false
enableServiceMonitor: true
enableIngress: true

# 認証
enableBasicAuth: true
```

---

7. 全コンポーネント設定詳細

7.1 Traefik（TCP/UDP対応）

helmfile/values/dev/traefik.yaml:

yaml

```yaml
# Traefik 3.5開発環境設定

# デプロイメント
deployment:
 replicas: 1

# Providers
providers:
 kubernetesCRD:
  enabled: true
  allowCrossNamespace: true
 kubernetesIngress:
  enabled: true
  publishedService:
   enabled: true

# エントリーポイント
ports:
 web:
  port: 80
  protocol: TCP
  expose:
   default: true

 websecure:
  port: 443
  protocol: TCP
  expose:
   default: true
  tls:
   enabled: true

 # TCP/UDPポート（CoreDNS, Vector等）
 dns-tcp:
  port: 5353
  protocol: TCP
  expose:
   default: true

 dns-udp:
  port: 5353
  protocol: UDP
  expose:
   default: true
```

```
  vector-tcp:
    port: 6000
    protocol: TCP
    expose:
      default: true

# Service
service:
  type: LoadBalancer  # Cilium LB IPAMが割り当て
  annotations:
    io.cilium/lb-ipam-ips: "192.168.100.10"

# リソース
resources:
  requests:
    cpu: 100m
    memory: 128Mi
  limits:
    cpu: 1000m
    memory: 512Mi

# メトリクス
metrics:
  prometheus:
    enabled: true
    service:
      enabled: true
    serviceMonitor:
      enabled: true

# ログ
logs:
  general:
    level: INFO
  access:
    enabled: true
    format: json

# ダッシュボード（開発環境のみ）
ingressRoute:
  dashboard:
    enabled: true
```

```
matchRule: Host(`traefik.dev.local`)
entryPoints: ["web"]
```

7.2 kube-prometheus-stack

helmfile/values/dev/prometheus.yaml:

yaml

```yaml
# kube-prometheus-stack開発環境設定

# Prometheus
prometheus:
  enabled: true
  prometheusSpec:
    replicas: 1
    retention: 7d
    retentionSize: "15GB"

    # ストレージ
    storageSpec:
      volumeClaimTemplate:
        spec:
          accessModes: ["ReadWriteOnce"]
          resources:
            requests:
              storage: 20Gi

    # ServiceMonitor自動検出
    serviceMonitorSelectorNilUsesHelmValues: false
    podMonitorSelectorNilUsesHelmValues: false

    # リソース
    resources:
      requests:
        cpu: 200m
        memory: 1Gi
      limits:
        cpu: 2000m
        memory: 4Gi

# Grafana
grafana:
  enabled: true
  replicas: 1
  adminPassword: "admin"  # 開発環境

  # データソース
  additionalDataSources:
    - name: Prometheus
      type: prometheus
      uid: prometheus
      access: proxy
```

```
    url: http://kube-prometheus-stack-prometheus:9090
    isDefault: true
    jsonData:
      httpMethod: POST
      timeInterval: 30s

  - name: Jaeger
    type: jaeger
    uid: jaeger
    access: proxy
    url: http://jaeger-query.observability:16686

#ダッシュボード
dashboardProviders:
  dashboardproviders.yaml:
    apiVersion: 1
    providers:
      - name: 'default'
        orgId: 1
        folder: ''
        type: file
        disableDeletion: false
        editable: true
        options:
          path: /var/lib/grafana/dashboards/default

dashboards:
  default:
    node-exporter:
      gnetId: 1860
      revision: 31
      datasource: Prometheus
    kubernetes-cluster:
      gnetId: 7249
      revision: 1
      datasource: Prometheus
    kubernetes-pods:
      gnetId: 6417
      revision: 1
      datasource: Prometheus
    cilium-metrics:
      gnetId: 16611
      revision: 1
      datasource: Prometheus
```

```yaml
  # Ingress
  ingress:
    enabled: true
    ingressClassName: traefik
    hosts:
      - grafana.dev.local

# AlertManager
alertmanager:
  enabled: true

# node-exporter
nodeExporter:
  enabled: true

# kube-state-metrics
kubeStateMetrics:
  enabled: true

# kubelet
kubelet:
  enabled: true
  serviceMonitor:
    interval: 30s
```

7.3 Vector

helmfile/values/dev/vector.yaml:

yaml

```yaml
# Vector開発環境設定

role: Agent

# ConfigMap
customConfig:
 data_dir: /var/lib/vector

 sources:
  # Kubernetesログ
  kubernetes_logs:
   type: kubernetes_logs

  # ホストメトリクス
  host_metrics:
   type: host_metrics
   filesystem:
    devices:
     excludes: ["binfmt_misc"]

  # Vector内部メトリクス
  internal_metrics:
   type: internal_metrics

 transforms:
  # ラベル追加
  add_labels:
   type: remap
   inputs: ["host_metrics", "internal_metrics"]
   source: |
    .tags.environment = "dev"
    .tags.cluster = "dev-cluster"

 sinks:
  # Prometheus Remote Write
  prometheus:
   type: prometheus_remote_write
   inputs: ["add_labels"]
   endpoint: "http://kube-prometheus-stack-prometheus.monitoring:9090/api/v1/write"
   compression: snappy
   healthcheck:
    enabled: true

  # コンソール出力（開発）
```

```
    console:
      type: console
      inputs: ["kubernetes_logs"]
      encoding:
        codec: json

  # リソース
  resources:
    requests:
      cpu: 100m
      memory: 128Mi
    limits:
      cpu: 1000m
      memory: 512Mi
```

7.4 Jaeger

helmfile/values/dev/jaeger.yaml:

yaml

```yaml
# Jaeger Operator開発環境設定

jaeger:
  create: true
  spec:
    strategy: allInOne
    allInOne:
      image: jaegertracing/all-in-one:1.65.0
      options:
        memory:
          max-traces: 10000

    storage:
      type: memory

    ingress:
      enabled: true
      ingressClassName: traefik
      hosts:
        - jaeger.dev.local

    agent:
      strategy: DaemonSet
```

7.5 ArgoCD

helmfile/values/dev/argocd.yaml:

yaml

```yaml
# ArgoCD開発環境設定

global:
  domain: argocd.dev.local

server:
  replicas: 1

  # Traefik経由アクセス用
  extraArgs:
    - --insecure  # TLS終端はTraefikで行う

  ingress:
    enabled: true
    ingressClassName: traefik
    hosts:
      - argocd.dev.local

  metrics:
    enabled: true
    serviceMonitor:
      enabled: true

repoServer:
  replicas: 1

controller:
  replicas: 1
  metrics:
    enabled: true
    serviceMonitor:
      enabled: true

redis:
  enabled: true

# Dex（SSO）は開発環境で無効
dex:
  enabled: false

# 初期パスワード
configs:
```

```yaml
  secret:
    argocdServerAdminPassword: "$2a$10$..."  # bcrypt hash
```

7.6 Kubernetes Dashboard

helmfile/values/dev/k8s-dashboard.yaml:

yaml

```yaml
# Kubernetes Dashboard開発環境設定

app:
  ingress:
    enabled: true
    ingressClassName: traefik
    hosts:
      - k8s-dashboard.dev.local

    # HTTPSバックエンド用
    annotations:
      traefik.ingress.kubernetes.io/router.tls: "true"

# メトリクススクレイパー
metricsScraper:
  enabled: true

# リソース
resources:
  requests:
    cpu: 100m
    memory: 128Mi
  limits:
    cpu: 500m
    memory: 512Mi
```

7.7 MinIO

minio-tenant.yaml（手動作成）:

yaml

```yaml
apiVersion: v1
kind: Namespace
metadata:
  name: minio
---
apiVersion: v1
kind: Secret
metadata:
  name: storage-config
  namespace: minio
stringData:
  config.env: |
    export MINIO_ROOT_USER="admin"
    export MINIO_ROOT_PASSWORD="DevPassword123"
---
apiVersion: minio.min.io/v2
kind: Tenant
metadata:
  name: dev-storage
  namespace: minio
spec:
  image: quay.io/minio/minio:latest
  pools:
    - name: pool-0
      servers: 4
      volumesPerServer: 2
      volumeClaimTemplate:
        spec:
          accessModes:
            - ReadWriteOnce
          resources:
            requests:
              storage: 50Gi
  configuration:
    name: storage-config
  requestAutoCert: false
  buckets:
    - name: prometheus
    - name: loki
    - name: grafana
```

8. メトリクス可視化設定

8.1 主要メトリクス一覧

ノードメトリクス（node-exporter）

- `node_cpu_seconds_total` - CPU使用時間
- `node_memory_MemAvailable_bytes` - 利用可能メモリ
- `node_memory_MemTotal_bytes` - 総メモリ
- `node_filesystem_avail_bytes` - ディスク空き容量
- `node_network_receive_bytes_total` - ネットワーク受信
- `node_network_transmit_bytes_total` - ネットワーク送信

Podメトリクス（cAdvisor）

- `container_cpu_usage_seconds_total` - コンテナCPU
- `container_memory_working_set_bytes` - コンテナメモリ
- `container_network_receive_bytes_total` - コンテナRX
- `container_network_transmit_bytes_total` - コンテナTX

Ciliumメトリクス

- `cilium_datapath_conntrack_gc_runs_total` - Connection tracking
- `cilium_drop_count_total` - パケットドロップ
- `cilium_forward_count_total` - パケット転送
- `cilium_policy_l7_total` - L7ポリシー適用

8.2 Traefik IngressRoute設定

ingress-routes.yaml:

yaml

```yaml
---
# Grafana
apiVersion: traefik.io/v1alpha1
kind: IngressRoute
metadata:
  name: grafana
  namespace: monitoring
spec:
  entryPoints:
    - web
  routes:
    - kind: Rule
      match: Host(`grafana.dev.local`)
      services:
        - name: kube-prometheus-stack-grafana
          port: 80

---
# Prometheus
apiVersion: traefik.io/v1alpha1
kind: IngressRoute
metadata:
  name: prometheus
  namespace: monitoring
spec:
  entryPoints:
    - web
  routes:
    - kind: Rule
      match: Host(`prometheus.dev.local`)
      services:
        - name: kube-prometheus-stack-prometheus
          port: 9090

---
# Hubble UI
apiVersion: traefik.io/v1alpha1
kind: IngressRoute
metadata:
  name: hubble-ui
  namespace: kube-system
spec:
  entryPoints:
    - web
```

```yaml
  routes:
    - kind: Rule
      match: Host(`hubble.dev.local`)
      services:
        - name: hubble-ui
          port: 80

---
# ArgoCD UI
apiVersion: traefik.io/v1alpha1
kind: IngressRoute
metadata:
  name: argocd-server
  namespace: argocd
spec:
  entryPoints:
    - web
  routes:
    - kind: Rule
      match: Host(`argocd.dev.local`)
      services:
        - name: argocd-server
          port: 80

---
# Jaeger UI
apiVersion: traefik.io/v1alpha1
kind: IngressRoute
metadata:
  name: jaeger-query
  namespace: observability
spec:
  entryPoints:
    - web
  routes:
    - kind: Rule
      match: Host(`jaeger.dev.local`)
      services:
        - name: jaeger-query
          port: 16686

---
# Kubernetes Dashboard
apiVersion: traefik.io/v1alpha1
```

```
kind: IngressRoute
metadata:
  name: k8s-dashboard
  namespace: kubernetes-dashboard
spec:
  entryPoints:
    - web
  routes:
    - kind: Rule
      match: Host(`k8s-dashboard.dev.local`)
      services:
        - name: kubernetes-dashboard-kong-proxy
          port: 443
          scheme: https
          serversTransport: dashboard-transport

---
# HTTPS Backend用トランスポート
apiVersion: traefik.io/v1alpha1
kind: ServersTransport
metadata:
  name: dashboard-transport
  namespace: kubernetes-dashboard
spec:
  serverName: kubernetes-dashboard
  insecureSkipVerify: true
```

8.3 hosts設定（ローカル開発）

bash

```
# /etc/hostsに追加
192.168.100.10 grafana.dev.local
192.168.100.10 prometheus.dev.local
192.168.100.10 hubble.dev.local
192.168.100.10 argocd.dev.local
192.168.100.10 jaeger.dev.local
192.168.100.10 k8s-dashboard.dev.local
192.168.100.10 traefik.dev.local
```

---

9. デプロイ手順

9.1 完全デプロイフロー

bash

```
# ============================
# Step 1: Terraformでインフラ構築
# ============================
cd terraform/
terraform init
terraform apply -auto-approve

# ============================
# Step 2: Ansibleでクラスタ構築
# ============================
cd ../ansible/

# 全Playbook実行
ansible-playbook -i inventory/hosts.ini playbooks/01-prerequisites.yml
ansible-playbook -i inventory/hosts.ini playbooks/02-container-runtime.yml
ansible-playbook -i inventory/hosts.ini playbooks/03-kubernetes-install.yml
ansible-playbook -i inventory/hosts.ini playbooks/04-cluster-init.yml

# kubeconfigコピー
mkdir -p ~/.kube
cp kubeconfig ~/.kube/config
chmod 600 ~/.kube/config

# クラスタ確認
kubectl get nodes
kubectl get pods -A

# ============================
# Step 3: Cilium LB IPAMセットアップ
# ============================
kubectl apply -f cilium-lb-ippool.yaml

# Cilium確認
kubectl -n kube-system exec ds/cilium -- cilium status

# ============================
# Step 4: helmfileで全コンポーネントデプロイ
# ============================
cd ../helmfile/

# 開発環境デプロイ
helmfile -e dev sync

# 進捗確認
```

```
watch kubectl get pods -A

# ============================
# Step 5: MinIO Tenantデプロイ
# ============================
kubectl apply -f minio-tenant.yaml

# ============================
# Step 6: IngressRoute設定
# ============================
kubectl apply -f ingress-routes.yaml

# ============================
# Step 7: 確認
# ============================

# 全Pod確認
kubectl get pods -A

# Service確認（LoadBalancer IP）
kubectl get svc -A | grep LoadBalancer

# IngressRoute確認
kubectl get ingressroute -A

# メトリクス確認
kubectl top nodes
kubectl top pods -A

# UIアクセス
echo "Grafana: http://grafana.dev.local"
echo "Prometheus: http://prometheus.dev.local"
echo "Hubble: http://hubble.dev.local"
echo "ArgoCD: http://argocd.dev.local"
echo "Jaeger: http://jaeger.dev.local"
echo "K8s Dashboard: http://k8s-dashboard.dev.local"
```

9.2 デプロイ時間目安

フェーズ	所要時間
Terraform VM作成	5-10分
Ansible前提条件	5分
Container Runtime	3分
Kubernetesインストール	5分
クラスタ初期化 + Cilium	10分
helmfile全コンポーネント	15-20分
合計	**40-50分**

---

10. トラブルシューティング

10.1 Cilium関連

問題: ノードがNotReady

bash

```
# Cilium Pod確認
kubectl get pods -n kube-system -l k8s-app=cilium

# ログ確認
kubectl logs -n kube-system ds/cilium --tail=50

# Cilium状態
kubectl -n kube-system exec ds/cilium -- cilium status

# Connectivity Test
cilium connectivity test
```

問題: LoadBalancer IPが割り当たらない

bash

```
# IPプール確認
kubectl get ciliumloadbalancerippool

# L2アナウンス確認
kubectl get ciliuml2announcementpolicy

# Ciliumログでエラー確認
kubectl logs -n kube-system deploy/cilium-operator | grep -i "lb-ipam"
```

10.2 Prometheus関連

問題: メトリクスが表示されない

bash

```
# ServiceMonitor確認
kubectl get servicemonitor -A

# Prometheusターゲット確認（Port Forward）
kubectl port-forward -n monitoring svc/kube-prometheus-stack-prometheus 9090:9090
# http://localhost:9090/targets

# Prometheusログ
kubectl logs -n monitoring sts/prometheus-kube-prometheus-stack-prometheus
```

問題: Grafanaダッシュボードが空

bash

```
# データソース確認
kubectl get cm -n monitoring kube-prometheus-stack-grafana -o yaml | grep -A 10 datasources

# Grafanaログ
kubectl logs -n monitoring deploy/kube-prometheus-stack-grafana
```

10.3 Traefik関連

問題: IngressRouteが動作しない

bash

```
# IngressRoute確認
kubectl get ingressroute -A

# Traefikログ
kubectl logs -n traefik deploy/traefik

# Traefik設定ダンプ
kubectl port-forward -n traefik svc/traefik 9000:9000
# http://localhost:9000/dashboard/
```

10.4 ArgoCD関連

問題: 502 Bad Gateway

bash

```
# ArgoCD server.insecure設定確認
kubectl get cm argocd-cmd-params-cm -n argocd -o yaml | grep insecure

# 設定追加
kubectl patch cm argocd-cmd-params-cm -n argocd \
 --type merge \
 -p '{"data":{"server.insecure":"true"}}'

kubectl rollout restart deployment argocd-server -n argocd
```

10.5 一般的な問題

問題: Podが起動しない

bash

```
# Pod詳細
kubectl describe pod <pod-name> -n <namespace>

# イベント確認
kubectl get events -n <namespace> --sort-by='.lastTimestamp'

# ログ確認
kubectl logs <pod-name> -n <namespace> --previous
```

問題: ノードリソース不足

bash

```
# リソース使用状況
kubectl top nodes
kubectl describe node <node-name>

# Pod退避
kubectl drain <node-name> --ignore-daemonsets --delete-emptydir-data
```

10.6 完全リセット手順

bash

```
# クラスタリセット
ansible-playbook -i inventory/hosts.ini playbooks/reset-cluster.yml

# または手動
sudo kubeadm reset -f
sudo rm -rf /etc/kubernetes /var/lib/etcd /var/lib/kubelet /var/lib/cni
sudo iptables -F && sudo iptables -t nat -F && sudo iptables -t mangle -F && sudo iptables -X
```

---

11. 次のステップ

11.1 本番環境への移行

1. **Control Plane 3台構成**へ拡張（etcd quorum確保）
2. **外部ロードバランサー**追加（HAProxy/keepalived）

3. **cert-manager**導入（Let's Encrypt自動化）
4. **Velero**バックアップ戦略
5. **OPA Gatekeeper**ポリシー管理

11.2 GitOps完全移行

bash

```
# ArgoCD Application作成
kubectl apply -f - <<EOF
apiVersion: argoproj.io/v1alpha1
kind: Application
metadata:
  name: infrastructure
  namespace: argocd
spec:
  project: default
  source:
    repoURL: https://github.com/yourorg/k8s-infrastructure
    targetRevision: main
    path: helmfile
  destination:
    server: https://kubernetes.default.svc
    namespace: default
  syncPolicy:
    automated:
      prune: true
      selfHeal: true
EOF
```

11.3 監視強化

- **Thanos**導入（Prometheus長期保存）
- **Loki**導入（ログ集約）
- **Tempo**導入（トレース保存）
- **アラートルール**カスタマイズ

まとめ

本ガイドでは、Proxmox + Terraform + Ansible + helmfileを用いた**Kubernetes 1.33基盤の完全自動構築**を解説しました。

重要なポイント：

1. ✅ **Cilium kube-proxy完全置換**でeBPF高性能化
2. ✅ **Cilium LB IPAM**でMetalLB不要
3. ✅ **Cilium Gateway API**はHTTP/HTTPS/TLS/gRPC専用（TCP/UDPは未実装）
4. ✅ **Traefik**でTCP/UDPルーティング対応
5. ✅ **helmfile環境管理**で開発/本番環境切り替え
6. ✅ **ノード・Podメトリクス完全可視化**（Prometheus + Grafana）
7. ✅ **全UI統一アクセス**（Traefik IngressRoute）

開発環境構成：Control Plane 2台 + Worker 3台でリソース最適化済み。本番環境へはhelm valuesとenvironments YAMLの変更のみで移行可能です。

デプロイ時間：約40-50分で完全自動構築が完了します。

作成日: 2025年10月19日
対象バージョン: Kubernetes 1.33.5, Cilium 1.18.2
次回更新: Kubernetes 1.34正式対応、Cilium TCP/UDPRoute実装時