Architektur & Identity

eBPF-Datapath

CNI mit eBPF-Datapath: Networking, Load-Balancing, Policy und Observability laufen als eBPF-Programme im Kernel – mit kube-proxy-Replacement ohne iptables-Ketten pro Service.
Komponenten: **cilium-agent** (DaemonSet, je Node), **cilium-operator** (clusterweite Aufgaben, z.B. IPAM), Hubble (Observability), **cilium**-CLI. CNCF-graduated.

Identity statt IP

Jeder Endpoint bekommt eine clusterweite Security-Identity, abgeleitet aus den security-relevanten Pod-Labels. Endpoints mit identischem Label-Set teilen eine Identity.
Policy-Entscheidungen laufen gegen die Identity, nicht gegen fluechtige Pod-IPs – das skaliert unabhaengig von der Endpoint-Zahl.

```
cilium status --wait
kubectl get ciliumendpoints -A   # Identity je Pod
kubectl get ciliumidentities
```

Install & kube-proxy-Replacement

Helm-Install

Ohne kube-proxy muessen **k8sServiceHost** / **k8sServicePort** gesetzt sein – sonst findet der Agent den API-Server nicht (den Service loest sonst niemand auf).

```
helm repo add cilium https://helm.cilium.io/
helm install cilium cilium/cilium \
  --namespace kube-system \
  --set kubeProxyReplacement=true \
  --set k8sServiceHost=<api-server-ip> \
  --set k8sServicePort=6443
```

kube-proxy-Replacement

eBPF uebernimmt ClusterIP, NodePort, LoadBalancer, externalIPs und HostPort.
Socket-LB: Service-zu-Backend-Uebersetzung schon beim **connect()**/**sendmsg()**-Syscall – kein Paket-Rewriting auf dem Pfad.

```
kubectl -n kube-system exec ds/cilium -- \
  cilium-dbg status --verbose | \
  grep KubeProxyReplacement
```

CiliumNetworkPolicy (L3/L4)

Selektoren, Entities, CIDR

endpointSelector waehlt die Ziel-Endpoints; **fromEndpoints**/**toEndpoints** selektieren per Label (Identity).
toEntities: **world**, **cluster**, **host**, **kube-apiserver**.
toCIDR/**toCIDRSet**: fuer externe Ziele – Cluster-intern per Label selektieren.

```
apiVersion: cilium.io/v2
kind: CiliumNetworkPolicy
metadata: {name: backend-allow, namespace: shop}
spec:
  endpointSelector:
    matchLabels: {app: backend}
  ingress:
  - fromEndpoints:
    - matchLabels: {app: frontend}
    toPorts:
    - ports:
      - {port: "8080", protocol: TCP}
```

Clusterwide & Enforcement-Modi

CiliumClusterwideNetworkPolicy: gleiche Syntax, nicht namespaced – fuer Baselines ueber alle Namespaces.
policyEnforcementMode: default (alles offen, bis eine Regel den Endpoint selektiert – dann default-deny je Richtung), always, never.

L7-Policies & toFQDNs

HTTP-Rules (L7)

L7-Regeln haengen unter **toPorts.rules.http**: **method**, **path**, **host**, Header (Regex). Durchsetzung via node-lokalem Envoy; Verstoss liefert HTTP 403 statt Paket-Drop. Weitere L7-Typen: **dns**, **kafka** (beta).

```
spec:
  endpointSelector:
    matchLabels: {app: api}
  ingress:
  - fromEndpoints:
    - matchLabels: {app: frontend}
    toPorts:
    - ports:
      - {port: "80", protocol: TCP}
      rules:
        http:
        - method: GET
          path: /v1/.*
```

toFQDNs (Egress nach Namen)

matchName exakt, **matchPattern** mit Wildcard.
Pflicht-Voraussetzung: eine **rules.dns**-Regel schaltet den DNS-Proxy ein – ohne DNS-Sichtbarkeit kann Cilium FQDNs nicht auf IPs mappen und die Policy laeuft leer.

```
egress:
- toEndpoints:
  - matchLabels: {k8s-app: kube-dns}
  toPorts:
  - ports: [{port: "53", protocol: ANY}]
    rules:
      dns: [{matchPattern: "*"}]
- toFQDNs:
  - matchName: api.example.com
  - matchPattern: "*.storage.example.com"
  toPorts:
  - ports: [{port: "443", protocol: TCP}]
```

Hubble (Observability)

Flows, Relay, UI

Hubble liest Flows (L3/L4/L7) direkt aus dem eBPF-Datapath – ohne Sidecars oder Paket-Mirroring.
Relay aggregiert clusterweit (auch ueber ClusterMesh), UI zeigt die Service-Dependency-Map, die CLI (**hubble**) filtert Flows live.

```
cilium hubble enable --ui
cilium hubble port-forward &
hubble status
hubble observe --namespace shop \
  --verdict DROPPED
hubble observe --protocol http --last 20
```

Metriken

Hubble-Metriken fuer Prometheus per Helm: **hubble.metrics.enabled="{dns,drop,tcp,flow,icmp,httpV2}"**.
Agent/Operator-Metriken: **prometheus.enabled=true**, **operator.prometheus.enabled=true**.

Encryption

WireGuard

Transparente Verschluesselung des Pod-Traffics zwischen Nodes (Device **cilium_wg0**, UDP 51871 – Firewall freischalten!). Key-Paar erzeugt jeder Node selbst, Public-Key-Austausch via CiliumNode-Annotation.
Node-zu-Node zusaetzlich: **encryption.nodeEncryption=true**. Same-Node-Traffic bleibt unverschluesselt.

```
helm upgrade cilium cilium/cilium \
  -n kube-system --reuse-values \
  --set encryption.enabled=true \
  --set encryption.type=wireguard
kubectl -n kube-system exec ds/cilium -- \
  cilium-dbg status | grep Encryption
```

IPsec

encryption.type=ipsec; Key liegt im Secret **cilium-ipsec-keys** (kube-system), Algorithmus waehlbar (z.B. GCM-128-AES).
Key-Rotation ist manuell (KEYID 1–15 inkrementieren) – nie waehrend eines Upgrades rotieren.

ClusterMesh

Multi-Cluster-Verbund

Pod-Konnektivitaet ueber Cluster-Grenzen, globale Services mit Cross-Cluster-Load-Balancing, Policies ueber Cluster hinweg.
Voraussetzungen: PodCIDRs aller Cluster disjunkt, eindeutiger **cluster.name** + **cluster.id** (1–255), IP-Konnektivitaet zwischen allen Nodes, **clustermesh-apiserver** gegenseitig erreichbar.

```
cilium clustermesh enable --context c1
cilium clustermesh connect --context c1 \
  --destination-context c2
cilium clustermesh status --wait
```

Globale Services

Annotation **service.cilium.io/global: "true"** auf dem gleichnamigen Service in beiden Clustern – Cilium load-balanct auf die Backends beider Cluster.
service.cilium.io/shared: "false": Remote-Backends nutzen, eigene nicht teilen.

Gateway API

Eingebauter Controller

Cilium implementiert die Gateway API (CRDs v1.4.1 vorab installieren): GatewayClass, Gateway, HTTPRoute, GRPCRoute, ReferenceGrant, TLSRoute (experimentell). Voraussetzungen: `kubeProxyReplacement=true`, `l7Proxy=true` (Default), `gatewayAPI.enabled=true`. GatewayClass heisst `cilium`.

```
apiVersion: gateway.networking.k8s.io/v1
kind: Gateway
metadata: {name: web-gw, namespace: shop}
spec:
  gatewayClassName: cilium
  listeners:
  - {name: http, port: 80, protocol: HTTP}
---
apiVersion: gateway.networking.k8s.io/v1
kind: HTTPRoute
metadata: {name: web, namespace: shop}
spec:
  parentRefs: [{name: web-gw}]
  rules:
  - matches:
    - path: {type: PathPrefix, value: /}
    backendRefs: [{name: web, port: 80}]
```

Diagnose (cilium-CLI)

Werkzeuge

`cilium status --wait`: Health aller Komponenten.
`cilium connectivity test`: E2E-Testsuite im Cluster.
`cilium sysdump`: Support-Archiv fuer Bug-Reports.
Im Agent-Pod: `cilium-dbg` (`endpoint list`, `policy get`, `monitor`).

```
cilium connectivity test
cilium sysdump --output-filename dump
kubectl -n kube-system exec ds/cilium -- \
  cilium-dbg endpoint list
kubectl -n kube-system exec ds/cilium -- \
  cilium-dbg monitor --type drop
```

Anti-Patterns

Was du nicht tun solltest

toFQDNs ohne DNS-Rule: ohne `rules.dns`-Sichtbarkeit kein FQDN-zu-IP-Mapping – Policy blockt oder laeuft leer.
kube-proxy-frei ohne k8sServiceHost/Port: Agent erreicht den API-Server nicht, Cluster-Bootstrap haengt.
CIDR-Regeln fuer Cluster-Traffic: `toCIDR` ist fuer externe Ziele gedacht; Pod-IPs sind fluechtig – Labels/Identities selektieren.
Default-Modus falsch verstanden: ohne selektierende Regel ist alles offen; die erste Regel schaltet den Endpoint auf default-deny (je Richtung).
WireGuard ohne UDP 51871: Nodes muessen sich auf dem Port erreichen, sonst kein Tunnel.
IPsec-Keys waehrend des Upgrades rotieren: erst alle Nodes auf gleiche Cilium-Version, dann rotieren.
ClusterMesh mit ueberlappenden PodCIDRs oder doppelter `cluster.id`: Verbund kommt nicht sauber hoch – vorab planen.

cilium-cheatsheet.de

Cilium Cheatsheet von OMNI52

Weiterfuehren, nicht selbst neu erfinden

Cilium-Rollout & Policy-Haertung

OMNI52™ hilft Plattform-Teams, Cilium fuer regulierte Enterprise-Umgebungen sauber aufzustellen.

Datapath- und IPAM-Design, Migration auf kube-proxy-Replacement ohne Traffic-Riss, Policy-Baseline von L3 bis L7 mit Hubble-Nachweis, transparente Verschluesselung (WireGuard/IPsec) und ClusterMesh-Topologien fuer Multi-Cluster-Betrieb. Output landet als GitOps-Repo bei euch im Cluster: Helm-Charts, Runbooks, Diagnose-Skripte. Euer Team operiert weiter, nicht wir.

OMNI52 GmbH, Ebern · istio-consulting.de · kostenloses 30-min Initial-Assessment

Aus der OMNI52 Cheatsheet-Fabrik

Verwandte Sheets: service-mesh-cheatsheet.de (Service Mesh im Vergleich) · istio-cheatsheet.de (Istio in der Tiefe) · kubernetes-cheatsheet.de (Kubernetes Core) · rke2-cheatsheet.de (RKE2 (CNI-Wahl)).