OMNI52

Istio Service Mesh

v1.30 – 2026.07

Architektur

Datapath & Komponenten

Client → Ingress GW → Sidecar(src) → Sidecar(dst) → App. Hop ≈ 0,5–2 ms p99. mTLS-Handshake nur beim Connect.
istiod: Pilot/Citadel/Galley vereint, xDS-Push + CA.
istio-proxy: Envoy-Sidecar (Mutating Webhook) oder Ambient ztunnel.
Gateway: dedizierte Envoys, Deployment + LB-Service.
Envoy-Version = Istio-Version + 8. Istio 1.30 → Envoy 1.38.

Install-Profile

default (prod), **minimal** (nur istiod), **ambient** (sidecar-less), **remote** (MC-Workload-Cluster), **demo** (laut). Canary via **--revision=1-30-1**.

```
istioctl install --set profile=default \
  --set values.global.proxy.resources.requests.cpu=100m
istioctl verify-install
```

Traffic Management

Gateway

Bindet Ports/Hosts/TLS am Edge-Proxy. **servers[].port** + **tls.mode**: **SIMPLE**, **MUTUAL**, **PASSTHROUGH**, **ISTIO_MUTUAL**. **credentialName** verweist auf Secret im selben Namespace wie der Gateway-Pod.

```
apiVersion: networking.istio.io/v1
kind: Gateway
metadata: {name: web-gw, namespace: istio-system}
spec:
  selector: {istio: ingressgateway}
  servers:
  - port: {number: 443, name: https, protocol: HTTPS}
    hosts: ["www.istio-cheatsheet.de"]
    tls:
      mode: SIMPLE
      credentialName: web-cert
```

VirtualService

Routing-Regeln. Bindet **hosts** an **gateways** (oder **mesh** für Ost-West). Match-Reihenfolge ist signifikant — erste Treffer-Regel gewinnt. Header-Match case-insensitive, Pfad-Match case-sensitive.

```
apiVersion: networking.istio.io/v1
kind: VirtualService
metadata: {name: reviews}
spec:
  hosts: [reviews]
  http:
  - match:
    - headers: {end-user: {exact: jason}}
    route:
    - destination: {host: reviews, subset: v2}
  - route:
    - destination: {host: reviews, subset: v1}
      weight: 90
    - destination: {host: reviews, subset: v3}
      weight: 10
    retries:
      attempts: 3
      perTryTimeout: 2s
      retryOn: gateway-error,connect-failure,refused-stream
    timeout: 10s
```

DestinationRule

Subsets (Versionen) + Traffic-Policy (LB, Connection-Pool, Outlier Detection, TLS). Wirkt erst nach VirtualService-Routing. **host** muss FQDN oder Short-Name im selben Namespace sein.

```
apiVersion: networking.istio.io/v1
kind: DestinationRule
metadata: {name: reviews}
spec:
  host: reviews
  trafficPolicy:
    connectionPool:
      tcp: {maxConnections: 100}
      http:
        http1MaxPendingRequests: 64
        http2MaxRequests: 1000
        maxRequestsPerConnection: 10
    outlierDetection:
      consecutive5xxErrors: 5
      interval: 30s
      baseEjectionTime: 60s
    loadBalancer:
      consistentHash:
        httpHeaderName: x-user-id
  subsets:
  - name: v1
    labels: {version: v1}
  - name: v2
    labels: {version: v2}
```

ServiceEntry

Externe Hosts in den Mesh-Registry holen (DB, SaaS, REST-APIs). **MESH_EXTERNAL** + **DNS** resolution sind die saubere Default-Kombi. Ohne ServiceEntry bleibt der Egress-Traffic **BlackHoleCluster** wenn **outboundTrafficPolicy=REGISTRY_ONLY**.

```
apiVersion: networking.istio.io/v1
kind: ServiceEntry
metadata: {name: stripe-api}
spec:
  hosts: ["api.stripe.com"]
  ports:
  - {number: 443, name: https, protocol: HTTPS}
  resolution: DNS
  location: MESH_EXTERNAL
```

Sidecar

Begrenzt was ein Workload aus der Mesh-Registry sieht — kritisch für Memory-Footprint und Push-Latenz in großen Clustern. Default: jeder Sidecar bekommt Config für alle Services. Auf > 200 Services pflicht!

```
apiVersion: networking.istio.io/v1
kind: Sidecar
metadata: {name: default, namespace: prod}
spec:
  egress:
  - hosts:
    - "./*"              # nur eigener Namespace
    - "istio-system/*"
    - "shared/*"
```

Security

PeerAuthentication

mTLS-Modus zwischen Sidecars. **STRICT**, **PERMISSIVE**, **DISABLE**. Geltung: Mesh (in **istio-system**), Namespace, oder Workload via **selector**. Migration zu STRICT: zuerst Mesh-weit **PERMISSIVE**, dann namespace-weise umstellen.

```
apiVersion: security.istio.io/v1
kind: PeerAuthentication
metadata: {name: default, namespace: prod}
spec:
  mtls: {mode: STRICT}
---
# Port-Ausnahme: Health-Check vom NLB ohne mTLS
spec:
  selector: {matchLabels: {app: legacy}}
  mtls: {mode: STRICT}
  portLevelMtls:
    8080: {mode: PERMISSIVE}
```

AuthorizationPolicy

L7-Zugriffskontrolle. **action**: **ALLOW** (default), **DENY** (precedence), **AUDIT**, **CUSTOM** (ext authz). Leere **rules**: blockiert/erlaubt alles (je nach action). Mehrere Policies kombinieren AND innerhalb, OR über Policies.

```
apiVersion: security.istio.io/v1
kind: AuthorizationPolicy
metadata: {name: reviews-allow, namespace: prod}
spec:
  selector: {matchLabels: {app: reviews}}
  action: ALLOW
  rules:
  - from:
    - source:
        principals:
        - cluster.local/ns/prod/sa/productpage
    to:
    - operation:
        methods: [GET]
        paths: ["/reviews/*"]
    when:
    - key: request.auth.claims[groups]
      values: ["reader", "admin"]
```

RequestAuthentication (JWT)

Validiert JWT, füllt **request.auth.*** für AuthorizationPolicy. Wichtig: ohne zusätzliche DENY-Policy mit **notRequestPrincipals=["*"]** können unauthentifizierte Requests weiterhin durch.

```
apiVersion: security.istio.io/v1
kind: RequestAuthentication
metadata: {name: jwt, namespace: prod}
spec:
  selector: {matchLabels: {app: api}}
  jwtRules:
  - issuer: "https://auth.example.com"
    jwksUri: "https://auth.example.com/jwks"
    audiences: ["api.example.com"]
    forwardOriginalToken: true
```

Identity

SPIFFE-URI **spiffe://<trust>/ns/<ns>/sa/<sa>**. Trust-Domain default **cluster.local**, in Multi-Cluster pro Cluster eindeutig. Workload-Cert-Rotation 24 h, automatisch via Citadel.

OMNI52 Istio Service Mesh

Observability

Telemetry-API

Metriken/Logs/Traces pro Workload oder Namespace. Ersetzt `EnvoyFilter`-Telemetry-Mods und die alten `values.telemetry.v2.*`-Settings.

```
apiVersion: telemetry.istio.io/v1
kind: Telemetry
metadata: {name: trace-prod, namespace: prod}
spec:
  tracing:
  - providers: [{name: tempo}]
    randomSamplingPercentage: 5.0
  metrics:
  - providers: [{name: prometheus}]
    overrides:
    - match: {metric: REQUEST_COUNT}
      tagOverrides:
        request_protocol: {operation: REMOVE}
  accessLogging:
  - providers: [{name: otel}]
    filter:
      expression: "response.code ≥ 400"
```

Metriken & Tracing

RED-Counter: `istio_requests_total`, Latency: `istio_request_duration_milliseconds_bucket`. Label `reporter` = source/destination — in Dashboards immer auf destination aggregieren (sonst Doppelzaehlung).

Istio propagiert B3/W3C, generiert sie nicht. App muss eingehende Trace-Header (`x-request-id`, `x-b3-*`, `traceparent`) beim Ausgang erneut setzen.

Performance & Tuning

Sidecar-Sizing

Default 100m CPU / 128Mi RAM: nicht prod-tauglich bei hohem RPS.
Faustregel: 1 mCore pro 100 RPS, +50 MiB pro 1000 Endpoints im Registry.
`Sidecar`-Resource: schneidet Registry → -70 % Memory typisch.

Pilot-Tuning

`PILOT_PUSH_THROTTLE`: default 100
`PILOT_DEBOUNCE_AFTER`: default 100 ms
`PILOT_DEBOUNCE_MAX`: default 10 s

Bei Push-Storms (viele Pod-Restarts) Debounce hochziehen, Throttle hoch für schnellere Konvergenz auf > 5000 Workloads.

Ambient Mode (1.30 GA)

Kein Sidecar-Inject mehr.
`ztunnel`: Node-DaemonSet, L4 mTLS.
`Waypoint Proxy`: optional, Namespace-scoped, L7.
Opt-in: Label `istio.io/dataplane-mode=ambient` pro Namespace.
Spart RAM bei vielen Pods, kostet Komplexität beim Debugging.

Jobs & CronJobs (Sidecar-Lifecycle)

App vor proxy ready → Connect-Refused. App fertig, Sidecar läuft → Job hängt.

Native Sidecar (K8s 1.29+ Beta, Istio 1.19+): proxy als `initContainer` (`restartPolicy: Always`), proper Lifecycle. Mesh-weit via istiod-Env `ENABLE_NATIVE_SIDECARS=true`.

Per-Pod (Istio 1.24+, Vorrang vor Mesh-Flag): Annotation `sidecar.istio.io/nativeSidecar: "true"` im Pod-Template (`"false"` erzwingt klassischen Sidecar).

Pre-Native: `holdApplicationUntilProxyStarts` gegen Start-Race, `trap` mit `POST :15020/quitquitquit` auf EXIT gegen Shutdown-Hänger (feuert auch bei Crash/Signal, nicht nur bei Success).

```
# Native per-Pod (Istio 1.24+, schlaegt Mesh-Flag):
    Pod-Template
metadata:
  annotations:
    sidecar.istio.io/nativeSidecar: "true"
---
# Pre-Native-Fallback (ohne Native Sidecars):
metadata:
  annotations:
    proxy.istio.io/config: |                # gegen
     Start-Race
      { "holdApplicationUntilProxyStarts": true }
spec:
  containers:
  - name: worker
    command: ["/bin/sh","-c"]
    args:
    - |
      trap 'curl -fsS -XPOST
     localhost:15020/quitquitquit||true' EXIT
      ./run-task
```

Graceful Drain: `EXIT_ON_ZERO_ACTIVE_CONNECTIONS`

Bei SIGTERM drainiert der Sidecar nur `terminationDrainDuration` (Default 5 s), dann Hard-Exit — lang lebende Verbindungen (gRPC-Streams, WebSockets, DB-Pools) werden mittendrin gekappt (503/Reset beim Rollout/Scale-Down).

Fix: `EXIT_ON_ZERO_ACTIVE_CONNECTIONS=true` (via `proxyMetadata` bzw. `proxy.istio.io/config`) — pilot-agent pollt aktive Envoy-Connections und beendet den Proxy sobald sie 0 erreichen statt nach fixem Timer.

Caveat: hängt eine Verbindung (Client schließt nie), blockt der Proxy bis `terminationGracePeriodSeconds` → SIGKILL. Grace-Period entsprechend hochsetzen. Mesh-weit via `meshConfig.defaultConfig.proxyMetadata`.

Multi-Cluster

Topologien & Setup

Primary-Remote: ein istiod, mehrere Workload-Cluster.
Multi-Primary: istiod pro Cluster, gemeinsame Root-CA.
External Control-Plane: istiod außerhalb.

Pflicht: gemeinsame `trustDomain` (oder `trustDomainAliases`), `network`-Label pro Cluster, Endpoint-Discovery via `istioctl create-remote-secret`.

```
istioctl create-remote-secret \
  --context=cluster-b \
  --name=cluster-b \
  | kubectl apply --context=cluster-a -f -
```

Diagnose

Erstes `istioctl`-Werkzeug bei Vorfall

`istioctl proxy-status` zeigt Sync-Stand jedes Sidecars. `SYNCED` = Config aktuell, `STALE` = Push läuft, `NOT SENT` = istiod hat nichts geschickt → Pilot-Logs prüfen.

```
istioctl proxy-status
istioctl proxy-config routes <pod>.<ns> -o json
istioctl proxy-config clusters <pod>.<ns>
istioctl proxy-config listeners <pod>.<ns>
istioctl proxy-config endpoints <pod>.<ns>
istioctl proxy-config secrets <pod>.<ns>

# Statisches Lint: VirtualService/DR-Konflikte etc.
istioctl analyze -n prod

# Vollständiger Bug-Report inkl. Konfig + Logs (anonymisiert)
istioctl bug-report
```

Live-Log eines Sidecars

`istioctl proxy-config log <pod> --level debug` setzt das Log-Level live ohne Pod-Restart. Komponenten-spezifisch z.B. `--level rbac:debug,jwt:debug`. Nach Diagnose zurueck auf `warning`.

Häufige Fehlerbilder

503 UC: Upstream-Connect-Fail, App-Port stimmt nicht mit Service-Target überein.
503 NR: No Route, VirtualService greift nicht (Host-Match falsch).
503 UF: Upstream-Failure, TLS-Mismatch (PeerAuth `STRICT` vs. Client ohne Sidecar).
404: Path-Match nicht getroffen, Reihenfolge der `http[]` prüfen.

`OUTPUT_CERTS`-Scraping: `read: connection reset by peer`

App-originated mTLS (Prometheus/Alloy-Federate via `ISTIO_META_OUTPUT_CERTS` + `/etc/istio-certs/{cert-chain,key,root-cert}.pem`) bricht mit RST, obwohl Ziel-`PeerAuthentication` `PERMISSIVE` ist und Allow-all greift.

Ursache fast immer: Client-Sidecar wrapt den App-mTLS in einen zweiten `ISTIO_MUTUAL`-Tunnel. Eine DR mit `host: "*.local"` + `exportTo: ["*"]` (typisch STRICT- Migration-Helper in `istio-system`) matcht via EndpointSlice-Lookup auch direkte Pod-IP-Calls. Ergebnis: TLS-in-TLS. Ziel-`15006` terminiert nur outer; inner TLS-Bytes landen als Klartext auf dem App-Port, Backend erwartet HTTP, schliesst.

PERMISSIVE ist Inbound-Entscheidung nach der Transport-Terminierung – heilt nichts, was outbound passiert.

Fix: `excludeOutboundPorts` am Scraper-Pod (port-scoped) oder eigene DR mit `tls.mode: DISABLE` auf den Ziel-Service (spezifischer als `*.local`, gewinnt).

```
# wraps der Client-Sidecar? alpn=istio* + sni=outbound_ =
    Beweis:
istioctl pc cluster <pod>.<ns> --fqdn <ziel> -o json \
  | jq
    '.[].transport_socket.typed_config|{sni,alpn_protocols}'
# welche *.local-DR greift?
kubectl get dr -A -o json | jq -r '.items[]|
  select(.spec.host|test("\\*\\.local"))|
  .metadata.namespace+"/"+.metadata.name'
```

Gateway API (v1, kubernetes-sigs)

Status in 1.30

Istio implementiert Gateway-API v1 (`Gateway`, `HTTPRoute`, `GRPCRoute`) als gleichberechtigte Alternative zu `Gateway`/`VirtualService`. Neue Projekte: Gateway-API. Bestand: `networking.istio.io` bleibt supportet, beide können koexistieren.

```yaml
apiVersion: gateway.networking.k8s.io/v1
kind: Gateway
metadata: {name: web, namespace: istio-system}
spec:
  gatewayClassName: istio
  listeners:
  - name: https
    hostname: www.istio-cheatsheet.de
    port: 443
    protocol: HTTPS
    tls:
      certificateRefs:
      - {name: web-cert}
---
apiVersion: gateway.networking.k8s.io/v1
kind: HTTPRoute
metadata: {name: site, namespace: web}
spec:
  parentRefs: [{name: web, namespace: istio-system}]
  hostnames: ["www.istio-cheatsheet.de"]
  rules:
  - matches: [{path: {type: PathPrefix, value: /}}]
    backendRefs: [{name: nginx, port: 80}]
```

Anti-Patterns

Was du nicht tun solltest

Default-Sidecar-Resources in Prod: OOM, Push-Throttle.
Mesh-VirtualService ohne `Sidecar`-Resource: jeder Sidecar bekommt jede Regel, Push-Storm.
`STRICT` ohne Migration: nicht injizierte Workloads brechen (Jobs, externe Health-Checks).
EnvoyFilter als Default-Tool: erst Telemetry-API, WasmPlugin, AuthorizationPolicy.
EnvoyFilter bricht zwischen Minors.
Tracing ohne App-Header-Propagation: Spans zerreißen.
Multi-Cluster ohne gemeinsame Root-CA: mTLS scheitert.
`OUTPUT_CERTS` ohne `excludeOutboundPorts`: Client-Sidecar wrapt App-mTLS doppelt (TLS-in-TLS) – `read: connection reset by peer` trotz PERMISSIVE PA.

Aus der OMNI52 Cheatsheet-Fabrik

Verwandte Sheets: service-mesh-cheatsheet.de (Istio + Linkerd + Cilium) · kubernetes-cheatsheet.de (Kubernetes Core) · k8s-cheatsheet.de (Kubernetes-Kurzref) · istio-spickzettel.de (Istio-Spickzettel, DE) · istio-quickref.de (English quick reference).

---

istio-cheatsheet.de

Service Mesh Cheatsheet von OMNI52

Weiterführen, nicht selbst neu erfinden

Workshops & Architektur-Reviews

OMNI52™ sichert Kubernetes-Cluster für regulierte Enterprise-Umgebungen durch ein automatisiertes Service Mesh auf Basis von Istio.

Architektur-Review, STRICT-mTLS-Migration, AuthorizationPolicy-Härtung, Telemetry-API-Pipeline. Output landet als GitOps-Repo bei euch im Cluster: Helm-Charts, Runbooks, Diagnose-Skripte. Euer Team operiert weiter, nicht wir.

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