Procedura Gateway API Envoy Helm v3 PRODUZIONE

Ultimo aggiornamento: 2026-08-18

Gateway API con Envoy Gateway Procedura Helm sicura e rollback

Versione 3 - procedura corretta e validata per replica in produzione (Helm 3.19.1)

Ambiente di riferimento: Kubernetes v1.33.8, RHEL 9.6, containerd, Flannel, MetalLB, ingress-nginx già operativo e Kubernetes Dashboard con Kong dedicato. Helm installato: v3.19.1. Obiettivo: introdurre Gateway API ed Envoy Gateway in parallelo, senza modificare gli Ingress esistenti né riavviare i workload applicativi.

1. Cosa installiamo e perché

Architettura di coesistenza:

Client/LAN | +--> IP MetalLB esistente --> ingress-nginx --> Ingress esistenti --> Service/Pod | +--> nuovo IP MetalLB -----> Envoy Proxy ----> HTTPRoute di test --> Service/Pod test

Envoy Gateway = control plane del secondo ramo. Nessuna migrazione degli Ingress esistenti è richiesta per il test.

2. Versioni fissate

La procedura usa Envoy Gateway v1.8.3 e Gateway API v1.5.1 Standard. La matrice ufficiale Envoy Gateway v1.8 indica supporto per Kubernetes v1.32-v1.35 e Gateway API v1.5.1; Kubernetes v1.33.8 è quindi nell'intervallo supportato.

- Kubernetes: v1.33.8
- Helm: v3.19.1
- Envoy Gateway: v1.8.3
- Gateway API: v1.5.1 - channel Standard

3. Stato iniziale richiesto

Dopo il rollback della precedente Fase 1, verificare che non restino Gateway API o Envoy:

kubectl get crd | grep gateway.networking.k8s.io kubectl get ns envoy-gateway-system kubectl get pods -A | grep -i envoy helm list -A | grep -E '^eg[[:space:]]'

Atteso: nessuna CRD gateway.networking.k8s.io; namespace envoy-gateway-system non presente; nessun pod Envoy; nessuna release Helm 'eg'.

Fotografare inoltre lo stato corrente:

mkdir -p /root/gateway-api-precheck kubectl get nodes -o wide > /root/gateway-api-precheck/nodes.txt kubectl get pods -A -o wide > /root/gateway-api-precheck/pods.txt kubectl get svc -A -o wide > /root/gateway-api-precheck/services.txt kubectl get ingress -A -o yaml > /root/gateway-api-precheck/ingress.yaml kubectl get ingressclass -o yaml > /root/gateway-api-precheck/ingressclass.yaml kubectl get ipaddresspools,l2advertisements -A -o

yaml > /root/gateway-api-precheck/metallb.yaml

4. Fase 1 - render e dry-run delle CRD Standard

Invece di installare prima Gateway API da un bundle e poi applicare install.yaml, generiamo con Helm un unico set coerente di CRD: Gateway API Standard + Envoy Gateway CRD.

4.1 Verificare che Helm possa leggere il chart OCI:

helm show chart oci://docker.io/envoyproxy/gateway-crds-helm --version v1.8.3

4.2 Dry-run server-side delle sole CRD:

helm template eg-crds oci://docker.io/envoyproxy/gateway-crds-helm
--version v1.8.3
--set crds.gatewayAPI.enabled=true
--set crds.gatewayAPI.channel=standard
--set crds.envoyGateway.enabled=true
| kubectl apply --server-side --dry-run=server -f -

Se compare qualunque errore relativo a safe-upgrades, channel, CRD già esistenti o conflitti, fermarsi e non eseguire la fase 5.

5. Fase 2 - installazione effettiva delle CRD

helm template eg-crds oci://docker.io/envoyproxy/gateway-crds-helm
--version v1.8.3
--set crds.gatewayAPI.enabled=true
--set crds.gatewayAPI.channel=standard
--set crds.envoyGateway.enabled=true
| kubectl apply --server-side -f -

Verificare subito:

kubectl get crd | grep gateway.networking.k8s.io kubectl get crd | grep gateway.envoyproxy.io

kubectl get crd gateways.gateway.networking.k8s.io
-o go-template='version={{ index .metadata.annotations
"gateway.networking.k8s.io/bundle-version" }} channel={{ index .metadata.annotations
"gateway.networking.k8s.io/channel" }}{{ "\n" }}'

kubectl get pods -A | grep -v Running | grep -v Completed

Atteso per l'annotazione: versione v1.5.1 e channel standard.

6. Fase 3 - installazione Envoy Gateway con Helm

Le CRD sono già state installate nella fase precedente. Per evitare che il chart principale tenti di gestirle di nuovo, si usa crds.enabled=false.

6.1 Dry-run Helm:

helm install eg oci://docker.io/envoyproxy/gateway-helm
--version v1.8.3
-n envoy-gateway-system
--create-namespace
--set crds.enabled=false

--dry-run

6.2 Se il dry-run è pulito, installazione:

helm install eg oci://docker.io/envoyproxy/gateway-helm
--version v1.8.3
-n envoy-gateway-system
--create-namespace
--set crds.enabled=false

6.3 Attendere che il control plane sia disponibile:

kubectl wait --timeout=5m -n envoy-gateway-system
deployment/envoy-gateway --for=condition=Available

6.4 Controlli:

helm list -n envoy-gateway-system kubectl get pods -n envoy-gateway-system -o wide kubectl get all -n envoy-gateway-system kubectl get gatewayclass kubectl get pods -A | grep -v Running | grep -v Completed

7. Rollback della sola installazione Envoy Gateway

Se la Fase 3 crea problemi, la prima azione è rimuovere la release Helm. Le CRD restano installate e quindi il rollback è meno distruttivo.

helm uninstall eg -n envoy-gateway-system

kubectl get pods -n envoy-gateway-system kubectl get pods -A | grep -v Running | grep -v Completed

Se il namespace rimane vuoto e non serve più, può essere rimosso solo dopo aver verificato che non contenga risorse utili:

kubectl get all -n envoy-gateway-system kubectl delete namespace envoy-gateway-system

8. Rollback completo delle CRD

Verifica preventiva:

kubectl get gatewayclass kubectl get gateway -A kubectl get httproute -A kubectl api-resources --api-group=gateway.envoyproxy.io

Se non esistono risorse da conservare e la release Helm è già stata disinstallata, rimuovere esattamente il set di CRD renderizzato:

helm template eg-crds oci://docker.io/envoyproxy/gateway-crds-helm
--version v1.8.3
--set crds.gatewayAPI.enabled=true
--set crds.gatewayAPI.channel=standard
--set crds.envoyGateway.enabled=true
| kubectl delete -f -

Controllo finale:

kubectl get crd | grep gateway.networking.k8s.io kubectl get crd | grep gateway.envoyproxy.io kubectl get pods -A | grep -v Running | grep -v Completed

9. Fase 4 - GatewayClass e test isolato (solo dopo il checkpoint)

Questa fase va eseguita solo dopo che Envoy Gateway è stabile. Prima del Gateway deve esistere una GatewayClass gestita da Envoy Gateway. Il test usa un namespace dedicato e non modifica Service, Ingress o pod applicativi esistenti.

kubectl create namespace gateway-test

kubectl -n gateway-test create deployment echo --image=registry.k8s.io/echoserver:1.10

kubectl -n gateway-test expose deployment echo --port=80 --target-port=8080 --name=echo

Salvare come gateway-test.yaml (include obbligatoriamente GatewayClass, Gateway e HTTPRoute):

apiVersion: gateway.networking.k8s.io/v1 kind: GatewayClass metadata: name: eg spec: controllerName: gateway.envoyproxy.io/gatewayclass-controller

apiVersion: gateway.networking.k8s.io/v1 kind: Gateway metadata: name: test-gateway namespace: gateway-test spec: gatewayClassName: eg listeners:

- name: http protocol: HTTP port: 80 allowedRoutes: namespaces: from: Same

apiVersion: gateway.networking.k8s.io/v1 kind: HTTPRoute metadata: name: echo namespace: gateway-test spec: parentRefs:

- name: test-gateway rules:
- backendRefs: name: echo port: 80

kubectl apply --dry-run=server -f gateway-test.yaml kubectl apply -f gateway-test.yaml

kubectl get gatewayclass eg kubectl wait --timeout=2m gatewayclass/eg --for=condition=Accepted

kubectl get gateway -n gateway-test -o wide kubectl wait --timeout=5m -n gateway-test gateway/test-gateway --for=condition=Programmed

kubectl get httproute -n gateway-test kubectl describe gatewayclass eg kubectl describe gateway -n gateway-test test-gateway kubectl describe httproute -n gateway-test echo

kubectl get svc -A -o wide | grep -E 'envoy|LoadBalancer' kubectl get pods -A | grep -v Running | grep -v Completed

Criteri GO/NO-GO: GatewayClass eg deve risultare Accepted=True; test-gateway deve risultare Programmed=True; HTTPRoute echo deve avere Accepted=True e ResolvedRefs=True. Solo dopo questi controlli verificare il Service LoadBalancer creato per Envoy e l'IP assegnato da MetalLB. L'IP deve essere libero e distinto dall'External IP di ingress-nginx.

10. Rollback del test

kubectl delete -f gateway-test.yaml --ignore-not-found=true kubectl delete namespace gateway-test --ignore-not-found=true

kubectl get gatewayclass kubectl get gateway -A kubectl get httproute -A kubectl get svc -A -o wide | grep -E 'envoy|LoadBalancer' kubectl get pods -A | grep -v Running | grep -v Completed

11. Sequenza raccomandata di esecuzione

1 Pre-check e fotografia del cluster.

2 helm show chart del chart CRD.

3 Dry-run server-side delle CRD Standard + Envoy Gateway CRD.

4 Apply effettivo delle CRD.

5 Verifica channel=standard e stato di tutti i pod.

6 helm install --dry-run del control plane con crds.enabled=false.

7 Installazione Helm reale.

8 Wait e verifica del solo control plane.

9 Solo in seguito: GatewayClass eg, namespace e Gateway/HTTPRoute di test.

10 Verifica Accepted/Programmed, quindi IP MetalLB distinto e test HTTP.

12. Diagnostica rapida

kubectl get nodes kubectl get pods -A -o wide kubectl get events -A --sort-by=.lastTimestamp helm list -A helm status eg -n envoy-gateway-system kubectl get gatewayclass kubectl get gateway -A kubectl get httproute -A kubectl get svc -A -o wide

13. Riferimenti ufficiali verificati

- Envoy Gateway v1.8 - Install with Helm: https://gateway.envoyproxy.io/v1.8/install/install-helm/
- Envoy Gateway compatibility matrix: https://gateway.envoyproxy.io/news/releases/matrix/
- Envoy Gateway CRD Helm chart values: https://gateway.envoyproxy.io/v1.8/install/gateway-crds-helm-api/
- Envoy Gateway v1.8 quickstart: https://gateway.envoyproxy.io/v1.8/tasks/quickstart/

| Principio operativo: Un cambiamento per volta, seguito da verifica. In caso di anomalia non si procede. La procedura evita il manifest monolitico install.yaml che nel test precedente ha tentato di sovrapporre CRD Experimental a CRD Standard. |

| --- |

| Componente | Funzione | Rapporto con il cluster esistente |

| --- | --- | --- |

| Gateway API CRD - Standard | Aggiungono GatewayClass, Gateway, HTTPRoute e API correlate. | Non sostituiscono Ingress e non riavviano pod. |

| Envoy Gateway CRD | Aggiungono le API specifiche di Envoy Gateway. | Usate solo dal nuovo controller. |

| Envoy Gateway | Control plane che osserva Gateway API e configura Envoy Proxy. | Installato in envoy-gateway-system. |

| Envoy Proxy | Data plane che riceve il traffico dei Gateway creati. | Creato successivamente da un Gateway; separato da ingress-nginx. |

| MetalLB | Assegna un External IP al Service LoadBalancer del nuovo data plane. | Già presente; deve assegnare un IP distinto. |

| ingress-nginx | Continua a servire gli Ingress esistenti. | Non viene modificato durante questa procedura. |

| Importante: Il chart CRD di Envoy Gateway ha il channel Gateway API Experimental come valore predefinito. Per questo ambiente il channel viene impostato esplicitamente a 'standard'. |

| --- |

| Checkpoint: A questo punto deve essere operativo solo il control plane Envoy Gateway. Non abbiamo ancora creato Gateway/HTTPRoute e non abbiamo migrato alcun workload. |

| --- |

| Attenzione: Eliminare le CRD solo dopo aver eliminato Gateway, HTTPRoute e tutte le risorse Envoy/Gateway API che si vogliono conservare. La cancellazione di una CRD elimina anche le relative custom resources. |

| --- |