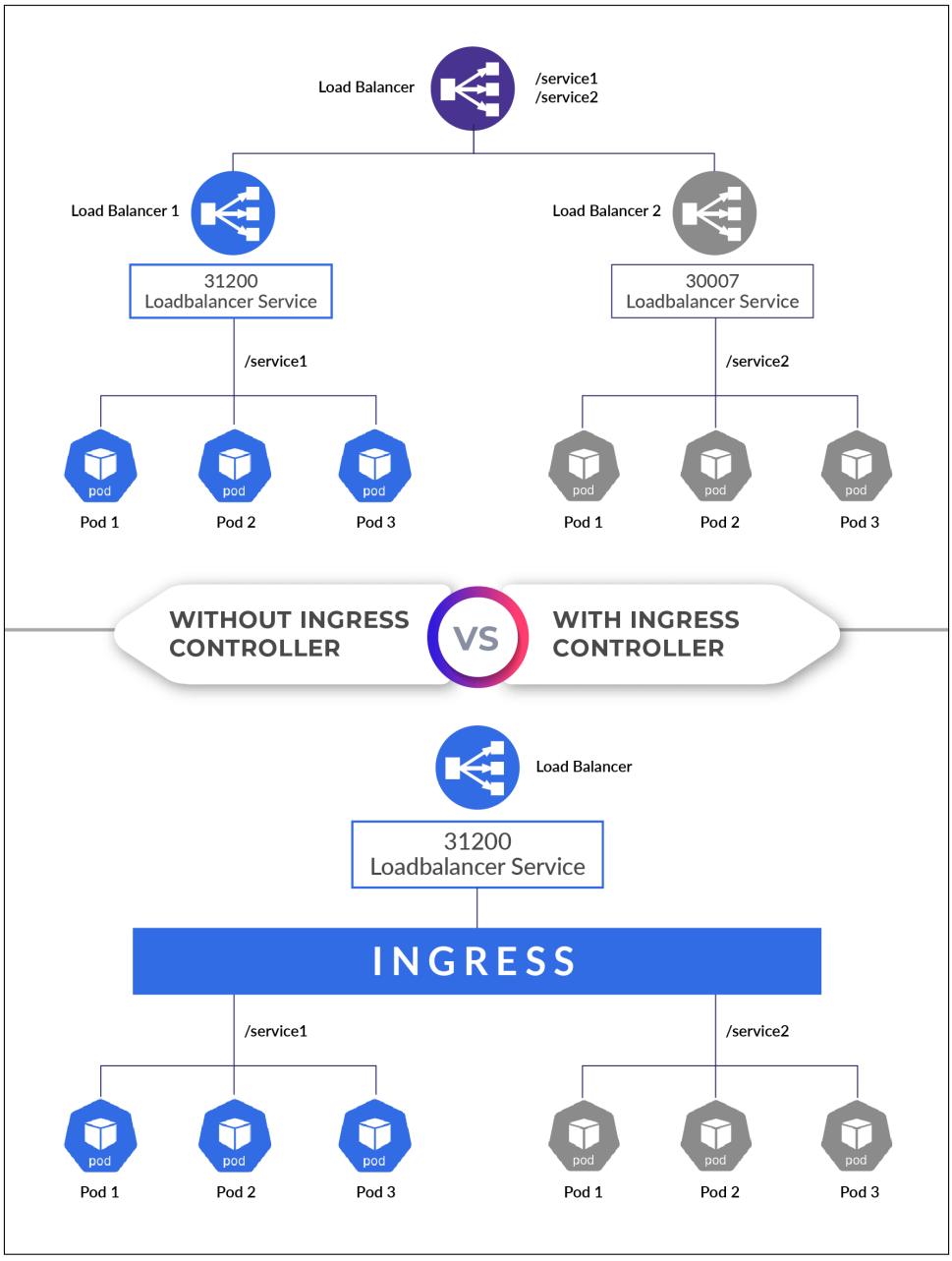
🧠 COURS COMPLET SUR INGRESS KUBERNETES



🔎 1. Rappel : Pourquoi Ingress ?

🚧 Problème

Tu exposes un service web (ex: backend-api, frontend, admin-panel). Sans Ingress, tu as 3 choix :

1. ClusterIP → privé, usage interne uniquement
2. NodePort → public, mais peu pratique et non flexible
3. LoadBalancer → pratique, mais un **load balancer coûteux par service**

🎯 Solution

Ingress = point d'entrée unique pour **plusieurs services** HTTP/HTTPS, avec :

- Routage par nom de domaine (app1.monapp.com, api.monapp.com)
- Routage par chemin (/admin, /api)
- Support SSL/TLS (HTTPS)
- Contrôles d'accès (authentification, headers, rate limit, etc.)

🧱 2. Architecture générale

```
Internet (Client)
  |
[Ingress Controller]
  |
[Ingress Resource]
  |
[Service]
  |
[Pod]
```

- **Ingress Controller** : pod spécial qui interprète les règles d'Ingress (NGINX, Traefik…)
- **Ingress** : objet YAML avec les règles de routage
- **Service** : connecte vers les pods
- **Pod** : contient le conteneur (app)

🛠️ 3. Installation d'un Ingress Controller (NGINX)

a) Pour Minikube (local)

```
minikube addons enable ingress
minikube tunnel
```

b) Pour cluster cloud ou kubeadm :

```
kubectl apply -f https://raw.githubusercontent.com/kubernetes/ingress-nginx/controller-v1.10.1/deploy/static/provider/cloud/deploy.yaml
```

Vérifie l'installation :

```
kubectl get pods -n ingress-nginx
kubectl get svc -n ingress-nginx
```

Tu dois voir un **pod NGINX Ingress Controller** tournant.

4. Création d'un Ingress (Exemple pas à pas)

a) Déployer deux services

app1-deployment.yaml

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: app1
spec:
  replicas: 1
  selector:
    matchLabels:
      app: app1
  template:
    metadata:
      labels:
        app: app1
    spec:
      containers:
      - name: app1
        image: hashicorp/http-echo
        args:
        - "-text=Hello from App1"
        ports:
        - containerPort: 5678
---
apiVersion: v1
kind: Service
metadata:
  name: app1
spec:
  selector:
```

```
    app: app1
  ports:
  - port: 80
    targetPort: 5678
```

app2-deployment.yaml
(similaire, juste changer les noms)

b) Définir un Ingress

ingress.yaml

```yaml
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: my-ingress
  annotations:
    nginx.ingress.kubernetes.io/rewrite-target: /
spec:
  rules:
  - http:
      paths:
      - path: /app1
        pathType: Prefix
        backend:
          service:
            name: app1
            port:
              number: 80
      - path: /app2
        pathType: Prefix
        backend:
          service:
            name: app2
            port:
              number: 80
```

☑ Ceci permet de dire : "quand on appelle /app1, envoie vers le service app1."

🌐 5. Tester en local avec Minikube

```
```

```
minikube tunnel
kubectl apply -f app1-deployment.yaml
kubectl apply -f app2-deployment.yaml
kubectl apply -f ingress.yaml
```

Ensuite visite :

- http://localhost/app1
- http://localhost/app2

🔐 6. HTTPS avec TLS

Pour activer TLS :

a) Crée un certificat

```
kubectl create secret tls tls-secret \
  --cert=chemin/vers/cert.crt \
  --key=chemin/vers/key.key
```

b) Ajoute TLS à ton ingress

```
spec:
  tls:
  - hosts:
    - monapp.local
    secretName: tls-secret
```

🧠 7. Annotations utiles

Annotation	Utilité
nginx.ingress.kubernetes.io/rewrite-target	Réécrit le chemin
nginx.ingress.kubernetes.io/auth-url	Rediriger vers un service d'auth
nginx.ingress.kubernetes.io/limit-rps	Limite les requêtes/s
nginx.ingress.kubernetes.io/ssl-redirect	Forcer le HTTPS

☸ 8. Ingress avancé (ex : sous-domaines)

```
spec:
  rules:
  - host: admin.monapp.com
    http:
      paths:
      - path: /
        pathType: Prefix
        backend:
          service:
            name: admin-service
            port:
              number: 80
```

🔚 9. Cert-Manager : Générer automatiquement les certificats Let's Encrypt

a) Installer cert-manager

```
kubectl apply -f https://github.com/cert-manager/cert-
manager/releases/latest/download/cert-manager.yaml
```

b) Créer un ClusterIssuer

```
apiVersion: cert-manager.io/v1
kind: ClusterIssuer
metadata:
  name: letsencrypt-prod
spec:
  acme:
    email: admin@monapp.com
    server: https://acme-v02.api.letsencrypt.org/directory
    privateKeySecretRef:
      name: letsencrypt-prod
    solvers:
    - http01:
        ingress:
          class: nginx
```

c) Utilisation dans Ingress

```yaml
metadata:
  annotations:
    cert-manager.io/cluster-issuer: letsencrypt-prod
spec:
  tls:
  - hosts:
    - monapp.com
    secretName: monapp-tls
```

□ 10. Problèmes frequents

Problème	Cause probable
404 Not Found	Ingress Controller absent ou mal configuré
HTTP au lieu de HTTPS	SSL pas activé ou certificat manquant
/app1 retourne vide	rewrite-target mal utilisé
Erreurs 503	Service cible non disponible ou mal nommé

□ 11. Outils à connaître

- kubectl describe ingress → voir les détails
- kubectl logs -n ingress-nginx → logs du controller
- Ingress NGINX docs
- Cert-Manager docs

🧠 12. Résumé visuel

Navigateur
↓
Ingress Controller (ex: NGINX)
↓
Ingress Rules
↓
Services
↓
Pods

🎓 13. À faire pour progresser

1. Configurer un **Ingress avec domaine local** (/etc/hosts)
2. Ajouter **TLS avec cert-manager**
3. Créer un **auth backend** avec nginx.ingress.kubernetes.io/auth-url
4. Utiliser **Traefik** comme Ingress Controller
5. Déployer une app réelle (Django, Laravel) derrière un Ingress

Principaux contrôleurs Ingress et leurs cas d'utilisation

1. NGINX Ingress Controller (ingress-nginx)

- **Description** : Contrôleur Ingress open-source basé sur NGINX, maintenu par la communauté Kubernetes.
- **Avantages** :
  - Large adoption et documentation abondante.
  - Supporte le routage basé sur le chemin et le nom d'hôte.
  - Compatible avec Cert-Manager pour la gestion des certificats TLS.
- **Cas d'utilisation** :
  - Environnements de développement et de production nécessitant une solution éprouvée.
  - Clusters auto-hébergés ou sur des fournisseurs de cloud sans contrôleur Ingress natif.

2. Traefik

- **Description** : Contrôleur Ingress moderne et léger, écrit en Go, avec une interface web intégrée.
- **Avantages** :
  - Configuration dynamique via annotations et CRD.
  - Intégration facile avec Let's Encrypt pour les certificats TLS.
  - Interface web pour la visualisation des routes.
- **Cas d'utilisation** :
  - Déploiements nécessitant une configuration dynamique et une interface utilisateur conviviale.
  - Environnements où la simplicité et la rapidité de configuration sont essentielles.

3. HAProxy Ingress

- **Description** : Contrôleur Ingress basé sur HAProxy, offrant des performances élevées et une faible latence.
- **Avantages** :
  - Excellente performance pour le trafic HTTP/HTTPS.
  - Supporte des fonctionnalités avancées comme le routage basé sur les en-têtes.
- **Cas d'utilisation** :
  - Applications nécessitant une haute performance et une faible latence.
  - Environnements avec des exigences de routage complexes.

4. Contour

- **Description** : Contrôleur Ingress basé sur Envoy, développé par VMware.
- **Avantages** :
  - Supporte le protocole HTTP/2 et gRPC.
  - Séparation claire entre le plan de contrôle et le plan de données.
- **Cas d'utilisation** :
  - Applications utilisant gRPC ou nécessitant HTTP/2.

    - Déploiements cherchant une architecture moderne et évolutive.Kubernetes+1KubeSphere+1

5. Kong Ingress Controller

- **Description** : Contrôleur Ingress basé sur Kong, offrant des fonctionnalités d'API Gateway.
- **Avantages** :
    - Fonctionnalités avancées comme la gestion des clés API, la limitation de débit, etc.
    - Extensible via des plugins.
- **Cas d'utilisation** :
    - Exposition d'API nécessitant des fonctionnalités de gestion avancées.
    - Environnements où la sécurité et la gestion des API sont primordiales.

6. AWS ALB Ingress Controller

- **Description** : Contrôleur Ingress pour les clusters Kubernetes sur AWS, utilisant l'Application Load Balancer.
- **Avantages** :
    - Intégration native avec les services AWS.
    - Supporte le routage basé sur le chemin et le nom d'hôte.
- **Cas d'utilisation** :
    - Clusters Kubernetes déployés sur AWS.
    - Environnements nécessitant une intégration étroite avec les services AWS.Alen Komljen

□ Comment choisir le bon contrôleur Ingress ?

Le choix du contrôleur Ingress dépend de plusieurs facteurs :

- **Environnement de déploiement** :
    - Sur AWS : AWS ALB Ingress Controller.
    - Sur GCP : GCE Ingress Controller.
    - Sur Azure : Azure Application Gateway Ingress Controller.
    - Auto-hébergé : NGINX, Traefik, HAProxy, etc.User Manual and Diagram Full List+1Alen Komljen+1KubeSphere+1Kubernetes+1Kubernetes
- **Fonctionnalités requises** :
    - Besoin de gestion d'API avancée : Kong.
    - Support de gRPC/HTTP2 : Contour.
    - Interface utilisateur intégrée : Traefik.Alen Komljen+1Reddit+1tigera.io
- **Performance** :
    - Trafic élevé nécessitant une faible latence : HAProxy.
    - Trafic modéré avec besoin de flexibilité : NGINX ou Traefik.
- **Simplicité de configuration** :
    - Débutants : NGINX ou Traefik.
    - Utilisateurs avancés : HAProxy, Contour, Kong.Kubernetes

📌 Recommandations

- **Débutants** : Commencez avec NGINX Ingress Controller pour sa simplicité et sa large adoption.
- **Besoins spécifiques** : Évaluez les fonctionnalités offertes par chaque contrôleur en fonction de vos besoins (performance, gestion d'API, support de protocoles, etc.).
- **Environnements cloud** : Utilisez les contrôleurs Ingress natifs proposés par votre fournisseur de cloud pour une intégration optimale.