Kubernetes: TLS et autorités de certification

Mathieu Corbin, Team Lead **@Exoscale**

Développeur

- Clojure
- Golang
- Emacs
- ...

Sysadmin

- Automatisation
- CI/CD
- Monitoring
- Systèmes distribués
- ...

@_mcorbin

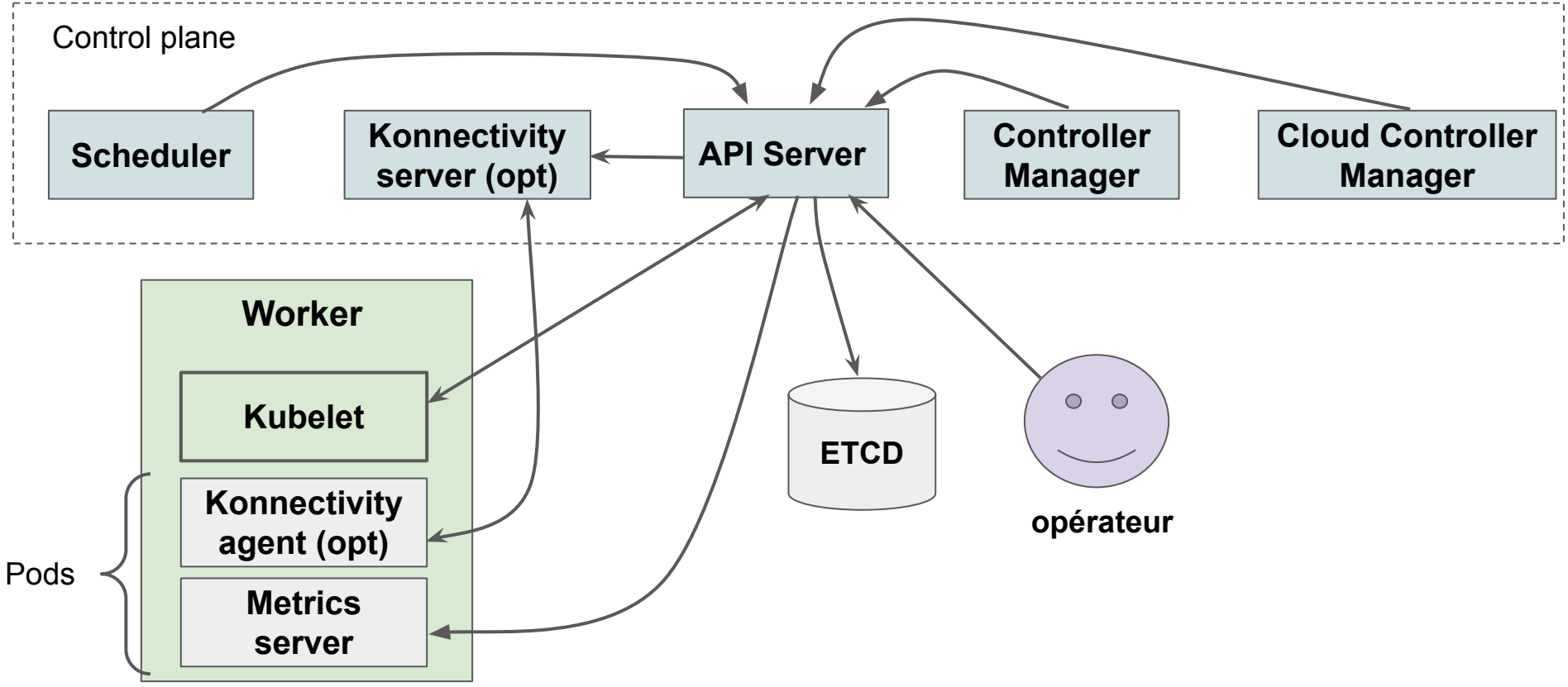
mcorbin.fr

EXOSCALE

Un Cloud Provider Européen

1.	FRANKFURT	DE-FRA-1
2.	MUNICH	DE-MUC-1
3.	VIENNA	AT-VIE-1
4.	GENEVA	CH-GVA-2
5.	ZURICH	CH-DK-2
6.	SOFIA	BG-SOF-1

Kubernetes: plusieurs composants



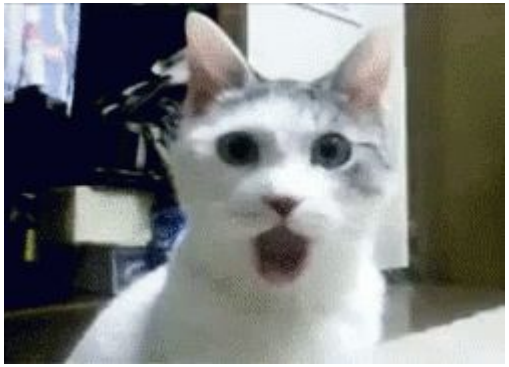
De nombreux flux

(m)TLS partout

Des flux de différents types

Des tonnes d’options en lien avec les certificats

Rien que pour l’API server (donc un seul composant), on a:

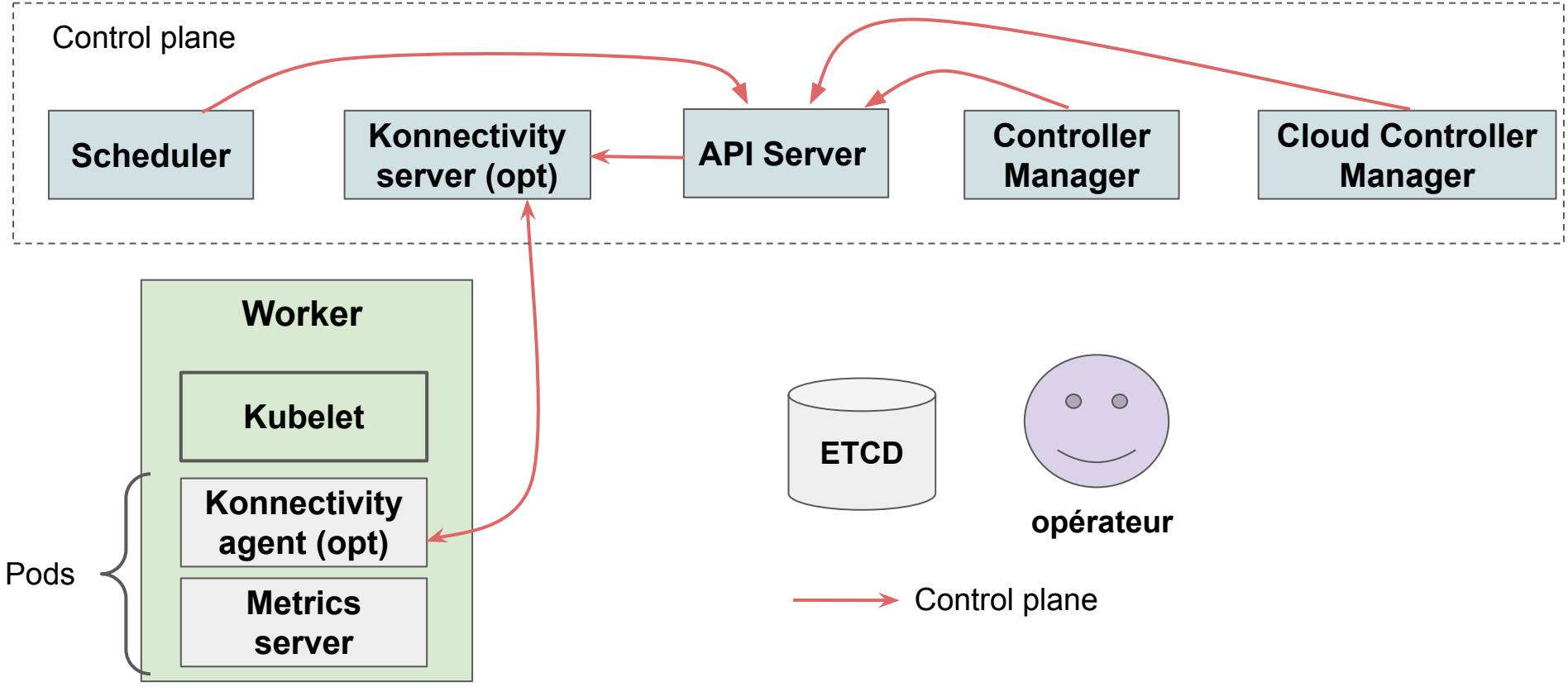


--client-ca-file	CAs acceptées par l’API server
--egress-selector-config-file	Configuration Konnectivity
--etcd-cafile	Certificats pour le client ETCD
--etcd-certfile	//
--etcd-keyfile	//
--kubelet-certificate-authority	CA des certificats Kubelet
--kubelet-client-certificate	Certificats pour le client Kubelet
--kubelet-client-key	//
--proxy-client-cert-file	Certificats pour le client de l’aggregation layer
--proxy-client-key-file	//
--requestheader-client-ca-file	CA de l’aggregation layer
--service-account-key-file	Clé pour signer les service accounts token
--service-account-signing-key-file	//
--tls-cert-file	Certificats pour le serveur HTTP de l’API server
--tls-private-key-file	//

Flux Control plane

Communication entre les différents composants du control plane

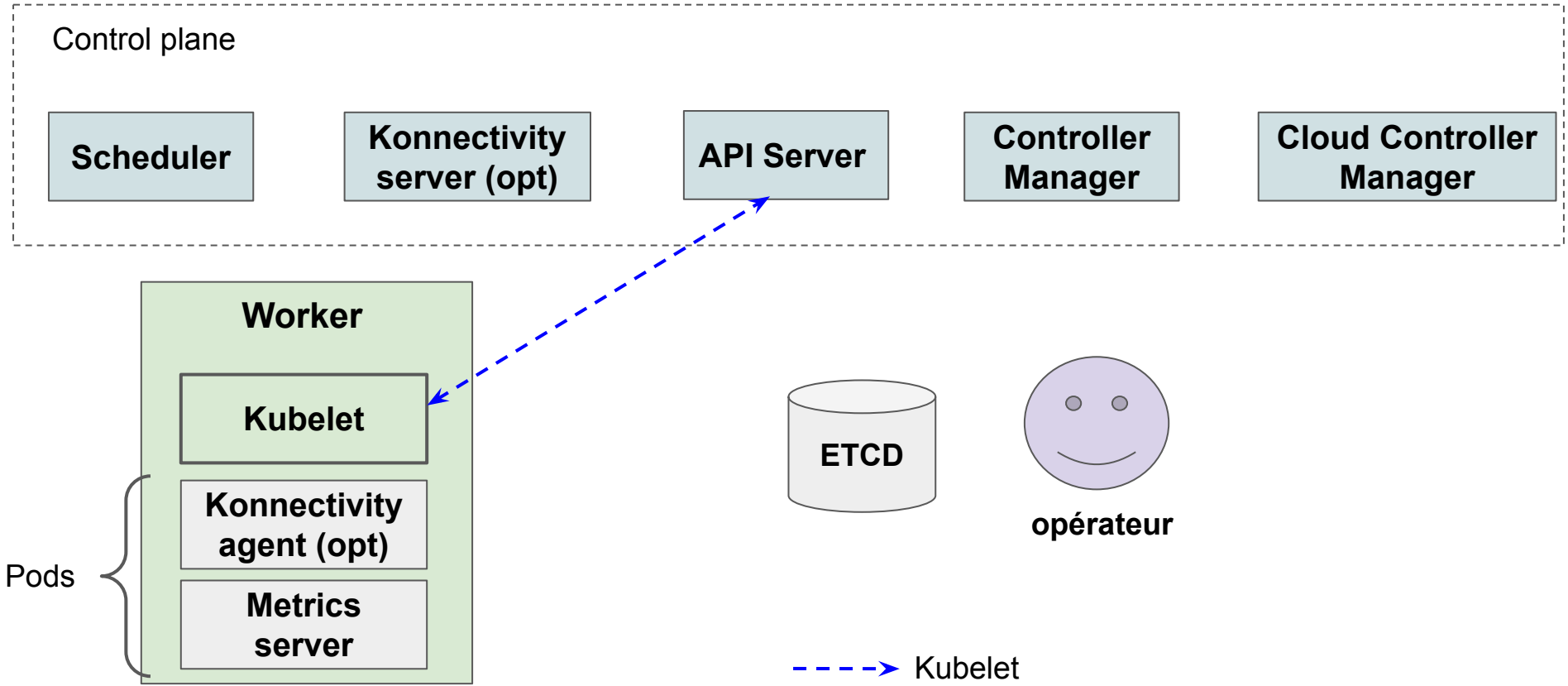
Flux Control plane



Flux Kubelet

Communication entre les workers (Kubelet) et l’API server

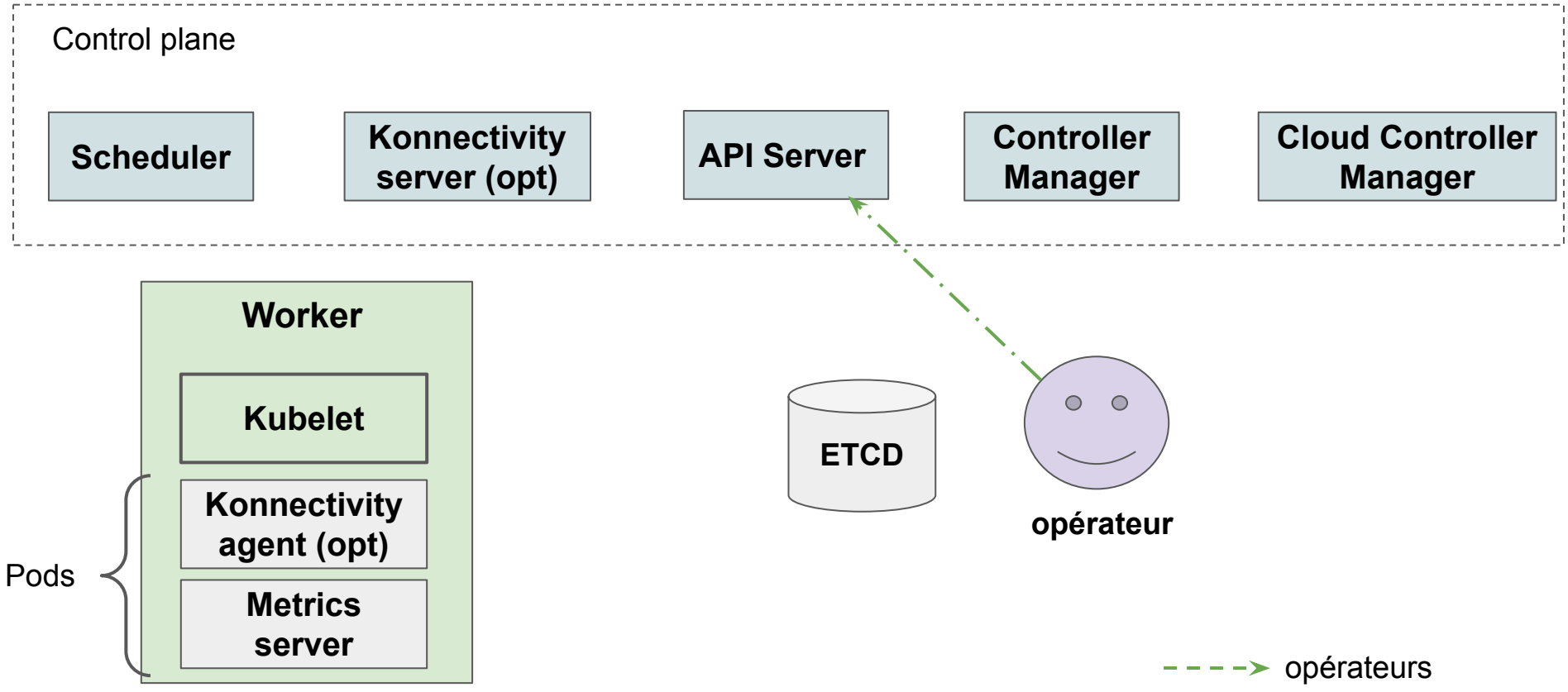
Flux Kubelet



Flux opérateurs

Communication entre les opérateurs et le cluster, pour l’administration par exemple

Flux opérateurs

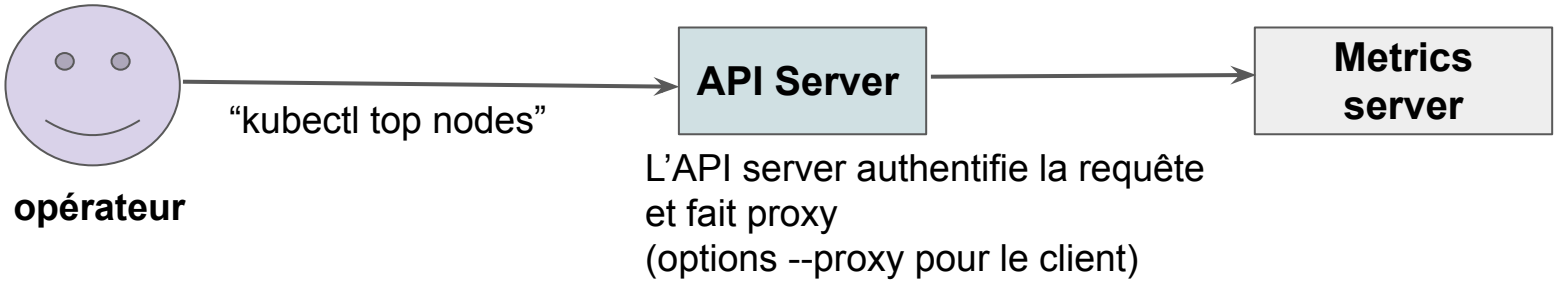


Flux aggregation layer

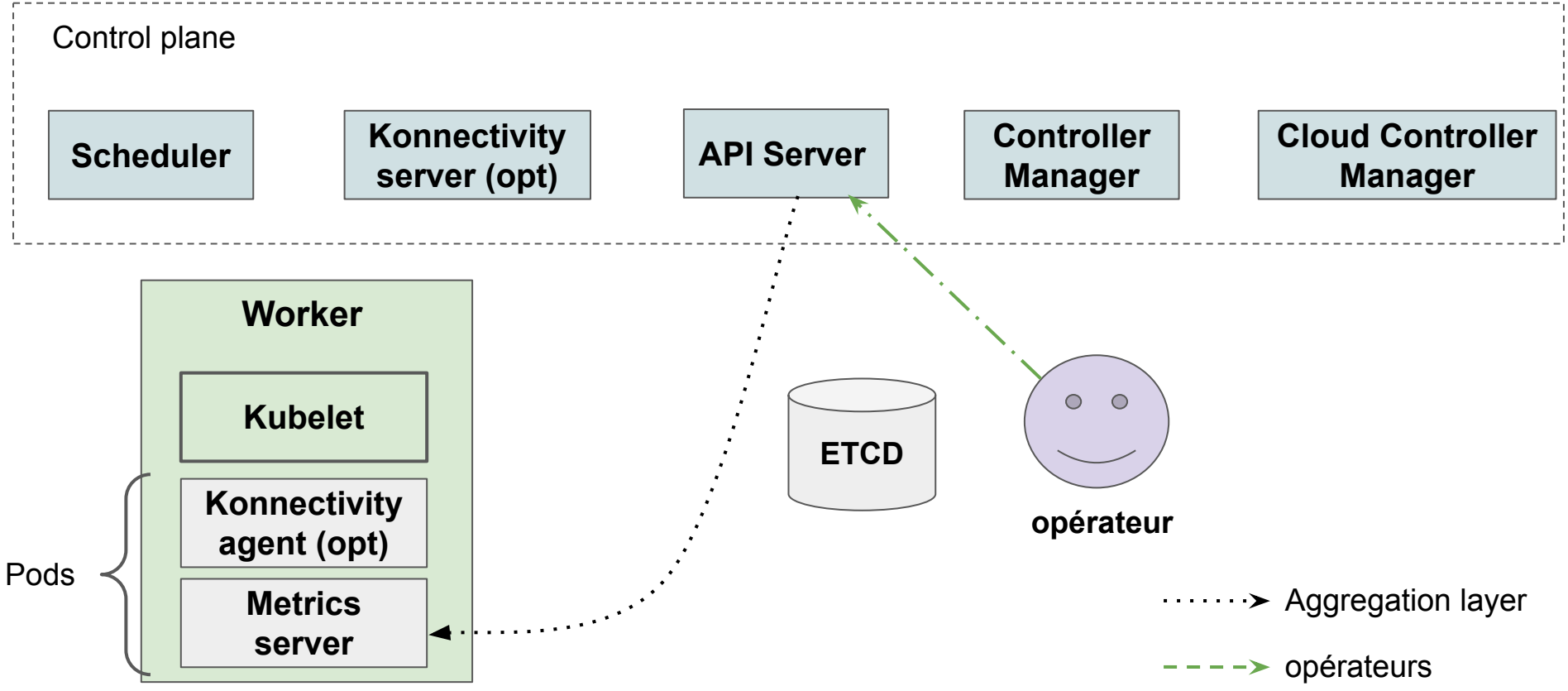
Communication entre l’API server qui sert de proxy vers des **extensions** de l’API server

Aggregation layer

- “The aggregation layer allows Kubernetes to be extended with additional APIs”
- Un client envoie une requête à l’API server
  - L’API server authentifie le client
  - L’API server proxy la requête (en utilisant un client et des certificats dédiés) à un autre service servant d’extension



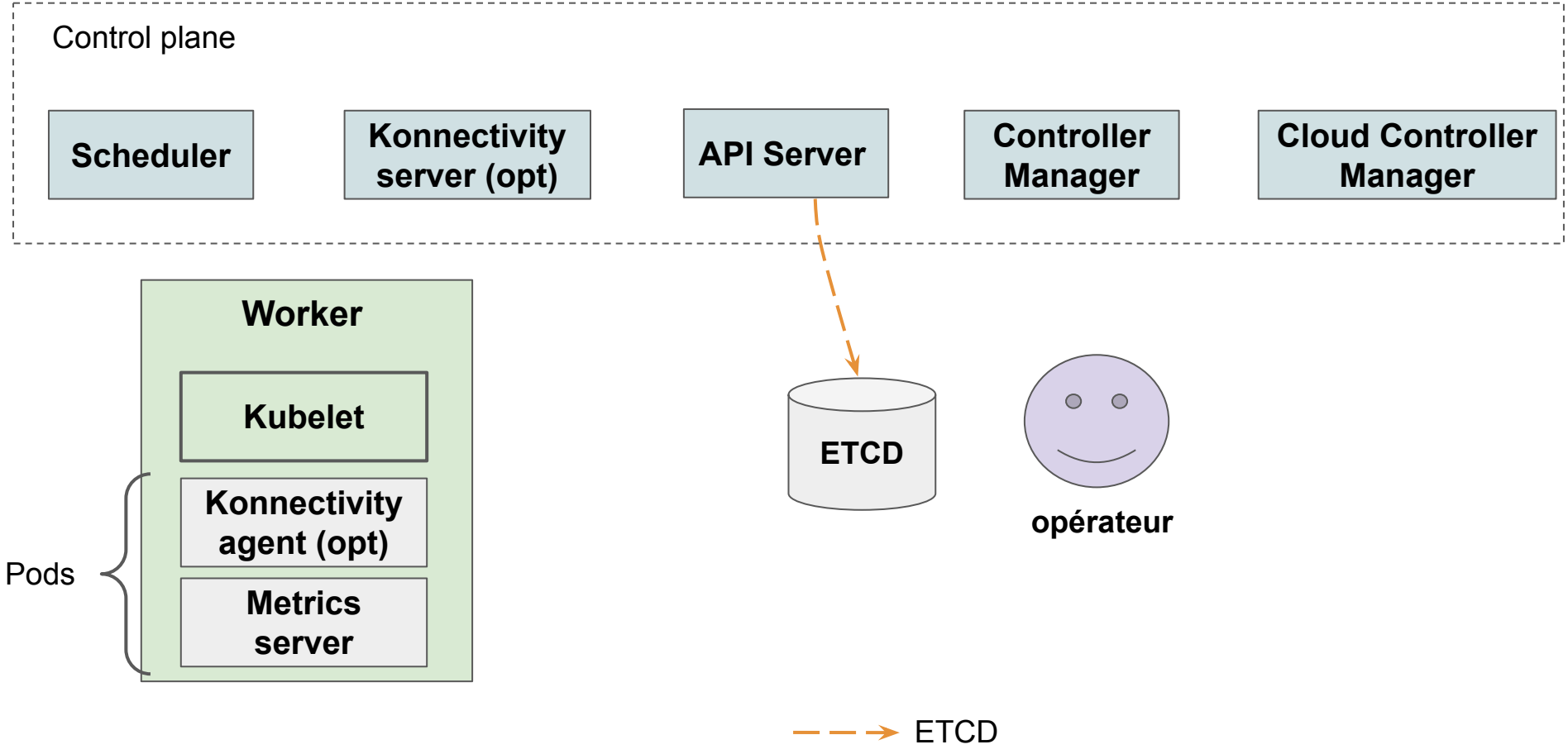
Kubernetes: aggregation layer



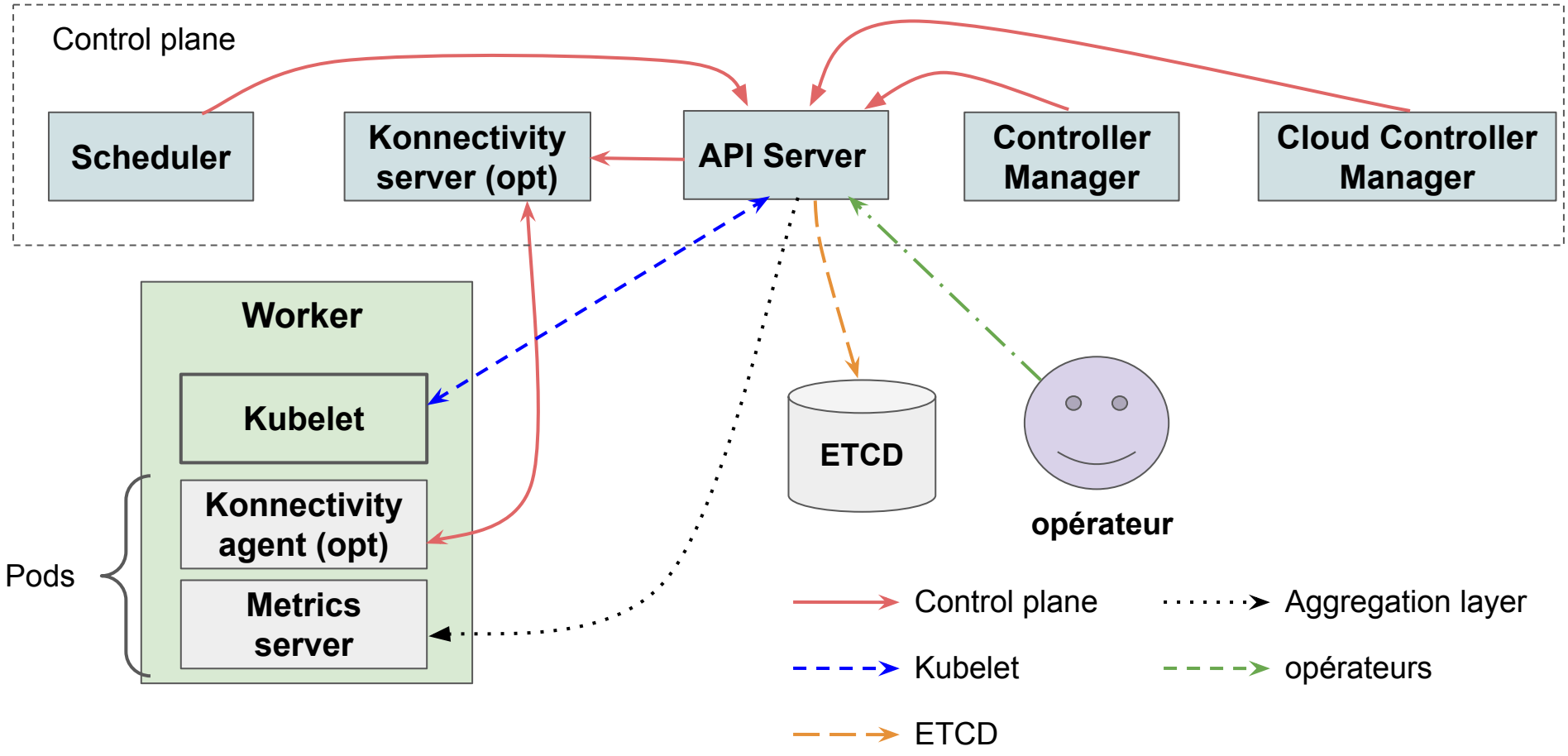
Flux ETCD

Communication entre l’API server et ETCD

ETCD


Control plane
Scheduler
Konnectivity server (opt)
API Server
Controller Manager
Cloud Controller Manager
Worker
Kubelet
Konnectivity agent (opt)
Metrics server
Pods
ETCD
opérateur
ETCD

Résumé


Control plane
Scheduler
Konnectivity server (opt)
API Server
Controller Manager
Cloud Controller Manager
Worker
Kubelet
Konnectivity agent (opt)
Metrics server
Pods
ETCD
opérateur
Control plane
Aggregation layer
Kubelet
opérateurs
ETCD

Combien d’autorités de certification ?

- 5 Flux
  - ETCD
  - Control plane
  - Opérateurs
  - Kubelet
  - Aggregation layer

- 5 autorités de certification pour le cluster
  - Permet de bien cloisonner les flux
    - Impossible qu’un client d’un flux parle au serveur d’un autre flux
  - Avoir plusieurs CA (notamment pour l’aggregation layer) est recommandé par Kubernetes

Authentification par certificat client

CA opérateurs

```
Issuer: C = CH, L = GVA2, O = Exoscale, OU = SKS, CN = operators-ca
Validity
    Not Before: Jun  4 19:13:29 2021 GMT
    Not After : Jul  4 19:13:34 2021 GMT
Subject: C = CH, L = GVA2, O = system:masters, OU = SKS, CN = mathieu
```

super admin (attention !)

Utilisateur (peut ne pas exister dans kubernetes)

Authentification par certificat client

- Les différents composants peuvent donc:
  - Référencer des groupes Kubernetes standards existant par défaut
    - Comme **system:masters** sur la slide précédente
  - Référencer des utilisateurs dans des ClusterRolesBindings existant par défaut
  - Référencer des utilisateurs ou groupes ajoutés par des administrateurs dans des ClusterRolesBindings/RoleBindings

- La doc Kubernetes maintient une liste de subjects conseillés:
  - https://kubernetes.io/docs/setup/best-practices/certificates/#certificate-paths

Exemple avec le controller manager

- Le Controller Manager aura un certificat avec “**system:kube-controller-manager**” en Common Name

- kubectl -n kube-system get clusterrolebinding system:kube-controller-manager -o yaml

```
...
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: system:kube-controller-manager
subjects:
- apiGroup: rbac.authorization.k8s.io
  kind: User
  name: system:kube-controller-manager
```

- Même principe pour les autres clients du cluster

Authentification par certificat

- Les certificats restent un moyen simple et sécurisé de s'authentifier à l'API
  - D'autres méthodes existent (OpenID) pour les opérateurs
  - Aussi possibilité d'utiliser des webhooks pour l'API server et kubelet

- Attention sur certaines offres Kubernetes as a service
  - Pas de contrôle sur le certificat du kubeconfig (TTL, sujet)
    - Donc tout le monde admin sans possibilité de changer cela
  - Voir kubeconfig qui utilisent des tokens "statiques"

- Chez Exoscale, contrôle du certificat:
  - exo sks kubeconfig **<cluster> <utilisateur/CN>** --ttl **<TTL seconde>** --group **<groupe k8s/org>**

Révocation

- Attention, pas moyen de révoquer un certificat dans Kubernetes
    - Ne jamais utiliser de groupes mais que des noms d'utilisateurs en sujet de certificat ?
        - Révoquer l'utilisateur dans Kubernetes si nécessaire
        - Nom d'utilisateur unique
    - Rotation de la CA "opérateurs" ?
        - Permet d'invalider tous les certificats externes sans toucher aux certificats internes (control plane, aggregation layer, etcd…)
        - Une bonne raison d'avoir des CA dédiées à chaque flux

Kubelet: TLS bootstrapping

- Kubelet doit pouvoir communiquer avec l'API server
  - Et donc avoir un certificat avec certaines permissions

- Nous voulons automatiser la génération de ce certificat
  - Nouvelle machine => la machine rejoint le cluster toute seule

- Utilisation de Bootstrap tokens
  - Un token créé dans Kubernetes (et avec un TTL) est passé à Kubelet au démarrage
  - Kubelet utilise ce token pour s'authentifier et demander un certificat via une CSR
  - Le Controller Manager est configuré avec la CA "Kubelet" et signe le certificat
    - Encore une bonne raison d'avoir une CA dédiée pour Kubelet
  - Kubelet gère aussi le renouvellement du certificat

TLS bootstrapping: limitations

- Ne permet de générer automatiquement que le certificat “client” de Kubelet
  - Flux Kubelet => API Server

- Le certificat server de kubelet (API Server => Kubelet) ne peut pas être signé de cette manière
  - Action manuelle (kubernetes certificate approve <CSR>), mais peu intéressant
  - Développer son propre controller Kubernetes pour approuver la CSR
    - Solution sélectionnée sur l’offre d’Exoscale
    - Le CCM vérifie (via une requête API) que la la CSR est correcte (machine appartenant au client, bonne IP…) et l’approuve

Aggregation layer, metrics server...

- Eviter le TLS “Insecure”

- Metrics server doit par exemple
  - Etre configuré pour trust la CA “aggregation” (--requestheader-client-ca-file)
    - Pour vérifier que l’API server utilise bien un certificat valide
  - Etre configuré pour trust la CA “kubelet” (--kubelet-certificate-authority)
    - En effet, Metrics server scrape Kubelet pour récupérer les métriques

- Possibilité chez Exoscale de récupérer les CA cert Kubelet/Aggregation
  - exo sks authority-cert <cluster> <autorité>
  - Nous gérons en fait pour vous Metrics server
    - … Mais pouvoir récupérer les certs publics peut être utile pour d’autres briques

Certificats clients/serveurs

- Evitez les certificats pouvant être utilisés comme client ET serveur
  - Configuration des extensions du certificat pour gérer cela

```
X509v3 Extended Key Usage: critical
    TLS Web Client Authentication
```

Merci

Questions ?