Des microservices pour des ascenseurs

Voyage au coeur du DevOps vertical

About me

Luc Juggery

Developer Relations Engineer

EXOSCALE

https://luc.run

Précédemment

- Co-founder startup IoT / efficacité énergétique
  - DevOps / architecte cloud
- Freelance
  - formateur Docker / Kubernetes
  - accompagnement DevOps

Exoscale

Cloud Provider Européen

- Compute
- Kubernetes
- Object Storage
- Block Storage
- GPU Servers
- DNS
- Virtual Private Cloud
- DBaaS

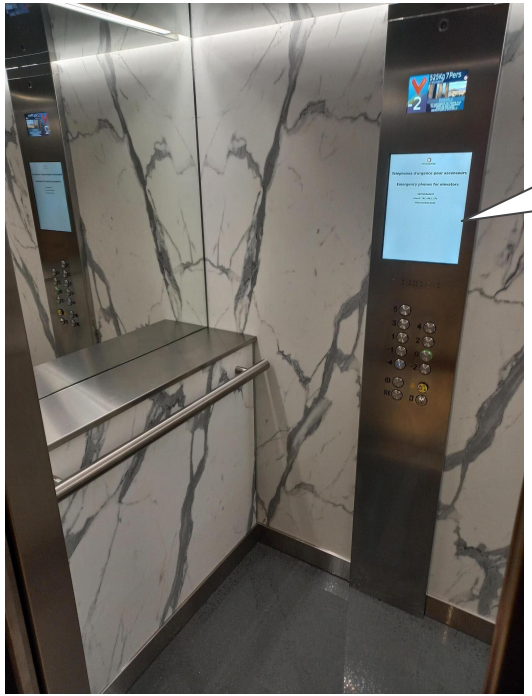
FRANKFURT	DE-FRA-1
VIENNA	AT-VIE-1
VIENNA-2	AT-VIE-2
GENEVA	CH-GVA-2
ZURICH	CH-DK-2
SOFIA	BG-SOF-1
MUNICH	DE-MUC-1
ZAGREB	HR-ZAG-1

EXOSCALE

Le contexte

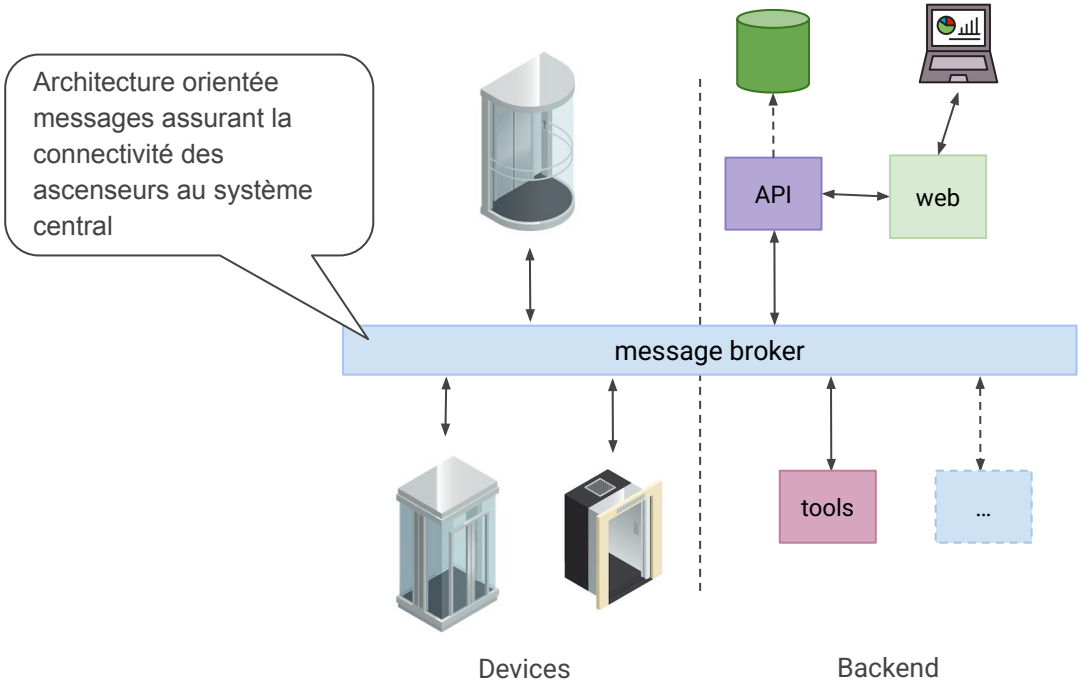
- Projet IoT d’équipement de plusieurs parcs d’ascenseurs à Monaco
- Fournir des services à valeurs ajoutée pour les copropriétaires et améliorer la gestion technique du parc
- Volonté de mettre en place une solution simple et maintenable
- Équipe réduite (~5 personnes) avec des disponibilités intermittentes

Côté ascenseurs



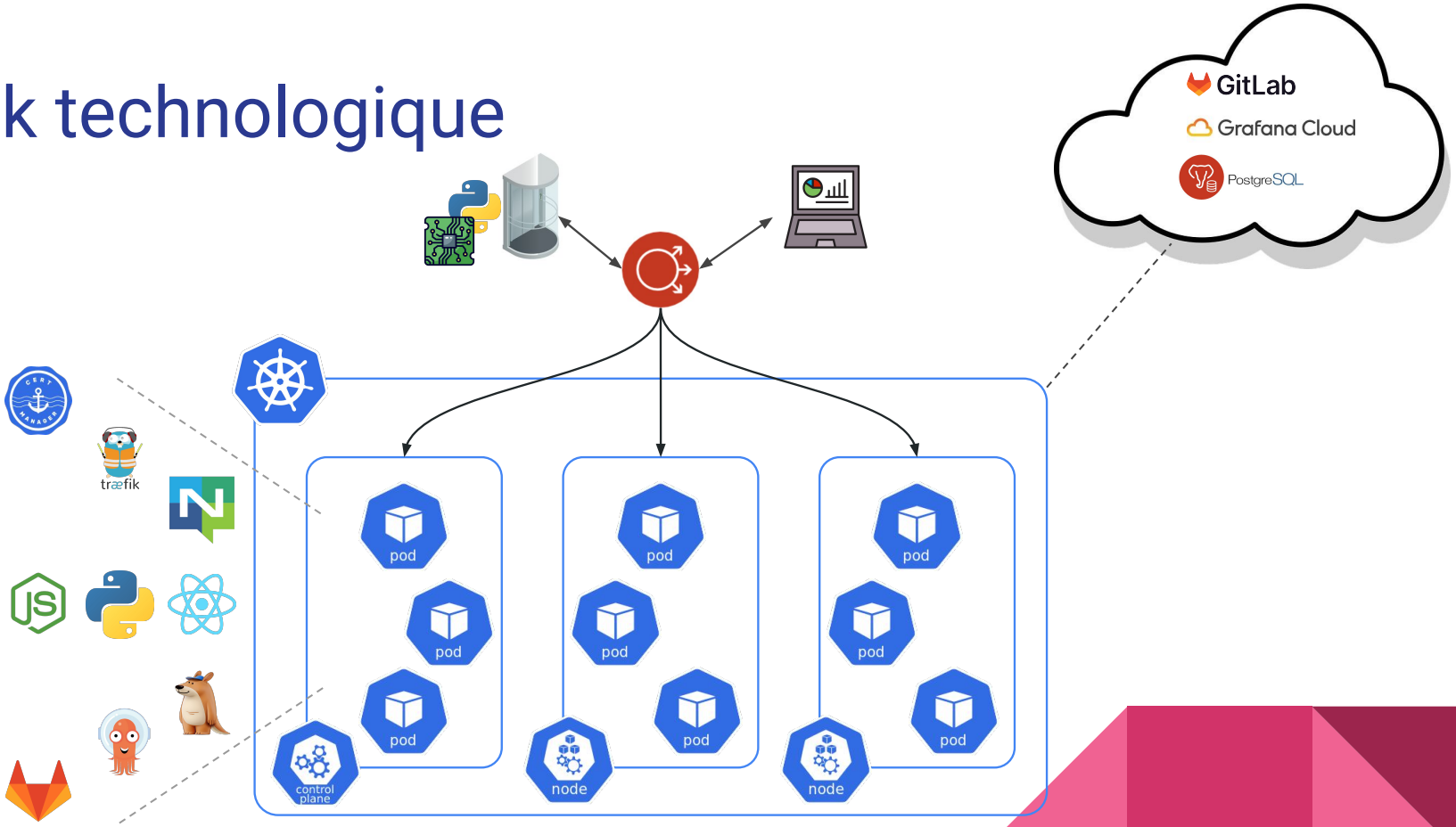
Écran affichant des informations relatives à la copropriété

Vue d’ensemble



Designed by macrovector / Freepik

Stack technologique



Les choix initiaux

Kubernetes managé

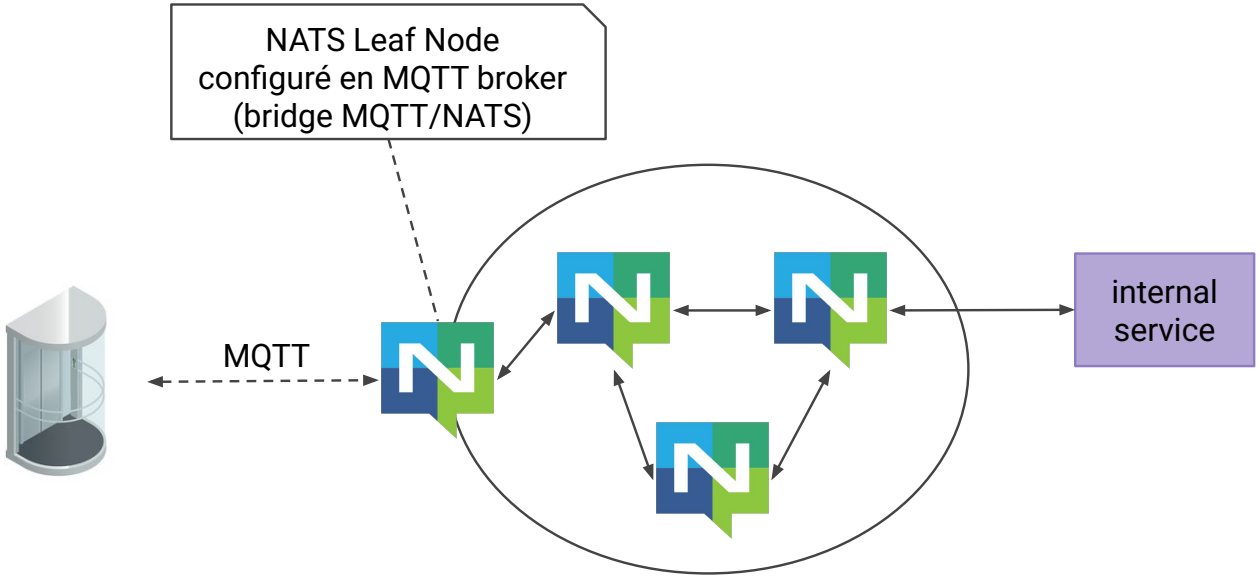
Bus de message basé sur NATS

GitLab (managé) - code/CI/registry

Approche GitOps avec Argo CD

Déploiement de l’application

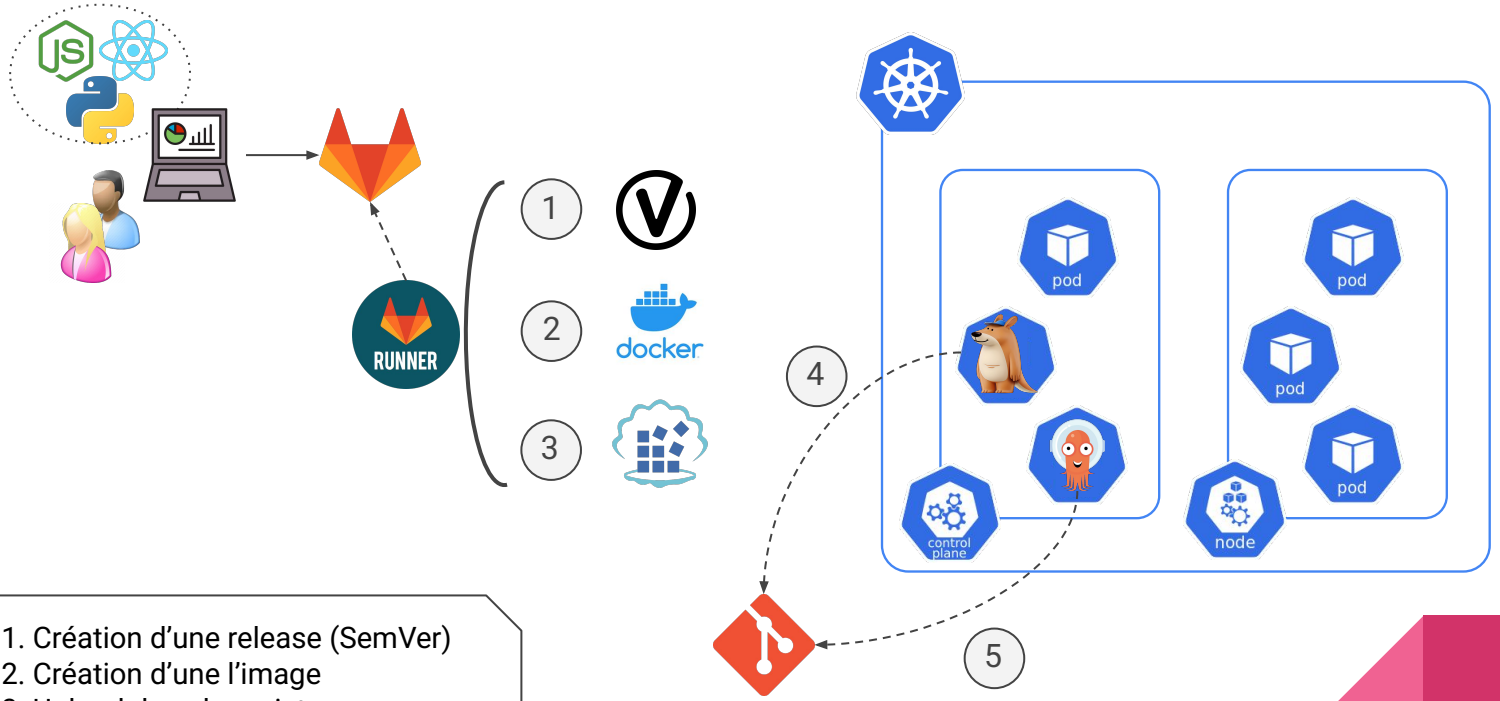
Communication device ↔ backend



Communication device ↔ backend

- Utilisation de MQTT pour le dialogue entre les devices et le backend
  - utilisée par la passerelle qui permet la connectivité des devices
  - standard de facto pour la communication dans l'IoT
- Pourquoi NATS, et pas Mosquitto ou autres ?
  - utilisation d'un cluster NATS en interne
  - facilité d'extension du cluster via un leaf node configuré en MQTT broker
  - payload MQTT traduit en message NATS, et inversement

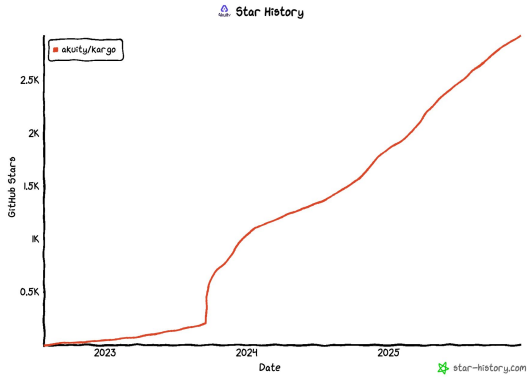
CI/CD



1. Création d'une release (SemVer)
2. Création d'une l'image
3. Upload dans le registry
4. Mise à jour config (GitOps) repo
5. Reconciliation

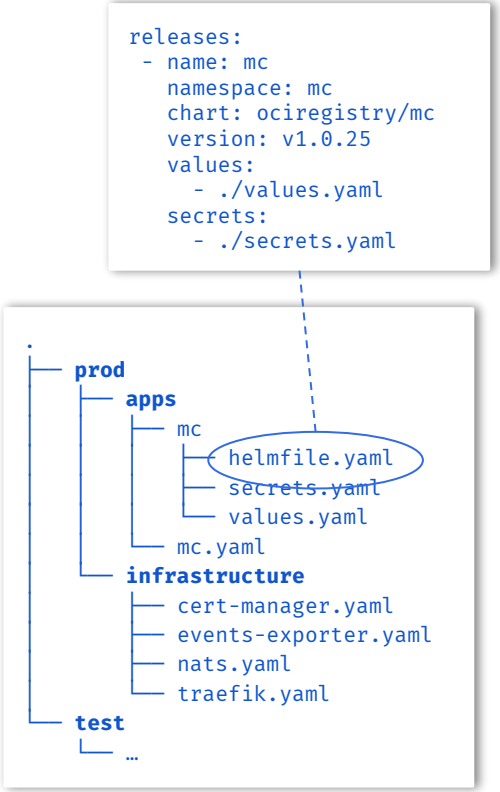
CI/CD - Évolution du workflow

- Utilisation de Kargo pour simplifier la CI et mettre à jour les fichiers values.yaml
- Pas encore utilisé pour la “vraie promotion” entre environnements
- Argo CD prend le relai en déployant les nouvelles versions



GitOps

- Sync automatique sur les différents environnements
- Plusieurs itérations sur la structure du repository
  - une seule branche
  - un répertoire par environnement
  - composants d'infrastructure / applications métier
- Utilisation de Helmfile
  - Spécification déclarative pour déployer des charts Helm



Gestion des secrets

- Encryption des fichiers de values
- SOPS (Secrets OPerationS) + age
- Décrypté par un plugin dédié dans Argo CD

```yaml
releases:
 - name: mc
   namespace: mc
   chart: ociregistry/mc
   version: v1.0.25
   values:
     - ./values.yaml
   secrets:
     - ./secrets.yaml
```

```yaml
database:
  url: postgresql://user:pass@postgres:5432/app
```

$ sops encrypt

```yaml
database:
    url: ENC[AES256_GCM,data:XieslPHwDM+Q==,iv:FD7JrBrd…A8=,tag:sZsJaMC01WFiEceBg==,type:str]
sops:
    age:
        - recipient: age10et4vwvyaar4xstqa95d6a3nhtr8rrnav8amsdcnvuf
          enc: |
            -----BEGIN AGE ENCRYPTED FILE-----
            …
            -----END AGE ENCRYPTED FILE-----
    mac: ENC[AES256_GCM,data:cAgU…xk=,iv:wXeso/czvY7+/R06c=,tag:UKklTtl2Sw==,type:str]
    version: 3.10.2
    …
```

Sécurité - une approche progressive

- Gestion des secrets (SOPS + age)
- RBAC et isolation par Namespace
- Durcissement des SecurityContext (Pod / Container)
- NetworkPolicies pour l’isolation réseau
- Évaluation de différentes solutions (Kyverno, Trivy Operator)

Les défis rencontrés

- Vouloir tout faire tourner dans Kubernetes
- Des choix temporaires qui perdurent dans le temps
- Importance des mises à jour régulières de Kubernetes
- Des microservices qui deviennent des “macroservices”
- Pas assez d’efforts sur l’observabilité

Les défis rencontrés

Vouloir tout faire tourner dans Kubernetes

- Ajout de couches applicatives
- Peut engendrer des problèmes (lors des mises à jour mais pas seulement)
- Exemple: utilisation de Longhorn + Postgres

Block storage distribué

⇒ Migration vers une base de données managée

Les défis rencontrés

Des choix temporaires qui perdurent dans le temps

- Certains raccourcis pris pour livrer une fonctionnalité rapidement
- Une dette technique qui s’accumule
- Exemple: faire passer des fichiers dans NATS au lieu de les uploader dans du stockage objet et d’envoyer la référence 😮

⇒ Nécessité d’une meilleure coordination en amont et des revues régulières de l’architecture

Les défis rencontrés

Importance des mises à jour régulières de Kubernetes

- Une release mineure de Kubernetes tous les 4 mois
- Plusieurs patchs entre les releases mineures (sécurité, bugs)
- Important de ne pas être trop en retard sur la dernière version en date

⇒ Retard qui peut se traduire par une nécessité de migrer vers un nouveau cluster au lieu d'appliquer les mises à jour

Les défis rencontrés

Des microservices qui deviennent des macroservices

- Ajout de fonctionnalités dans un composant au lieu de lui dédier un service
- Pollution du comportement initial
- Exemple: l'ajout de génération de fichiers PDF dans un composant dédié à faire des requêtes en base ⇒ OOM réguliers 😠

⇒ Nécessite une meilleure coordination et revue régulière des nouvelles fonctionnalités

Les défis rencontrés

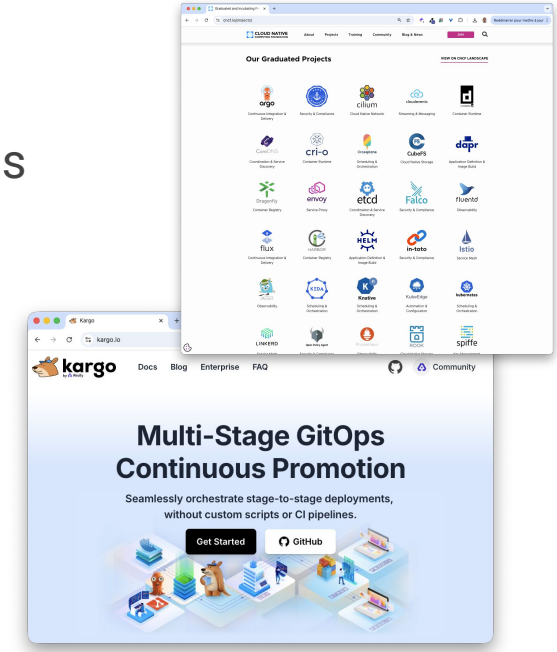
Pas assez d'efforts sur l'observabilité

- Mise en place de quelques briques
  - kube-prometheus-stack
  - tests avec Loki
  - kube-event-forwarder
- Pas d'effort suffisant pour construire une solution cohérente

⇒ Instrumentation avec OpenTelemetry et centralisation dans Grafana Cloud (en cours)

Importance de l’écosystème

- Échanges techniques dans des conférences et meetups
- Suivi de l’évolution des projets open-source / CNCF
- Permet d’identifier et de tester des composants



Les évolutions à considérer

- Gestion des secrets avec ESO https://external-secrets.io/
- Meilleur découpage des microservices en fonction des responsabilités
- Amélioration de la gestion des ressources (Karpenter)
- Simplifier la création d'environnement (cluster + apps)
- Aller plus loin dans l'utilisation de Kargo

Si c’était à refaire

- DB managée dès le début
- Meilleur cadrage sur le découpage des fonctionnalités
- Mise à jour régulières des clusters dès le jour 1
- Plus d’effort sur l’observabilité

Key takeaways

- R&D (Rapid & Dirty) OK si le temporaire reste temporaire

  ⇒ sinon, ça devient de la dette technique très vite

- Mettre à jour Kubernetes régulièrement

  ⇒ moins risqué que migrer un cluster obsolète

- Simplification en continu

  ⇒ refactorer tôt, éviter les macroservices, clarifier les responsabilités

Questions ?

Merci !

https://luc.run/devopsrex25.pdf