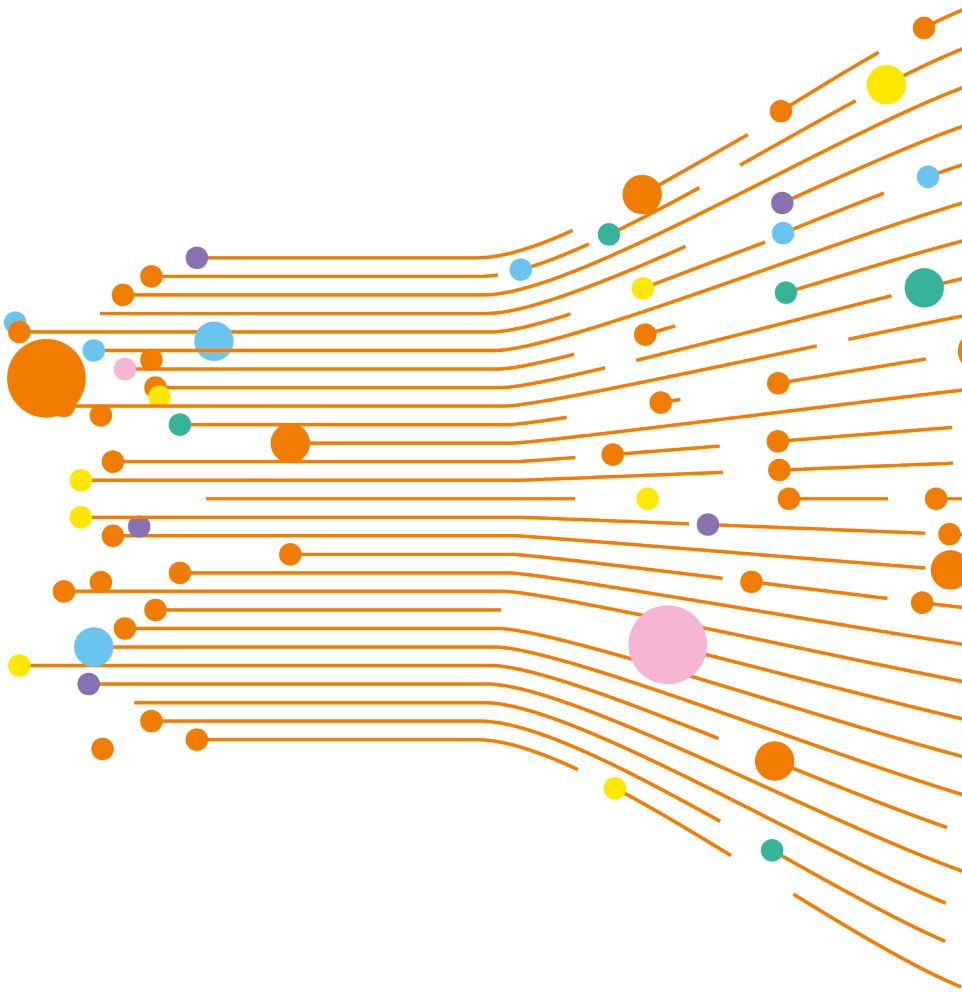
Datasource Loki



Objectif

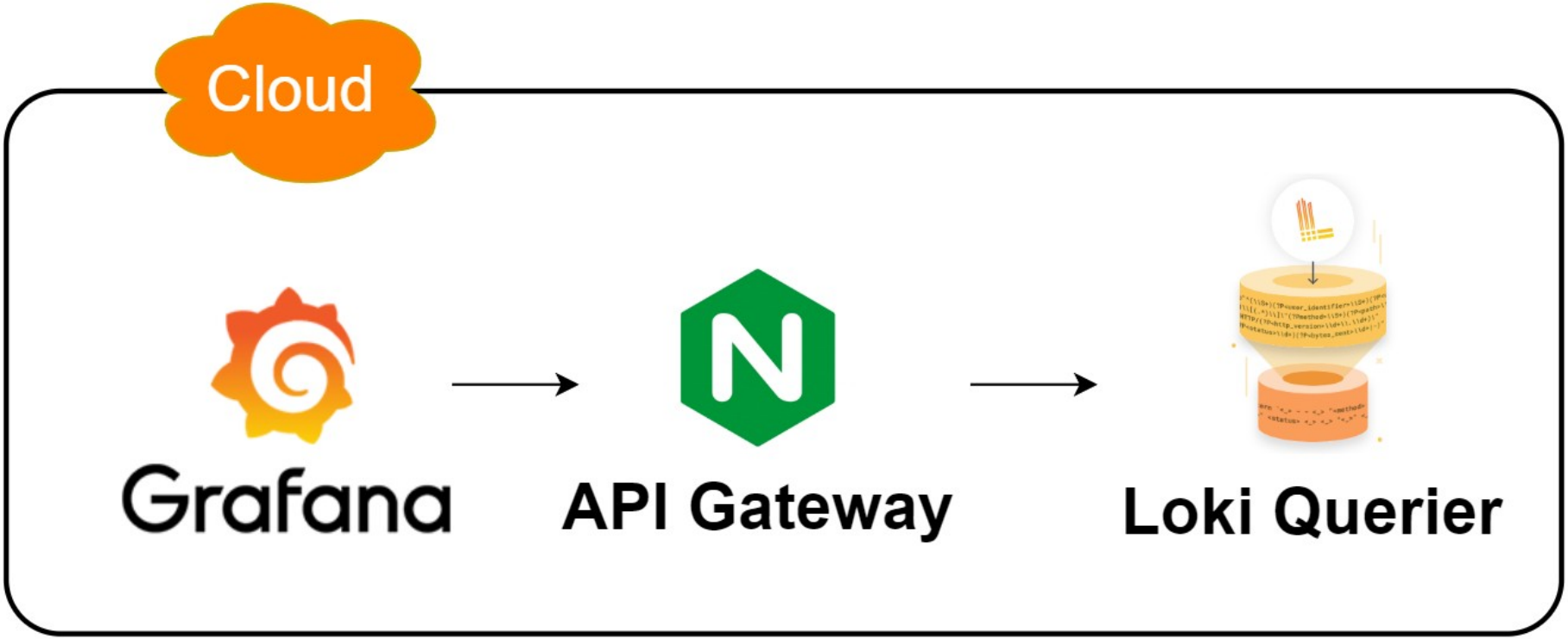
Ouverture de la datasource Loki au client final Athos d’ocb-esagso.

Pour lui permettre de :

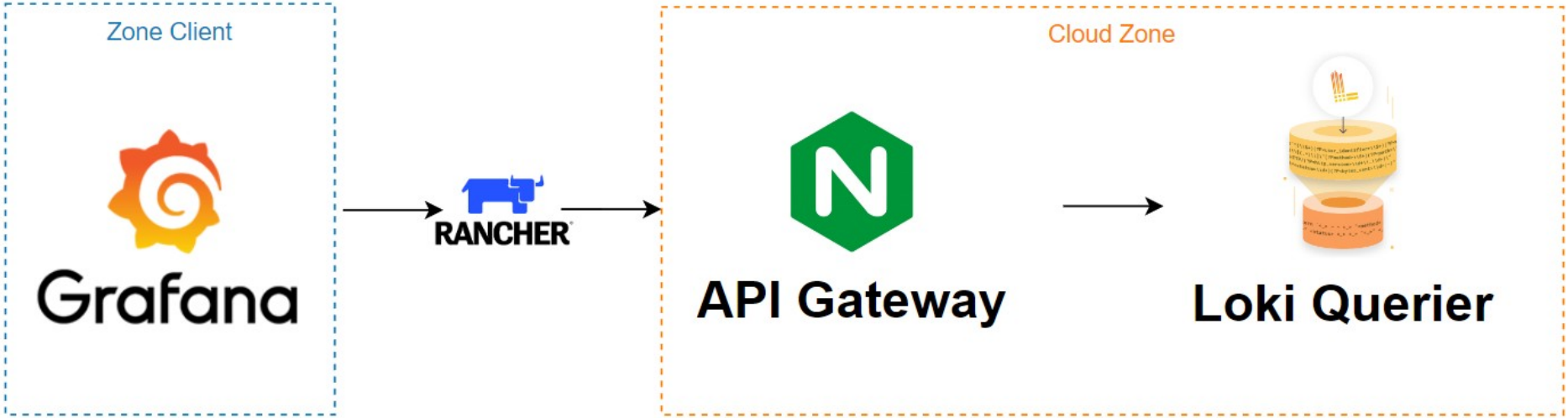
connecter son Grafana déployé sur le cluster client au Loki de la zone Cloud ;

faire ses propres dashboards, créer des alerting rules et notification policies via son Grafana.

Notre architecture actuelle :



Solution 1 :
Donner accès au Nginx Gateway de la zone cloud



Solution 1 : Exemple de datasource

```yaml
apiVersion: v1
stringData:
  datasource_loki_gateway_cloud.yaml: |
    apiVersion: 1
    datasources:
    - name: "Loki Gateway cloud"
      type: "loki"
      access: proxy
      orgId: 1
      url: "https://rancher.ocb-test06.caascad.com/k8s/clusters/local/api/v1/namespaces/logging-client/services/gateway:80/proxy"
      jsonData:
        httpHeaderName1: 'HeaderName'
        httpHeaderName2: 'Authorization'
      secureJsonData:
        httpHeaderValue1: 'HeaderValue'
        httpHeaderValue2: 'Bearer xxxxxxxxx'
kind: Secret
type: Opaque
metadata:
  labels:
    grafana-datasource: "1"
  name: grafana-datasource-loki-gateway-cloud
  namespace: pf-1054
```

Solution 1:
Donner accès au Nginx Gateway de la zone cloud

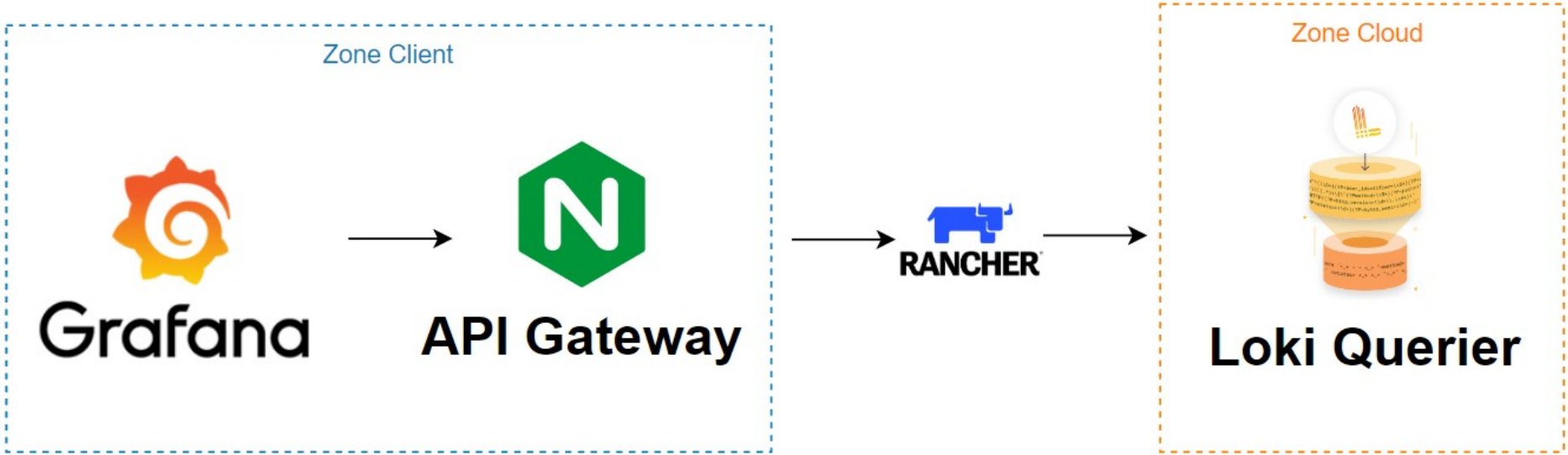
Dans cette solution la tâche à faire est :

- création d'un user Rancher spécifiquement pour ce cas d'usage.

  Plusieurs choix de configurations :

  - accès large : accès sur tous les services du cluster cloud en lecture (→ implique notamment que le client pourrait lire les logs Caascad).
  - accès limité : accès sur tous les services du namespace logging-client en lecture (→ ça implique d'utiliser des projets Rancher, ce n'est pas encore industrialisé).
  - accès strict : accès sur les services dont le nom est gateway (→ implique de changer le nom du service de Nginx Gateway, car le service a le même nom pour logging et logging-client).

Solution 2 :
Déployer Nginx Gateway dans la zone cliente



Solution 2 : Exemple de configuration Nginx Gateway

```
1
2   server {
3     listen               80;
4     location = /loki/api/v1/tail {
5       proxy_pass https://rancher.ocb-test06.caascad.com/k8s/clusters/local/api/v1/namespaces/logging-
  client/services/querier:3100/proxy$request_uri;
6       proxy_set_header Upgrade $http_upgrade;
7       proxy_set_header Connection "upgrade";
8       proxy_set_header Authorization "Bearer xxxxxxxx";
9     }
10    location ~ /loki/api/.* {
11      proxy_pass https://rancher.ocb-test06.caascad.com/k8s/clusters/local/api/v1/namespaces/logging-
  client/services/querier:3100/proxy$request_uri;
12      proxy_set_header Authorization "Bearer xxxxxxxx";
13    }
14  }
15 }
```

Solution 2 : Exemple de datasource

```yaml
apiVersion: v1
data:
  datasource_loki_gateway_client.yaml: |
    apiVersion: 1
    datasources:
    - name: "Loki Gateway client"
      type: "loki"
      access: proxy
      orgId: 1
      url: "http://gateway.caascad-logging.svc.cluster.local"
kind: ConfigMap
metadata:
  labels:
    grafana-datasource: "1"
  name: grafana-datasource-loki-gateway-client
  namespace: pf-1054
```

Solution 2 :
Déployer Nginx Gateway dans la zone cliente

Dans cette solution les tâches à faire sont :

- création d'un user Rancher spécifiquement pour ce cas d'usage : idem que la solution 1 ;

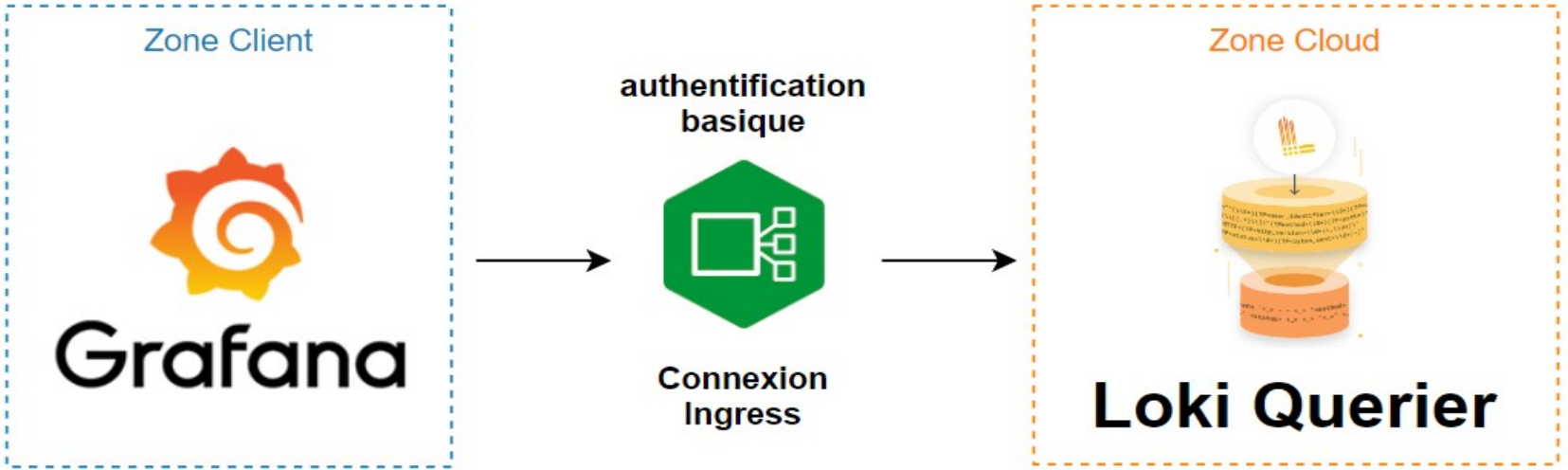
- déploiement d'un Nginx Gateway en zone cliente :

  - nouvelle configuration envs-ng : on peut utiliser la chart Loki et activer seulement le composant Nginx Gateway (on désactive tout le reste) ;

  - point à faire attention : token Rancher dans la configuration de Nginx (secret au lieu de configmap ?) ;

  - tests fonctionnels : pouvoir lancer le test fonctionnel pour le déploiement Nginx Gateway seulement sur le cluster client prdgso et pas sur les autres zones.

Solution 3 :
Donner accès à Loki Querier dans la zone Cloud



Solution 3 : Exemple de configuration Ingress

```yaml
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  annotations:
    cert-manager.io/cluster-issuer: zerossl
    kubernetes.io/ingress.class: ingress-nginx-public
    nginx.ingress.kubernetes.io/auth-realm: Enter your credentials
    nginx.ingress.kubernetes.io/auth-secret: loki-basic-auth
    nginx.ingress.kubernetes.io/auth-type: basic
  name: loki-querier
  namespace: logging-client
spec:
  rules:
  - host: loki-querier.ocb-test06.caascad.com
    http:
      paths:
      - backend:
          serviceName: querier
          servicePort: 3100
        path: /loki/api
        pathType: Prefix
  tls:
  - hosts:
    - loki-querier.ocb-test06.caascad.com
    secretName: mon-loki-querier-tls
---
apiVersion: v1
stringData:
  auth: demo:$apr1$afTorrh7$kChCke6xCsl8/HSbeBFXH.
kind: Secret
type: Opaque
metadata:
  name: loki-basic-auth
  namespace: logging-client
```

Solution 3 : Exemple de datasource

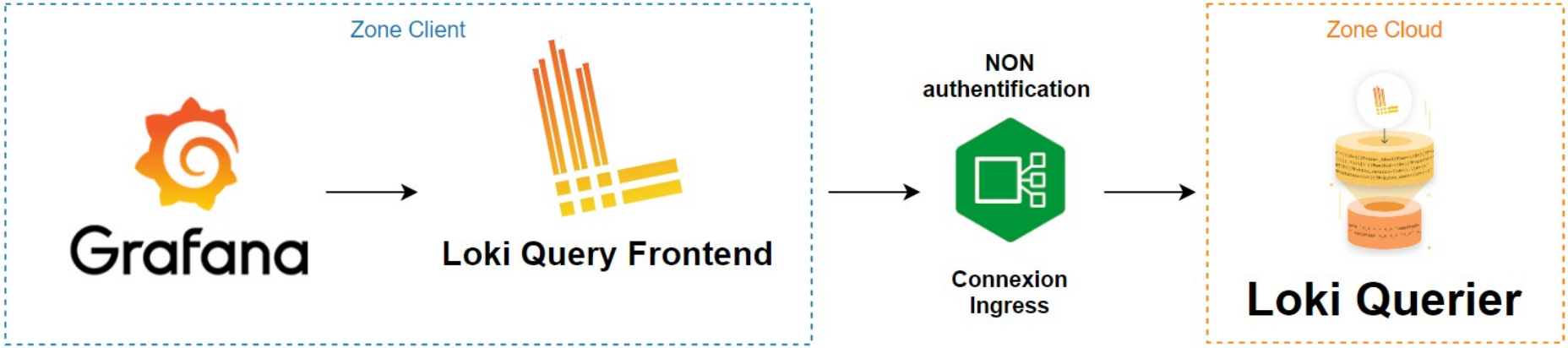
```yaml
apiVersion: v1
stringData:
  datasource_loki_querier_ingress_cloud.yaml: |
    apiVersion: 1
    datasources:
    - name: "Loki Querier Ingress cloud"
      type: "loki"
      access: proxy
      orgId: 1
      url: "https://loki-querier.ocb-test06.caascad.com"
      basicAuthUser: demo
      basicAuth: true
      secureJsonData:
        basicAuthPassword: xxxx
kind: Secret
type: Opaque
metadata:
  labels:
    grafana-datasource: "1"
  name: grafana-datasource-loki-querier-ingress-cloud
  namespace: pf-1054
```

Solution 3 :
Donner accès à Loki Querier dans la zone Cloud

Dans cette solution les tâches à faire sont :

- création d'un secret contenant le user/password pour l'ingress ;
- déploiement d'un ingress pour Loki Querier en zone cloud.

Solution 4 :
Déployer Loki Query Frontend dans la zone cliente



Solution 4 : Exemple de Datasource

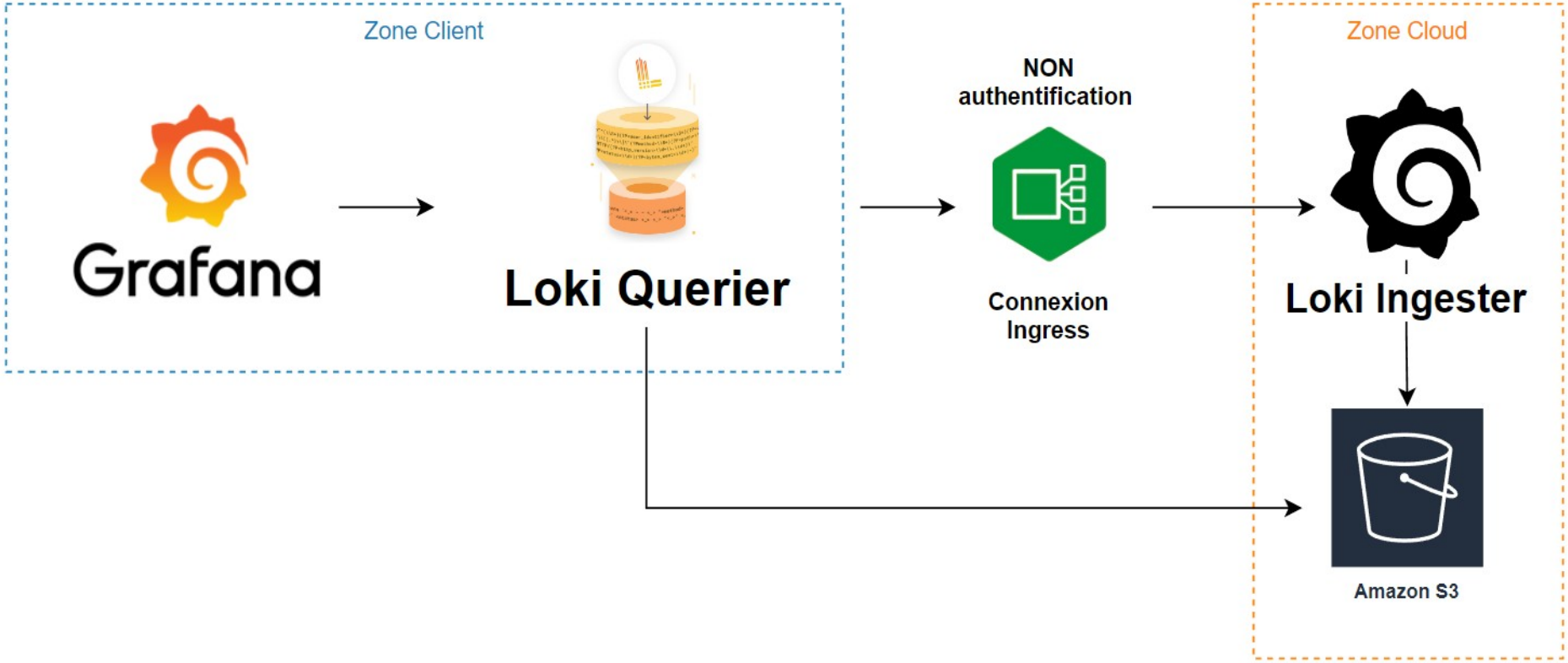
```yaml
apiVersion: v1
data:
  datasource_loki_query_frontend.yaml: |
    apiVersion: 1
    datasources:
    - name: "Loki Query Frontend"
      type: "loki"
      access: proxy
      orgId: 1
      url: "http://query-frontend.pf-1054.svc.cluster.local:3100"
kind: ConfigMap
metadata:
  labels:
    grafana-datasource: "1"
  name: grafana-datasource-loki-query-frontend
  namespace: pf-1054
```

Solution 4 :
Déployer Loki Query Frontend dans la zone cliente

Dans cette solution les tâches à faire sont :

- déploiement d'un Loki Query Frontend en zone cliente :
    - nouvelle configuration envs-ng : on peut utiliser la chart Loki, créer des nouveaux templates, et activer seulement ce composant de Loki ;
    - tests fonctionnels : pouvoir lancer le test fonctionnel pour le déploiement Loki Query Frontend seulement sur le cluster client prdgso et pas sur les autres zones.
- déploiement d'un ingress pour Loki Querier en zone cloud.

Solution 5 :
Déployer Loki Querier dans la zone cliente



Comparaison

Solutions	Méthode de connexion	Rapide à mettre en place	Performances	Accès au cœur du logiciel	Cloisonnement par rapport aux autres applications
Solution 1	+ +	+ +	−	−	−
Grafana zone cliente -> Nginx Gateway zone cloud -> Loki Querier zone cloud	Rancher (authentification avec un Token)	Presque tout est prêt	Performance actuelle	Accès direct aux composants	Selon la configuration Rancher, + ou - d'accès sur les services de la zone cloud
Solution 2	+ +	+	−	+	−
Grafana zone cliente -> Nginx Gateway zone cliente -> Loki Querier zone cloud	Rancher (authentification avec un Token)	Déploiement de Loki Gateway	Performance actuelle	Accès via Nginx	Selon la configuration Rancher, + ou - d'accès sur les services de la zone cloud
Solution 3	+	+ +	−	−	+
Grafana zone cliente -> Loki Querier zone cloud	Ingress (authentification basique)	Presque tout est prêt	Performance actuelle	Accès direct à Loki Querier	Le client n'a accès qu'à Loki Querier (logging-client)
Solution 4	−	−	+	+	+
Grafana zone cliente -> Loki Query Frontend -> Loki Querier zone cloud	Ingress (pas d'authentification)	Déploiement de Loki Query Frontend	Plus rapide	Accès via Loki Query Frontend	Le client n'a accès qu'à Loki Querier (logging-client)

Merci