**Seguimiento de Tráfico Aéreo en Tiempo Real con Kafka y Kubernetes**

Este proyecto implementa un sistema de seguimiento de tráfico aéreo en tiempo real utilizando datos de la API de OpenSky. La arquitectura se basa en microservicios Node.js que se comunican a través de Kafka, desplegados en Kubernetes y con escalado automático mediante KEDA.

Interfaz de usuario gráfica

El contenido generado por IA puede ser incorrecto.

**Requisitos Previos**

Antes de comenzar, necesitarás tener instalados:

1. **Kubernetes Cluster** - Puede ser Minikube, Kind, o un clúster gestionado como EKS, AKS o GKE
2. **kubectl** - Para interactuar con el clúster
3. **Helm 3** - Para instalar operadores como Strimzi (Kafka) y KEDA
4. **Docker** - Para construir imágenes si necesitas modificar el código

**Estructura del Proyecto**

opensky-kafka-keda/

├── kubernetes/

│ ├── namespace-proyecto.yaml # Definición del namespace

│ ├── configmap-proyecto.yaml # Configuración de la aplicación

│ ├── secret-proyecto.yaml # Secretos (credenciales)

│ ├── kafka-cluster.yaml # Definición del cluster Kafka (Strimzi)

│ ├── kafka-topic.yaml # Tópico para los datos de vuelo

│ ├── producer-deployment.yaml # Despliegue del productor

│ ├── consumer-deployment.yaml # Despliegue del consumidor

│ ├── expose-opensky-producer.yaml # Servicios para exponer componentes

│ ├── keda-opensky-consumer-scaler.yaml # Configuración de autoescalado del consumidor

│ ├── keda-opensky-producer-scaler.yaml # Configuración de autoescalado del productor (opcional)

│ ├── opensky-producer-monitor.yaml # ServiceMonitor para Prometheus (observabilidad)

│ ├── opensky-keda-alerts.yaml # Reglas de alertas en Prometheus

│ └── opensky-grafana-dashboard.yaml # Dashboard de Grafana

└── README.md # Este archivo

**Guía de Despliegue**

**Paso 1: Instalar Operadores Necesarios**

Primero, necesitamos instalar los operadores que gestionarán Kafka y el autoescalado:

# Crear un namespace para los operadores (opcional pero recomendado)

kubectl create namespace kafka

kubectl create namespace operators

# Instalar el operador Strimzi para Kafka

helm repo add strimzi https://strimzi.io/charts/

helm repo update

helm install strimzi-kafka-operator strimzi/strimzi-kafka-operator

--namespace kafka --create-namespace

helm install strimzi-kafka-operator strimzi/strimzi-kafka-operator \

--namespace operators --create-namespace

**helm install strimzi strimzi/strimzi-kafka-operator --version** 0.45.0 **-n** kafka **--create-namespace**

# Instalar KEDA para el autoescalado basado en eventos

helm repo add kedacore https://kedacore.github.io/charts

helm repo update

helm install keda kedacore/keda

--namespace kafka --create-namespace

kubectl delete scaledobject.keda.sh opensky-consumer-scaler -n opensky --force --grace-period=0

for so in $(kubectl get scaledobjects.keda.sh -A --no-headers | awk '{print $2" -n "$1}'); do

kubectl patch scaledobject.keda.sh $so -p '{"metadata":{"finalizers":[]}}' --type=merge;

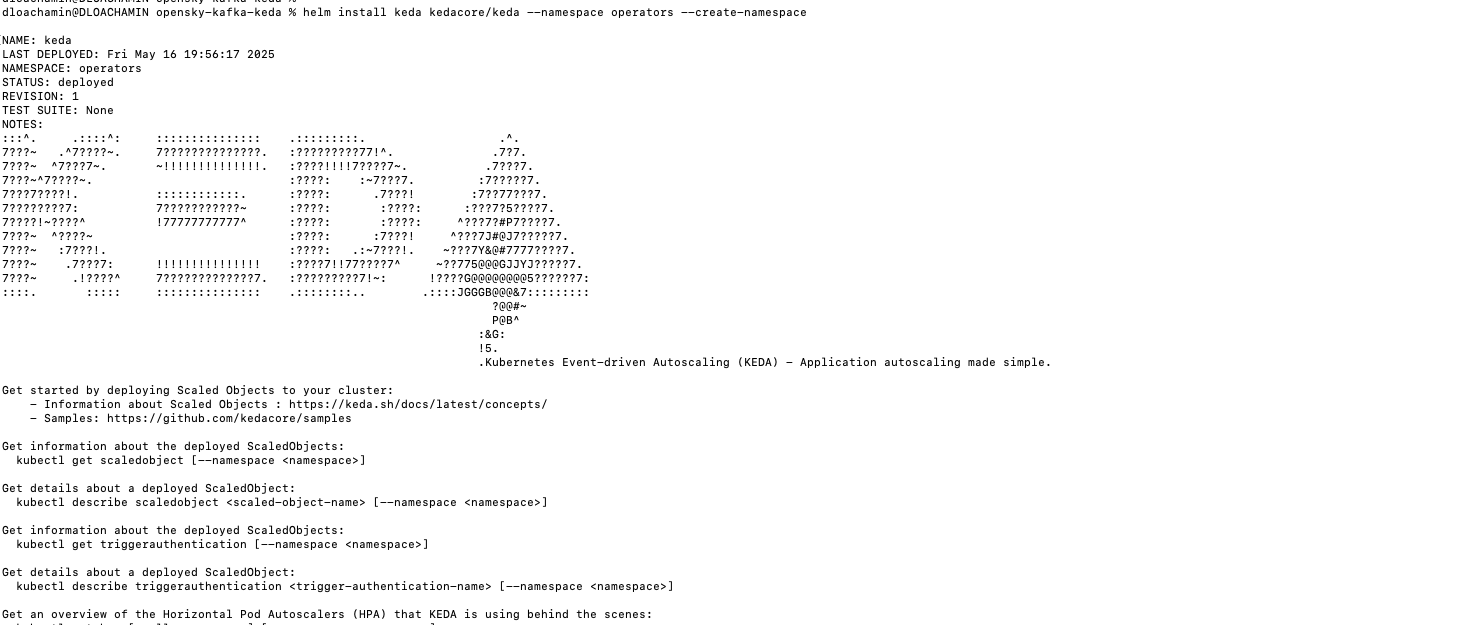
kubectl delete scaledobject.keda.sh $so --force --grace-period=0;

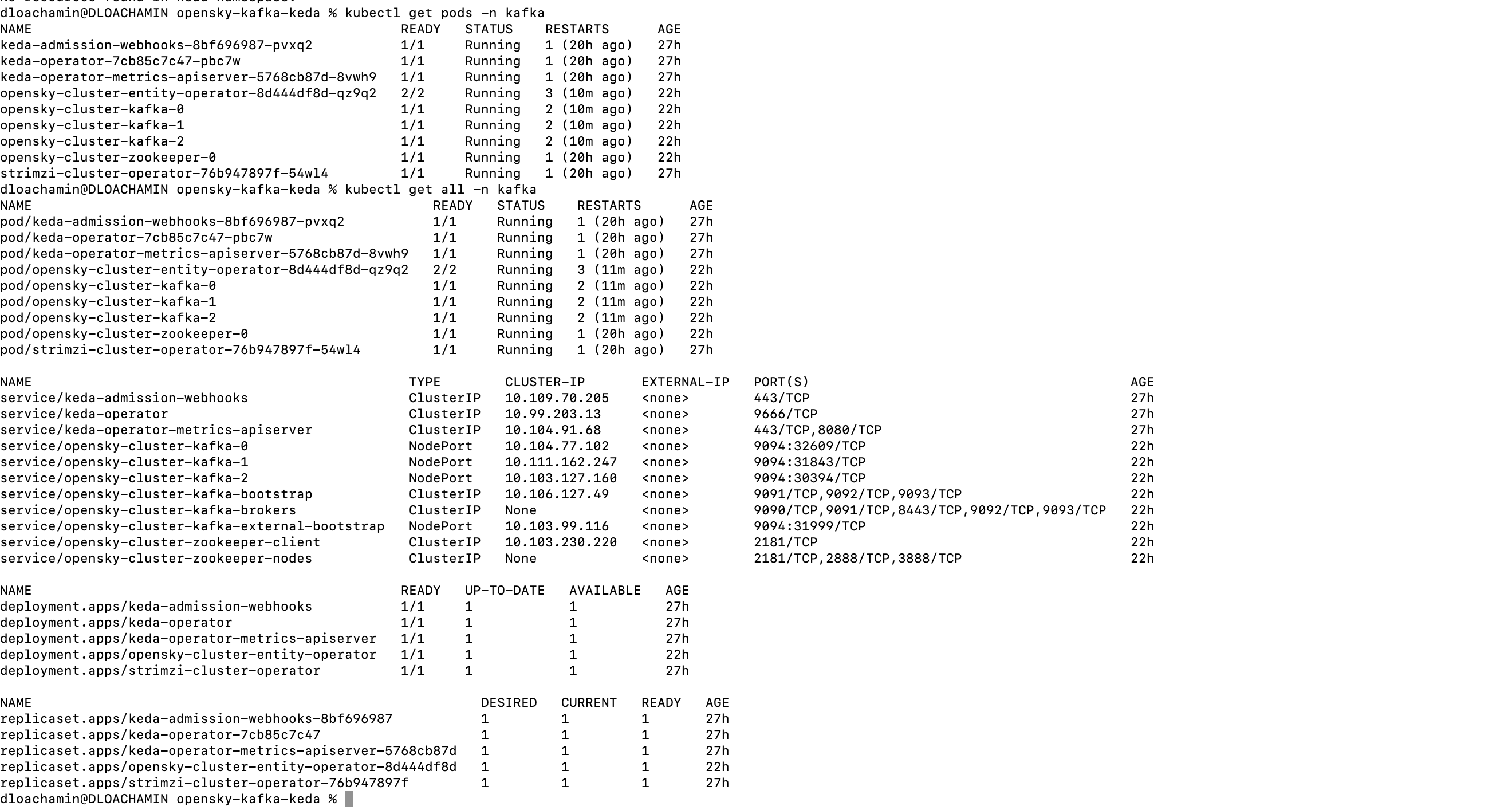
done

helm install keda kedacore/keda

--namespace operators --create-namespace

**helm uninstall** keda **-n operators**





# Verificar que los operadores estén instalados correctamente

kubectl get pods -n kafka

kubectl get pods -n operators

ver los logs

kubectl logs -n kafka strimzi-cluster-operator-76b947897f-m6jcq --follow

**Paso 2: Configurar el Namespace y Recursos Básicos**

Ahora, configuraremos el namespace para nuestro proyecto y aplicaremos los recursos básicos:

# Crear el namespace para el proyecto

kubectl apply -f kubernetes/namespace-proyecto.yaml

# Crear ConfigMap con configuración de la aplicación

kubectl apply -f kubernetes/configmap-proyecto.yaml

# Crear Secret con credenciales (si es necesario)

kubectl apply -f kubernetes/secret-proyecto.yaml

**Paso 3: Desplegar Kafka con Strimzi**

Ahora desplegaremos un clúster de Kafka utilizando el operador Strimzi:

# Desplegar el clúster Kafka

kubectl apply -f kubernetes/kafka-cluster.yaml

# Esperar a que el clúster esté listo (esto puede tardar unos minutos)

kubectl wait kafka/opensky-cluster --for=condition=Ready --timeout=300s -n opensky

Ver el estado de Kafka

kubectl get kafka -A

**1. Verifica el NodePort asignado por Kubernetes**

kubectl get svc -n Kafka

Busca un servicio como este:

opensky-cluster-kafka-external-bootstrap NodePort 10.96.x.x <none> 9094:32XXX/TCP

**2. Redirige el puerto con minikube service**

minikube service opensky-cluster-kafka-external-bootstrap -n kafka –url

Esto te dará una URL como:

<http://192.168.49.2:32XXX>

**3. Configura tu cliente Kafka**

kafka-console-producer.sh --broker-list 192.168.49.2:32XXX --topic test

o

kafka-console-consumer.sh --bootstrap-server 192.168.49.2:32XXX --topic test --from-beginning

1. En advertised.listeners, Strimzi configurará automáticamente algo como:

PLAINTEXT://opensky-cluster-kafka-0.opensky-cluster-kafka-brokers.kafka.svc:9092

EXTERNAL://<node-ip>:<nodeport>

# Crear el tópico Kafka para los datos de vuelo

kubectl apply -f kubernetes/kafka-topic.yaml

# Verificar que el clúster y el tópico estén creados

kubectl get kafka -n opensky

kubectl get kafkatopic -n opensky

**Paso 4: Desplegar Productor y Consumidor**

Con Kafka funcionando, podemos desplegar los microservicios que producen y consumen datos:

# Desplegar el productor que obtiene datos de OpenSky

kubectl apply -f kubernetes/producer-deployment.yaml

# Desplegar el consumidor que procesa los datos

kubectl apply -f kubernetes/consumer-deployment.yaml

# Exponer los servicios

kubectl apply -f kubernetes/expose-opensky-producer.yaml

# Verificar que los pods estén funcionando

kubectl get pods -n opensky



**Paso 5: Configurar Autoescalado con KEDA**

Ahora vamos a configurar KEDA para escalar automáticamente el consumidor basado en el lag de Kafka:

# Aplicar la configuración de escalado para el consumidor

kubectl apply -f consumidor/keda-opensky-consumer-scaler.yaml

# Opcionalmente, si quieres escalar también el productor

# kubectl apply -f productor/keda-opensky-producer-scaler.yaml

# Verificar que los ScaledObjects estén creados

kubectl get scaledobjects -n opensky

🎶 

# Verificar los HorizontalPodAutoscalers creados por KEDA

kubectl get hpa -n opensky



**Paso 6: Configurar Observabilidad (Opcional)**

Para monitorear el sistema, podemos instalar Prometheus y Grafana:

# Instalar Prometheus Operator (si no está instalado)

helm repo add prometheus-community https://prometheus-community.github.io/helm-charts

helm repo update

helm install prometheus prometheus-community/kube-prometheus-stack \

--namespace monitoring --create-namespace

helm status prometheus -n monitoring

A white background with black text

AI-generated content may be incorrect.

# Aplicar ServiceMonitor para recolectar métricas

kubectl apply -f productor/opensky-producer-monitor.yaml

# Aplicar reglas de alertas

kubectl apply -f productor/opensky-keda-alerts.yaml

# Aplicar dashboard de Grafana

kubectl apply -f productor/opensky-grafana-dashboard.yaml

# Acceder a Grafana (portforward)

kubectl port-forward svc/prometheus-grafana 3000:80 -n monitoring

# Usuario por defecto: admin / Contraseña: prom-operator

**Resultado Esperado**

Una vez desplegado correctamente, deberías tener:

1. Un productor consultando la API de OpenSky para obtener datos de tráfico aéreo en tiempo real
2. Un clúster Kafka recibiendo y almacenando estos datos en un tópico
3. Un consumidor procesando los datos del tópico Kafka
4. KEDA escalando automáticamente el consumidor según la carga de mensajes
5. (Opcional) Monitorización con Prometheus y Grafana

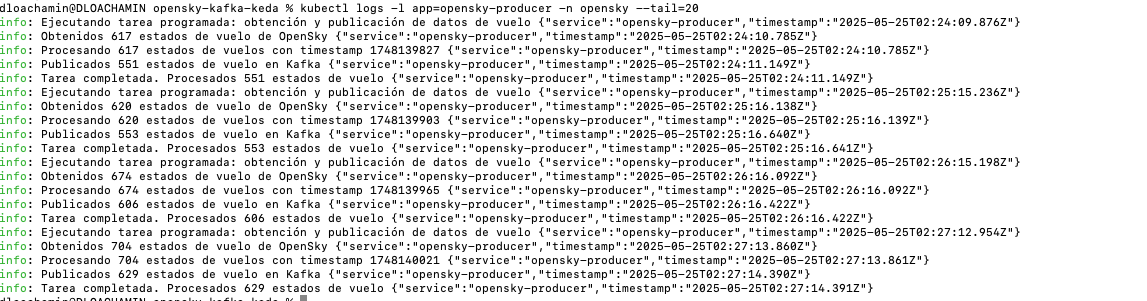
**Validación y Pruebas**

Para verificar que todo funciona correctamente:

**1. Verificar que el productor está obteniendo datos**

# Ver logs del productor

kubectl logs -l app=opensky-producer -n opensky --tail=20



# Deberías ver mensajes indicando que se están obteniendo datos de OpenSky y publicando a Kafka

**2. Verificar que el consumidor está procesando datos**

# Ver logs del consumidor

kubectl logs -l app=opensky-consumer -n opensky --tail=20



# Deberías ver mensajes indicando que se están procesando datos de Kafka

**3. Verificar el autoescalado**

Para probar el autoescalado, puedes:

# Ver el estado actual del HPA

kubectl get hpa -n opensky



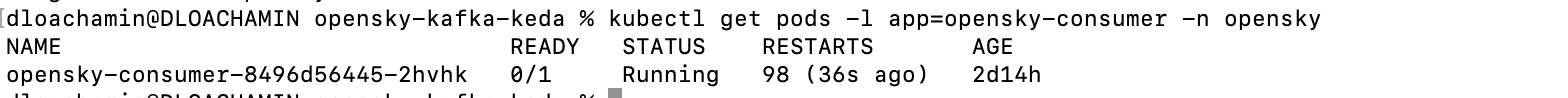
# Verificar el comportamiento de escalado

kubectl describe scaledobject opensky-consumer-scaler -n opensky



# Verificar el número de réplicas a lo largo del tiempo

watch kubectl get pods -l app=opensky-consumer -n opensky



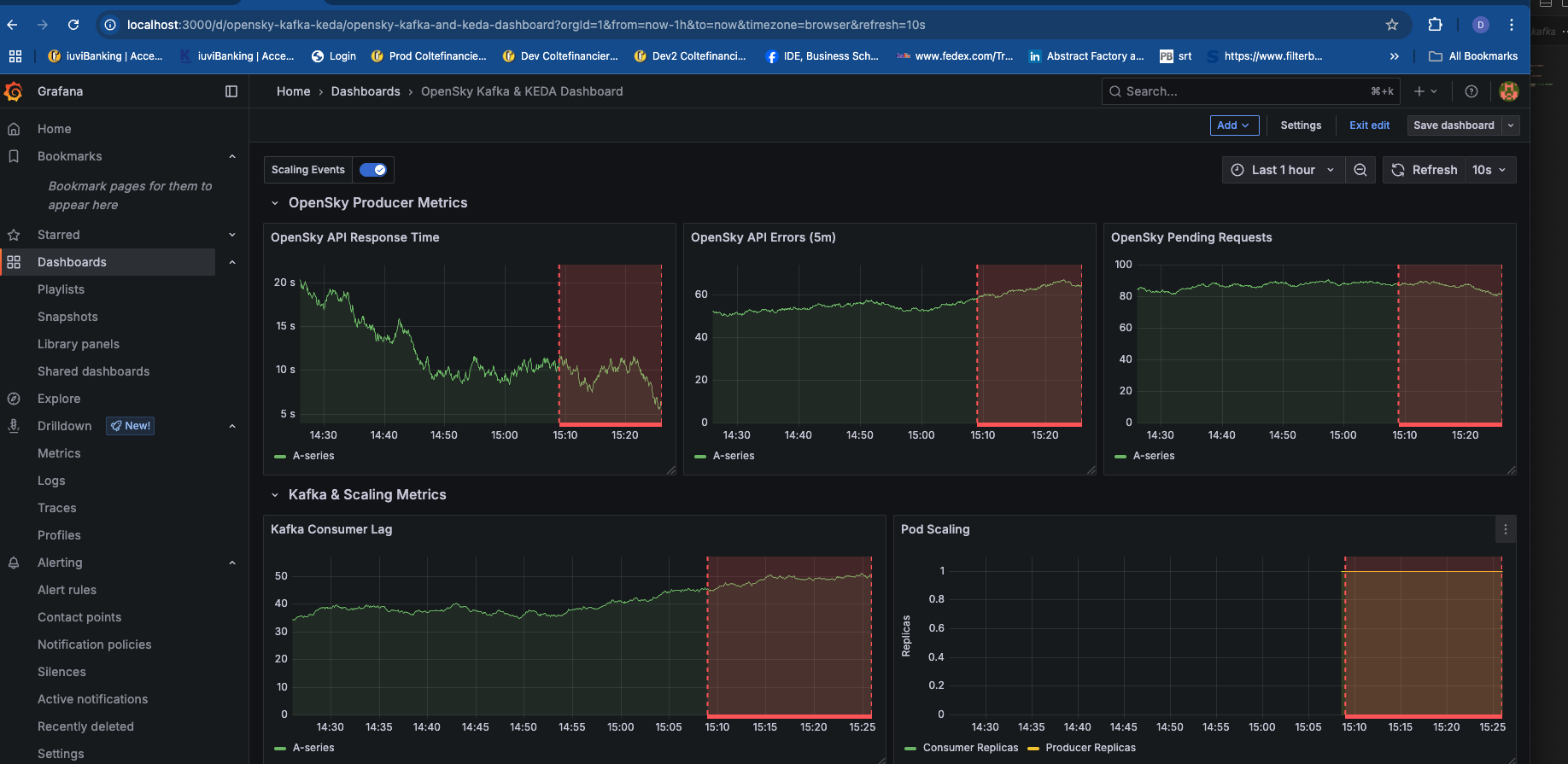
**4. Verificar la monitorización (si está configurada)**

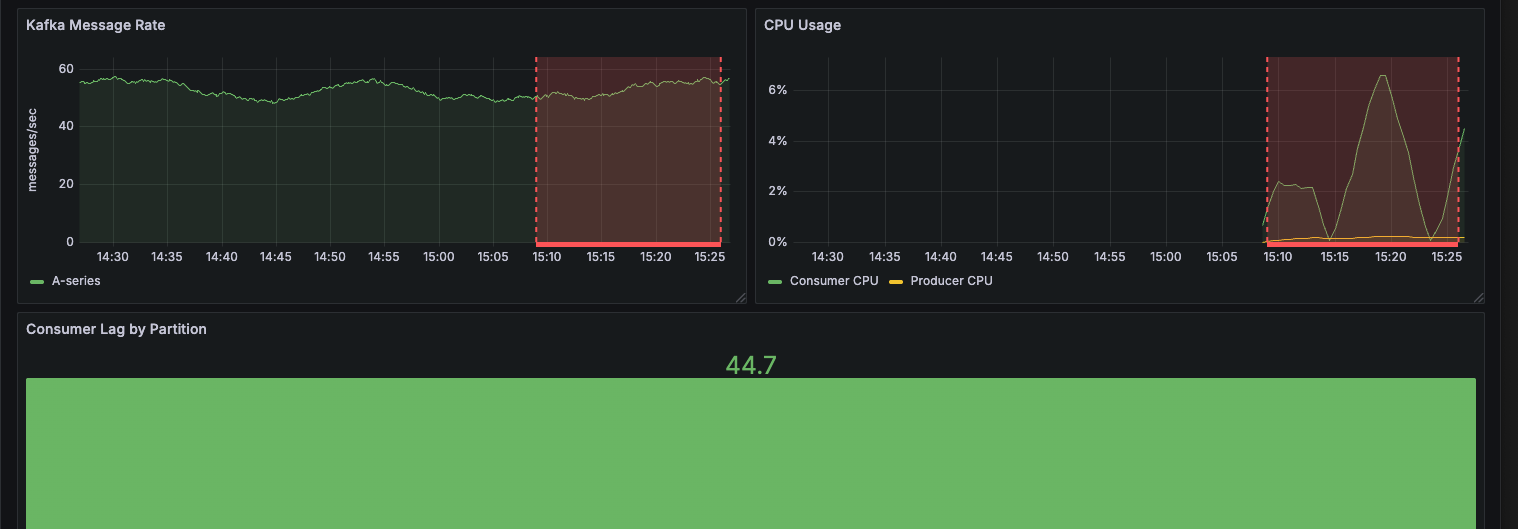
# Acceder a Grafana

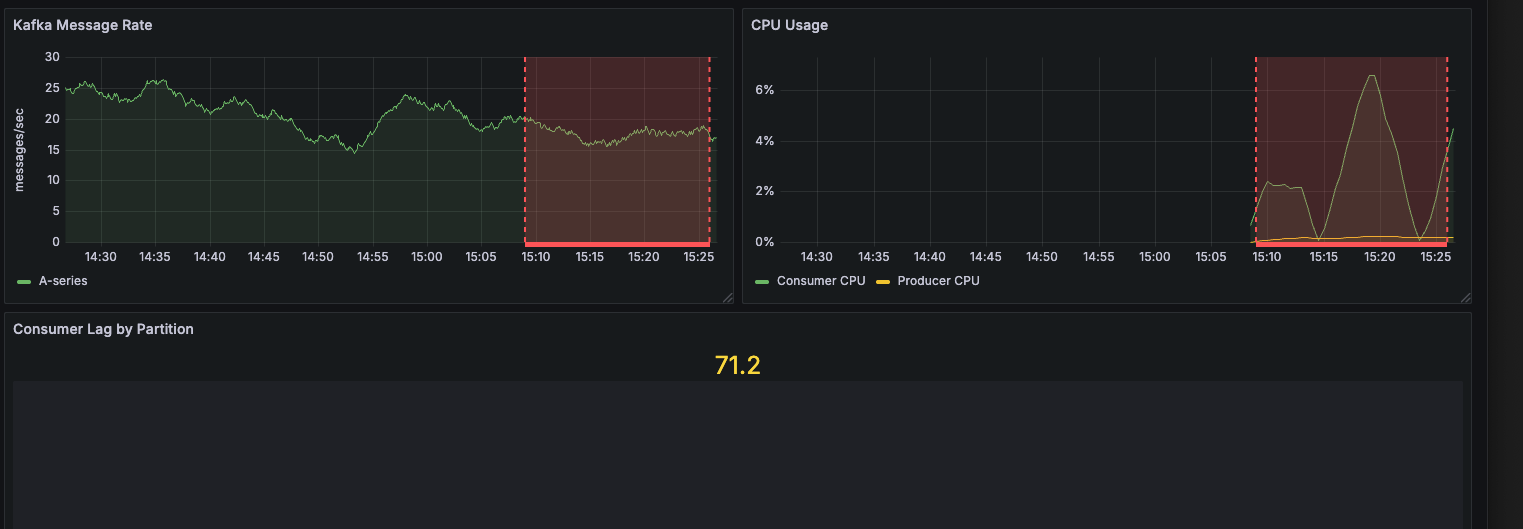
kubectl port-forward svc/prometheus-grafana 3000:80 -n monitoring

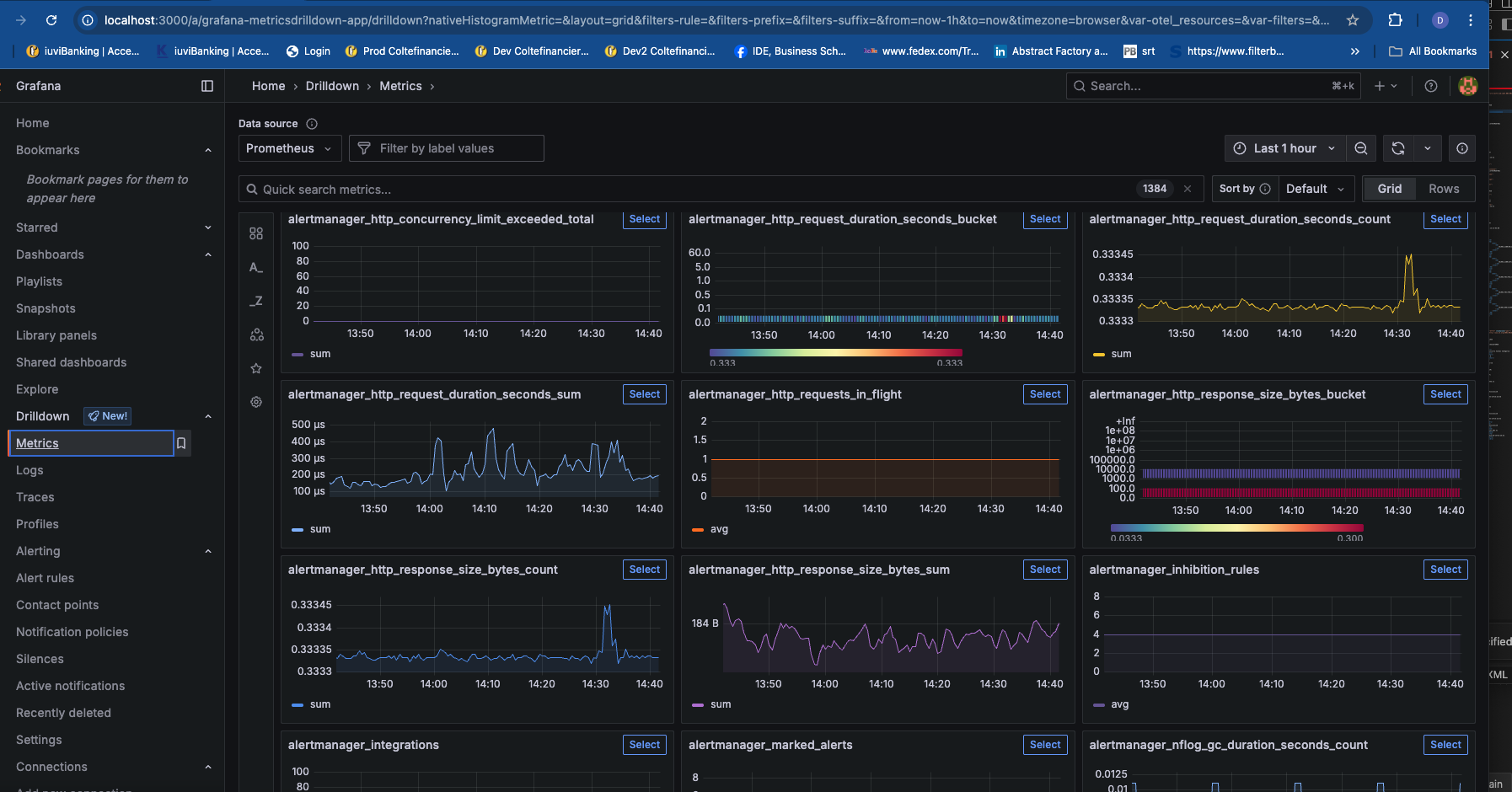


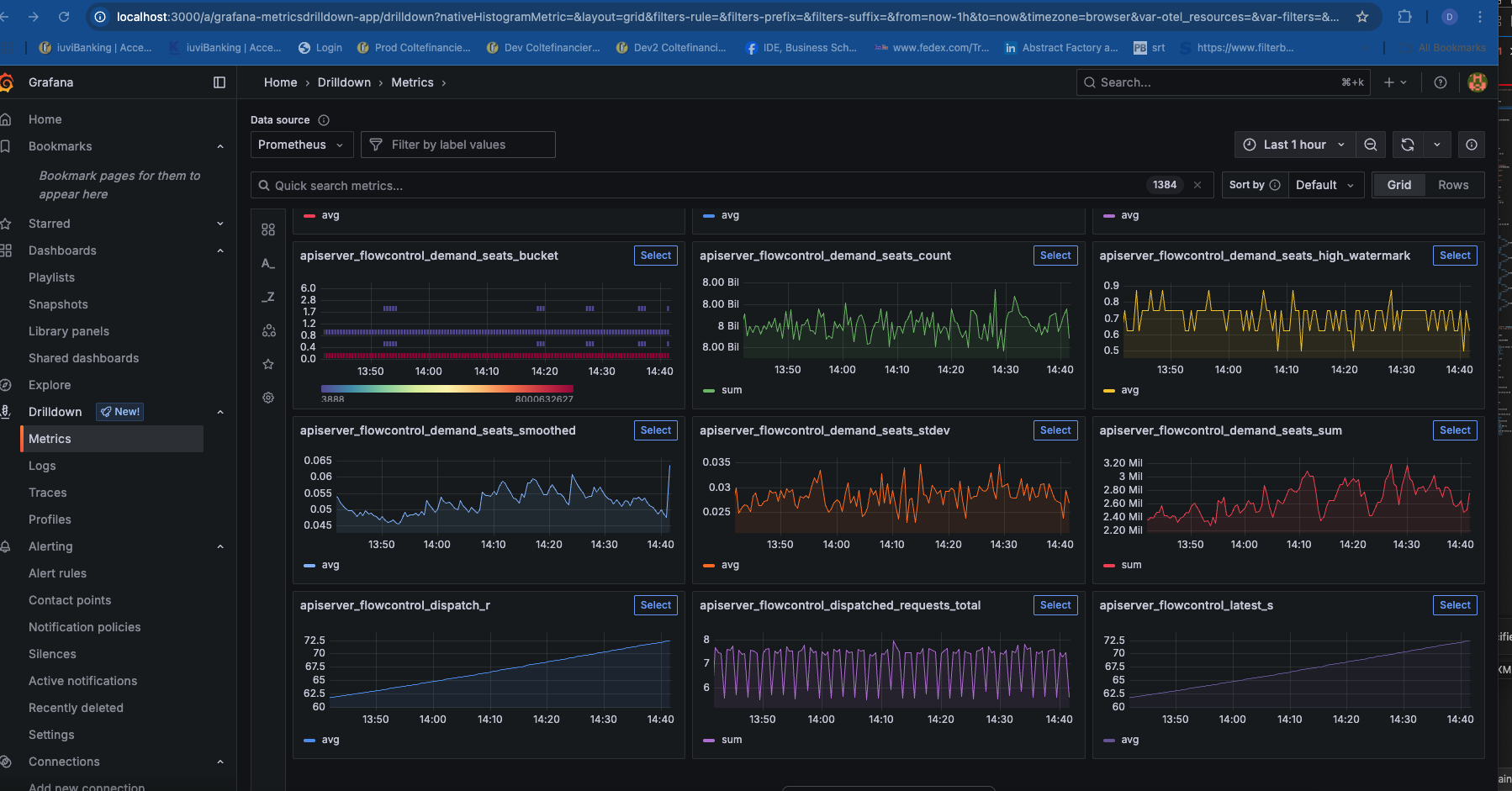
# En tu navegador, abre http://localhost:3000 y busca el dashboard "OpenSky Kafka & KEDA"











**Consejos y Solución de Problemas**

* **Error 429 en el productor**: La API de OpenSky tiene límites de tasas estrictos. Si ves errores 429, aumenta el intervalo de consulta en el ConfigMap.
* **Problemas con KEDA**: Asegúrate de que la configuración de ScaledObject está correcta y que KEDA tiene permisos para acceder a las métricas de Kafka.
* **Kafka no inicia**: Verifica los recursos disponibles en el clúster. Kafka puede necesitar bastante memoria y CPU.
* **Monitorización no funciona**: Asegúrate de que los ServiceMonitor están en el mismo namespace que los pods o tienen etiquetas correctas para ser seleccionados.

**Referencias y Recursos Adicionales**

* [Documentación de Strimzi](https://strimzi.io/documentation/)
* [Documentación de KEDA](https://keda.sh/docs/)
* [API de OpenSky](https://openskynetwork.github.io/opensky-api/index.html)
* [Arquitectura de Microservicios con Kafka](https://microservices.io/patterns/data/event-driven-architecture.html)