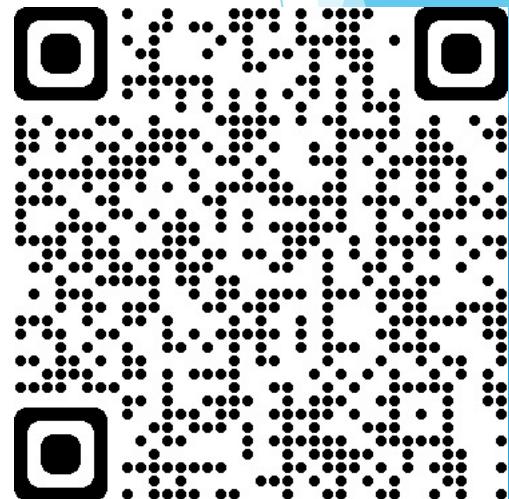


Estudio de frameworks y técnicas serverless en Kubernetes

Autor: Óscar Illescas Jiménez

Tutor: Micael Gallego Carrillo



Objetivo

- ↑ Encontrar y probar al menos un sustituto en Kubernetes para cada uno de los servicios que componen el ecosistema Serverless.



Funciones como servicio (FaaS)

Cloud

- ↑  AWS Lambda
- ↑  Azure Functions
- ↑  Google Cloud Functions
- ↑ ...

Kubernetes

- ↑ Knative
- ↑ OpenFaas

Knative

- ↑ Instalación
- ↑ Ejemplos



Knative

↑ Instalación

```
kubectl apply --filename https://github.com/knative/serving/releases/download/v0.15.0/serving-crds.yaml
kubectl apply --filename https://github.com/knative/serving/releases/download/v0.15.0/serving-core.yaml
kubectl apply --filename https://github.com/knative/net-istio/releases/download/v0.15.0/release.yaml
kubectl apply --filename https://github.com/knative/eventing/releases/download/v0.16.0/eventing-crds.yaml
kubectl apply --filename https://github.com/knative/eventing/releases/download/v0.16.0/eventing-core.yaml
kubectl apply --filename https://github.com/knative/eventing/releases/download/v0.16.0/in-memory-channel.yaml
kubectl apply --filename https://github.com/knative/eventing/releases/download/v0.16.0/mt-channel-broker.yaml
kubectl apply --filename https://github.com/knative/serving/releases/download/v0.16.0/monitoring-core.yaml
kubectl apply --filename https://github.com/knative/serving/releases/download/v0.16.0/monitoring-metrics-prometheus.yaml
```



Knative

↑ Hola mundo

Ejemplo básico de una función en nodejs con express.
Un pequeño api REST simulando una bbdd en memoria.



<https://github.com/MasterCloudApps-Projects/Serverless-Kubernetes/tree/main/1.faas/knative#rest-api>

Knative

↑ Hola mundo

```
apiVersion: serving.knative.dev/v1
kind: Service
metadata:
  name: knative-rest
  namespace: default
spec:
  template:
    spec:
      containers:
        - image: docker.io/oillescas/knative-rest
```

<https://github.com/MasterCloudApps-Projects/Serverless-Kubernetes/tree/main/1.faas/knative#rest-api>



Funciones como servicio (FaaS)

OpenFaas

- ↑ Instalación
- ↑ Custom Templates
- ↑ Ejemplos
 - ↑ Hello-world node
 - ↑ Hello-world Java



OpenFaaS

↑ Instalación

```
git clone https://github.com/openfaas/faas-netes

# generate a random password
PASSWORD=$(head -c 12 /dev/urandom | shasum| cut -d' ' -f1)

kubectl -n openfaas create secret generic basic-auth \
--from-literal=basic-auth-user=admin \
--from-literal=basic-auth-password="$PASSWORD"

cd faas-netes && \
kubectl apply -f namespaces.yml
kubectl apply -f ./yaml
```



OpenFaaS

↑ Custom templates

↑ Node12-files

Partiendo de la plantilla node12 hemos hecho una modificación para integrar un plugin de expressjs para recibir archivos en la request además de aumentar los timeouts.

↑ Node12-nats

También partiendo de la plantilla node12 hemos modificado el tratamiento de la cabecera para facilitar la lectura de los mensajes de una cola Nats



OpenFaaS

↑ Hola mundo

Ejemplo básico procedente de la plantilla de node12.

Devuelve en un json el body de la petición original.



<https://github.com/MasterCloudApps-Projects/Serverless-Kubernetes/tree/main/1.faas/OpenFaaS#hello-world>

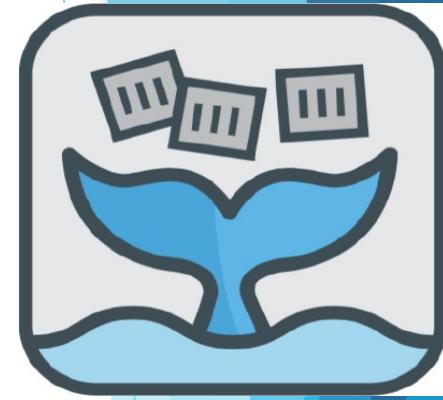
OpenFaaS

↑ Hello-world

```
'use strict'

module.exports = async (event, context) => {
  const result = {
    'status': 'Received input: ' + JSON.stringify(event.body)
  }

  return context
    .status(200)
    .succeed(result)
}
```



<https://github.com/MasterCloudApps-Projects/Serverless-Kubernetes/tree/main/1.faas/OpenFaaS#hello-world>

OpenFaaS

↑ hello-java

Ejemplo básico procedente de la plantilla de java11.

Devuelve en el texto “Hello, world!” en el body de la respuesta.



<https://github.com/MasterCloudApps-Projects/Serverless-Kubernetes/tree/main/1.faas/OpenFaaS#hello-java>

OpenFaaS

↑ hello-java

```
package com.openfaas.function;

import com.openfaas.model.IHandler;
import com.openfaas.model.IResponse;
import com.openfaas.model.IRequest;
import com.openfaas.model.Response;

public class Handler extends com.openfaas.model.AbstractHandler
{
    public IResponse Handle(IRequest req) {
        Response res = new Response();
        res.setBody("Hello, world!");

        return res;
    }
}
```



<https://github.com/MasterCloudApps-Projects/Serverless-Kubernetes/tree/main/1.faas/OpenFaaS#hello-java>

OpenFaaS

↑ hello-java



```
version: 1.0
provider:
  name: openfaas
functions:
  hello-java:                                //Nombre de la función
    lang: java11                               //Planilla de la que partimos
    handler: ./hello-java                      //Path del handler
    image: oillescas/hello-java:latest          //Nombre de la imagen docker que se creará
```

<https://github.com/MasterCloudApps-Projects/Serverless-Kubernetes/tree/main/1.faas/OpenFaaS#hello-java>

Gestión de ficheros

Cloud

- ↑  AWS S3
- ↑  Azure Blob Storage
- ↑  Google Cloud Storage
- ↑ ...

Kubernetes

- ↑  Minio

Gestión de ficheros

Minio

↑ Instalación

↑ Ejemplos

↑ Static-site

↑ REST OpenFaaS function



Minio

↑ Instalación

```
#Instalación con helm  
helm repo add minio https://helm.min.io/  
helm install --set service.type=LoadBalancer my-minio minio/minio
```



Minio

↑ Static Site

En este ejemplo subimos una web estática a un bucket de minio

lo configuramos para poder usar el servicio de minio detrás de un ingress.

Y configuramos el ingress para que sirva esta web estática.



Minio

Static Site

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: simpleapp-ingress
  annotations:
    nginx.ingress.kubernetes.io/rewrite-target: "/static-site/$1"
    nginx.ingress.kubernetes.io/configuration-snippet: |
      rewrite ^/?$ /static-site/index.html break;
spec:
  # ingressClassName: nginx
  rules:
  - host: miniostatic.192.168.0.100.nip.io
    http:
      paths:
      - path: /(.*)
        pathType: Prefix
        backend:
          service:
            name: my-minio
            port:
              number: 9000
```



Minio

↑ Minio api REST

En este ejemplo desarrollamos una función basada en la custom template node12-files.

Usa el api de minio para listar los buckets, listar los objetos de un bucket eliminar objetos de un bucket y para subir nuevos objetos a un bucket.



Minio

Minio api REST

```
version: 1.0
provider:
  name: openfaas
functions:
  openfaas-minio-api:
    lang: node12-files
    handler: ./minio-api
    image: oillescas/minio-api:latest
    environment:
      endpoint: my-minio.default.svc.cluster.local
      minio-port: 9000
      use-ssl: false
    secrets:
      - openfaas-minio
```



Bases de datos

Cloud

- ↑  AWS DynamoDB
- ↑  Azure CosmosDB
- ↑  Google Cloud Spanner
- ↑ ...

Kubernetes

- ↑ MongoDB
- ↑ PostgreSQL

PostgreSQL by Zalando



Instalación

```
# Clonamos el repositorio
git clone https://github.com/zalando/postgres-operator.git
cd postgres-operator

# Instalamos el operador de PostgreSQL con helm
helm install postgres-operator ./charts/postgres-operator -f ./charts/
postgres-operator/values-crd.yaml

# Desplegamos la interfaz web (UI)
# Instalamos la UI con con helm
helm install postgres-operator-ui ./charts/postgres-operator-ui

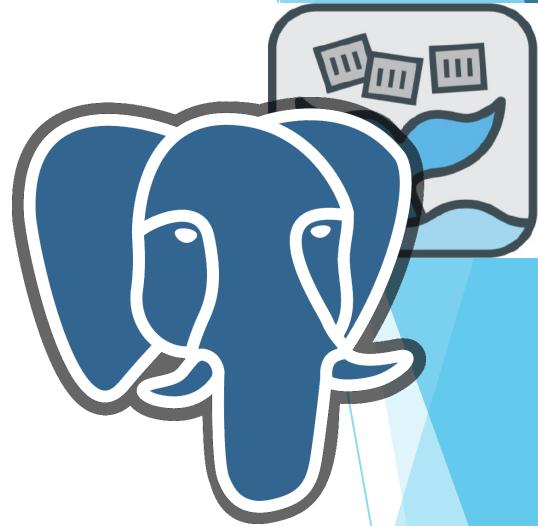
# Abrimos el puerto a la UI con port-forward
kubectl port-forward svc/postgres-operator-ui 8081:8081
```



PostgreSQL by Zalando

↑ Despliegue de un cluster

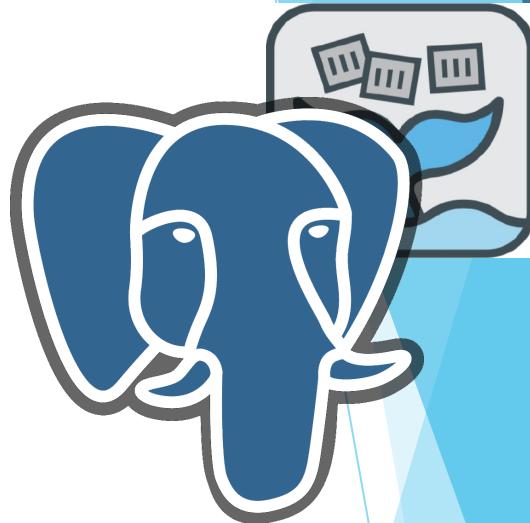
```
apiVersion: "acid.zalan.do/v1"
kind: postgresql
metadata:
  name: acid-minimal-cluster
  namespace: default
spec:
  teamId: "acid"
  volume:
    size: 1Gi
  numberOfInstances: 2
  users:
    zalando: # database owner
    - superuser
    - createdb
    foo_user: [] # role for application foo
  databases:
    foo: zalando # dbname: owner
  preparedDatabases:
    bar: {}
  postgresql:
    version: "12"
```



PostgreSQL by Zalando

↑ Ejemplo

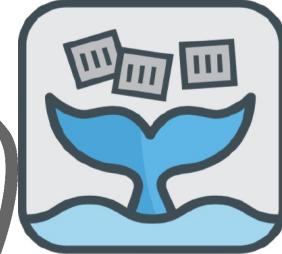
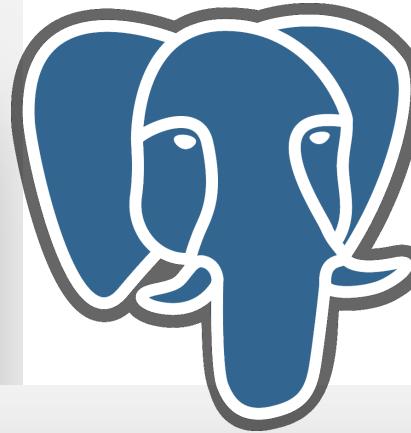
En este ejemplo adaptamos una función para gestionar dispositivos IOT para que use el cluster PostgreSQL desplegado en el ejemplo anterior.



PostgreSQL by Zalando

Ejemplo

```
version: 1.0
provider:
  name: openfaas
functions:
  crud-postgre:
    lang: node12
    handler: ./crud-postgre
    image: ollescas/crud-postgre:latest
    environment:
      db_host: acid-minimal-cluster-repl.default.svc.cluster.local
      db_port: 5432
      db_name: foo
    secrets:
      - db
```



```
function initPool() {
  return new Pool({
    user: fs.readFileSync('/var/openfaas/secrets/db-username', 'utf-8'),
    host: process.env['db_host'],
    database: process.env['db_name'],
    password: fs.readFileSync('/var/openfaas/secrets/db-password', 'utf-8'),
    port: process.env['db_port'],
    ssl: {
      rejectUnauthorized: false,
    },
  });
}
```

MongoDB by Percona

↑ Instalación



mongoDB®

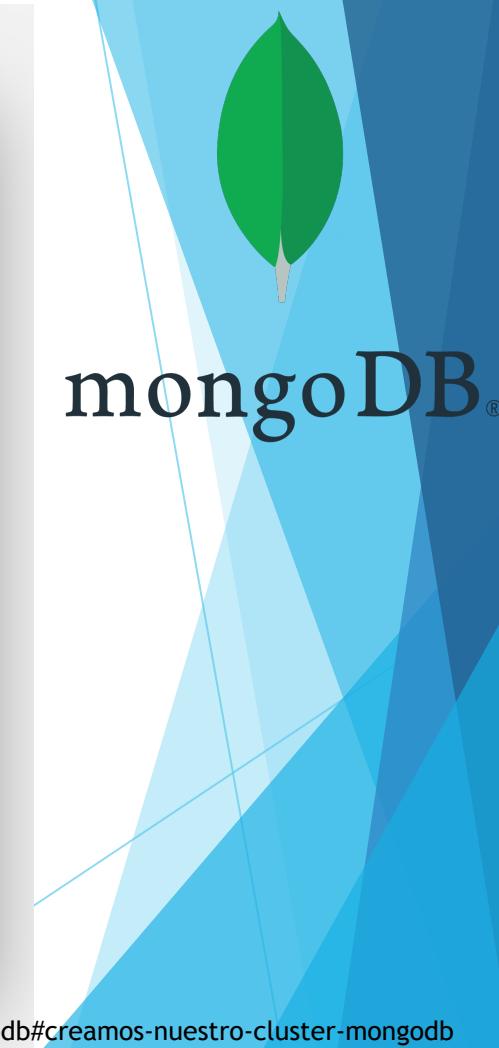
```
# Despliegue del operador
kubectl apply -f https://raw.githubusercontent.com/percona/percona-server-mongodb-operator/v1.7.0/deploy/bundle.yaml
```

MongoDB by Percona



Despliegue del cluster

```
apiVersion: psmdb.percona.com/v1-5-0
kind: PerconaServerMongoDB
metadata:
  name: my-mongodb
spec:
  image: percona/percona-server-mongodb:4.2.8-8
  # ...
  # ...
  replsets:
  - name: rs0
    size: 3
    # ...
    # ...
    resources:
      limits:
        cpu: "300m"
        memory: "0.5G"
      volumeSpec:
        persistentVolumeClaim:
          resources:
            requests:
              storage: 3Gi
  mongod:
    net:
      port: 27017
      hostPort: 0
    security:
      # ...
      # ...
    setParameter:
      # ...
      # ...
    storage:
      # ...
      # ...
```

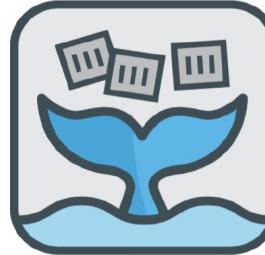


mongoDB®

MongoDB by Percona

↑ Ejemplo

En este ejemplo desarrollamos un pequeño api REST para gestionar usuarios y comentarios haciendo uso del cluster MongoDB que hemos desplegado en el ejemplo anterior.



mongoDB®

MongoDB by Percona



Ejemplo

```
version: 1.0
provider:
  name: openfaas
functions:
  get:
    lang: node10-express
    handler: ./crud-mongo
    image: oillescas/crud-mongo:latest
    environment:
      db_host: openfaas-mongo-rs0.default.svc.cluster.local
      db_name: openfaas
    secrets:
      - openfaas-mongodb
```



mongoDB®

```
function initConexion() {
  const user = fs.readFileSync("/var/openfaas/secrets/username", "utf-8"),
  password = fs.readFileSync("/var/openfaas/secrets/password", "utf-8"),
  host = process.env["db_host"];

  return `mongodb+srv://${user}:${password}@${host}
/admin?replicaSet=rs0&ssl=false`;
}
```

Colas de mensajería y gestión de eventos

Cloud

- ↑  SQS,
- ↑  SNS, Event Bridge
- ↑  Azure Service Bus
- ↑  Google Cloud Pub/Sub
- ↑ ...

Kubernetes

- ↑ Knative event broker
- ↑ OpenFaaS Conectors
- ↑ Argo Events

Knative event broker

↑ Ejemplo

En este ejemplo probamos el sistema de eventos de Knative. Para el ejemplo desplegaremos un broker, y dos event triggers y veremos como el broker reparte el trafico en función de los parámetros de entrada.

Hemos usado una implementación en memoria muy simple pero la implementación recomendada para producción esta basada en Apache Kafka



Knative event broker

↑ Ejemplo

```
apiVersion: eventing.knative.dev/v1
kind: Broker
metadata:
  name: default
  namespace: event-example
```

```
apiVersion: eventing.knative.dev/v1
kind: Trigger
metadata:
  name: hello-display
spec:
  broker: default
  filter:
    attributes:
      type: greeting
  subscriber:
    ref:
      apiVersion: v1
      kind: Service
      name: hello-display
```

```
apiVersion: eventing.knative.dev/v1
kind: Trigger
metadata:
  name: goodbye-display
spec:
  broker: default
  filter:
    attributes:
      source: sendoff
  subscriber:
    ref:
      apiVersion: v1
      kind: Service
      name: goodbye-display
```



Knative event broker

↑ Ejemplo

```
curl -v  
"http://broker-ingress.knative-eventing.svc.cluster.local/event-example/de  
fault"  
\\  
-X POST \  
-H "Ce-Id: say-hello" \  
-H "Ce-Specversion: 1.0" \  
-H "Ce-Type: greeting" \  
-H "Ce-Source: not-sendoff" \  
-H "Content-Type: application/json" \  
-d '{"msg":"Hello Knative!"}'
```



```
curl -v  
"http://broker-ingress.knative-eventing.svc.cluster.local/event-example/de  
fault"  
\\  
-X POST \  
-H "Ce-Id: say-goodbye" \  
-H "Ce-Specversion: 1.0" \  
-H "Ce-Type: not-greeting" \  
-H "Ce-Source: sendoff" \  
-H "Content-Type: application/json" \  
-d '{"msg":"Goodbye Knative!"}'
```

OpenFaaS Cron Connector

↑ Ejemplo

En este ejemplo probaremos el cron conector que habilita una anotación en los descriptores de las funciones OpenFaaS para que se ejecuten automáticamente.

```
kubectl apply -f https://raw.githubusercontent.com/MasterCloudApps-Projects/Serverless-Kubernetes/main/Examples/openfaas/kubernetes/cron-connector.yaml
```

OpenFaaS Cron Connector

↑ Ejemplo

```
version: 1.0
provider:
  name: openfaas
functions:
  download-forecast:
    lang: node12
    handler: ./download-forecast
    image: oillescas/download-forecast:latest
    annotations:
      topic: cron-function
      schedule: "1 */4 * * *"
    environment:
      endpoint: my-minio.default.svc.cluster.local
      minio-port: 9000
      use-ssl: false
      BUCKET_NAME: forecast
    secrets:
    - openfaas-minio
```

OpenFaaS Nats connector

↑ Ejemplo

En este ejemplo probaremos nats-conector que nos permitirá anotar las funciones para que se ejecuten cuando a una cola nats llegue un mensaje de un topic concreto.

```
kubectl apply -f https://raw.githubusercontent.com/MasterCloudApps-Projects/Serverless-Kubernetes/main/Examples/openfaas/kubernetes/nats-connector.yaml
```

OpenFaaS Nats connector

↑ Ejemplo

```
version: 1.0
provider:
  name: openfaas
functions:
  nats-reciver:
    lang: node12-nats
    handler: ./nats-reciver
    image: oillescas/nats-reciver:latest
    annotations:
      topic: "nats-test"
    environment:
      RAW_BODY: 'true'
```

OpenFaaS Minio webhook

↑ Ejemplo

En este ejemplo hacemos uso del sistema de eventos de minio para con un webhook llamar a una función OpenFaaS

```
# Creamos el bucket  
  
mc mb minio-tfm/forecast  
  
# Configuramos los eventos del bucket  
  
mc admin config set minio-tfm notify_webhook:1 endpoint=http://gateway.openfaas:8080/function/minio-webhook  
mc event add minio-tfm/forecast arn:minio:sqs::1:webhook --event put --suffix .xml
```

Argo Events

Argo Events es un broker de eventos que soporta multitud de eventos de entrada (minio, nats, mqtt, sns, sqs, gcp pub/sub,...) y otros tantos sistemas de salida (HTTP, Lambda, nats, Kafka,...) lo cual nos permite conectar distintos componentes del ecosistema Serverless de una manera sencilla.

↑ Instalación



```
# Crear un namespace
kubectl create namespace argo-events

# Desplegar los componentes de Argo Events, SA, ClusterRoles, Sensor Controller, EventBus Controller and EventSource Controller
kubectl apply -f https://raw.githubusercontent.com/argoproj/argo-events/stable/manifests/install.yaml

# Desplegar un eventbus
kubectl apply -n argo-events -f https://raw.githubusercontent.com/argoproj/argo-events/stable/examples/eventbus/native.yaml
```

Argo Events

↑ Ejemplo

En este ejemplo desplegamos un eventSource de tipo webhook que recibe las peticiones y un sensor que mediante un trigger las conduce a una función OpenFaaS



Argo Events

Ejemplo

```
apiVersion: argoproj.io/v1alpha1
kind: EventSource
metadata:
  name: webhook
spec:
  service:
    ports:
      - port: 12000
        targetPort: 12000
  webhook:

# event-source can run multiple HTTP servers. Simply define a unique port to start a new HTTP server
  example:
    # port to run HTTP server on
    port: "12000"
    # endpoint to listen to
    endpoint: /example

# HTTP request method to allow. In this case, only POST requests are accepted
  method: POST
```



Argo Events



Ejemplo

```
apiVersion: argoproj.io/v1alpha1
kind: Sensor
metadata:
  name: openfaas-sensor
spec:
  template:
    serviceAccountName: argo-events-sa
  dependencies:
    - name: test-dep
      eventSourceName: webhook
      eventName: example
  triggers:
    - template:
        name: openfaas-trigger
        http:
          url: http://gateway.openfaas.svc.cluster.local:8080/function/hello-world
          payload:
            - src:
                dependencyName: test-dep
              dest: bucket
            method: POST
```



Gestión de usuarios

Cloud

- ↑  Cognito
- ↑  Active Directory B2C
- ↑  Cloud IAM
- ↑ ...

Kubernetes

- ↑ Keycloak

Keycloak

Keycloak es un software opensource que permite la gestión de usuario y el control de acceso de los mismos a las aplicaciones y servicios.



↑ Instalación

```
# Instalamos Keycloak.  
kubectl create -f https://raw.githubusercontent.com/keycloak/keycloak-quickstarts/latest/kubernetes-examples/keycloak.yaml
```

Keycloak



↑ Ejemplo static-site

En este ejemplo vamos a segurizar el acceso a la web que desplegamos en minio.

Para esto vamos a conectar el ingress nginx que desplegamos en el anterior ejemplo con Keycloak mediante un servicio llamado oauth2-proxy que será el encargado de validar las peticiones de los usuarios.

Keycloak

Ejemplo static-site

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: simpleapp-ingress
annotations:
  nginx.ingress.kubernetes.io/rewrite-target: "/static-site/$1"
  nginx.ingress.kubernetes.io/configuration-snippet: |
    rewrite ^/?$ /static-site/index.html break;
  nginx.ingress.kubernetes.io/auth-url: "https://$host/oauth2/auth"
  nginx.ingress.kubernetes.io/auth-signin:
    "https://$host/oauth2/start?rd=$escaped_request_uri"
spec:
  tls:
  - hosts:
    - miniostatic.192.168.64.4.nip.io
  # ingressClassName: nginx
  rules:
  - host: miniostatic.192.168.64.4.nip.io
    http:
      paths:
      - path: /(.*)
        pathType: Prefix
        backend:
          service:
            name: my-minio
            port:
              number: 9000
```



```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: oauth2-proxy
spec:
  tls:
  - hosts:
    - miniostatic.192.168.64.4.nip.io
  # ingressClassName: nginx
  rules:
  - host: miniostatic.192.168.64.4.nip.io
    http:
      paths:
      - path: /oauth2
        pathType: Prefix
        backend:
          service:
            name: oauth2-proxy
            port:
              number: 4180
```

Keycloak

↑ Ejemplo static-site

```
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    k8s-app: oauth2-proxy
  name: oauth2-proxy
spec:
# ...
# ...
  spec:
    containers:
      - args:
          - --provider=keycloak
          - --email-domain=*
          - --client-id=openfaas
          - --client-secret=a92d17a9-7302-4c23-bc53-86b187ed928d
          - --login-url=https://keycloak.192.168.64.4.nip.io/auth/realmstfm/protocol/openid-connect/auth
          - --redeem-url=https://keycloak.192.168.64.4.nip.io/auth/realmstfm/protocol/openid-connect/token
          -
          --validate-http://keycloak.192.168.64.4.nip.io/auth/realmstfm/protocol/openid-connect/userinfo
          - --scope=profile
          - --ssl-insecure-skip-verify=true
          - --ssl-upstream-insecure-skip-verify=true
          - --pass-access-token=true
          - --pass-authorization-header=true
          - --set-xauthrequest=true
          - --set-authorization-header=true
        env:
          - name: OAUTH2_PROXY_CLIENT_ID
            value: openfaas
          - name: OAUTH2_PROXY_CLIENT_SECRET
            value: a92d17a9-7302-4c23-bc53-86b187ed928d
          - name: OAUTH2_PROXY_COOKIE_SECRET
            value: MEZXM1RPYU5jYU5JWlVjUi9kbkdHzz09
        image: quay.io/oauth2-proxy/oauth2-proxy:latest
        imagePullPolicy: Always
        name: oauth2-proxy
      ports:
        - containerPort: 4180
          protocol: TCP
```



Keycloak

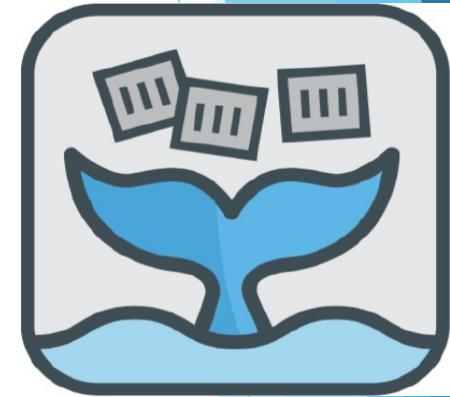


↑ Segurizar función OpenFaaS

En este ejemplo vamos a volver a usar el oauth2-proxy para en este caso usarlo de proxy del OpenFaaS Gateway y segurizar las llamadas a las funciones

```
const token = event.headers['x-forwarded-access-token'];

var decoded = jwt.decode(token);
console.log(decoded);
jwt.verify(token, cert, { algorithms: ['RS256'] }, function (err, payload) {
  console.error(err);
  console.log(payload);
});
```



Keycloak

↑ Segurizar función OpenFaaS

```
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    k8s-app: oauth2-proxy
  name: oauth2-proxy
  namespace: openfaas
spec:
  replicas: 1
  selector:
    matchLabels:
      k8s-app: oauth2-proxy
  template:
    metadata:
      labels:
        k8s-app: oauth2-proxy
    spec:
      containers:
        - args:
            # ...
            - --upstream=http://gateway.openfaas.svc.cluster.local:8080/
            # ...
```

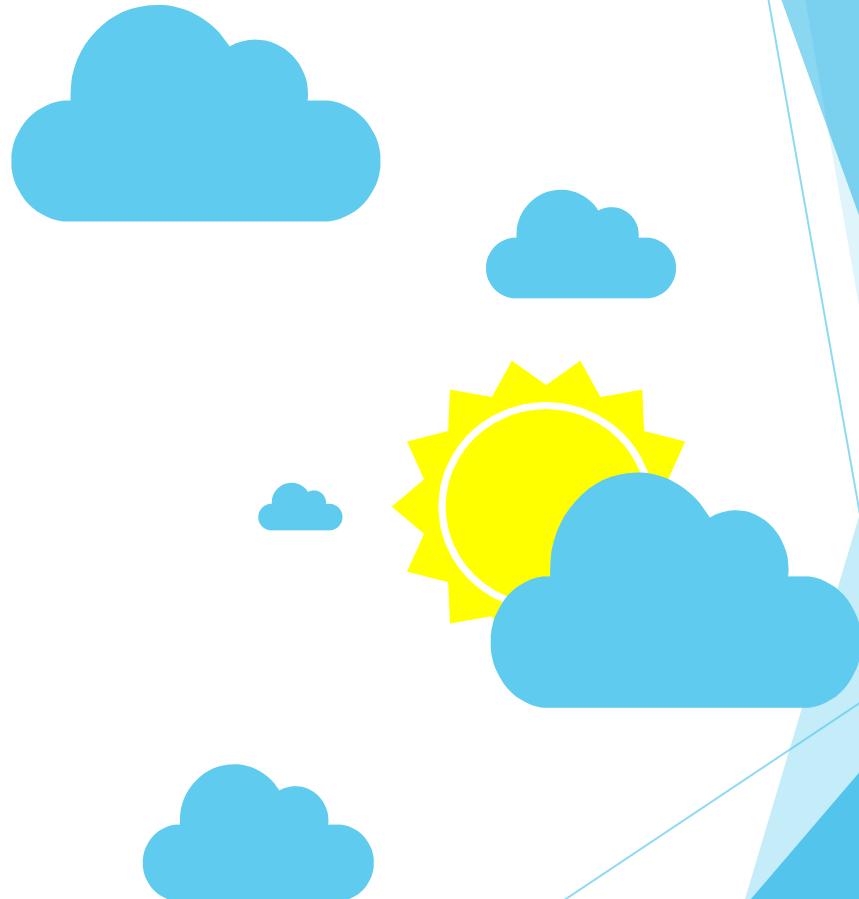


```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: openfaas-ingress
  namespace: openfaas
  annotations:
    nginx.ingress.kubernetes.io/rewrite-target: "/$1"
spec:
  tls:
    - hosts:
        - openfaas.192.168.64.4.nip.io
  rules:
    - host: openfaas.192.168.64.4.nip.io
      http:
        paths:
          - path: /(.*)
            pathType: Prefix
            backend:
              service:
                name: oauth2-proxy
                port:
                  number: 4180
```

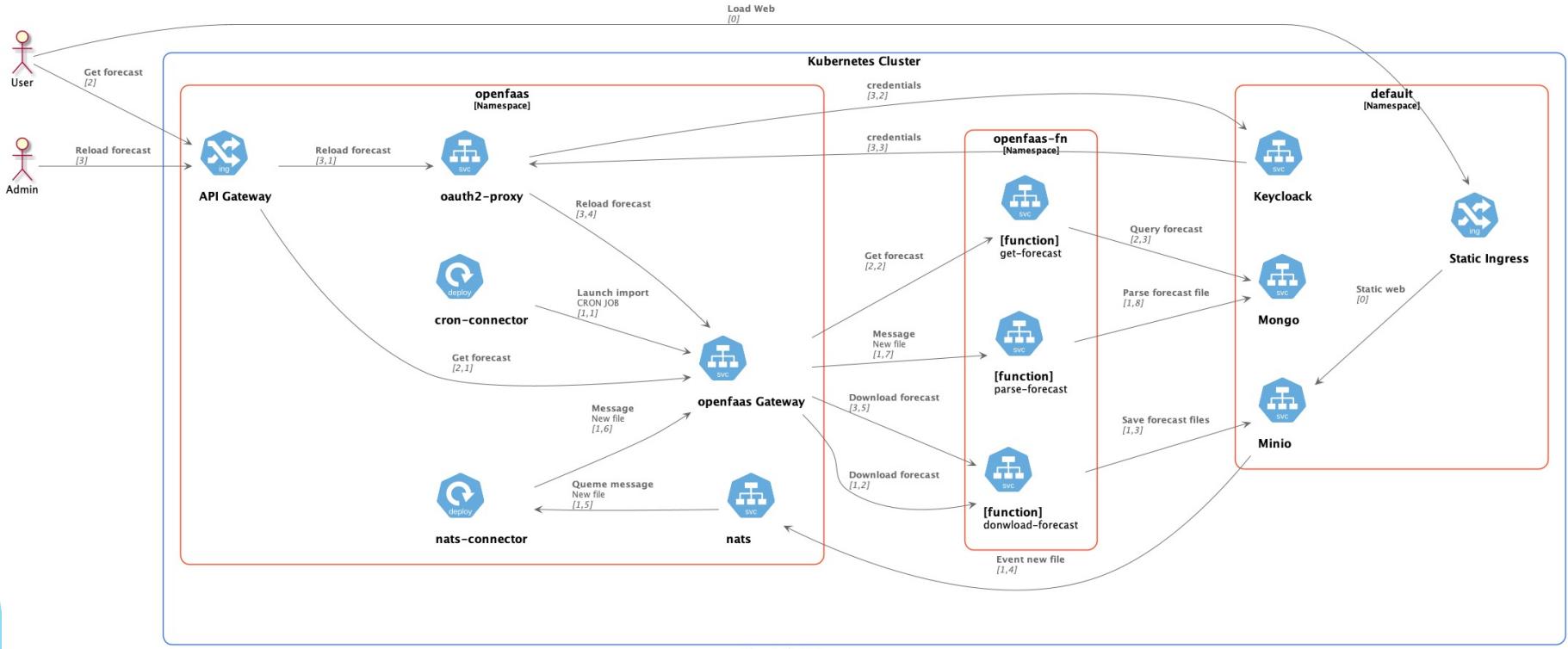
APP

Forecast

Ejemplo de uso de las distintas tecnologías

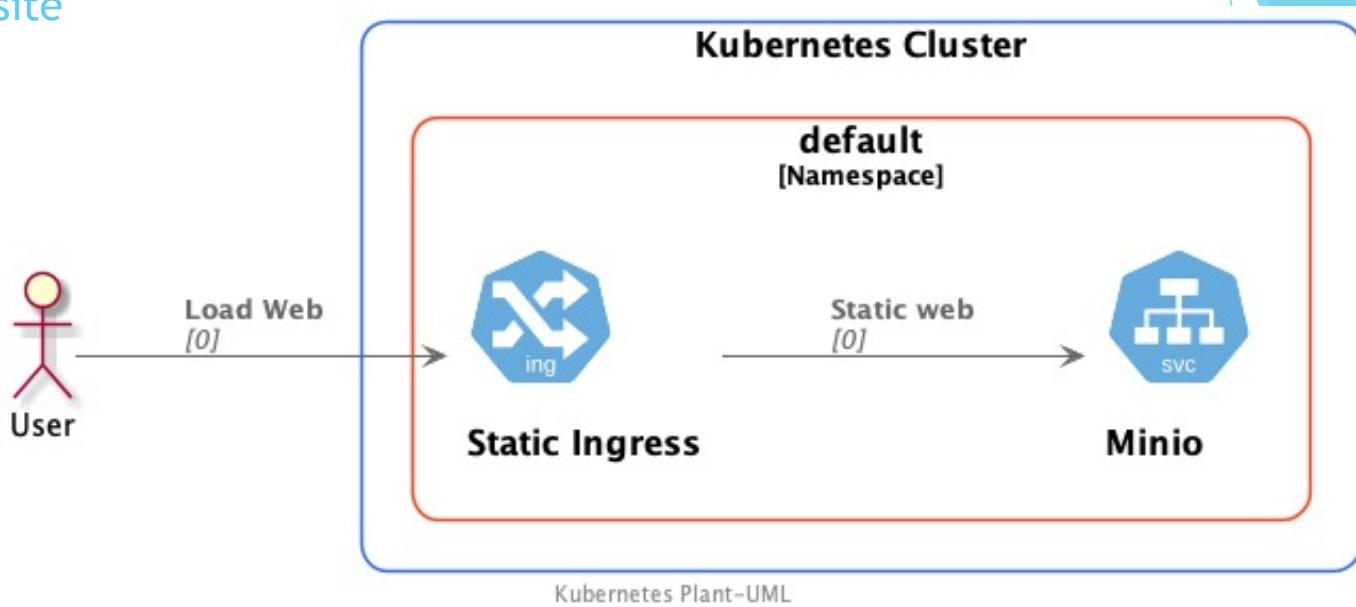


Forecast App



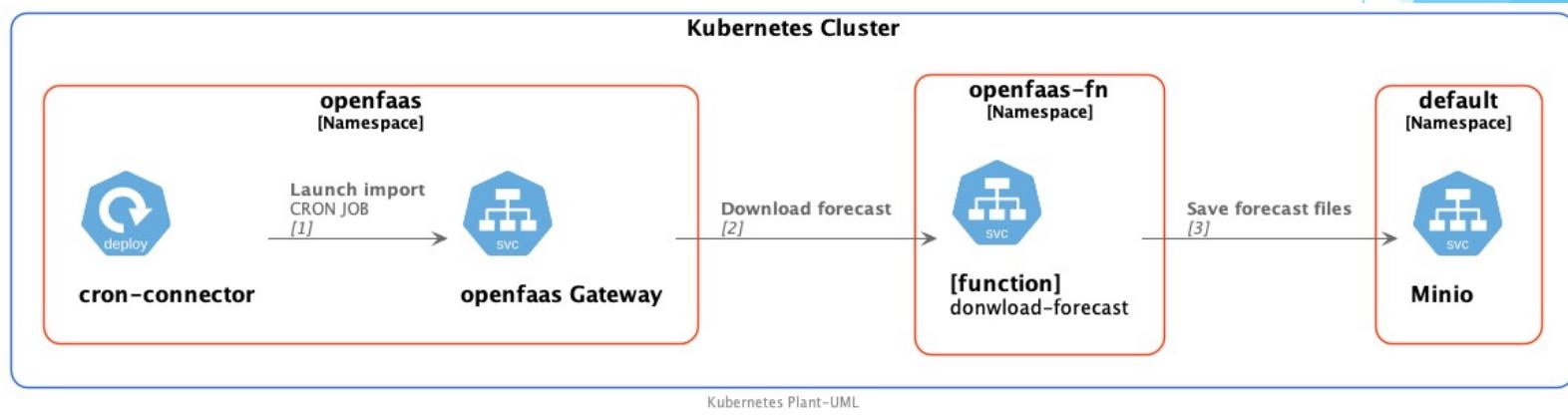
Forecast App

Static website



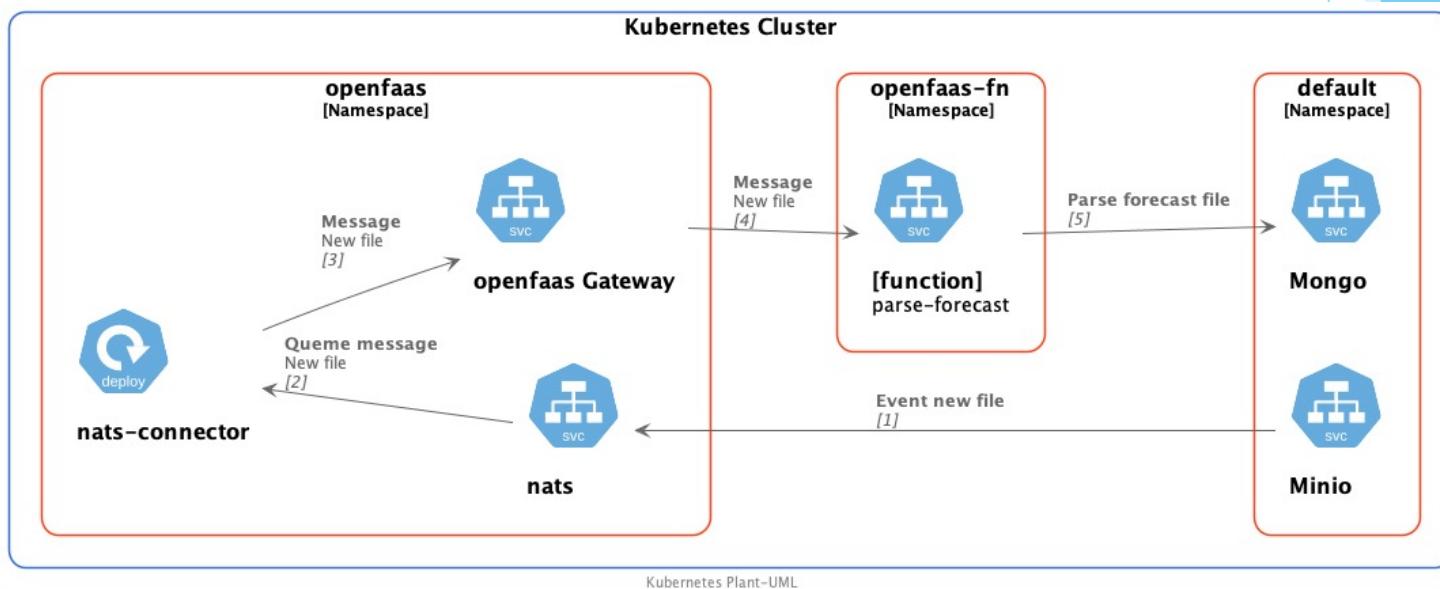
Forecast App

Static website



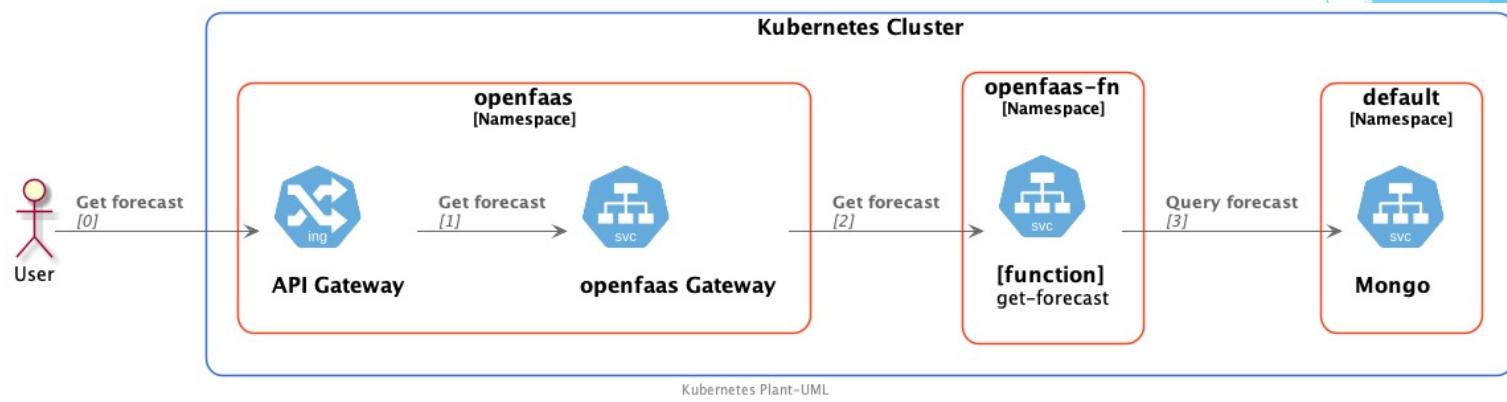
Forecast App

parse-forecast



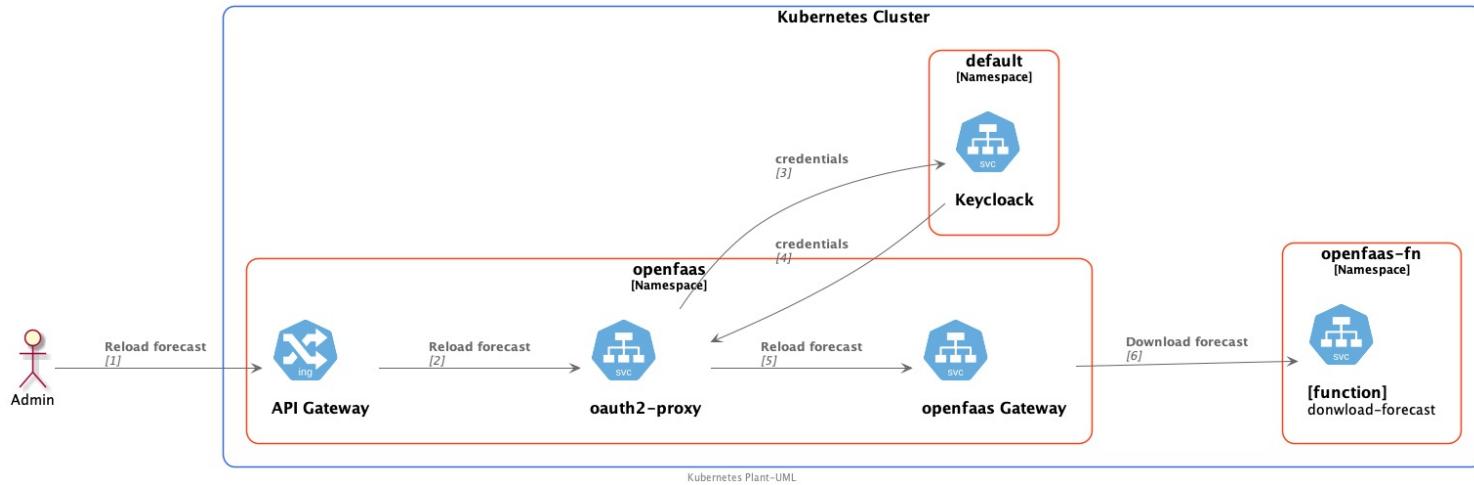
Forecast App

get-forecast

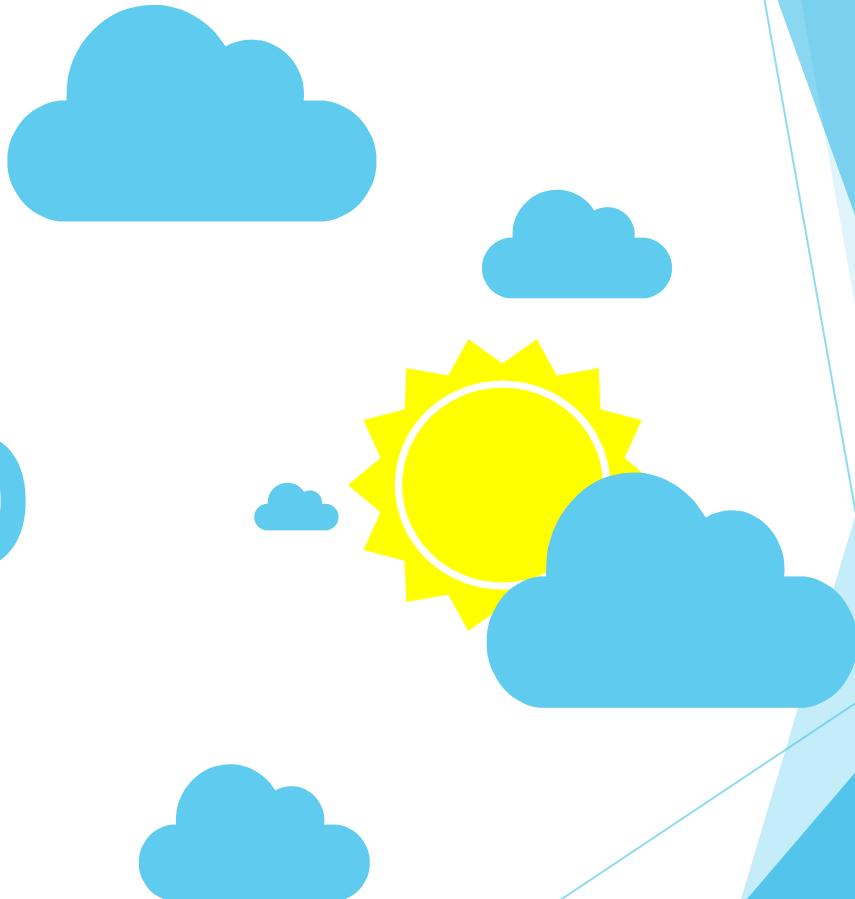


Forecast App

get-forecast



Demo



Conclusiones

Conclusiones

- ⬆ Hay sustitutos en Kubernetes para todas las tecnologías serverless
- ⬆ No todas son equivalentes a las soluciones de los proveedores de cloud
- ⬆ Las soluciones en Kubernetes pueden convivir con las de los proveedores de cloud en un modelo híbrido

Proximos pasos

- ⬆ Estudiar más soluciones de Faas (openWishk)
- ⬆ Estudiar el ci/cd de las faas especialmente OpenFaaS
- ⬆ Probar otros operadores de bases de datos (Scylla Alternator)

THANKS!

Any questions?

You can find me at:



@oillescas



oscarillescas

