Documento de Instalación del Nodo Autonómico

Minsait

Agosto 2025

Histórico de versiones:

Versión	Realizado por:	Fecha
1.0	Minsait	17/07/2025
1.1	Minsait	30/07/2025
1.2	Minsait	08/08/2025
1.3	Minsait	13/08/2025
1.4	Minsait	14/08/2025
1.5	Minsait	18/08/2025
1.6	Minsait	22/08/2025
1.7	Minsait	04/09/2025
1.8	Minsait	12/09/2025

Contenido

1. Objetivo

El presente documento tiene como finalidad describir de forma detallada el proceso de instalación, configuración y validación de los componentes que integran el Nodo Autonómico de la plataforma UNICAS. Incluye los requisitos previos, la guía de instalación de cada paquete funcional, las dependencias entre módulos y las consideraciones técnicas necesarias para garantizar un despliegue exitoso y conforme a las especificaciones del proyecto.

Este documento servirá como referencia para los equipos técnicos responsables de la implantación, asegurando la homogeneidad y la calidad en la instalación de los distintos entornos.

2. Requisitos de la instalación

2.1. Infraestructura para el despliegue de Onesait Healthcare

Onesait Healthcare es una plataforma modular que se distribuye de forma totalmente contenerizada y permite su despliegue tanto en entornos On Premise como en proveedores Cloud como AWS, Microsoft Azure o Google Cloud.

Con respecto a la infraestructura necesaria para su despliegue, se necesitará tener disponible los siguientes elementos:

- Load Balancer donde se expondrán los servicios de la plataforma hacia el exterior y permite el balanceo de carga y alta disponibilidad de la plataforma
- Nombre de dominio configurado y certificados para ese dominio correctamente configurados a nivel de balanceador. Este nombre de dominio se deberá informar durante el despliegue de los diferentes módulos de la solución.
- Docker Container Registry donde desplegar las imágenes de los módulos que forman parte de la plataforma Onesait Healthcare y van a ser desplegadas en cada uno de los entornos.
- Base de datos relacional (Oracle o Mysql).
- Clúster de Kubernetes en versión 1.30 o superior
- Máquina virtual adicional con acceso al clúster y que disponga de los clientes de línea de comando "kubectl" y "helm" instalados.

2.2. Preconfiguración del clúster de Kubernetes para el despliegue de Onesait Healthcare

2.2.1. Creación namespace dentro del clúster

- Crear el **namespace** para la instalación de los módulos de la solución, normalmente a este namespace se le llama: **oh-modules**. Se puede crear mediante el comando:

kubectl create namespace

```
kubectl create namespace oh-modules
```

2.2.2. Creación de credenciales para Docker Registry

Se requiere un Docker Registry que contendrá las imágenes Docker de los módulos de Onesait Healthcare.

Se debe crear un secret con las credenciales para poder acceder a dicho Registry.

- Crear un secret con las **credenciales** para el **Docker Registry**, el nombre de este secret debe ser: **oh-docker-creds**, se crearía con un comando de esta forma:

kubectl create secret

```
kubectl create secret docker-registry oh-docker-creds --docker-server=<host-del-docker-registry> --docker-username=<user-docker-registry> --docker-password=<pass-docker-registry> -n oh-modules
```

2.2.3. Configuración de Gateway Controller

Se requiere instalar y configurar una implementación de la Gateway API de Kubernetes y habilitar el Gateway Controller. Esta guía usa la implementación de traefik.

Esta configuración está ideada para una única instancia. Si se requiere una tipología múltiple ver “ANEXO: Configuración para varios entornos en el mismo clúster de kubernetes”.

Instalación de traefik y configuración como Gateway Controller

Si el clúster en el que se va a realizar el despliegue ya tiene traefik instalado ignorar este apartado.

Para comprobar si traefik está instalado

```
kubectl get pods -n traefik
```

Si se obtiene algún resultado traefik está instalado.

Para instalar traefik ejecutamos los siguientes comandos.

```
kubectl create namespace traefik
helm repo add traefik https://helm.traefik.io/traefik
helm repo update
helm search repo traefik/traefik --versions
```

Del último comando anterior apuntar la versión deseada a instalar (como mínimo 34.2.0), lanzar el siguiente comando para obtener los valores por defecto de la instalación de traefik:

```
helm show values traefik/traefik --version 34.2.0 > values_traefik.yaml
```

Editamos el fichero obtenido (se puede usar nano o el editor que se desee):

```
nano values_traefik.yaml
```

Buscamos la cadena: **providers.kubernetesGateway.enabled**, que tendrá el valor **false** y lo cambiamos a **true**, guardamos los cambios en el fichero (con nano sería CTRL+O, enter para confirmar, CTRL+X).

Finalmente instalamos traefik con el fichero de parámetros modificado:

```
helm install traefik  --values=values_traefik.yaml --namespace traefik --set
dashboard.enabled=true traefik/traefik --version 34.2.0
```

Una vez finalice la instalación comprobar que el pod de traefik se crea y queda en estado Running con el comando:

```
kubectl get pods --namespace traefik
```

Configuración Gateway Controller en instalación traefik existente

Si el clúster en el que se va a realizar el despliegue ya tiene traefik instalado, pero no tiene habilitado el Gateway Controller (por ejemplo, en la instalación por defecto de k3s), se deberán seguir los pasos de este apartado.

En caso de que traefik esté instalado y configurado como Gateway Controller se ignorará este apartado.

Para comprobar si traefik está habilitado como Gateway Controller se lanza el siguiente comando control el clúster:

```
kubectl get gatewayclasses
```

Si está habilitado obtendremos una respuesta de esta forma:

```
NAME      CONTROLLER                      ACCEPTED    AGE
traefik   traefik.io/gateway-controller   True        2d22h
```

Si no se obtienen resultados no se tiene habilitado ningún gateway controller, si se obtienen resultados que no contienen la clase traefik es que hay instaladas otras implementaciones de gateway controller.

Para habilitarlo creamos los recursos de la URL: https://doc.traefik.io/traefik/providers/kubernetes-gateway/

Ejecutando el siguiente comando:

```
kubectl apply -f https://github.com/kubernetes-sigs/gateway-
api/releases/download/v1.2.1/standard-install.yaml
```

Hay que añadir ciertos permisos al service account de traefik, para ello creamos el siguiente ClusterRole:

traefik-gateway-role

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
```

```
  name: traefik-gateway-role
rules:
  - apiGroups:
      - ""
    resources:
      - namespaces
    verbs:
      - list
      - watch
  - apiGroups:
      - ""
    resources:
      - services
      - secrets
      - configmaps
    verbs:
      - get
      - list
      - watch
  - apiGroups:
      - discovery.k8s.io
    resources:
      - endpointslices
    verbs:
      - list
      - watch
  - apiGroups:
      - gateway.networking.k8s.io
    resources:
      - gatewayclasses
      - gateways
      - httproutes
      - grpcroutes
      - tcproutes
      - tlsroutes
      - referencegrants
      - backendtlspolicies
    verbs:
      - get
      - list
      - watch
  - apiGroups:
      - gateway.networking.k8s.io
    resources:
      - gatewayclasses/status
      - gateways/status
      - httproutes/status
      - grpcroutes/status
      - tcproutes/status
      - tlsroutes/status
      - referencegrants/status
      - backendtlspolicies/status
    verbs:
      - update
```

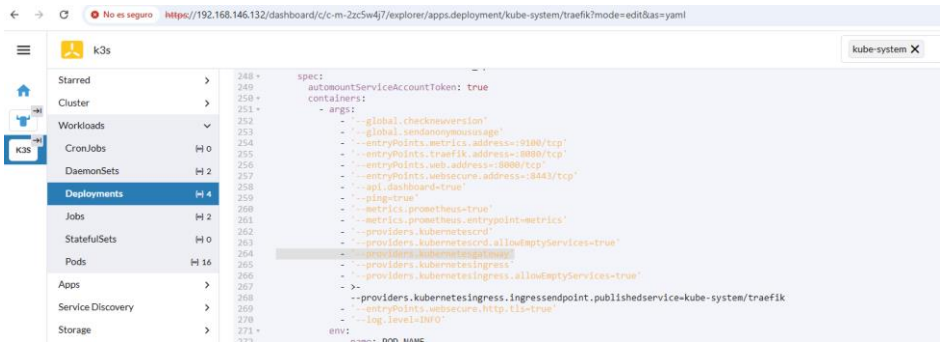
Asignamos los permisos creando el siguiente ClusterRoleBinding:

traefik-gateway-role-binding

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: traefik-gateway-role-binding
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: traefik-gateway-role
subjects:
  - kind: ServiceAccount
    name: traefik
    namespace: <namespace de instalación de traefik, suele ser traefik o kube-system>
```

Finalmente habilitamos el Gateway Controller accediendo al **Deployment** de **traefik** (en el namespace de instalación de traefik, que suele ser **traefik** o **kube-system**), y en la sección de argumentos se le añade el siguiente:

```
- '--providers.kubernetesgateway'
```



Se crea el GatewayClass:

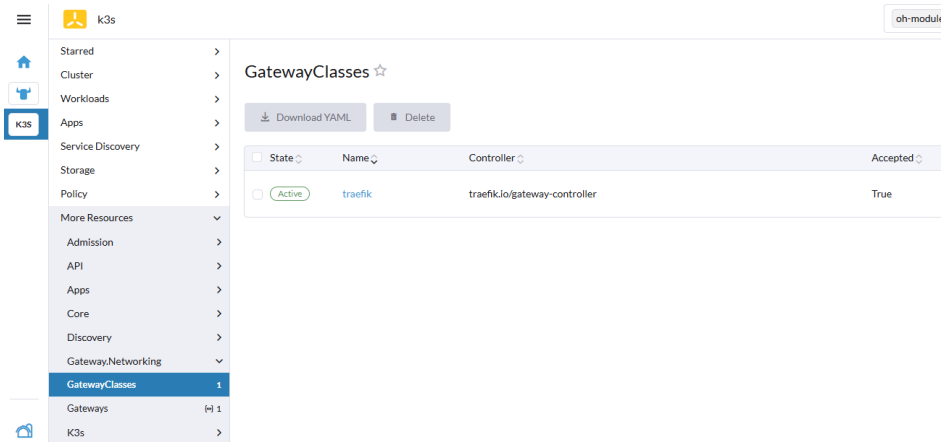
traefik

```
apiVersion: gateway.networking.k8s.io/v1
kind: GatewayClass
metadata:
  name: traefik
  namespace: <namespace de instalación de traefik, suele ser traefik o kube-system>
spec:
  controllerName: traefik.io/gateway-controller
```

Tras recrear el pod de traefix con el kubernetesgateway debería aparecernos la GatewayClass de traefik

Si estamos en rancher sería visible en el menú:

More Resources -> Gateway.Networking y dentro de este: **GatewayClasses**



Creación del Gateway HTTP

Si la configuración de HTTPS para el dominio que se expone se realiza en un balanceador externo el Gateway se creará como HTTP y el balanceador es el que gestiona las conexiones HTTPS y expone el certificado y vuelca las peticiones a los nodos worker del clúster.

Una vez traefik se encuentra instalado y configurado como Gateway Controller creamos el Gateway importando el siguiente yaml (el Gateway se debe llamar **"oh-modules-gateway"** ya que los charts de helm hacen referencia a dicho nombre tal cual):

oh-modules-gateway

```
apiVersion: gateway.networking.k8s.io/v1
kind: Gateway
metadata:
  name: oh-modules-gateway
  namespace: <namespace, normalmente oh-modules>
spec:
  gatewayClassName: traefik
  listeners:
    - allowedRoutes:
        namespaces:
          from: Same
      name: <namespace, normalmente oh-modules>
      hostname: <host-expuesto, debe coincidir con el expuesto en el balanceador>
      port: 8000
      protocol: HTTP
```

Por ejemplo, para el clúster de desarrollo quedaría:

oh-modules-gateway

```
apiVersion: gateway.networking.k8s.io/v1
kind: Gateway
metadata:
  name: oh-modules-gateway
  namespace: oh-modules
spec:
  gatewayClassName: traefik
  listeners:
    - allowedRoutes:
        namespaces:
```

```
        from: All
    name: oh-modules
    hostname: oh-modules.ohrancherha1-1.indra.es
    port: 8000
    protocol: HTTP
```

Creación del Gateway HTTPS

En caso de que el balanceador no configure el HTTPS y el certificado entonces se tendrá que crear el Gateway HTTPS.

Para ello debemos disponer del certificado del dominio, tanto la parte pública como la privada. Teniendo los ficheros crt (parte pública del certificado en formato PEM incluyendo el raíz y CA intermedios si los tuviera) y key (parte privada del certificado) se crearía un secret con dicho certificado con un comando de la forma:

```
kubectl create secret tls <nombre-cert> --cert=<nombre-cert>.crt --key=<nombre-cert>.key -n <namespace, normalmente oh-modules>
```

Una vez creado el secret creamos el Gateway importando el siguiente yaml (el Gateway se debe llamar **"oh-modules-gateway"** ya que los charts de helm hacen referencia a dicho nombre tal cual):

oh-modules-gateway

```
apiVersion: gateway.networking.k8s.io/v1
kind: Gateway
metadata:
  name: oh-modules-gateway
  namespace: <namespace, normalmente oh-modules>
spec:
  gatewayClassName: traefik
  listeners:
    - allowedRoutes:
        namespaces:
          from: Same
      name: <namespace, normalmente oh-modules>
      hostname: <host-expuesto, debe coincidir con el expuesto en el balanceador>
      port: 8443
      protocol: HTTPS
      tls:
        mode: Terminate
        certificateRefs:
          - name: <nombre-cert>
            namespace: <namespace, normalmente oh-modules>
```

Creamos el Gateway

```
kubectl apply -f oh-modules-gateway.yaml
```

Comprobamos que se ha creado el Gateway

```
kubectl get gateways -n oh-modules
```

```
sgvillanueva@ip-10-164-97-140:~$ kubectl get gateways -n oh-modules
NAME                 CLASS     ADDRESS   PROGRAMMED   AGE
oh-modules-gateway   traefik             True         15d
```

Creación de Network Load Balancer para AWS

Este apartado sólo aplica si la instalación se está realizando en un clúster EKS de AWS, en caso contrario ignorar este apartado.

Se debe disponer de una Elastic IP pública registrada en AWS y un dominio que se asociará a dicha IP.

Se accede a la consola de AWS → EC2 → Red y Seguridad → Direcciones IP elásticas, seleccionar la IP que se tiene registrada y anotar el valor de ID de asignación (que tendrá el prefijo eipallloc-).

Usamos ese valor para crear el siguiente yaml:

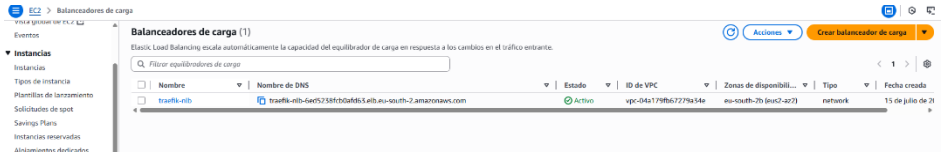
nlb-epi-to-traefik

```
apiVersion: v1
kind: Service
metadata:
  name: nlb-eip-to-traefik
  namespace: traefik
  annotations:
    service.beta.kubernetes.io/aws-load-balancer-attributes:
load_balancing.cross_zone.enabled=true
    service.beta.kubernetes.io/aws-load-balancer-name: "traefik-nlb"
    service.beta.kubernetes.io/aws-load-balancer-type: "nlb"
    service.beta.kubernetes.io/aws-load-balancer-eip-allocations: "<valor de
eipalloc anotado en el paso anterior>"
    service.beta.kubernetes.io/aws-load-balancer-scheme: "internet-facing"
spec:
  type: LoadBalancer
  ports:
    - name: web
      port: 80
      targetPort: 8000
      protocol: TCP
    - name: websecure
      port: 443
      targetPort: 8443
      protocol: TCP
  selector:
    app.kubernetes.io/instance: traefik-traefik
    app.kubernetes.io/name: traefik
```

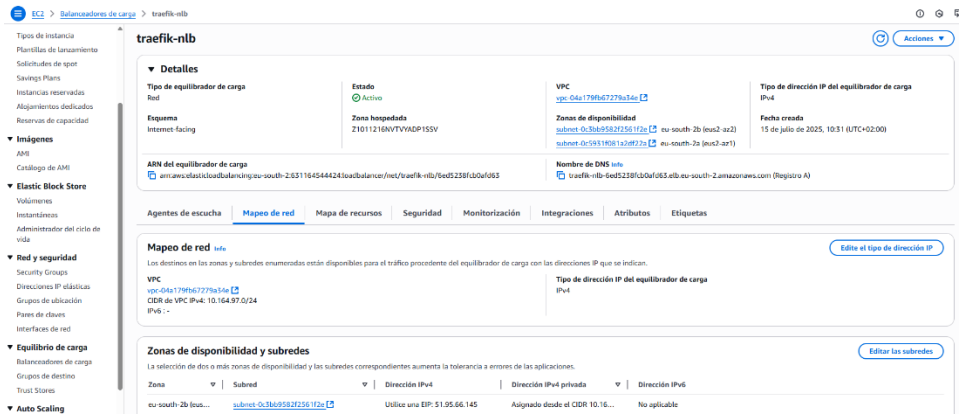
Se crea el servicio del balanceador con el siguiente comando:

```
kubectl -n traefik apply -f <fichero yaml con en el que se guardó>
```

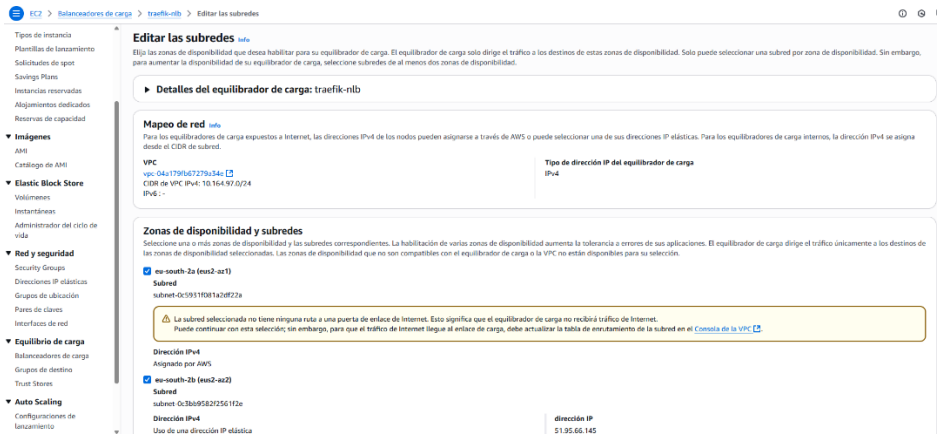
En la consola de AWS → EC2 → Equilibrio de carga → Balanceadores de carga deberá aparece un balanceador correspondiente al servicio que hemos creado, inicialmente en estado "Aprovisionando" y pasado un tiempo debería cambiar a "Activo".



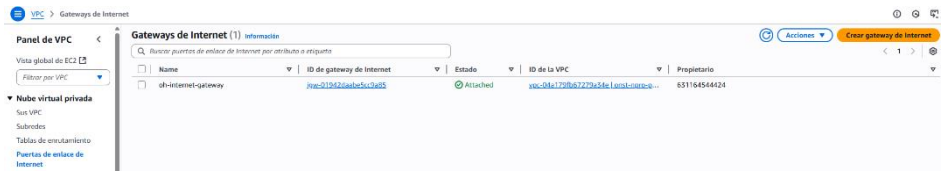
Accedemos al detalle del balanceador y a la pestaña Mapeo de Red (se mostrará sólo una zona) pulsar en Editar las Subredes.



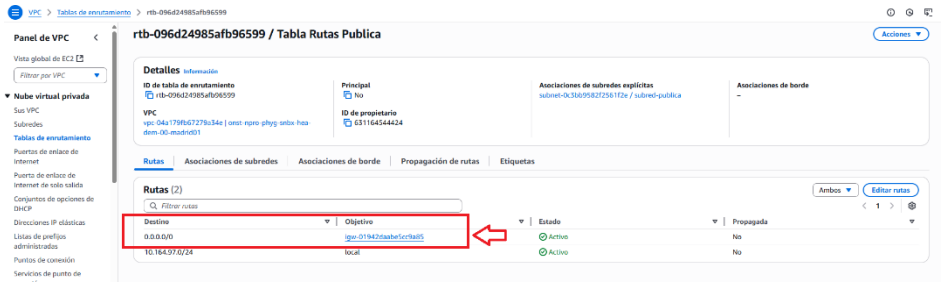
Y añadimos el resto de las zonas:



Será necesario crear un gateway de internet, para ello accedemos a la consola AWS → VPC → Nube virtual privada → Puertas de enlace de internet.



En la tabla de rutas pública hay que añadir la siguiente regla:



Una vez se tiene realizada esta configuración se elimina el balanceador que crea traefik por defecto:

```
kubectl -n traefik delete service traefik
```

Y se crea de nuevo el service (sin balanceo) con el siguiente yaml;

traefik service

```
apiVersion: v1
kind: Service
metadata:
  name: traefik
  namespace: traefik
spec:
  ipFamilies:
  - IPv4
  ports:
  - name: web
    port: 80
    protocol: TCP
    targetPort: web
  - name: websecure
    port: 443
    protocol: TCP
    targetPort: websecure
  selector:
    app.kubernetes.io/instance: traefik-traefik
    app.kubernetes.io/name: traefik
  sessionAffinity: None
  type: ClusterIP
```

Ejecutando el comando:

```
kubectl -n traefik apply -f <fichero yaml con en el que se guardó>
```

NOTA: Este es un ejemplo de cómo hacerlo, pero no la única forma. Las anotaciones del yalm[1] dependerán de los criterios de creación del Load Balancer que decida cada Nodo Autonómico, teniendo que ser ajustarlos al comportamiento deseado.

2.3. Helm como gestor de paquetes para el despliegue de Onesait Healthcare

La instalación de los módulos se realiza mediante charts de helm por lo tanto se debería registrar el repositorio en la herramienta con la que se vaya la realizar la instalación.

El repositorio que contiene los charts de Onesait Healthcare es:

Repositorio Charts Onesait Healthcare
`https://nexus.devops.onesait.com/repository/onesait-healthcare-helm-charts`

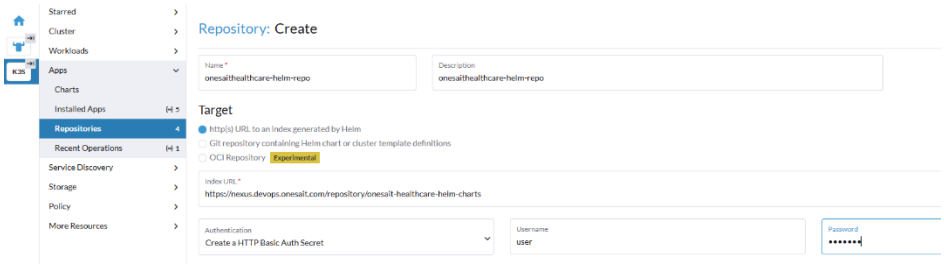
Para acceder a este repositorio son necesarias credenciales que serán proporcionadas por el equipo de Onesait Healthcare.

2.3.1. Instalación Helm desde Rancher

Desde un clúster de Kubernetes gestionado por Rancher se puede registrar el repositorio Helm e instanciar los charts mediante formularios visuales.

Para ello se accede a la consola de administración de Rancher y al clúster de Kubernetes en el que se vaya a realizar la instalación accediendo al menú Apps → Repositories y se pulsa en Create.

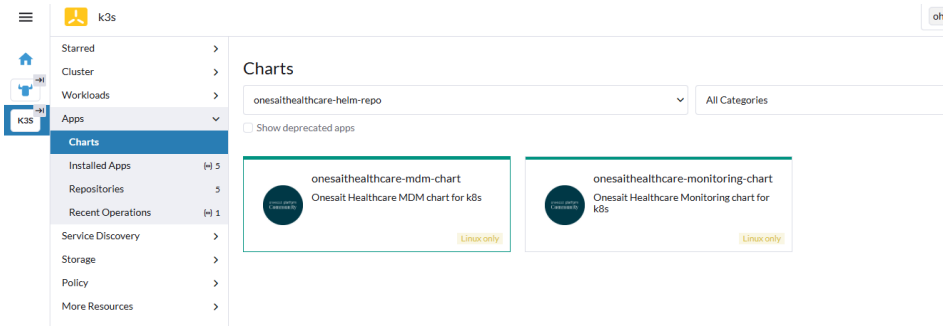
Se informa el nombre y la descripción con que se desee mostrar el repositorio, el índex URL indicada anteriormente y las credenciales de acceso proporcionadas.



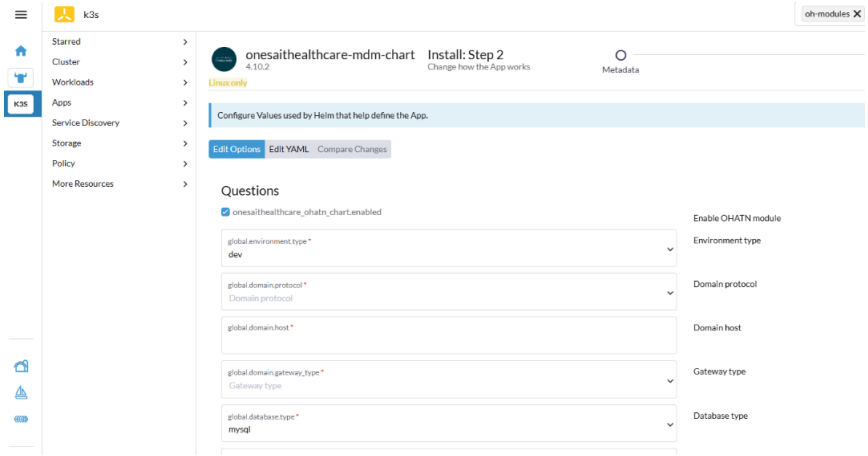
Tras crearlo se mostrará el listado de repositorios y deberá aparecer el nuevo en estado Active.

Si vamos al menú Apps → Charts y en el filtro se selecciona sólo el repositorio que hemos creado mostrará los charts disponibles en el mismo:

[1] Relación de anotaciones disponible en https://github.com/kubernetes-sigs/aws-load-balancer-controller/blob/main/docs/guide/service/annotations.md en función de la versión finalmente instalada



Desde esta vista pulsando el en chart que se desee instalar se mostrará el README del mismo, pulsando en Install se indica el namespace de instalación y un nombre para la instanciación y solicitará los valores parametrizables mostrando un formulario:



2.3.2. Instalación Helm desde Openshift

En clúster Openshift se puede registrar el repositorio helm para realizar la instalación de los charts mediante formularios visuales.

Para ello se accede al namespace donde se quiera realizar la instalación (en Openshift los repositorios con credenciales hay que registrarlos a nivel de namespace, si se desea usar un mismo repositorio para distintos namespaces hay que registrarlo en cada uno de ellos).

Creamos un secret con las credenciales de acceso al repositorio proporcionadas por el equipo de Onesait Healthcare:

onesait-healthcare-helm-repo-creds

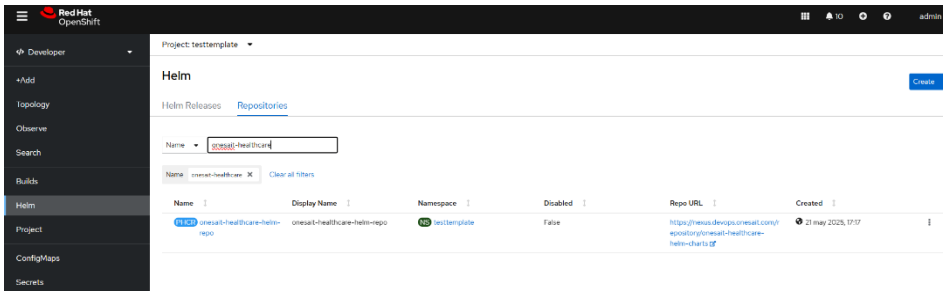
```
kind: Secret
apiVersion: v1
metadata:
  name: onesait-healthcare-helm-repo-creds
  namespace: <namespace>
stringData:
  username: <usuario>
  password: <password>
type: Opaque
```

Una vez creado el secret se crea el repositorio con el siguiente yaml:

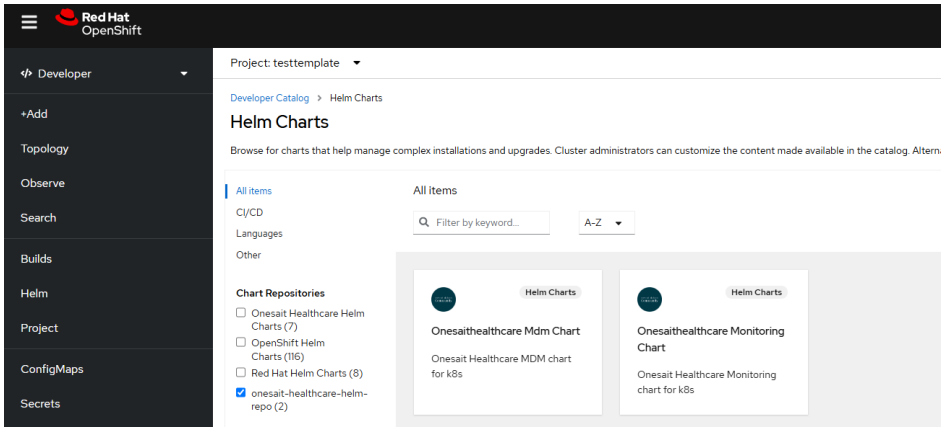
onesait-healthcare-helm-repo

```
apiVersion: helm.openshift.io/v1beta1
kind: ProjectHelmChartRepository
metadata:
  name: onesait-healthcare-helm-repo
  namespace: <namespace>
spec:
  connectionConfig:
    basicAuthConfig:
      name: onesait-healthcare-helm-repo-creds
    url:     'https://nexus.devops.onesait.com/repository/onesait-healthcare-helm-
charts'
  description: onesait-healthcare-helm-repo
  name: onesait-healthcare-helm-repo
```

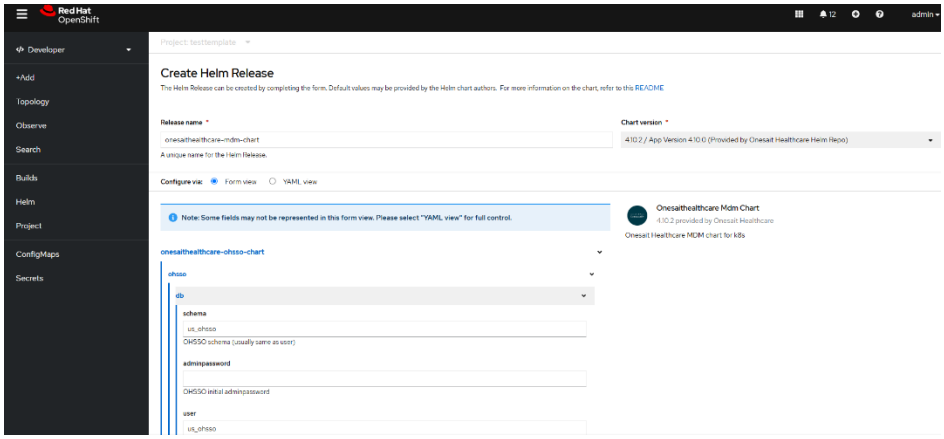
Si se accede a Developer → Helm → Repositories debe verse el nuevo repositorio:



Con el repositorio registrado podemos ir al menú Developer → Add -Helm Chart y filtrar por el repositorio creado, nos mostrará los charts publicados en el mismo:



Pulsando en el que se desee instalar muestra el README, pulsando en Create nos muestra un formulario para introducir los valores parametrizados:



2.3.3. Instalación Helm mediante helm client

Además de los asistentes visuales ofrecidos por Rancher y Openshift se puede seguir usando la instalación básica mediante cliente helm.

Para ello desde una máquina del clúster con el cliente helm instalado se ejecutaría el siguiente comando (para registrar el repositorio y las credenciales de acceso al mismo):

helm repo add

```
helm repo add onesait-healthcare-helm-repo
https://nexus.devops.onesait.com/repository/onesait-healthcare-helm-charts --
username <usuario> --password <password>
helm repo update
```

Para ver todos los charts disponibles ejecutamos el comando:

helm repo search

```
helm search repo --max-col-width 70
```

Para instalar un chart (mostramos el ejemplo con el de MDM) obtenemos los valores por defecto:

helm show values

```
helm show values onesait-healthcare-helm-repo/onesaithealthcare-mdm-chart >
values_mdm.yaml
```

Se edita el fichero obtenido y se informan los valores de los campos que se quiera dar a la instalación.

Se instala el chart con el comando:

helm install

```
helm install -f values_mdm.yaml mdm onesait-healthcare-helm-
repo/onesaithealthcare-mdm-chart -n oh-modules
```

3. Instalación Paquete MDM

3.1. Módulos que incluye

- Recursos comunes a todos los módulos
- SSO
- Settings Manager
- Ontology Server
- Users & Resources
- MPI
- Audit & Logs
- Professional Desktop

3.2. Prerrequisitos

Se deben cumplir los prerrequisitos generales y haber realizado los pasos indicados en la guía correspondiente.

3.2.1. Prerrequisitos de Base de Datos

Se debe disponer de un gestor de base de datos MySQL que tenga visibilidad desde los nodos worker del clúster de Kubernetes en el que se desplegarán los módulos.

A continuación, se indica como crear los usuarios y esquemas necesarios para los módulos de MDM.

MySQL con acceso de administrador

Disponiendo de usuario administrador se pueden crear los usuarios y esquemas para los módulos MDM con el siguiente lanzador:

[FTP_ENTREGAS]
/oradata/Versiones_Producto_OH/OH_v4/sistemas_configuraciones_iniciales_v4/mdm_scripts/MySQL/mdm-global-01-create-mysql/mdm-create-schemas-mysql.sh

Sin acceso de administrador

Si no se dispone de acceso de administrador se tendrá que solicitar la creación de los siguientes usuarios, cada uno de ellos con un esquema sobre el que tenga permisos y con el mismo nombre del usuario:

- us_ohsso
- us_ohcon
- us_ohont
- us_ohaut (el usuario debe tener permiso "grant option" sobre su esquema para poder asignar permiso a otros usuarios a objetos de su esquema)
- us_ohmpi (el usuario debe tener permiso "grant option" sobre su esquema para poder asignar permiso a otros usuarios a objetos de su esquema)

- us_ohatn
- us_ohatn_hist (el usuario debe tener permisos sobre su esquema y sobre el esquema us_ohatn, también debe tener permiso "grant option" sobre su esquema y sobre el us_ohatn)

3.3. Procedimiento de despliegue de Capa de Persistencia

Se crearán los modelos de datos de los módulos de MDM ejecutando el lanzador **mdm-create-models-mysql.sh** de la carpeta:

[FTP_ENTREGAS]
/oradata/Versiones_Producto_OH/OH_v4/sistemas_configuraciones_iniciales_v4/mdm_scripts/MySQL/mdm-global-02-models-mysql

Este lanzador solicitará todos los datos necesarios del gestor de BD y usuarios y passwords.

3.4. Paso previo a despliegue de módulos – certificado del dominio

Si el certificado usado para el dominio que se expone NO es verificable por una CA oficial, se producirían errores en los backends de los módulos debido a que las máquinas virtuales Java de los mismo no son capaces de verificar el certificado.

Si se da esta situación es necesario que se cree un secret con la parte pública del certificado, la instalación de los módulos está preparada para inyectar dicho certificado en el almacén de certificados de confianza de las máquinas virtuales Java de los contenedores con lo que ya no se producirán errores de verificación. En caso contrario, es decir, si el certificado es oficial no es necesario crear este secret.

Para crear este secret (**IMPORTANTE EL NOMBRE DEL SECRET DEBE SER EXACTAMENTE: cert-domain**) debemos disponer de la parte pública del certificado en un fichero .crt y ejecutar sobre el clúster un comando de la forma:

kubectl create secret

```
kubectl create secret generic cert-domain --from-file=tls.crt=<fichero.crt> -n
<namespace donde se vaya a realizar la instalación>
```

Una vez creado el secret se puede proceder a la instalación de los módulos. Hay que tener en cuenta que hay que indicar en los parámetros del chart que el certificado del dominio ya está instalado (esto se explica en el siguiente apartado).

3.5. Procedimiento de despliegue de Módulos

Para instalar los módulos de MDM se empleará el chart de Helm correspondiente, **onesaithealthcare-mdm-chart**.

Se pueden consultar los prerrequisitos generales donde se indica el repositorio helm a usar, así como distintas formas de instalación.

Se deben tener disponibles los datos de la base de datos (host, puerto, usuarios y passwords) así como conocer el dominio con el que se exponen los módulos.

Estos son los parámetros que requiere el chart, se explica a continuación el valor que se debe informar, los parámetros que no se mencionan en esta lista es porque tienen valores por defecto que en la mayoría de los casos no es necesario modificarlos:

- **global.domain.protocol**: http o https, es el protocolo con el que se exponen los módulos (normalmente https)
- **global.domain.host:** dominio con el que se exponen los módulos
- **global.domain.gateway_type:** k8s_gateway o istio, para indicar si el gateway que se ha configurado es de la api general de kubernetes (gateway.networking.k8s.io/v1) o de la api de istio (networking.istio.io/v1beta1)
- **global.domain.cert_official:** true o false (por defecto true), true indica que el certificado es oficial, **si el certificado no es oficial se informará false y se deberá haber ejecutado el apartado anterior para instalar el secret**
- **global.database.type:** mysql
- **global.database.host**: host o IP de la base de datos
- **global.database.port:** puerto de la base de datos
- **global.database.props:** opcional, propiedades que se añadirán a la cadena de conexión jdbc, por ejemplo para mysql: ?serverTimezone=UTC)
- **global.docker.registry.host:** host del repositorio docker donde se encuentran las imágenes de los módulos (por ejmplo para el caso del repositorio de onesait healthcare el valor sería: docksdtr.indra.es)
- **global.docker.registry.project:** project del repositorio docker donde se encuentran las imágenes de los módulos (por ejemplo para el caso del repositorio de onesait healthcare el valor sería: multhn_cmhn20_2)
- **onesaithealthcare_ohsso_chart.ohsso.db.adminpassword:** password inicial que se asignará al usuario admin de OHSSO
- **onesaithealthcare_ohsso_chart.ohsso.db.schema:** esquema de la BD para el módulo OHSSO
- **onesaithealthcare_ohsso_chart.ohsso.db.user:** usuario de BD para el módulo OHSSO (normalmente será el mismo que el esquema)
- **onesaithealthcare_ohsso_chart.ohsso.db.pass:** password de BD para el módulo OHSSO

A partir de aquí se encontrarán las mismas propiedades para el resto de módulo, lo indicamos de forma genérica para no repetir:

- **onesaithealthcare_<MODULO>_chart.<MODULO>.db.schema:** esquema de la BD para el módulo <MODULO>
- **onesaithealthcare_<MODULO>_chart.<MODULO>.db.user:** usuario de BD para el módulo <MODULO> (normalmente será el mismo que el esquema)
- **onesaithealthcare_<MODULO>_chart.<MODULO>.db.pass**: password de BD para el módulo <MODULO>

La instalación con helm client se realizaría con los siguientes comandos.

Registrar el repositorio y las credenciales de acceso al mismo (si no se ha realizado ya previamente):

helm repo add

```
helm repo add onesait-healthcare-helm-repo
https://nexus.devops.onesait.com/repository/onesait-healthcare-helm-charts --
username <usuario> --password <password>
helm repo update
```

Obtenemos el values.yaml por defecto de MDM:

helm show values

```
helm show values onesait-healthcare-helm-repo/onesaithealthcare-mdm-chart >
values_mdm.yaml
```

Se edita el fichero obtenido y se informan los valores de los campos que se quiera dar a la instalación.

Se instala el chart con el comando:

helm install

```
helm install -f values_mdm.yaml mdm onesait-healthcare-helm-
repo/onesaithealthcare-mdm-chart -n <namespace de instalación, normalmente oh-
modules>
```

El comando helm indicará deployed pero eso únicamente indica que ha sido capaz de crear todos los recursos en el cluster, hay que esperar a que los pods terminen de levantar, para ello chequeamos con el siguiente comando:

kubectl get pods

```
kubectl get pods -n <namespace de instalación>
```

Lo lanzaremos periodicamente hasta que veamos que todos los pods están en estado Running con los contenedores Ready 1/1, es decir, toda la lista de pods debería acabar de esta forma:

output kubectl get pods

```
NAME                                                    READY   STATUS
RESTARTS      AGE
<pod name>                                                              1/1
Running   0             99m
...
```

En caso de que algún no se mostrara Running o presentara reinicios (columna Restarts mayor que 0) se pueden consultar los logs del pod con el comando:

kubectl logs

```
kubectl logs <pod name> -n <namespace de instalación>
```

Para ver los eventos del pod se ejecuta el comando:

kubectl describe pod

```
kubectl describe pod <pod name> -n <namespace de instalación>
```

3.5. Operaciones post-instalación

Una vez se ha instanciado el helm correspondiente y todos los módulos han desplegado correctamente se deben realizar las siguientes tareas.

3.5.1. Scripts POST

Se lanzarán los scripts post instalación ejecutando el lanzador **mdm-post-mysql.sh** de la carpeta:

[FTP_ENTREGAS]
/oradata/Versiones_Producto_OH/OH_v4/sistemas_configuraciones_iniciales_v4/mdm_scripts/MySQL/mdm-global-03-post-mysql

Tras la ejecución de los scripts se deberán reiniciar los siguientes pods:

- ohaut-back
- ohont-back
- ohmpi-back

Se puede hacer ejecutando los siguientes comandos:

kubectl create secret

```
kubectl -n oh-modules scale deployment ohaut-back --replicas 0
kubectl -n oh-modules scale deployment ohont-back --replicas 0
kubectl -n oh-modules scale deployment ohmpi-back --replicas 0
kubectl -n oh-modules scale deployment ohaut-back --replicas 1
kubectl -n oh-modules scale deployment ohont-back --replicas 1
kubectl -n oh-modules scale deployment ohmpi-back --replicas 1
```

3.5.2. OHSSO

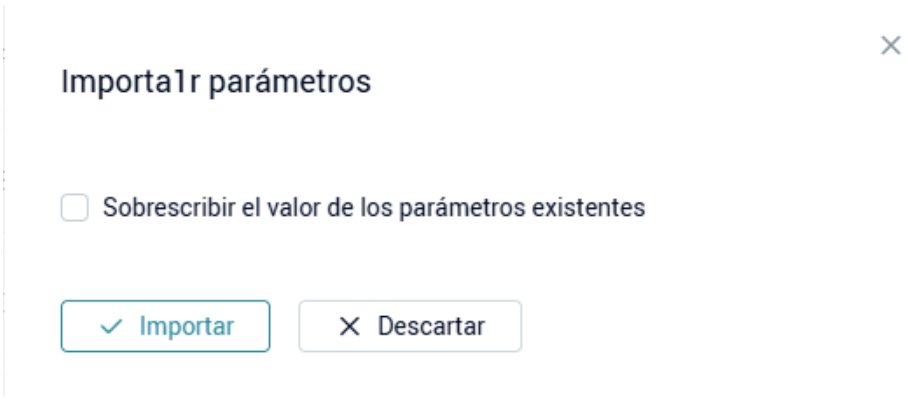
- Acceder a la consola de administración de OHSSO e importar el realm: **oh-base**, a partir del fichero:

[FTP_ENTREGAS]
/oradata/Versiones_Producto_OH/OH_v4/sistemas_configuraciones_iniciales_v4/ohsso/realm-export-oh-base.json

- Acceder a los clients: **hnrole, ruleengine, oh-monitoring y hn-install,** revisar sus **redirectUris** y añadir la del entorno en que se está desplegando (y pulsar **Save** tras añadirlas), es decir, la url pública con la que se expone desde haproxy, por ejemplo, para el clúster de desarrollo sería: https://oh-modules.ohrancherha1-1.indra.es/*
- En el client **hn-install** en la sección: **Authentication Flow Overrides → Browser Flow** → seleccionar: **hn_install**, y pulsar Save.
- Tras realizar esta configuración será posible logarse a los módulos usando las credenciales: **us_install/12345678a**

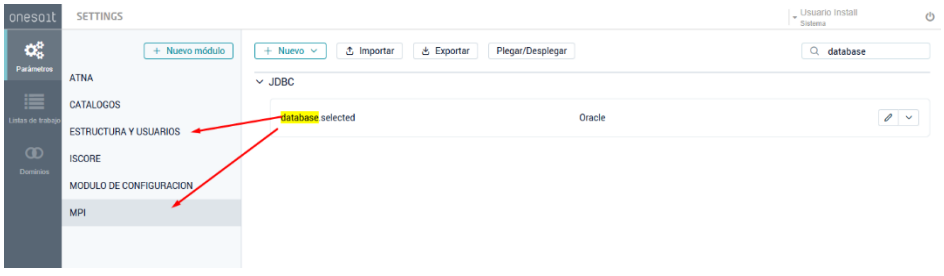
3.5.3. OHCONF

Acceder al módulo OHCON (con las credenciales indicadas anteriormente) e importar las propiedades de los siguientes ficheros, teniendo en cuenta que **NO se debe marcar la opción de sobrescribir**



- **(FTP ENTREGAS) /oradata/Versiones_Producto_OH/OH_v4/OH_CON_v4.8.0/conf/Totales/OHCON/OHCON_HNCONF.csv**
- **(FTP ENTREGAS) /oradata/Versiones_Producto_OH/OH_v4/OH_CON_v4.8.0/conf/Totales/OHCON/OHCON_ISCORE.csv**
- **(FTP ENTREGAS) /oradata/Versiones_Producto_OH/OH_v4/OH_AUT_v4.9.0/conf/Totales/OHCON/OHCON_HNAUT.csv**
- **(FTP ENTREGAS) /oradata/Versiones_Producto_OH/OH_v4/OH_MPI_v4.11.0/conf/Totales/OHCON/OHCON_ISPOB.csv**
- **(FTP ENTREGAS) /oradata/Versiones_Producto_OH/OH_v4/OH_ONT_v4.7.0/conf/Totales/OHCON/OHCON_HNCAT.csv**
- **(FTP ENTREGAS) /oradata/Versiones_Producto_OH/OH_v4/OH_ATN_v4.3.0/conf/Totales/OHCON/OHCON_HNATNA.csv**

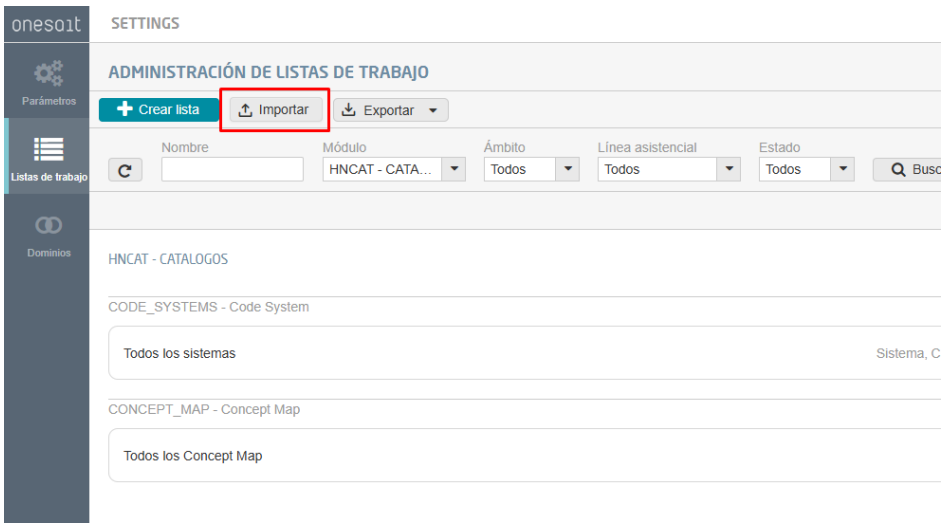
Una vez cargadas las propiedades de los módulos, buscar en MPI y OHAUT la propiedad **"database.selected"**, como valor inicial tiene Oracle, pero según el sistema utilizado podría otro sistema de BD, como por ejemplo MySql.



Deslogarse del módulo OHCON y volver a logarse.

Acceder a la opción de "*Listas de trabajo*" del módulo OHCON e importar el CSV de la ruta:

- **(FTP ENTREGAS) /oradata/Versiones_Producto_OH/OH_v4/OH_CON_v4.8.0/conf/Totales/OHCON/HNCAT_WorkList.csv**



Tras estos cambios reiniciar los despliegues de ohcon-back y ohont-back con los siguientes comandos:

kubectl create secret

```
kubectl -n oh-modules scale deployment ohont-back --replicas 0
kubectl -n oh-modules scale deployment ohcon-back --replicas 0
kubectl -n oh-modules scale deployment ohont-back --replicas 1
kubectl -n oh-modules scale deployment ohcon-back --replicas 1
```

3.5.4. OHONT

Acceder al módulo OHONT (con las credenciales indicadas anteriormente) e importar los catálogos de los siguientes ficheros:

- **(FTP ENTREGAS) /oradata/Versiones_Producto_OH/OH_v4/OH_ONT_v4.7.0/conf/Post/CodeSystem**

Ir al listado de ConceptMap, acceder a listado de elementos del ConceptMap "*Profesion_Roles*" importar los elementos del siguiente fichero:

- **(FTP ENTREGAS) /oradata/Versiones_Producto_OH/OH_v4/OH_ONT_v4.7.0/conf/Post/ConceptMap/*Profesion_Roles.csv***

3.5.5. OHAUT

Crear la configuración de los nodos de la estructura funcional de OHAUT en caso de ser necesaria, para ello seguimos los pasos indicados en [Instalación inicial configuración nodos estructura - StructureDefinition]

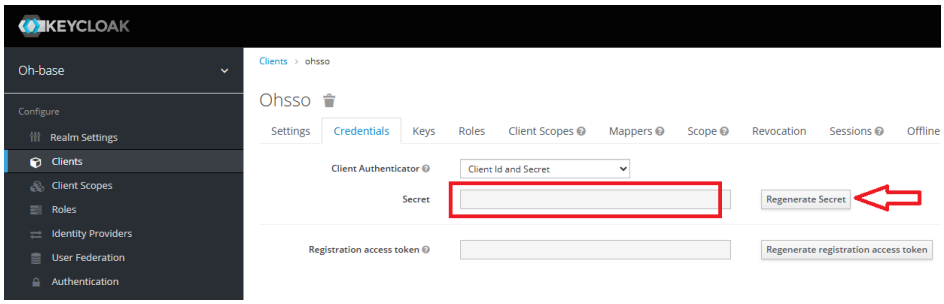
Tras la ejecución de los pasos realizados en los módulos anteriores, se deberán reiniciar los siguientes pods:

- ohaut-back
- ohont-back
- ohmpi-back

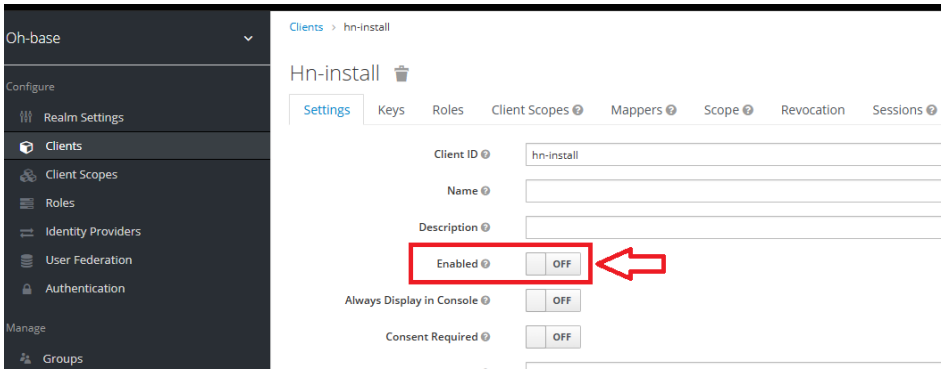
3.5.6. CONFIGURACIÓN DEFINITIVA OHSSO

Tras haber importado correctamente las propiedades de OHCONF y catálogos de OHONT y funcionar correctamente los módulos se configurará el flujo de login estándar de OHSSO contra los profesionales registrados en OHAUT.

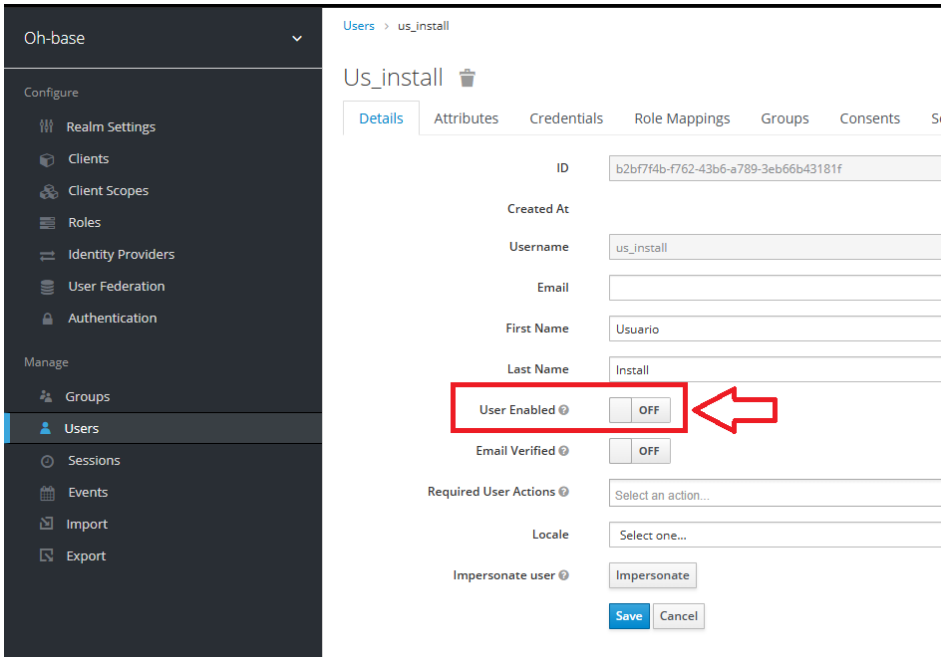
- Acceder a la consola de administración de OHSSO, al client ohsso, y regenerarle las credenciales, anotar el nuevo secret ya que se usará en un paso posterior.



- Deshabilitar el client hn-install, y salvar los cambios.



- Deshabilitar usuario us_install y salvar los cambios.



- En Openshift modificar el configmap **hnhome-iscore-sso-props**, cambiar la propiedad **keycloak.client** al valor **hnrole**. Cambiar **keycloak.login.client_secret** con el nuevo secret regenerado en pasos anteriores, **SE DEBE PEGAR CODIFICADO EN BASE64 (USAR LA PÁGINA** https://www.base64encode.org/ **)**. Se haría con los siguientes comandos:

```
kubectl get configmap hnhome-iscore-sso-props -n oh-modules -o yaml > hnhome-iscore-sso-props.yaml

-- editar el fichero para aplicar los cambios indicados

kubectl apply -f hnhome-iscore-sso-props.yaml -n oh-modules
```

- En Openshift modificar el configmap **ohsso-configmap-realm-oh-base**, cambiar la propiedad **keycloak.login.client_secret** con el nuevo secret regenerado en pasos anteriores, **SE DEBE PEGAR CODIFICADO EN BASE64 (USAR LA PÁGINA** https://www.base64encode.org/ **)**. Se haría con lo siguientes comandos:

```
kubectl get configmap ohsso-configmap-realm-oh-base -n oh-modules -o yaml > ohsso-configmap-realm-oh-base.yaml

-- editar el fichero para aplicar los cambios indicados

kubectl apply -f ohsso-configmap-realm-oh-base.yaml -n oh-modules
```

- En Openshift modificar el confgmap **hnhome-ohsso-front**, cambiar el valor del campo **resource** al valor **hnrole**. Se haría con los siguientes comandos:

```
kubectl get configmap hnhome-ohsso-front -n oh-modules -o yaml > hnhome-ohsso-front.yaml

-- editar el fichero para aplicar los cambios indicados

kubectl apply -f hnhome-ohsso-front.yaml -n oh-modules
```

- Reiniciar todos los pods del namespace con el siguiente comando. Una vez se complete el reinicio ya se podrá acceder a todos los módulos con el usuario us_admin configurado por defecto.

```
kubectl delete --all pods --namespace=oh-modules
```

4. Instalación Paquete DATA

4.1. Módulos que incluye

- Global Repository (OH_HDR)
- Visor Historia Clínica (OH_HDA)
- Consent Manager (OH_CSM)

4.2. Prerrequisitos

- Se deben cumplir los prerrequisitos generales y haber realizado los pasos indicados en la guía correspondiente.
- Debe instalarse previamente el paquete MDM.

4.2.1. Prerrequisitos de Base de Datos

Se debe disponer de un gestor de base de datos MySQL que tenga visibilidad desde los nodos worker del clúster de Kubernetes en el que se desplegarán los módulos.

A continuación, se indica como crear los usuarios y esquemas necesarios para los módulos de DATA.

MySQL con acceso de administrador

Disponiendo de usuario administrador se pueden crear los usuarios y esquemas para los módulos DATA con los siguientes scripts:

- Global Repository

[FTP_ENTREGAS] /oradata/Versiones_Producto_OH/OH_v[MAJOR VERSION]/OH_HDR_v[VERSION]/bbdd/MySQL8/1-ohhdr-user.sql

- Consent Manager
[FTP_ENTREGAS] /oradata/Versiones_Producto_OH/OH_v[MAJOR VERSION]/OH_CSM_v[VERSION]/bbdd/total/mysql/1-ohcsm-user.sql

Sin acceso de administrador

Si no se dispone de acceso de administrador se tendrá que solicitar la creación de los siguientes usuarios, cada uno de ellos con un esquema sobre el que tenga permisos y con el mismo nombre del usuario:

- us_hdr (el usuario debe tener permiso "grant option" sobre su esquema para poder asignar permiso a otros usuarios a objetos de su esquema)
- us_ohcsm (el usuario debe tener permiso "grant option" sobre su esquema para poder asignar permiso a otros usuarios a objetos de su esquema)

4.3. Procedimiento de despliegue de Capa de Persistencia

4.3.1. MySQL

- Global Repository

Con el usuario propio del módulo lanzar los siguientes scripts:

[FTP_ENTREGAS] /oradata/Versiones_Producto_OH/OH_v[MAJOR VERSION]/OH_HDR_v[VERSION]/bbdd/MySQL8/2-ohhdr-create-database.sql

[FTP_ENTREGAS] /oradata/Versiones_Producto_OH/OH_v[MAJOR VERSION]/OH_HDR_v[VERSION]/bbdd/MySQL8/3-ohhdr-tables.sql

- Consent Manager

Con el usuario propio del módulo lanzar los siguientes scripts:

[FTP_ENTREGAS] /oradata/Versiones_Producto_OH/OH_v[MAJOR VERSION]/OH_CSM_v[VERSION]/bbdd/total/mysql/2-ohcsm-create-database.sql

[FTP_ENTREGAS] /oradata/Versiones_Producto_OH/OH_v[MAJOR VERSION]/OH_CSM_v[VERSION]/bbdd/total/mysql/3-ohcsm-tables.ddl.sql

4.4. Procedimiento de despliegue de Módulos

Para instalar los módulos del paquete DATA se deberá realizar la instalación del chart de Helm **onesaithealthcare-data-chart**. Las opciones de instalación del chart en función del tipo de entorno vienen descritas en el apartado inicial de "Requisitos de la instalación".

La instalación con helm client se realizaría con los siguientes comandos.

Obtenemos el values.yaml por defecto de DATA:

helm show values

```
helm show values onesait-healthcare-helm-repo/onesaithealthcare-data-chart > values_data.yaml
```

Se edita el fichero obtenido y se informan los valores de los campos que se quiera dar a la instalación.

Se deben tener disponibles los datos de la base de datos (host, puerto, usuarios y passwords) así como conocer el dominio con el que se exponen los módulos.

Estos son los parámetros que requiere el chart, se explica a continuación el valor que se debe informar, los parámetros que no se mencionan en esta lista es porque tienen valores por defecto que en la mayoría de los casos no es necesario modificarlos:

- **global.domain.protocol**: http o https, es el protocolo con el que se exponen los módulos (normalmente https)
- **global.domain.host:** Dominio con el que se exponen los módulos

- **global.domain.gateway_type:** k8s_gateway o istio, para indicar si el gateway que se ha configurado es de la api general de kubernetes (gateway.networking.k8s.io/v1) o de la api de istio (networking.istio.io/v1beta1)
- **global.database.type:** mysql
- **global.docker.registry.host:** Host del repositorio docker donde se encuentran las imágenes de los módulos (por ejemplo para el caso del repositorio de onesait healthcare el valor sería: docksdtr.indra.es)
- **global.docker.registry.project:** project del repositorio docker donde se encuentran las imágenes de los módulos (por ejemplo para el caso del repositorio de onesait healthcare el valor sería: multhn_cmhn20_2)
- **global.docker.registry.secret:** Secret con las credenciales de acceso al docker registry. (Por ejemplo en nuestro caso el secret sería: oh-docker-creds)
- **dependencies.ohhdr/ohcsm/ohhda**: Por defecto se dejan en true.
- **onesaithealthcare_ohhdr_chart.ohhdr.database.url:** Cadena de conexión a la BD. (para BD MySQL sería: jdbc:mysql://database_host:port/db_schema).
- **onesaithealthcare_ohhdr_chart.ohhdr.database.user:** Usuario de BD para el módulo OHHDR (normalmente será el mismo que el esquema). En este caso sería: us_hdr
- **onesaithealthcare_ohhdr_chart.ohhdr.database.pass:** Password de BD para el módulo OHHDR
- **onesaithealthcare_ohcsm_chart.ohcsm.database.url:** Cadena de conexión a la BD. (para BD MySQL sería: jdbc:mysql://database_host:port/db_schema).
- **onesaithealthcare_ohcsm_chart.ohcsm.database.user:** Usuario de BD para el módulo OHCSM (normalmente será el mismo que el esquema). En este caso sería: us_ohcsm
- **onesaithealthcare_ohcsm_chart.ohcsm.database.pass:** Password de BD para el módulo OHCSM.

Se instala el chart con el comando:

helm install

```
helm install -f values_data.yaml data onesait-healthcare-helm-
repo/onesaithealthcare-data-chart -n <namespace de instalación, normalmente oh-
modules>
```

El comando helm indicará deployed pero eso únicamente indica que ha sido capaz de crear todos los recursos en el cluster, hay que esperar a que los pods terminen de levantar, para ello chequeamos con el siguiente comando:

kubectl get pods

```
kubectl get pods -n <namespace de instalación>
```

Lo lanzaremos periódicamente hasta que veamos que todos los pods están en estado Running con los contenedores Ready 1/1, es decir, toda la lista de pods debería acabar de esta forma:

output kubectl get pods

```
NAME                                                    READY   STATUS
RESTARTS      AGE
<pod name>                                                      1/1
Running   0             99m
```

...

En caso de que algún no se mostrara Running o presentara reinicios (columna Restarts mayor que 0) se pueden consultar los logs del pod con el comando:

kubectl logs

```
kubectl logs <pod name> -n <namespace de instalación>
```

Para ver los eventos del pod se ejecuta el comando:

kubectl describe pod

```
kubectl describe pod <pod name> -n <namespace de instalación>
```

4.5. Operaciones post-instalación

4.5.1. Visor Historia Clínica (OH_HDA)

Setting

Importar los ficheros de la ruta:

[FTP_ENTREGAS] /oradata/Versiones_Producto_OH/OH_v[MAJOR VERSION]/OH_HDA_v[VERSION]/configuración OHCON/

4.5.2. Consent Manager (OH_CSM)

Setting

Importar los ficheros de la ruta:

[FTP_ENTREGAS] /oradata/Versiones_Producto_OH/OH_v[MAJOR VERSION]/OH_CSM_v[VERSION]/Configuración OHCON/

Ontology

- Comprobar que existe el siguiente catálogo en Ontology: http://hl7.org/fhir/CodeSystem/CONSENT_TYPE
  - Si no existe, crearlos con las siguientes propiedades:
    Title: CONSENT_TYPE
    Name: CONSENT_TYPE
    Hierarchy: Is-A
    Content: Complete
    Status: Activo
    Properties:
      Boolean - createConsentManager
      Boolean - editConsentManager
      String - tipo
      Boolean - singleConsent
- Crear las siguientes entradas en el catálogo CONSENT_TYPE:
  * Code: CRL
    Display: Solicitar Cuidador
    tipo: C
    createConsentManager: true

editConsentManager: true
singleConsent: false

* Code: IPP
Display: Inclusión de paciente a programa
tipo:
createConsentManager: false
8editConsentManager: false
singleConsent: false

5. Instalación Paquete Integration & Interoperability

5.1. Módulos que incluye

- **OHIEN (Control Panel)**

Panel centralizado de control para visualizar topics, grupos de consumidores, integraciones, esquemas y estado del ecosistema Kafka.

- **Clúster Kafka**

Cluster de Kafka con configuración adaptada para OHIEN en modo KRaft (sin ZooKeeper).

- **Kafka Connect**

Plataforma de integración para conectar fuentes y destinos con Kafka, gestionado desde el módulo Control Panel.

- **Schema Registry**

Almacena y gestiona los esquemas Avro, JSON Schema y Protobuf utilizados en Kafka.

- **Kafka JMX Exporter (opcional, en kafkaien)**

Exponer métricas Kafka, configurable en `values.yaml`. Permite exportar MBeans estándar como:
- `kafka.server`, `kafka.controller`, `java.lang`, etc.

- **Operador camel-k**

Instalación del operador camel-k necesario para el despliegue de las integraciones desarrolladas en camelk.

Importante:

```
La instalación de OHIEN y su entorno (Kafka, Kafka Connect, Schema Registry,
Control Panel) se realiza con un chart de Helm específico.

Si necesitas usar integraciones desarrolladas con Camel, debes instalar un
segundo chart para Camel K.
Ambos charts pueden instalarse de forma independiente, pero para un entorno
completo se recomienda instalar ambos.
```

5.2. Prerrequisitos

- Kubernetes (v1.24+)

- Helm (v 3.2+)

- Para OHIEN es necesario la instalación previa de MDM y debe realizarse sobre el mismo namespace en el que se encuentre desplegado MDM.

- Se debe disponer de un *StorageClass* con capacidad de aprovisionamiento dinámico de volúmenes persistentes que soporte *block storage*.

- Para que la instalación del operador Camel K funcione correctamente, se debe tener acceso a un registro de contenedores Docker (tipo Docker Registry) accesible desde los nodos del clúster, con soporte tanto para operaciones de push como de pull de imágenes. Esto es necesario porque Camel K construye y publica imágenes personalizadas (IntegrationKits) en tiempo de ejecución, las cuales luego serán utilizadas por las integraciones desplegadas.

5.3. Pasos previos a la instalación

5.3.1. StorageClass

Debes disponer de un StorageClass adecuado para los volúmenes persistentes de Kafka y otros módulos. La creación dependerá de los drivers disponibles en tu clúster Kubernetes.

Importante:

```
Asegúrate de que el StorageClass seleccionado soporte block storage y
aprovisionamiento dinámico.
```

Ejemplo para AWS EKS

A continuación se muestran 2 ejemplos de configuración de StorageClass en clústeres EKS

Ejemplo 1: EKS con aprovisionamiento automático habilitado (EBS CSI Driver)

En un cluster EKS de AWS con modo automático habilitado ser puede crear el StorageClass con los siguientes yaml.

StorageClass kafka-ebs auto mode

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: kafka-ebs
provisioner: ebs.csi.eks.amazonaws.com
volumeBindingMode: WaitForFirstConsumer
reclaimPolicy: Retain
parameters:
  type: gp3
```

Ejemplo 2: EKS sin aprovisionamiento automático habilitado

En un cluster EKS de AWS sin el modo automático habilitado, el aprovisionamiento de volúmenes EBS requiere que el EBS CSI Driver esté instalado y configurado manualmente. El StorageClass se define de la siguiente manera:

StorageClass kafka-ebs auto mode

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: kafka-ebs
provisioner: ebs.csi.aws.com
volumeBindingMode: WaitForFirstConsumer
reclaimPolicy: Retain
parameters:
  type: gp3
```

5.4. Procedimiento de despliegue de Módulos

5.4.1. Instalación de OHIEN con helm

Instalación OHIEN con helm

Se puede consultar el apartado 00 Requisitos de instalación donde se explica la instalación con helm,

1. Obtén el archivo de valores por defecto, para ello utiliza el siguiente comando.

```
helm show values onesait-healthcare-helm-repo/onesaithealthcare-iengine-chart > values_ohien_iengine.yaml
```

2. Edita `values_ohien_iengine.yaml` generado en el paso anterior. En nuestro ejemplo utilizaremos `nano` para añadir los parámetros necesarios (se guarda con CTRL+O y se sale con CTRL+X) o se puede usar cualquier otro editor con el que esté familiarizado:

```
nano values_ohien_iengine.yaml
```

Estos son los parámetros que requiere el chart, se explica a continuación el valor que se debe informar, los parámetros que no se mencionan en esta lista es porque tienen valores por defecto que en la mayoría de los casos no es necesario modificarlos:

global

- `**global.registry.host**`: (Obligatorio) El hostname del registro de contenedores donde están las imágenes (ejemplo: dock.sdtr.indra.es, index.docker.io, etc.).
- `**global.registry.project**`: (Obligatorio) El proyecto o namespace dentro del registro donde están las imágenes.
- `**global.domain.protocol**`: (Obligatorio) Protocolo para acceder al dominio, normalmente `http` o `https`.
- `**global.domain.host**`: (Obligatorio) El nombre de dominio o IP del host donde se expondrán los servicios.
- `**global.domain.port**`: (Obligatorio) Puerto de acceso al dominio. Usa 80 para HTTP o 443 para HTTPS.

- `**global.istio_enabled**`: (Opcional) Si usas Istio para malla de servicios, pon `true`. Si no, deja `false`.

onesaithealthcare_ohien_kafkaien_chart

- `**onesaithealthcare_ohien_kafkaien_chart.image.tag**`: Versión de la imagen Docker de KafkaIEN. Por defecto `4.0.0`.
- `**onesaithealthcare_ohien_kafkaien_chart.kafka.createMode**`: `verify` (no reemplaza recursos existentes) o `create` (reemplaza si existen). Usa `verify` salvo que sepas que necesitas sobrescribir recursos.
- `**onesaithealthcare_ohien_kafkaien_chart.kafka.replicas**`: Número de brokers Kafka. Para entornos productivos, usa al menos 3 y cambia el tipo de almacenamiento a `persistent`.
- `**onesaithealthcare_ohien_kafkaien_chart.kafka.storage.type**`: `ephemeral` (datos temporales, se pierden al reiniciar) o `persistent` (datos se conservan). Para pruebas, puedes dejar `ephemeral`. *Para producción, usa `persistent`.*
- `**onesaithealthcare_ohien_kafkaien_chart.kafka.storage.size**`: Tamaño del volumen de almacenamiento (ejemplo: `2Gi`, `10Gi`).
- `**onesaithealthcare_ohien_kafkaien_chart.kafka.storage.storageClass**`: StorageClass de Kubernetes a usar (ejemplo: `standard`, `kafka-ebs`). Puede estar vacío en modo `ephemeral`.
- `**onesaithealthcare_ohien_kafkaien_chart.kafka.storage.deleteClaim**`: Si es `true`, el PVC se elimina al borrar el release de Helm. Para producción, normalmente se deja en `false`.

onesaithealthcare_ohien_kafka_connect_chart

- `**onesaithealthcare_ohien_kafka_connect_chart.image.tag**`: Versión de la imagen Docker de Kafka Connect. Por defecto `1.2.8`.

onesaithealthcare_ohien_schemaregistry_chart

- `**onesaithealthcare_ohien_schemaregistry_chart.image.tag**`: Versión de la imagen Docker de Schema Registry. Por defecto `5.0.0`.

onesaithealthcare_ohien_control_panel_chart

- `**onesaithealthcare_ohien_control_panel_chart.image.backend.tag**`: Versión de la imagen Docker del backend del Control Panel. Por defecto `3.13.0`.
- `**onesaithealthcare_ohien_control_panel_chart.image.frontend.tag**`: Versión de la imagen Docker del frontend del Control Panel. Por defecto `3.13.0`.
- `**onesaithealthcare_ohien_control_panel_chart.replicas**`: Número de réplicas del Control Panel. Por defecto `0`
- `**onesaithealthcare_ohien_control_panel_chart.timezone**`: Zona horaria de la aplicación. Por defecto `UTC`.

- `**onesaithealthcare_ohien_control_panel_chart.elasticsearch_enabled**`: Habilita o deshabilita la integración con Elasticsearch. Por defecto `false`.

Importante:

```
Reemplaza todos los valores `<CHANGE_ME_...>` por los datos reales de tu entorno
antes de instalar.
```

Finalmente se instala con:

```
helm install -f values_iengine.yaml iengine onesait-healthcare-helm-
repo/onesaithealthcare-iengine-chart -n oh-modules
```

Se indicará que se ha instalado el chart (el tiempo puede ser elevado hasta que responda).

Se comprueba que los siguientes pods están en estado Running 1/1: kafka-connect-xxxxxx (es decir un pod con el prefijo kafka-connect)

ohien-kakfa-<N> (apareceán tantos pods de esta forma como nodos de cluster kafka se haya indicado en la instalación)

schemaregistry-xxxxxx (es decir un pod con el prefijo schemaregistry)

Con el comando:

```
kubectl -n <namespace de instalacion> get pods
```

5.4.2. Operaciones post-instalación

Una vez se ha instanciado el helm correspondiente y todos los módulos han desplegado correctamente se deben realizar las siguientes tareas.

Configuración Posterior en HNCONF

Una vez desplegado el ecosistema, debe completarse la configuración del módulo **OHIEN** dentro del sistema HNCONF.

Desde la dirección FTP Entregas, descargar el fichero de

(FTP ENTREGAS) /oradata/Versiones_Producto_OH/OH_v3/OH_IEN_v3.14.0/hnconf/V4/total/OHCON_OHIEN_202485.csv

Luego de ingresar en HNCONF, utilizar la opción de "Importar" y seleccionar el fichero: OHCON_OHIEN_FIXED.csv.

Importación en HNCONF

1. Ingrese a **HNCONF**.
2. Diríjase a **Settings > Integration Engine**.
3. Haga clic en **Importar** y seleccione el fichero descargado:
   `OHCON_OHIEN_FIXED.csv`
4. Una vez importado, actualice los siguientes parámetros con los valores adecuados según el despliegue actual.

Parámetros de Configuración

- GENERAL

Parámetro	Descripción
`ohien.kafka_connect.namespace`	Namespace Kubernetes donde se despliega Kafka Connect (normalmente: `oh-modules`)

- KAFKA

Parámetro	Descripción
`ohien.kafka.config.connect.password`	Contraseña del usuario Kafka utilizado por OHIEN. Obtener desde el `Secret` llamado `ohien` en el namespace correspondiente, revelando su valor.

Diríjase a **Settings > ISCORE

- ISCORE

Parámetro	Descripción
`ohien.url`	Se debe cambiar el valor a: "${general.url}/ohien/api"

Finalmente arrancamos el módulo Control Panel, para ello ejecutamos el siguiente comando:

```
kubectl -n <namespace de instalación> scale deployment ohien-control-panel --
replicas 1
```

5.4.3. Instalación de Camel K con Helm

Prerrequisitos

Es necesario disponer en un docker registry donde se generarán las imágenes de las integraciones desplegadas con Camel K.

Se debe crear un secret con las credenciales a dicho repositorio, la cuenta usada debe tener permisos de pull y push.

Importante el secret debe tener como nombre "**camel-registry**" puesto que dicho nombre se referencia en el chart de instalación.

```
kubectl create secret docker-registry camel-registry --docker-server=<host-del-
docker-registry> --docker-username=<user-docker-registry> --docker-
password=<pass-docker-registry> -n oh-modules
```

Instalación Camel K con helm

Se lanzaría el siguiente comando para obtener el ficheros de configuración por defecto.

```
helm show values onesait-healthcare-helm-repo/onesaithealthcare-ohien-camelk-
chart > values_ohien_camelk.yaml
```

Estos son los parámetros que requiere el chart, se explica a continuación el valor que se debe informar, los parámetros que no se mencionan en esta lista es porque tienen valores por defecto que en la mayoría de los casos no es necesario modificarlos:

- `**ohien.image**`: Indica la imagen Docker que se usará para la aplicación principal (por ejemplo, Eclipse Temurin JDK 17).
- `**ohien.toolImage**`: Indica la imagen Docker que se usará para el entorno de herramientas de compilación (por ejemplo, Quarkus Mandrel builder con JDK 21).
- `**camel-k-operator.operator.enabled**`: Activa (`true`) o desactiva (`false`) el operador Camel K.
- `**camel-k-operator.operator.image**`: Indica la imagen Docker que usará el operador Camel K.
- `**integrationPlatform.registry.mode**`: Define el modo del registro de contenedores de la plataforma de integración. Los valores posibles son: `external` (externo) o `local` (interno).
- `**integrationPlatform.registry.address**`: Dirección (host) del registro de contenedores externo. Sustituye `<CHANGE_ME_REGISTRY_HOST>` por el nombre o dirección de tu registro.
- `**integrationPlatform.registry.organization**`: Nombre de la organización o proyecto dentro del registro de contenedores. Sustituye `<CHANGE_ME_REGISTRY_PROJECT>` por el nombre de tu proyecto.
- `**integrationPlatform.registry.secret**`: Nombre del secreto en Kubernetes usado para autenticarse en el registro (`camel-registry`).
- `**integrationPlatform.registry.insecure**`: Si lo pones en `true`, permite conexiones no seguras al registro; si quieres que sean seguras, usa `false`.

- `**maven.customRepo.enabled**`: Activa (`true`) o desactiva (`false`) el uso de un repositorio Maven personalizado.
- `**maven.customRepo.id**`: Identificador del repositorio Maven personalizado (por ejemplo, `local-nexus`).
- `**maven.customRepo.url**`: Dirección (URL) del repositorio Maven personalizado. Sustituye `<CHANGE_ME_REPOSITORY_URL>` por la URL de tu repositorio.
- `**maven.customRepo.username**`: Usuario para autenticarse en el repositorio Maven personalizado. Sustituye `<CHANGE_ME_REPOSITORY_USERNAME>` por tu usuario.
- `**maven.customRepo.password**`: Contraseña para autenticarse en el repositorio Maven personalizado. Sustituye `<CHANGE_ME_REPOSITORY_PASSWORD>` por tu contraseña.

Se edita el fichero `values_ohien_camelk.yaml` generado en el paso anterior. En nuestro ejemplo utilizaremos `nano` para añadir los parámetros necesarios (se guarda con CTRL+O y se sale con CTRL+X) o se puede usar cualquier otro editor con el que esté familiarizado:

```
nano values_ohien_camelk.yaml
```

Importante:

Reemplaza todos los valores `<CHANGE_ME_...>` por los datos reales de tu entorno antes de instalar.

Finalmente se instala con:

```
helm install -f values_camelk.yaml camelk onesait-healthcare-helm-repo/onesaithealthcare-ohien-camelk-chart -n oh-modules
```

6. Instalación Paquete Monitorización

6.1. Módulos que incluye

- Prometheus
- Grafana
- Dashboards

6.2. Prerrequisitos

- Este chart se debe instalar en el namespace donde se encuentre instalado MDM.
- Se debe haber instalado y configurado previamente el chart MDM (onesaithealthcare-mdm-chart).
- Si se desea tener persistencia de las métricas frente a reinicios de los pods se debe disponer de un Storage Class con provisión dinámica.
- Acceder a OHSSO, si no se tiene instalado el client oh-monitoring seguir siguientes pasos, en caso contrario ignorar este punto.

Acceder a Client Scopes y añadir los scopes (de tipo openid-connect): openid, groups.

Importar el client oh-monitoring con el json:

(FTP_ENTREGAS) /oradata/Versiones_Producto_OH/OH_v4/sistemas_configuraciones_iniciales_v4/ohsso/client-oh-monitoring.json

Revisar que las redirectUris se corresponden con las del entorno (añadir también las redirect uris que se vayan a asignar a prometheus y grafana).

Acceder a User Federation y en hn-provider en la lista de Included clients añadir: oh-monitoring.

- Tanto si existía el client oh-monitoring como si se acaba de instalar, **acceder a su pestaña de credenciales, regenerar el secret y apuntarlo** ya que se requerirá en la instalación del chart.
- En Openshift editar el SecurityContextConstraint anyuid y en la lista de users añadirle:

```
system:serviceaccount:<namespace-donde-se-instalara_el_chart>:oh-prometheus
system:serviceaccount:<namespace-donde-se-instalara_el_chart>:oh-grafana
```

Por ejemplo:

```
system:serviceaccount:oh-modules:oh-prometheus
system:serviceaccount:oh-modules:oh-grafana
```

- Comprobar que la NetworkPolicy cluster-network-policy-<namespace_instalacion> contiene los puertos: 9090, 9091 y 3000.

6.3. Procedimiento de despliegue de Capa de Persistencia

No aplica ya que los módulos desplegados no requieren de base de datos.

En caso de querer habilitar la persistencia de prometheus, como se indica en los prerrequisitos, se deberá disponer de un StorageClass con provisión dinámica.

6.4. Procedimiento de despliegue de Módulos

Consultar la guía general de prerrequisitos donde se indican varias alternativas para registrar el repositorio helm e instalar el chart.
El repositorio se encuentra en:
https://nexus.devops.onesait.com/repository/onesait-healthcare-helm-charts
Se deberá disponer de credenciales para poder acceder al mismo, dichas credenciales serán proporcionadas por el equipo de Onesait Healthcare.

6.5. Operaciones post-instalación

1. Verificar el acceso a prometheus con la URL <url_oh_modules>/prometheus y que en la sección Status -> Target se están alcanzando los endpoints de métricas de Kafka, OH Modules y OHSSO.

2. Verificar el acceso a grafana con la URL <url_oh_modules>/grafana y que en la sección dashboards aparecen los paneles correspondientes a los targets de prometheus y se visualizan correctamente.

7. Instalación Paquete Analytics

7.01. Instalación Módulo Ingesta

7.1. Módulos que incluye

- Módulo de Ingesta (OHDTS)

7.2. Prerrequisitos

Se deben cumplir los prerrequisitos generales y haber realizado los pasos indicados en la guía correspondiente.

7.2.1. Prerrequisitos de Base de Datos

Se debe disponer de un gestor de base de datos MySQL que tenga visibilidad desde los nodos worker del clúster de Kubernetes en el que se desplegará el módulo.

Tener creados los siguientes esquemas con sus tablas correspondientes (No es un prerrequisito necesario para la instalación, pero si para la ejecución correcta de este módulo.):

- us_hdr
- us_hnatna
- us_hnaut
- us_hncat
- us_hncard
- us_hnpob
- us_ohbpm

7.3. Procedimiento de despliegue de Capa de Persistencia

Si la capa de persistencia va sobre MySql, se creará el modelo de datos del módulo OH_DTS:

[FTP_ENTEGRAS]
/oradata/Versiones_Producto_OH/OH_v4/OH_DTS_v[version]/bbdd/total/mysql

7.4. Procedimiento de despliegue de Módulos

Para instalar el módulo del BI se empleará el chart de Helm correspondiente, **onesaithealthcare-dts-chart**.

Se pueden consultar los prerrequisitos generales donde se indica el repositorio helm a usar, así como distintas formas de instalación.

La instalación con helm client se realizaría con los siguientes comandos.

Obtenemos el values.yaml por defecto de DTS:

helm show values

```
helm show values onesait-healthcare-helm-repo/onesaithealthcare-dts-chart > values_dts.yaml
```

Se edita el fichero obtenido y se informan los valores de los campos que se quiera dar a la instalación.

Se deben tener disponibles los datos de la base de datos (host, puerto, usuarios y passwords) así como conocer el dominio con el que se exponen los módulos.

Estos son los parámetros que requiere el chart, se explica a continuación el valor que se debe informar, los parámetros que no se mencionan en esta lista es porque tienen valores por defecto que en la mayoría de los casos no es necesario modificarlos:

- **global.domain.gateway_type:** k8s_gateway o istio, para indicar si el gateway que se ha configurado es de la api general de kubernetes (gateway.networking.k8s.io/v1) o de la api de istio (networking.istio.io/v1beta1)
- **global.docker.registry.host:** Host del repositorio docker donde se encuentran las imágenes de los módulos (por ejemplo para el caso del repositorio de onesait healthcare el valor sería: docksdtr.indra.es)
- **global.docker.registry.project:** project del repositorio docker donde se encuentran las imágenes de los módulos (por ejemplo para el caso del repositorio de onesait healthcare el valor sería: multhn_cmhn20_2)
- **global.docker.registry.secret:** Secret con las credenciales de acceso al docker registry. (Por ejemplo en nuestro caso el secret sería: oh-docker-creds)
- **dts.module.app_name:** Nombre de la aplicación. Por defecto sera oh-dts
- **dts.module.schedule:** Periodicidad de ejecución del cronJob. Se debe poner en formato crontab.
- **dts.module.max_days:** Indica los días a ingestar a partir de la fecha final del último periodo ingestado. Será 0 si se quiere ingestar todo.
- **dts.module.start_date.value:** Fecha de inicio de la ingesta para su primera ejecución. Con esto hacemos que la primera ingesta empiece en la fecha que mejor nos convenga, evitando así ingestar periodos en los que no hay datos.
- **dts.module.cfg.(insercion,extracion,routes).deploy:** Check para indicar si se quieren desplegar los ConfigMaps de inserción, extracción y routes. Por defecto lo dejamos a true

- **dts.database.type:** Mysql. Indicamos el tipo de BBDD a la que apuntamos

A partir de aquí se encontrarán las mismas propiedades de BBDD para el resto de componentes de dts (datasource, iengine, datastore) lo indicamos de forma genérica para no repetir:

- **dts.database.<MODULO>.driver:** Driver de conexión a la BD. (para BD MySQL sería: database.driverClassName:com.mysql.cj.jdbc.Driver)
- **dts.database.<MODULO>.url:** Cadena de conexión a la BD. (para BD MySQL sería: jdbc:mysql://database_host:port/db_schema)

- **dts.database.<MODULO>.username:** usuario de BD para el módulo <MODULO> (normalmente será el mismo que el esquema)
- **dts.database.<MODULO>.password**: password de BD para el módulo <MODULO>

Se instala el chart con el comando:

helm install

```
helm install -f values_dts.yaml dts onesait-healthcare-helm-repo/onesaithealthcare-dts-chart -n <namespace de instalación, normalmente oh-modules>
```

El comando helm indicará deployed. A diferencia de otros modulos que al desplegar levantan un pod, **DTS es un cronjob que lanza su pod a la hora que se haya indicado en el despliegue**, si todo va bien a la hora correspondiente se creara el pod de DTS, para ello chequeamos con el siguiente comando:

kubectl get pods

```
kubectl get pods -n <namespace de instalación>
```

Comprobamos que se ha creado el cronjob con el siguiente comando:

kubectl get cronjobs

```
kubectl get cronjobs -n <namespace de instalación>
```

```
:~$ kubectl get cronjobs -n
NAME     SCHEDULE      TIMEZONE   SUSPEND   ACTIVE   LAST SCHEDULE   AGE
oh-dts   12 12 * * *   <none>     False     0        20h             20h
```

Se pueden consultar los logs del pod con el comando:

kubectl logs

```
kubectl logs <pod name> -n <namespace de instalación>
```

Para ver los eventos del pod se ejecuta el comando:

kubectl describe pod

```
kubectl describe pod <pod name> -n <namespace de instalación>
```

7.02. Instalación Framework BI

7.1. Módulos que incluye

- Módulo OHBI

7.2. Prerrequisitos

Se deben cumplir los prerrequisitos generales y haber realizado los pasos indicados en la guía correspondiente.

7.2.1. Prerrequisitos de Base de Datos

Si los objetos creados por esta aplicación se van a persistir en una bbdd MySql, se debe disponer de un gestor de base de datos MySQL que tenga visibilidad desde los nodos worker del cluster de kubernetes en el que se desplegará el módulo. Crear un esquema vacío y un usuario de acceso al mismo (el modelo de datos se creará automáticamente al arrancar el módulo).

Para la creación del esquema se ejecutará el siguiente script 'Creación esquema OHBI.sql' **(FTP ENTREGAS) /oradata/Versiones_Producto_OH/OH_v4/OH_BI_v[VERSION]/bbdd/total/mysql**

La otra opción sería persistir estos objetos creados sobre la BBDD Derby embebida que incluye el módulo, en este caso no habrá que hacer lo anterior expuesto como prerrequisitos de base de datos.

7.2.2. Prerrequisitos PV

Si el storaceClass que se vaya a indicar en la instalación del módulo no tiene la capacidad de provisionar dinámicamente el PV habrá que crear un volumen persistente manualmente, este volumen persistente contendrá los drivers de conexión disponibles en el módulo.

En la siguiente ruta hemos dejado el yaml necesario para crear el PV correspondiente: **(FTP ENTREGAS) /oradata/Versiones_Producto_OH/OH_v4/OH_BI_v[VERSION]/src/openshift/oh-bi/PersistentVolume_driver.yaml**

Si se va a usar Derby para la persistencia de los objetos creados por el módulo, y el storageClass referenciado por el PVC no tiene la capacidad de provisionar dinámicamente el PV, habrá que crear otro volumen persistente más.

En la siguiente ruta hemos dejado el yaml necesario para crear el PV correspondiente: **(FTP**

ENTREGAS) /oradata/Versiones_Producto_OH/OH_v4/OH_BI_v[VERSION]/src/openshift/oh-bi/PersistentVolume_db.yaml

Tan solo habrá que cambiar en cada uno de los PV's a crear respecto a los entregados, lo siguiente (acorde a lo proporcionado por el administrador de sistemas una vez haya creado el filesystem para cada uno):

- la capacidad del espacio físico (storage),
- el servidor donde se encuentra este espacio físico (server)
- y su ruta (path)
- y el storageClassName por el vuestro

Si no se tiene un storageClass creado, se podrá dejar el mismo que trae los PV de la entrega, creando antes el storageClass mediante el siguiente fichero:

(FTP ENTREGAS) /oradata/Versiones_Producto_OH/OH_v4/OH_BI_v[VERSION]/src/openshift/oh-bi/StorageClass.yaml

kubectl create StorageClass

```
kubectl apply -f StorageClass.yaml
```

7.3. Procedimiento de despliegue de Módulos

Para instalar el módulo OH_BI se empleará el chart de Helm correspondiente, **onesaithealthcare-bi-chart**.

Se pueden consultar los prerrequisitos generales donde se indica el repositorio helm a usar así como distintas formas de instalación.

La instalación con helm client se realizaría con los siguientes comandos.

Obtenemos el values.yaml por defecto de BI:

helm show values

```
helm show values onesait-healthcare-helm-repo/onesaithealthcare-bi-chart > values_bi.yaml
```

Se edita el fichero obtenido y se informan los valores de los campos que se quiera dar a la instalación.

Se deben tener disponibles los datos de la base de datos (host, puerto, usuarios y passwords) así como conocer el dominio con el que se exponen los módulos.

Estos son los parámetros que requiere el chart, se explica a continuación el valor que se debe informar, los parámetros que no se mencionan en esta lista es porque tienen valores por defecto que en la mayoría de los casos no es necesario modificarlos:

- **global.domain.protocol**: http o https, es el protocolo con el que se exponen los módulos (normalmente https)
- **global.domain.host:** Dominio con el que se exponen los módulos
- **global.domain.gateway_type:** k8s_gateway o istio, para indicar si el gateway que se ha configurado es de la api general de kubernetes (gateway.networking.k8s.io/v1) o de la api de istio (networking.istio.io/v1beta1)
- **global.docker.registry.host:** Host del repositorio docker donde se encuentran las imágenes de los módulos (por ejemplo para el caso del repositorio de onesait healthcare el valor sería: docksdtr.indra.es)
- **global.docker.registry.project:** project del repositorio docker donde se encuentran las imágenes de los módulos (por ejemplo para el caso del repositorio de onesait healthcare el valor sería: multhn_cmhn20_2)
- **global.docker.registry.secret:** Secret con las credenciales de acceso al docker registry. (Por ejemplo en nuestro caso el secret sería: oh-docker-creds)
- **bi.module.namespace:** Namespace en el que estamos realizando el despliegue
- **bi.keyclaok.realmname:** Nombre del realm de keycloak. Por defecto seria oh-base
- **bi.keyclaok.resource:** Resource del keycloak. Por defecto seria hnrole
- **bi.mail.ohbi_mail_configuration_host_name:** Host del servidor de correo con el que se enviarán los informes.
- **bi.mail.ohbi_mail_configuration_port:** Puerto del servidor de correo.
- **bi.mail.ohbi_mail_configuration_from:** Mail remitente desde el que se enviarán los informes.
- **bi.mail.ohbi_mail_configuration_password:** Contraseña del mail.
- **bi.database.type:** MySql o Derby
- **bi.database.mysql.ohbi_db_driver:** Driver de la base de datos MySql (Ejemplo: com.mysql.cj.jdbc.Driver)
- **bi.database.mysql.ohbi_db_host**: Host de la base de datos MySql
- **bi.database.mysql.ohbi_db_port:** Puerto de la base de datos MySql
- **bi.database.mysql.ohbi_db_schema:** Esquema de la base de datos MySql
- **bi.database.mysql.ohbi_db_parameters:** Parámetros de la base de datos MySql
- **bi.database.mysql.ohbi_db_dialect:** Dialecto de la base de datos MySql (Ejemplo: org.hibernate.dialect.MySQLDialect)
- **bi.database.mysql.ohbi_db_user:** Usuario de BD para el módulo OHBI (normalmente será el mismo que el esquema). En este caso sería: us_ohbi
- **bi.database.mysql.ohbi_db_pass:** Password de BD para el módulo OHBI.
- **bi.persistence.enable_derby**: true o false, si se va a usar la bbdd embebida Derby que trae por defecto OHBI

- **bi.persistence.storageclassname:** Nombre del STORAGECLASS que vayamos a usar para los VOLUMENES. Es un objeto que ya debe existir en el entorno con independencia de este despliegue.
- **bi.persistence.storage_derby:** espacio en disco para la bbdd embebida Derby usada para persistir los objetos creados con OHBI (si no usan bbdd externalizada)

Se instala el chart con el comando:

helm install

```
helm install -f values_bi-oh.yaml bi onesait-healthcare-helm-
repo/onesaithealthcare-bi-chart -n <namespace de instalación, normalmente oh-
modules>
```

El comando helm indicará deployed pero eso únicamente indica que ha sido capaz de crear todos los recursos en el cluster, hay que esperar a que los pods terminen de levantar, para ello chequeamos con el siguiente comando:

kubectl get pods

```
kubectl get pods -n <namespace de instalación>
```

Lo lanzaremos periódicamente hasta que veamos que todos los pods están en estado Running con los contenedores Ready 1/1, es decir, toda la lista de pods debería acabar de esta forma:

output kubectl get pods

```
NAME                                                              READY   STATUS
RESTARTS      AGE
<pod name>
1/1     Running   0             99m
...
```

En caso de que algún no se mostrara Running o presentara reinicios (columna Restarts mayor que 0) se pueden consultar los logs del pod con el comando:

kubectl logs

```
kubectl logs <pod name> -n <namespace de instalación>
```

Para ver los eventos del pod se ejecuta el comando:

kubectl describe pod

```
kubectl describe pod <pod name> -n <namespace de instalación>
```

7.4. Operaciones post-instalación

Una vez se ha instanciado el helm correspondiente y haya desplegado el módulo correctamente se deben lanzar los siguientes scripts que habilitará todo el mecanismo necesario (perfiles, roles, etc) para poder logarse en este módulo.

Si el módulo MDM está en una bbdd MySql:

- OHONT_Permisos_OHBI_mysql.sql **(FTP ENTREGAS)/oradata/Versiones_Producto_OH/OH_v4/OH_BI_v[VERSION]/bbdd/total/PermisosOHBI_MySql**
- OHAUT_Permisos_OHBI_mysql.sql **(FTP ENTREGAS)/oradata/Versiones_Producto_OH/OH_v4/OH_BI_v[VERSION]/bbdd/total/PermisosOHBI_MySql**

Si el módulo MDM está en una bbdd Oracle:

- OHONT_Permisos_OHBI_oracle.sql **(FTP ENTREGAS) /oradata/Versiones_Producto_OH/OH_v4/OH_BI_v[VERSION]/bbdd/total/PermisosOHBI_Oracle**
- OHAUT_Permisos_OHBI_oracle.sql **(FTP ENTREGAS) /oradata/Versiones_Producto_OH/OH_v4/OH_BI_v[VERSION]/bbdd/total/PermisosOHBI_Oracle**

A continuación desde el módulo OHAUT, habrá que asignar a los usuarios con los que se quiera logar en la aplicación, la profesión creada (sin especialidad) y uno de los roles creados (ROL_OHBI_ADMIN o ROL_OHBI_USER, según se quiera que el usuario entre como administrador o como usuario) y asociados a esa profesión. O asignar uno de los perfiles creados (ADMINISTRADOR_OHBI o USUARIO_OHBI, según se quiera que el usuario entre como administrador o como usuario) a un rol ya existente y que tenga ya asignado el usuario con el que se desea entrar.

NOTA: Para que aparezca el acceso al módulo en el escritorio, hay que pedir al módulo OH_DSK que creen ese acceso cuando el usuario logado tenga una de las dos funcionalidades(permisos) que hemos creado (OHBI_ADMIN y OHBI_USER).

8. Instalación Paquete Process Management

8.1. Módulos que incluye

- Process Manager (OH_BPM): Designer / Todo List / Smart de procesos
- Program Manager (OH_PRM)
- Forms Builder (OH_GEN)

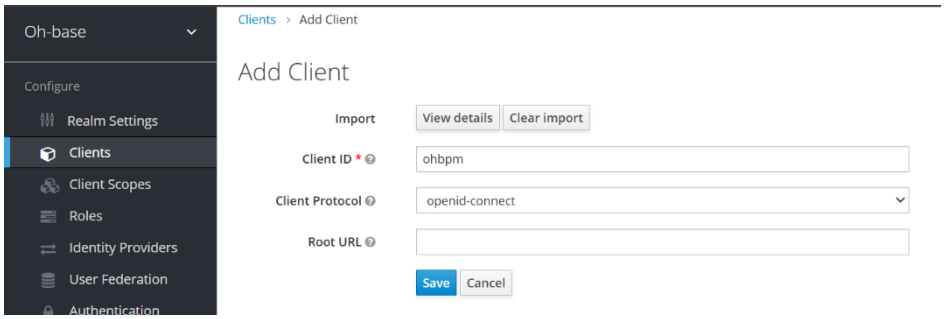
8.2. Prerrequisitos

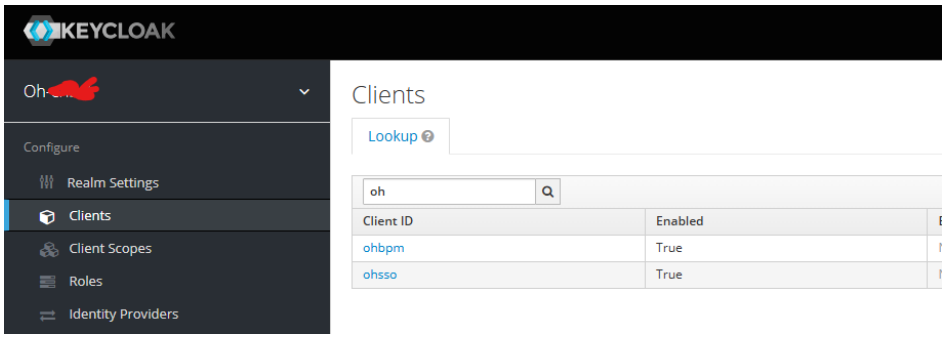
- Se deben cumplir los prerrequisitos generales y haber realizado los pasos indicados en la guía correspondiente.
- Deben instalarse previamente los paquetes MDM y DATA.
- Se debe crear un usuario en keycloack ohbpm para todo los accesos a BPM

8.2.1. Prerrequisito KEYCLOACK

Instalar cliente ohbpm o usar el ohsso para el uso de acciones offline (creacion de actividades dinamicas o temporizadas)

Para instalar el client **ohbpm** vamos a "Clients" → "Create" → y ahi importamos el json que se proporciona a continuacion.





ohbpm.json

```
{
    "clientId": "ohbpm",
    "surrogateAuthRequired": false,
    "enabled": true,
    "alwaysDisplayInConsole": false,
```

```
"clientAuthenticatorType": "client-secret",
"redirectUris": [],
"webOrigins": [],
"notBefore": 0,
"bearerOnly": false,
"consentRequired": false,
"standardFlowEnabled": false,
"implicitFlowEnabled": false,
"directAccessGrantsEnabled": true,
"serviceAccountsEnabled": true,
"publicClient": false,
"frontchannelLogout": false,
"protocol": "openid-connect",
"attributes": {
    "saml.force.post.binding": "false",
    "saml.multivalued.roles": "false",
    "frontchannel.logout.session.required": "false",
    "oauth2.device.authorization.grant.enabled": "false",
    "backchannel.logout.revoke.offline.tokens": "false",
    "saml.server.signature.keyinfo.ext": "false",
    "use.refresh.tokens": "true",
    "oidc.ciba.grant.enabled": "false",
    "backchannel.logout.session.required": "true",
    "client_credentials.use_refresh_token": "false",
    "require.pushed.authorization.requests": "false",
    "saml.client.signature": "false",
    "saml.allow.ecp.flow": "false",
    "id.token.as.detached.signature": "false",
    "saml.assertion.signature": "false",
    "client.secret.creation.time": "1700223643",
    "saml.encrypt": "false",
    "saml.server.signature": "false",
    "exclude.session.state.from.auth.response": "false",
    "saml.artifact.binding": "false",
    "saml_force_name_id_format": "false",
    "acr.loa.map": "{}",
    "tls.client.certificate.bound.access.tokens": "false",
    "saml.authnstatement": "false",
    "display.on.consent.screen": "false",
    "token.response.type.bearer.lower-case": "false",
    "saml.onetimeuse.condition": "false"
},
"authenticationFlowBindingOverrides": {},
"fullScopeAllowed": true,
"nodeReRegistrationTimeout": -1,
"protocolMappers": [
    {
        "name": "Client IP Address",
        "protocol": "openid-connect",
        "protocolMapper": "oidc-usersessionmodel-note-mapper",
        "consentRequired": false,
        "config": {
            "user.session.note": "clientAddress",
            "id.token.claim": "true",
            "access.token.claim": "true",
            "claim.name": "clientAddress",
            "jsonType.label": "String"
        }
    },
    {
        "name": "Client ID",
```

```
            "protocol": "openid-connect",
            "protocolMapper": "oidc-usersessionmodel-note-mapper",
            "consentRequired": false,
            "config": {
                "user.session.note": "clientId",
                "id.token.claim": "true",
                "access.token.claim": "true",
                "claim.name": "clientId",
                "jsonType.label": "String"
            }
        },
        {
            "name": "Client Host",
            "protocol": "openid-connect",
            "protocolMapper": "oidc-usersessionmodel-note-mapper",
            "consentRequired": false,
            "config": {
                "user.session.note": "clientHost",
                "id.token.claim": "true",
                "access.token.claim": "true",
                "claim.name": "clientHost",
                "jsonType.label": "String"
            }
        }
    ],
    "defaultClientScopes": [
        "web-origins",
        "acr",
        "profile",
        "roles",
        "email"
    ],
    "optionalClientScopes": [
        "address",
        "phone",
        "offline_access",
        "microprofile-jwt"
    ],
    "access": {
        "view": true,
        "configure": true,
        "manage": true
    }
}
```

8.2.2. Prerrequisitos de Base de Datos

Se debe disponer de un gestor de base de datos MySQL que tenga visibilidad desde los nodos worker del clúster de Kubernetes en el que se desplegarán los módulos.

A continuación, se indica como crear los usuarios y esquemas necesarios para los módulos de Process Manager.

MySQL con acceso de administrador

Disponiendo de usuario administrador se pueden crear los usuarios y esquemas para los módulos de Process Manager con los siguientes scripts:

- Process Manager

[FTP_ENTREGAS] /oradata/Versiones_Producto_OH/OH_v[MAJOR VERSION]/OH_BPM_v[VERSION]/bbdd/mysql/1-ohbpm-database.sql

[FTP_ENTREGAS] /oradata/Versiones_Producto_OH/OH_v[MAJOR VERSION]/OH_BPM_v[VERSION]/bbdd/mysql/2-ohbpm-user.sql

- Forms Builder

[FTP_ENTREGAS] /oradata/Versiones_Producto_OH/OH_v[MAJOR VERSION]/OH_BPM_v[VERSION]/bbdd/totales/mysql/HNFOR_inicial_000_bduser_mysql_ddl.sql

Sin acceso de administrador

Si no se dispone de acceso de administrador se tendrá que solicitar la creación de los siguientes usuarios, cada uno de ellos con un esquema sobre el que tenga permisos y con el mismo nombre del usuario:

- us_ohbpm (el usuario debe tener permiso "grant option" sobre su esquema para poder asignar permiso a otros usuarios a objetos de su esquema)
- us_hnfor(el usuario debe tener permiso "grant option" sobre su esquema para poder asignar permiso a otros usuarios a objetos de su esquema)

8.3. Procedimiento de despliegue de Capa de Persistencia

MySQL

- Process Manager

Con el usuario propio del módulo lanzar los siguientes scripts:

[FTP_ENTREGAS] /oradata/Versiones_Producto_OH/OH_v[MAJOR VERSION]/OH_BPM_v[VERSION]/bbdd/mysql/3-ohbpm-camunda-engine.sql

[FTP_ENTREGAS] /oradata/Versiones_Producto_OH/OH_v[MAJOR VERSION]/OH_BPM_v[VERSION]/bbdd/mysql/4-ohbpm-tables.sql

- Forms Builder

Con el usuario propio del módulo lanzar los siguientes scripts:

[FTP_ENTREGAS] /oradata/Versiones_Producto_OH/OH_v[MAJOR VERSION]/OH_GEN_v[VERSION]/bbdd/totales/mysql/HNFOR_inicial_001_tablas_mysql_ddl.sql

[FTP_ENTREGAS] /oradata/Versiones_Producto_OH/OH_v[MAJOR VERSION]/OH_GEN_v[VERSION]/bbdd/totales/mysql/HNFOR_inicial_002_ini_datos_mysql_dml.sql

[FTP_ENTREGAS] /oradata/Versiones_Producto_OH/OH_v[MAJOR VERSION]/OH_GEN_v[VERSION]/bbdd/totales/mysql/HNFOR_inicial_003_funciones_mysql_ddl.sql

8.4. Procedimiento de despliegue de Módulos

Para instalar los módulos del paquete Process Management se se deberá realizar la instalación del chart de Helm **onesaithealthcare-bpm-chart**. Las opciones de instalación del chart en función del tipo de entorno vienen descritas en el apartado inicial de "Requisitos de la instalación".

La instalación con helm client se realizaría con los siguientes comandos.

Obtenemos el values.yaml por defecto de BPM:

helm show values

```
helm show values onesait-healthcare-helm-repo/onesaithealthcare-bpm-chart > values_bpm.yaml
```

Se edita el fichero obtenido y se informan los valores de los campos que se quiera dar a la instalación.

Se deben tener disponibles los datos de la base de datos (host, puerto, usuarios y passwords) así como conocer el dominio con el que se exponen los módulos.

Estos son los parámetros que requiere el chart, se explica a continuación el valor que se debe informar, los parámetros que no se mencionan en esta lista es porque tienen valores por defecto que en la mayoría de los casos no es necesario modificarlos:

- **global.domain.protocol**: http o https, es el protocolo con el que se exponen los módulos (normalmente https)
- **global.domain.host:** Dominio con el que se exponen los módulos
- **global.domain.gateway_type:** k8s_gateway o istio, para indicar si el gateway que se ha configurado es de la api general de kubernetes (gateway.networking.k8s.io/v1) o de la api de istio (networking.istio.io/v1beta1)
- **global.docker.registry.host:** Host del repositorio docker donde se encuentran las imágenes de los módulos (por ejemplo para el caso del repositorio de onesait healthcare el valor sería: docksdtr.indra.es)
- **global.docker.registry.project:** project del repositorio docker donde se encuentran las imágenes de los módulos (por ejemplo para el caso del repositorio de onesait healthcare el valor sería: multhn_cmhn20_2)
- **global.docker.registry.secret:** Secret con las credenciales de acceso al docker registry. (Por ejemplo en nuestro caso el secret sería: oh-docker-creds)
- **global.database.type:** mysql
- **dependencies.ohbpm/ohprm/ohgen**: Por defecto se dejan en true.

- **onesaithealthcare_ohbpm_chart.ohbpm.database.url:** Cadena de conexión a la BD. (para BD MySQL sería: jdbc:mysql://database_host:port/db_schema).
- **onesaithealthcare_ohbpm_chart.ohbpm.database.user:** Usuario de BD para el módulo OHBPM (normalmente será el mismo que el esquema). En este caso sería: us_ohbpm
- **onesaithealthcare_ohbpm_chart.ohbpm.database.pass:** Password de BD para el módulo OHBPM
- **onesaithealthcare_ohgen_chart.ohgen.database.url:** Cadena de conexión a la BD. (para BD MySQL sería: jdbc:mysql://database_host:port/db_schema).
- **onesaithealthcare_ohgen_chart.ohgen.database.user:** Usuario de BD para el módulo OHGEN (normalmente será el mismo que el esquema). En este caso sería: us_hnfor
- **onesaithealthcare_ohgen_chart.ohgen.database.pass:** Password de BD para el módulo OHGEN.

Se instala el chart con el comando:

helm install

```
helm install -f values_bpm.yaml bpm onesait-healthcare-helm-
repo/onesaithealthcare-bpm-chart -n <namespace de instalación, normalmente oh-
modules>
```

El comando helm indicará deployed pero eso únicamente indica que ha sido capaz de crear todos los recursos en el cluster, hay que esperar a que los pods terminen de levantar, para ello chequeamos con el siguiente comando:

kubectl get pods

```
kubectl get pods -n <namespace de instalación>
```

Lo lanzaremos periódicamente hasta que veamos que todos los pods están en estado Running con los contenedores Ready 1/1, es decir, toda la lista de pods debería acabar de esta forma:

output kubectl get pods

```
NAME                                                    READY   STATUS
RESTARTS      AGE
<pod name>
1/1     Running   0             99m
...
```

En caso de que algún no se mostrara Running o presentara reinicios (columna Restarts mayor que 0) se pueden consultar los logs del pod con el comando:

kubectl logs

```
kubectl logs <pod name> -n <namespace de instalación>
```

Para ver los eventos del pod se ejecuta el comando:

kubectl describe pod

```
kubectl describe pod <pod name> -n <namespace de instalación>
```

8.5. Operaciones post-instalación

8.5.1. Process Manager (OH_BPM)

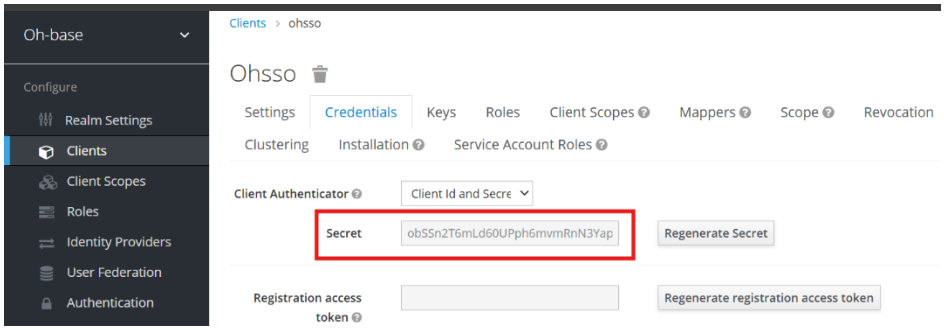
Modificar el configmap **ohbpm (ohbpm-back-cfg)**, cambiar el login y clave del usuario offline creado antes de la instalación en el keycloack. **SE DEBE PEGAR CODIFICADO EN BASE64 (USAR LA PÁGINA** https://www.base64encode.org/ **)**. Se haría con los comandos:

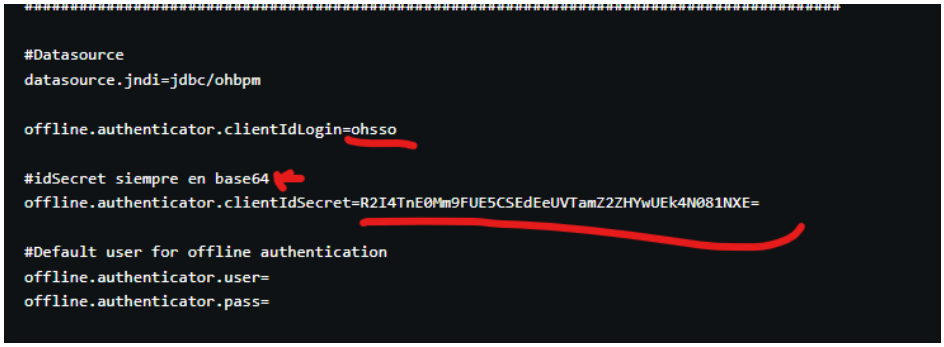
kubectl create secret

```
kubectl get configmap ohbpm-back-cfg -n oh-modules -o yaml > ohbpm-back-cfg.yaml

-- editar el fichero para aplicar los cambios indicados

kubectl apply -f ohbpm-back-cfg.yaml -n oh-modules
```





Configuración en módulo Settings

Se adjunta en la carpeta **[FTP_ENTREGAS] /oradata/Versiones_Producto_OH/OH_v[MAJOR VERSION]/OH_BPM_v[VERSION]/src/totales** los siguientes archivos de configuración:

- **OHCON_OHBPM_[X].csv** que deberá importarse en la sección de configuración.
- **OHCON_OHBPM_WL_[X].csv que** deberá importarse en la sección de listas de trabajo.

Configuración en módulo Ontology

Se adjunta en la carpeta **/oradata/Versiones_Producto_OH/OH_v[MAJOR VERSION]/OH_BPM_v[VERSION]/src/totales** los siguientes archivos de catálogos (si existiera algún fichero json más, importarlo también, se tratan de incrementales)

- **OHONT_BPM_ActivityFilters.json** filtros para las actividades de la smart
- **OHONT_BPM_Exclusion_Reason.json** motivos de exclusión
- **OHONT_BPM_Instantiation_Reason.json** motivos de instanciación
- **OHONT_BPM_KPIs.json** nombre de los kpi a controlar
- **OHONT_BPM_Pause_Reason.json** motivos para pausar un proceso
- **OHONT_BPM_Process.json** nombre de los procesos
- **OHONT_BPM_Resume_Reason.json** motivos de reanudación
- **OHONT_BPM_Status.json** estados de negocio
- **OHONT_CM_BPMProcessExclusion.json** conceptMap proceso y exclusión
- **OHONT_CM_BPMProcessInstantiationReason.json** conceptMap proceso y motivo de instanciacion
- **OHONT_CM_BPMProcessKpis.json** conceptMap proceso y kpis de control
- **OHONT_CM_BPMProcessStatus.json** conceptMap proceso y estados de negocio

Configuración de permisos en módulo Users & Resources

Verificar que existen los siguientes permisos desde el aplicativo para el módulo del Process Manager. Si no es así, darlos de alta desde el módulo Users & Resources, en la pestaña Funcionalidades del menú Permisos.

Los permisos deben crearse vinculados al módulo OHBPM, con el ámbito **Sistema** y la categoría **Contexto profesional**.

MÓDULO	CÓDIGO	DESCRIPCIÓN
OHBPM	OHBPM_ADMIN	ADMINISTRACION TOTAL
OHBPM	OHBPM_PROCESS_LIST	Acceso al listado de procesos. (Estadísticas)
OHBPM	OHBPM_PROCESS_TODO_LIST	Acceso al listado de profesional (TO-DO List) Seguimiento de actividades
OHBPM	OHBPM_PATIENT_ACCESS	Acceso al listado de procesos de un paciente. (SMART)
OHBPM	OHBPM_CAREPLAN_W	Escritura sobre instancias de proceso
OHBPM	OHBPM_CAREPLAN_R	Lectura sobre instancias de proceso
OHBPM	OHBPM_DEFINITION_R	Lectura sobre definiciones de proceso
OHBPM	OHBPM_DEFINITION_W	Escritura sobre definiciones de proceso

OHBPM	OHBPM_TASK_W	Escritura sobre etapas y actividades
OHBPM	OHBPM_TASK_R	Lectura sobre etapas y actividades
OHBPM	OHBPM_PROCESS_DESIGNER	Acceso diseñador de procesos y subprocesos.
OHBPM	OHBPM_PROCESS_DESIGNER_R	Acceso diseñador de procesos y subprocesos. (**SOLO LECTURA**)
OHBPM	OHBPM_PROCESS_DESIGNER_PUBLISH	Permiso de publicación de Procesos
OHBPM	OHBPM_PROCESS_DESIGNER_ADMIN	Permiso de actualización del modelo **SIN VERSIONAR - Esta acción debe realizarse con cuidado**

8.5.2. Program Manager (OH_PRM)

Configuración en módulo Settings

Se adjunta en la carpeta **[FTP_ENTREGAS] /oradata/Versiones_Producto_OH/OH_v[MAJOR VERSION]/OH_PRM_v[VERSION]/conf** los siguientes archivos de configuración:

- **OHCON_OHPRM_Propiedades.csv** que deberá importarse en la sección de configuración.
- **OHCON_OHPRM_ListasDeTrabajo.csv que** deberá importarse en la sección de listas de trabajo.

8.5.3. Forms Builder (OH_GEN)

Configuración en el módulo Settings

Se adjunta en la carpeta **[FTP_ENTREGAS] /oradata/Versiones_Producto_OH/OH_v[MAJOR VERSION]/OH_GEN_v[VERSION]/src/totales** los siguientes archivos de configuración:

- **OHCON_HNFORM_[X].csv** que deberá importarse en la sección de configuración.
- **OHCON_HNFORM_WL_[X].csv que** deberá importantes en la sección de listas de trabajo.

Configuración en el módulo Ontology

Se adjunta en la carpeta **/oradata/Versiones_Producto_OH/OH_v[MAJOR VERSION]/OH_GEN_v[VERSION]/src/totales** los siguientes archivos de catálogos (si existiera algún fichero json más, importarlo también, se tratan de incrementales)

- **OHONT_tipoFormulario.json** tipos de formularios permitidos (escalas, hojas, información adicional…)
- **OHONT_FORM_OWNER.json** propietarios que se pueden asignar a los formularios
- **OHONT_idiomas.json** idiomas para las traducciones
- **OHONT_sexo.json** sexo de los pacientes
- **OHONT_tipo_profesional.json** tipo de profesionales que se pueden asignar a los campos
- **OHONT_typeRegex.json** listado de tipo de campos a validar
- **OHONT_typeFhirPath.json** listado de tipos de extracción estándar

Configuración de permisos en módulo Users & Resources.

Verificar que existen los siguientes permisos desde el aplicativo para el módulo del Forms Builder. Si no es así, darlos de alta desde el módulo Users & Resources, en la pestaña Funcionalidades del menú Permisos.

CÓDIGO	DESCRIPCIÓN	AMBITO	CATEGORÍA
IMPRIMIR_INF	Imprimir información	SISTEMA	Formularios
HNHDW_FORM	Visualización de Smart de Cuestionarios	SISTEMA	Formularios
HNFORM_MANAG_READ	Lectura administración formularios.	SISTEMA	Formularios
HNHDW_FORM_READ	Permiso consulta de cuestionarios	SISTEMA	Formularios
HNFORM_MANAG_DEACTIVATE	Desactivar formulario	SISTEMA	Formularios
HNHDW_FORM_WRITE	Permiso de edición de cuestionarios	SISTEMA	Formularios
HNFORM_MANAG_PUBLISH	Publicar formulario	SISTEMA	Formularios
HNFORM_REPORT_WRITE	Gestionar informes.	SISTEMA	Uso de Informes
HNFORM_MANAG_TRANSLATE	Permite traducir formularios.	SISTEMA	Formularios
HNFORM_REPORT_READ	Consultar informes	SISTEMA	Uso de Informes
HNFORM_MANAG_WRITE	Escritura administración formularios.	SISTEMA	Formularios

ANEXO: Configuración para varios entornos en el mismo clúster de kubernetes

Esta guía describe como preparar el cluster de kubernetes para el despliegue de varias instancias de Onesait Healthcare bajo la misma infraestructura. En escenarios donde se quiera instalar diferentes entornos de pruebas (desarrollo / test / formacion / etc) y no se dispongan de suficientes recursos, podría reutilizarse el mismo clúster de kubernetes haciendo una separación lógica de los entornos para tener diferentes instalaciones independientes de Onesait Healthcare

DISCLAIMER: Esta guía se debe aplicar si ya se han ejecutado todos los pasos indicados en el epígrafe "2. Requisitos de instalación".

Esta configuración NO ESTÁ RECOMENDADA para entornos PRODUCTIVOS.

Preparación del nuevo namespace

En primer lugar, crearemos el nuevo namespace para la instalación de los módulos de la solución. En este ejemplo usaremos el nombre oh-develop. Se puede crear mediante el comando:

```
kubectl create namespace oh-develop
```

Para poder desplegar en este namespace las imágenes docker de los módulos de Onesait Healthcare, se deberá crear un secret con las creden ciales para el docker registry, el nombre de este secret debe ser: oh-docker-creds

```
kubectl create secret docker-registry oh-docker-creds --docker-server= --
dockerusername= --docker-password= -n oh-develop
```

Opción 1: Configuración del Gateway HTTP en el nuevo namespace

Si la configuración de HTTPS para el dominio que se expone se realiza en un balanceador externo el Gateway se creará como HTTP y el balanceador es el que gestiona las conexiones HTTPS y expone el certificado y vuelca las peticiones a los nodos worker del cluster. Una vez traefik se encuentra instalado y configurado como Gateway Controller creamos el Gateway importando el siguiente yaml (el Gateway se debe llamar "oh-modules-gateway" ya que los charts de helm hacen referencia a dicho nombre tal cual):

```
apiVersion: gateway.networking.k8s.io/v1
kind: Gateway
metadata:
    name: oh-modules-gateway
    namespace: <namespace>
spec:
    gatewayClassName: traefik
```

```
    listeners:
        - allowedRoutes:
            namespaces:
            from: Same
        name: <namespace>
        hostname: <host-expuesto, debe coincidir con el expuesto en el
balanceador>
        port: 8000
        protocol: HTTP
```

Por ejemplo, para el namespace "oh-develop" quedaría configurado de la siguiente forma:

```
apiVersion: gateway.networking.k8s.io/v1
kind: Gateway
metadata:
    name: oh-modules-gateway
    namespace: oh-develop
spec:
    gatewayClassName: traefik
    listeners:
        - allowedRoutes:
            namespaces:
            from: Same
        name: oh-develop
        hostname: oh-develop.healthcare.onesait.com
        port: 8000
        protocol: HTTP
```

Creamos el Gateway:

```
kubectl apply -f oh-modules-gateway.yaml
```

Comprobamos que se ha creado el Gateway:

```
kubectl get gateways -n oh-develop
```

Opción 2: Configuración del Gateway HTTPS en el nuevo namespace

En caso de que el balanceador no confgure el HTTPS y el certificado entonces se tendrá que crear el Gateway HTTPS. Para ello debemos disponer del certificado del dominio, tanto la parte pública como la privada. Teniendo los ficheros crt (parte pública del certificado en formato PEM incluyendo el raíz y CA intermedios si los tuviera) y key (parte privada del certificado) se crearía un secret con dicho certificado con un comando de la forma:

```
kubectl create secret tls <nombre-cert> --cert=<nombre-cert>.crt --
key=<nombre-cert>.key -n oh-develop
```

Una vez creado el secret creamos el Gateway importando el siguiente yaml (el Gateway se debe llamar "oh-modules-gateway" ya que los charts de helm hacen referencia a dicho nombre tal cual):

```
apiVersion: gateway.networking.k8s.io/v1
kind: Gateway
metadata:
```

```
    name: oh-modules-gateway
    namespace: oh-develop
spec:
    gatewayClassName: traefik
    listeners:
        - allowedRoutes:
            namespaces:
            from: Same
        name: oh-develop
        hostname: <host-expuesto, debe coincidir con el expuesto en el
balanceador>
        port: 8443
        protocol: HTTPS
        tls:
          mode: Terminate
          certificateRefs:
            - name: <nombre-cert>
              namespace: oh-develop
```

Creamos el Gateway:

```
kubectl apply -f oh-modules-gateway.yaml
```

Comprobamos que se ha creado el Gateway:

```
kubectl get gateways -n oh-develop
```

Configuración DNS para el nuevo entorno

La configuración actual tras la ejecución de esta guía es que disponemos del namespace original "oh-modules" y un nuevo namespace "oh-develop" cada uno con su propio Gateway HTTP.

Existe una única IP pública asociada a un balanceador (configurado en la guía 00.0 Requisitos de instalación).

Es necesario dar de alta en el servidor de DNS una entrada para el nuevo nombre de dominio asociado al nuevo Gateway (atributo "hostname" del recurso "Gateway" creado en esta guía) y asociarla a la IP actual.

De esta forma nuestro Gateway Controller (traefik) enrutará las peticiones al Gateway adecuado según el "hostname" y se podrá tener varios entornos desplegados en diferentes namespaces bajo el mismo clúster de kubernetes.