Documento de Instalación del Nodo Autonómico

Minsait

Diciembre 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
2.0	Minsait	20/11/2025
3.0	Minsait	16/12/2025
4.0	Minsait	19/12/2025
4.1	Minsait	23/12/2025
4.2	Minsait	16/01/2026
4.3	Minsait	21/01/2026
4.4	Minsait	20/02/2026
4.5	Minsait	27/02/2026
4.6	Minsait: Cambios en el apartado 5.4.2 y el Anexo1	27/03/2026
4.7	Minsait: Eliminación punto 7.01, modificación de la versión del punto 7.02 y modificación del Anexo I	30/06/2026
4.8	Minsait: - Modificación punto 4.5.2: se ha añadido que tienen que comprobar que está creado el catálogo “designation-use.json”. - Modificación punto 3.2.1: se ha especificado que el listado únicamente es para identificar el usuario de cada uno de los módulos. - Modificación punto 5.4.1: se ha puesto en los dos sitios que el nombre del fichero que se tiene que importar es “OHCON_OHIEN_20251125.csv”. - Modificación punto 3.5.5: se ha explicado de manera más concreta como se tiene que realizar posteriormente la configuración eliminando un enlace erróneo.	15/05/2026

	- Anexo I: se cambia la tabla por una ruta a las versiones de las imágenes. - Anexo II: se añade este anexo con una tabla con las urls de los módulos a los que se accede de manera directa.	
4.9	Minsait: - Se modifican los puntos “2.3.3”, “3.5” y “4.4”, cambiando las imágenes de las versiones. - Se modifica la hoja Excel de “Versiones.xlsx” que se menciona en el Anexo I	03/09/206

Contenido

1. Objetivo 6
2. Requisitos de la instalación 6
2.1. Infraestructura para el despliegue de Onesait Healthcare 6
2.2. Preconfiguración del clúster de Kubernetes para el despliegue de Onesait Healthcare 6
2.2.1. Creación namespace dentro del clúster 6
2.2.2. Creación de credenciales para Docker Registry 7
2.2.3. Configuración de Gateway Controller 7
2.3.1. Instalación Helm desde Rancher 18
2.3.2. Instalación Helm desde Openshift 19
2.3.3. Instalación Helm mediante helm client 21
3. Instalación Paquete MDM 23
3.1. Módulos que incluye 23
3.2. Prerrequisitos 23
3.2.1. Prerrequisitos de Base de Datos 23
3.3. Procedimiento de despliegue de Capa de Persistencia 25
3.4. Paso previo a despliegue de módulos – certificado del dominio 25
3.5. Procedimiento de despliegue de Módulos 26
3.5. Operaciones post-instalación 29
3.5.1. Scripts POST 29
3.5.2. OHSSO 30
3.5.3. OHCONF 31
3.5.4. OHONT 32
3.5.5. OHAUT 32
3.5.6. CONFIGURACIÓN DEFINITIVA OHSSO 33
4. Instalación Paquete DATA 36
4.1. Módulos que incluye 36
- 4.2. Prerrequisitos 36
4.2.1. Prerrequisitos de Base de Datos 36
4.3. Procedimiento de despliegue de Capa de Persistencia 37
4.3.1. MySQL 37
4.3.2. Oracle 38
4.4. Procedimiento de despliegue de Módulos 38
4.5. Operaciones post-instalación 40

4.5.1. Visor Historia Clínica (OH_HDA) 40
4.5.2. Consent Manager (OH_CSM) 41
5. Instalación Paquete Integration & Interoperability 43
5.1. Módulos que incluye 43
5.2. Prerrequisitos 44
5.3. Pasos previos a la instalación 44
5.3.1. StorageClass 44
5.3.2. Ejemplo para AWS EKS 44
5.4. Procedimiento de despliegue de Módulos 45
5.4.1. Instalación de OHIEN 45
5.4.2. Instalación de Camel K con Helm 49
6. Instalación Paquete Monitorización 51
6.1. Módulos que incluye 51
6.2. Prerrequisitos 51
6.3. Procedimiento de despliegue de Capa de Persistencia 52
6.4. Procedimiento de despliegue de Módulos 53
6.5. Operaciones post-instalación 53
7. Instalación Paquete Analytics 54
7.01. Instalación Framework BI 54
7.1. Módulos que incluye 54
7.2. Prerrequisitos 54
7.2.1. Prerrequisitos de Base de Datos 54
7.2.2. Prerrequisitos PV 54
7.3. Procedimiento de despliegue de Módulos 55
7.4. Operaciones post-instalación 57
7.02. Instalación DTC 59
7.1. Módulos que incluye 59
7.2. Prerrequisitos 59
7.3. Procedimiento de despliegue de Capa de Persistencia 59
7.4. Procedimiento de despliegue de Módulos 59
7.4. Operaciones post-instalación 62
8. Instalación Paquete Process Management 65
8.1. Módulos que incluye 65
8.2. Prerrequisitos 65
8.2.1. Prerrequisito KEYCLOACK 65

8.2.2. Prerrequisitos de Base de Datos 67
8.3. Procedimiento de despliegue de Capa de Persistencia 69
8.3.1. MySQL 69
8.3.2. Oracle 69
8.4. Procedimiento de despliegue de Módulos 70
8.5. Operaciones post-instalación 72
8.5.1. Process Manager (OH_BPM) 72
8.5.2. Program Manager (OH_PRM) 75
8.5.3. Forms Builder (OH_GEN) 76
9. Guías de configuración adicionales 77
9.1. Configuración de persistencia sobre NFS 77
9.2. Configuración para varios entornos en el mismo clúster de kubernetes 81
Anexo I 0
Anexo II 0

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 ÚNICAS. 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 permitirá 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
```

Una vez creado el secret, se puede verificar que se ha creado correctamente ejecutando el siguiente comando:

```
kubectl get secret oh-docker-creds -n oh-modules
```

Si el secret se ha creado correctamente, el comando devolverá una entrada con el nombre "oh-docker-creds". En caso contrario, se deberá revisar el comando de creación y las credenciales proporcionadas.

2.2.3. Configuración de Gateway Controller

Para exponer el Sistema al exterior del Clúster de Kubernetes, es necesario instalar y configurar una implementación de la Gateway API de Kubernetes y habilitar el Gateway Controller. Esta guía usa la implementación de traefik, por ser compatible con todos los tipos de Clúster[1] propuestos y tener carácter de distribución libre.

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 Traefik está instalado, el comando devolverá una lista de pods en el namespace traefik, con un resultado similar al siguiente:

```
NAME                          READY   STATUS    RESTARTS   AGE
traefik-xxxxxxxxxx-xxxxx      1/1     Running   0          2m
```

[1] Rancher, Openshift y AWS EKS

Si Traefik no está instalado o el namespace no existe, el comando devolverá un mensaje indicando que no se han encontrado recursos o que el namespace no existe.

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
```

Si la instalación se ha realizado correctamente, el comando devolverá una lista de pods en el namespace traefik con el estado Running, con un resultado similar al siguiente:

```
NAME                          READY    STATUS     RESTARTS    AGE

traefik-xxxxxxxxxx-xxxxx      1/1      Running    0           2m
```

Si el pod no se encuentra en estado Running, será necesario revisar el estado del despliegue para identificar posibles incidencias durante la instalación.

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.

En caso de que existan otras implementaciones de gateway controller instaladas en el clúster, será necesario tenerlas en cuenta en la configuración del entorno. La coexistencia de múltiples Gateway Controllers dependerá de la arquitectura y configuración específica del clúster Kubernetes.

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
```

Guardar el contenido anterior en un fichero yaml y crear el ClusterRole en el clúster ejecutando el siguiente comando:

```
kubectl apply -f traefik-gateway-role.yaml
```

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>
```

Guardar el contenido anterior en un fichero yaml y crear el ClusterRoleBinding en el clúster ejecutando el siguiente comando:

```
kubectl apply -f traefik-gateway-role-binding.yaml
```

Finalmente habilitamos el Gateway Controller accediendo al **Deployment** de **traefik** (ubicado en el namespace de instalación de traefik, que suele ser **traefik** o **kube-system**).

El acceso al Deployment de Traefik puede realizarse mediante las herramientas habituales de administración del clúster Kubernetes, ya sea a través de una interfaz gráfica o mediante línea de comandos.

Una vez localizado el Deployment de Traefik, en la sección de argumentos del contenedor se deberá añadir el siguiente parámetro:

```
--providers.kubernetesgateway
```



Se crea el GatewayClass:

traefik

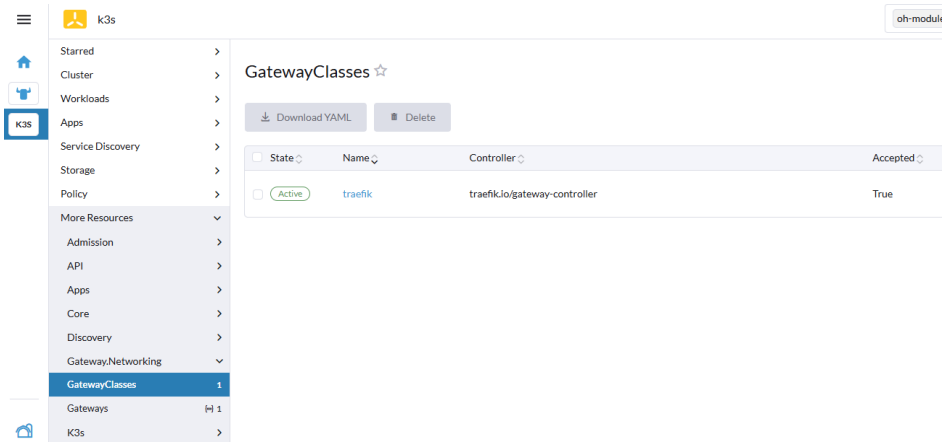
```
apiVersion: gateway.networking.k8s.io/v1
kind: GatewayClass
metadata:
  name: traefik
  namespace: <namespace de instalación de traefik, suele ser traefik o kube-system>
spec:
  controllerName: traefik.io/gateway-controller
```

Guardar el contenido anterior en un fichero yaml y crear el recurso GatewayClass en el clúster ejecutando el siguiente comando:

```
kubectl apply -f traefik-gatewayclass.yaml
```

Tras recrear el pod de traefik 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 el dominio se expone mediante HTTPS en un balanceador externo, el Gateway dentro del clúster debe configurarse como HTTP. En este caso, el balanceador se encarga de gestionar las conexiones seguras (TLS), presentar el certificado y redirigir las peticiones hacia 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 de forma literal):

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
```

Para importarlo guardar el contenido anterior en un fichero yaml y crear el recurso Gateway en el clúster ejecutando el siguiente comando:

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

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 ni el certificado, 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 de manera literal):

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.

Para comprobar si se dispone de una Elastic IP, se debe acceder a la consola de AWS y navegar a EC2 → Red y Seguridad → Direcciones IP elásticas. En este apartado se mostrará el listado de Elastic IPs asignadas a la cuenta. Se deberá seleccionar la IP que se tiene registrada y anotar el valor de ID de asignación (que tendrá el prefijo eipallloc-).

En caso de no disponer de una Elastic IP, será necesario solicitarla previamente desde la consola de AWS antes de continuar con este apartado.

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
```

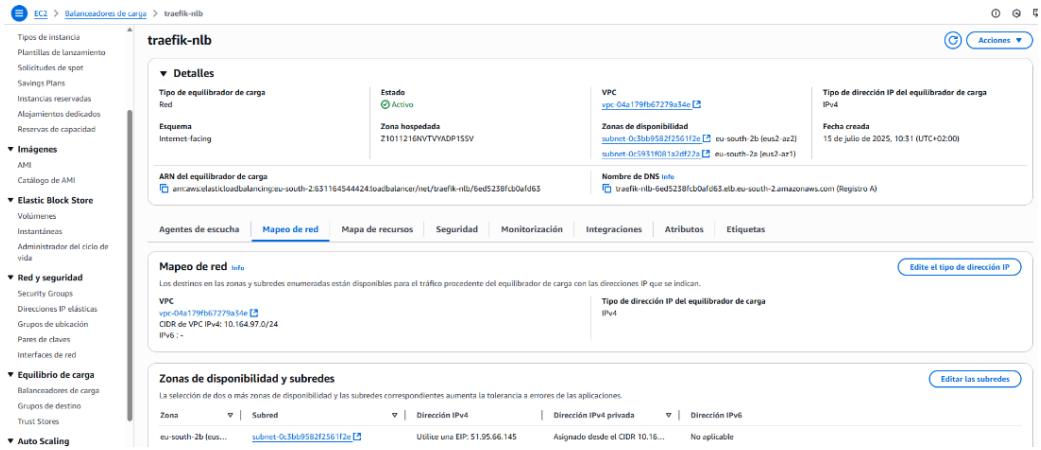
Se crea el servicio del balanceador ejecutando el siguiente comando:

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

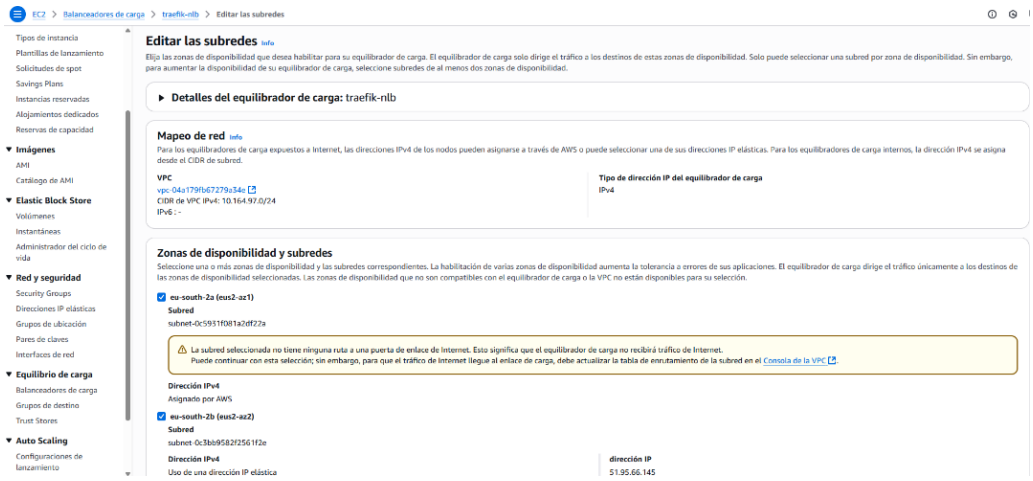
En la consola de AWS → EC2 → Equilibrio de carga → Balanceadores de carga, deberá aparecer un balanceador correspondiente al servicio que hemos creado, inicialmente en estado "Aprovisionando" y pasado un tiempo, que suele ser de varios minutos dependiendo de la carga del servicio, debería cambiar a "Activo".



Accedemos al detalle del balanceador pulsando sobre su nombre desde el listado de balanceadores y, una vez dentro, accedemos a la pestaña Mapeo de Red (se mostrará inicialmente sólo una zona) y pulsamos en Editar las Subredes.

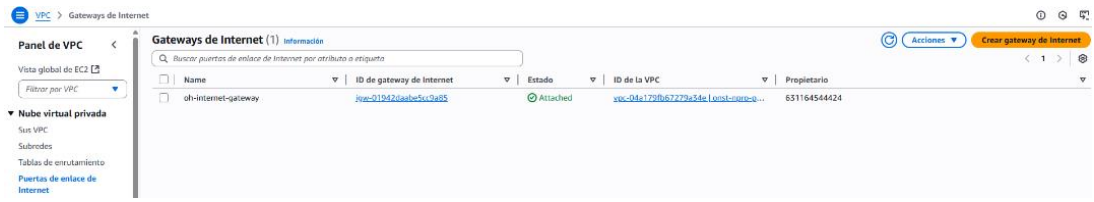


Y añadimos el resto de las zonas:



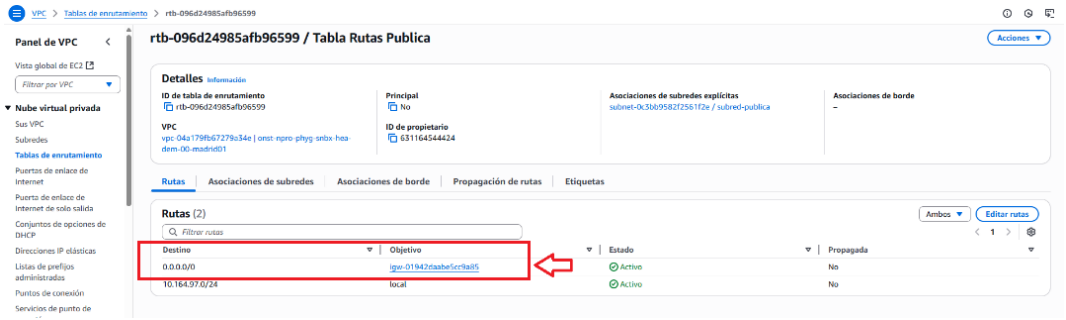
Será necesario crear un gateway de internet, para ello accedemos a la consola AWS → VPC → Nube virtual privada → Puertas de enlace de internet.

En caso de no existir una previamente creada, deberá pulsarse en “Crear puerta de enlace de internet”, crearla y asociarla a la VPC correspondiente.



Para acceder a la tabla de rutas pública, desde la consola de AWS se debe navegar a VPC → Tablas de enrutamiento, seleccionar la tabla asociada a la subred pública y acceder a la pestaña Rutas.

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 yaml[2] asociadas al Service dependerán de los criterios de creación del Load Balancer que decida cada Nodo Autonómico, teniendo que ser ajustarlos al comportamiento deseado.

El uso de otros mecanismos o proveedores de exposición queda bajo la responsabilidad de cada Comunidad, tanto su instalación como configuración, los Helm Charts propuestos también permiten la instalación sobre Gateways y VirtualServices de Istio, como los provistos por Openshift Service Mesh 2.x, no se recomienda su uso. No obstante, sí se quisiera seguir esta vía deberemos asignar el valor “true” a los parámetros marcados como “istio”. Recordar que el hecho de configurar este parámetro no exime de realizar una correcta configuración de este mecanismo de exposición, fuera del alcance del presente manual. Los Helm Charts así configurados se limitarán a crear los VirtualService necesarios, del mismo modo que se crean los HttpRoutes en el caso de haber seleccionado la opción de exposición recomendada.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 a 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`

[2] 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.

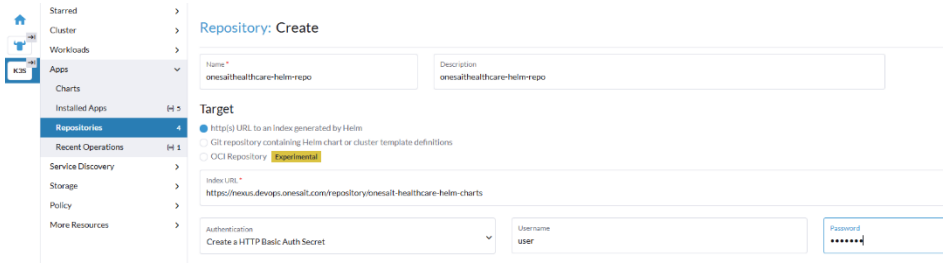
Para acceder a este repositorio son necesarias credenciales que serán proporcionadas por el equipo de Onesait Healthcare. En caso de no disponer de ellas pueden solicitarlas al correo: oficinatecnica.unicas@gencat.cat.

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.

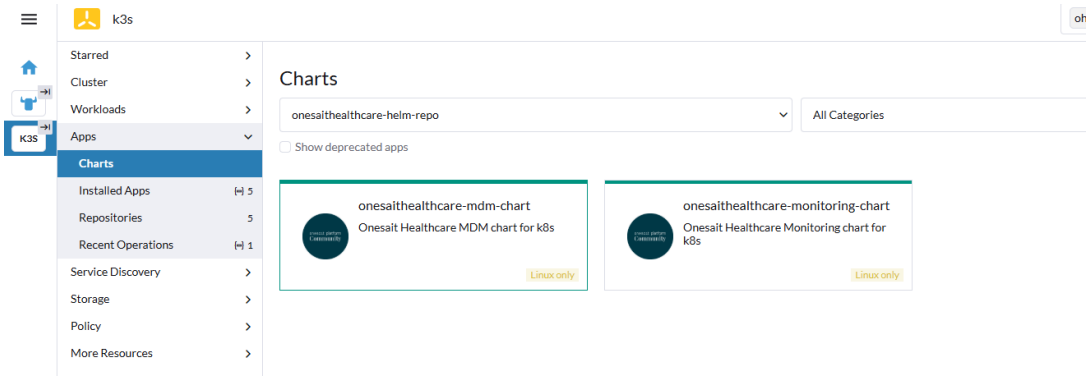
Para ello se accede a la consola de administración de Rancher y al clúster de Kubernetes en el que se vaya a realizar la instalación accediendo al menú Apps → Repositories y se pulsa en Create.

Se informa el nombre y la descripción con que se desee mostrar el repositorio, el índex URL indicada anteriormente (url del Repositorio Charts Onesait Healthcare) y las credenciales de acceso proporcionadas.

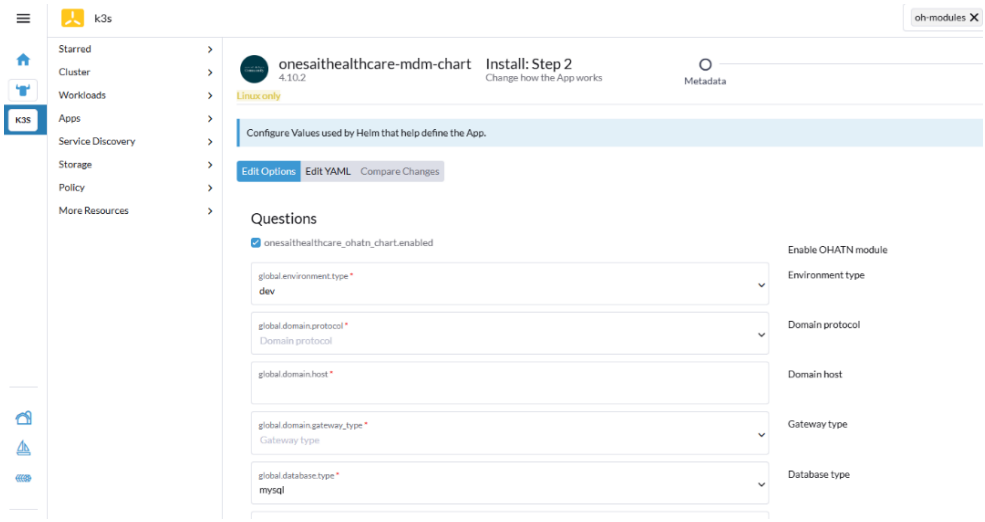


Tras crearlo se mostrará el listado de repositorios y deberá aparecer disponible en el menú Apps → Charts.

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:



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
```

La creación de secrets en Kubernetes se describe previamente en el apartado 2.2.2. Creación de credenciales para Docker Registry.

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

onesait-healthcare-helm-repo

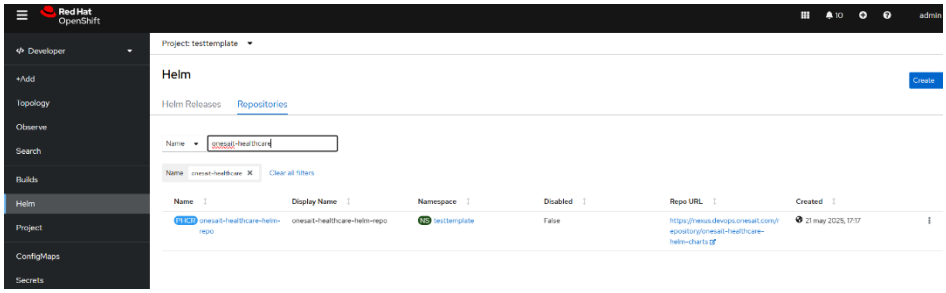
```
apiVersion: helm.openshift.io/v1beta1
kind: ProjectHelmChartRepository
metadata:
  name: onesait-healthcare-helm-repo
  namespace: <namespace>
spec:
  connectionConfig:
    basicAuthConfig:
      name: onesait-healthcare-helm-repo-creds
```

```
    url:     'https://nexus.devops.onesait.com/repository/onesait-healthcare-helm-
charts'
  description: onesait-healthcare-helm-repo
  name: onesait-healthcare-helm-repo
```

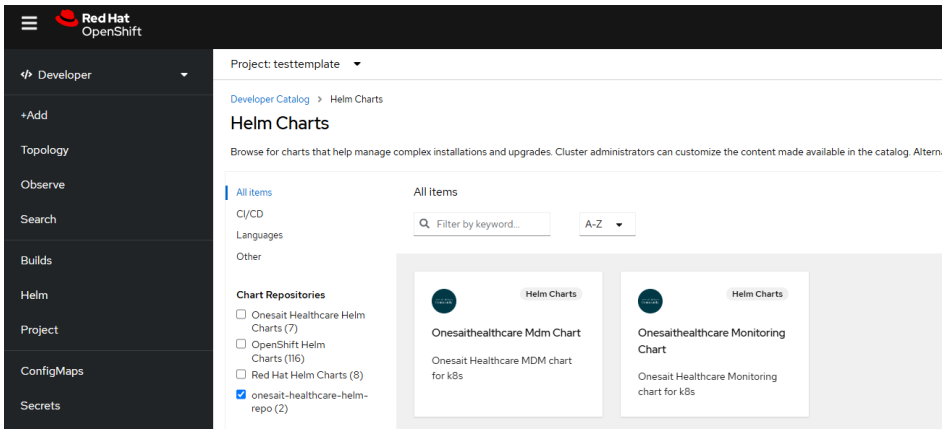
Guardar el contenido anterior en un fichero yaml y crear el repositorio en el clúster ejecutando el siguiente comando:

```
kubectl apply -f onesait-healthcare-helm-repo.yaml
```

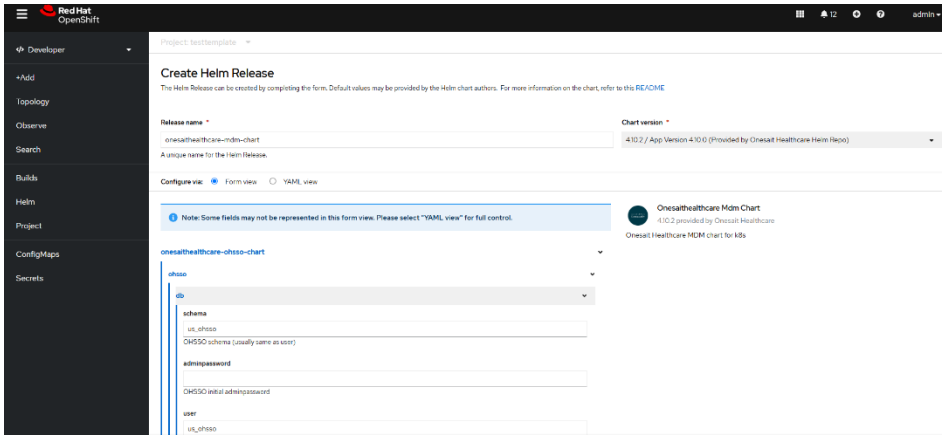
Si se accede a Developer → Helm → Repositories debe verse el nuevo repositorio:



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):

Para ello desde una máquina con acceso al clúster Kubernetes y con los clientes de línea de comandos kubectl y helm correctamente configurados, se ejecutarán los siguientes comandos 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, de acuerdo con el README del chart y las necesidades del entorno.

NOTA: Se tiene que instalar la versión de MDM que viene en el anexo I. Para ello tendremos que modificar el fichero "values_mdm.yaml" poniendo las versiones que se muestran en las columnas Images Values:

Paquete/Charts	Versión Paquete/Charts	Módulo	Images values 1	Images values 2	Images values 3	Images values 4	Otras variables
MDM	4.1.0	OH_SSO	image_tag_oracle: 4.13.0.oracle image_tag_mysql: 4.13.0.mysql				
		OH_CON	image_tag_front: 4.11.0	image_tag_front_ng15: 4.11.0	image_tag_front_webc: 4.11.0	image_tag_back: 4.11.0	
		OH_AUT	image_tag_front: 4.13.4	image_tag_back: 4.13.4	image_tag_front_wcint: 4.7.0		
		OH_ONT	image_tag_front: 4.11.2	image_tag_back: 4.11.3	image_tag_ng15_front: 4.11.2		
		OH_MPI	image_tag_front: 4.15.2	image_tag_back: 4.15.2			
		OH_ATN	image_tag_front: 4.4.1	image_tag_back: 4.4.2			
		OH_ALE		image_tag_back: 3.7.0			
		OH_WCFORM	image_tag_front: 4.15.0				
		OH_WCVIEWER	image_tag_front: 4.10.0				
		OH_DSK	image_tag_front: 3.30.0				
		OH_WCWIDGET	image_tag_front: 4.17.0				
		OH_WCHIRDOC	image_tag_front: 4.7.0				
		OH_INTEGRATOR	image_tag_front: 4.10.0	image_tag_webcomponent: 4.10.0			
		OH_DOC	image_tag_front: 4.2.0				

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 requisitos de la instalación (epígrafe 2) 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/Oracle 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 Onesait Healthcare incluidos en el paquete Master Data Management (MDM).

Existen dos escenarios, un primer escenario en el cual se dispone de un usuario administrador de la base de datos con el que crear los diferentes esquemas necesarios y un segundo escenario en el que la gestión de la base de datos es externa y se debe pedir a un externo que se creen los esquemas necesarios y nos proporcionen los usuarios y credenciales para acceder a los mismos.

Para la base de datos Oracle, se debe tener instalado con anterioridad Oracle Text

Para el primer escenario:

MySQL con acceso de administrador

Disponiendo de usuario administrador se pueden crear los usuarios y esquemas para los módulos MDM ejecutando el siguiente script en un entorno con acceso a la base de datos:

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

Oracle con acceso de administrador

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

[FTP_SERVER]
/oradata/Versiones_Producto_OH/OH_v4/sistemas_configuraciones_iniciales_v4/mdm_scripts/Oracle/mdm-global-01-create-oracle/mdm-create-schemas-oracle.sh

Los módulos Ontology Server, MPI y Audit & logs lanzan los scripts de creación de las tablas y datos iniciales al iniciar la aplicación.

NOTA: Al ejecutar el script mdm-create-schemas-oracle.sh, se solicitarán por parámetro los nombres de los usuarios de base de datos y sus contraseñas, y el propio script se encargará de crearlos.

La relación entre cada módulo y el nombre de usuario recomendado por defecto en la base de datos es la siguiente:

OHSSO → US_OHSSO

OHCON → US_HNCONF

OHONT → US_HNCAT

OHAUT → US_HNAUT

OHMPI → US_HNPOB

OHATN → US_HNATNA

Este listado únicamente es para identificar el usuario de cada uno de los módulos.

No se recomienda utilizar nombres de usuario distintos a los indicados por defecto, salvo que sea estrictamente necesario. En algunos scripts posteriores estos nombres pueden venir fijados, por lo que el uso de nombres diferentes implicaría modificar dichos scripts manualmente antes de su ejecución.

Para el segundo escenario:

Oracle o MySQL 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. Usuario de base de datos utilizado por el módulo SSO (OHSSO) para la persistencia de su información interna.
- us_ohcon. Usuario de base de datos utilizado por el módulo Settings Manager (OHCON) para el almacenamiento de su configuración y propiedades.
- us_ohont. Usuario de base de datos utilizado por el módulo Ontology Server (OHONT) para el acceso a su esquema y la gestión de ontologías, catálogos y terminologías.

- us_ohaut. Usuario de base de datos utilizado por el módulo Audit & Logs (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. Usuario de base de datos utilizado por el módulo MPI (Master Patient Index). El usuario debe tener permiso "grant option" sobre su esquema para poder asignar permiso a otros usuarios a objetos de su esquema.
- us_ohatn. Usuario de base de datos utilizado por el módulo de Auditoría y Notificaciones (ATN) para el acceso a su esquema principal.
- us_ohatn_hist. Usuario de base de datos utilizado para la gestión del histórico del módulo ATN. 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

MySQL

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

[FTP_SERVER] /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, usuarios y passwords.

Oracle

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

[FTP_SERVER] /oradata/Versiones_Producto_OH/OH_v4/sistemas_configuraciones_iniciales_v4/mdm_scripts/Oracle/mdm-global-02-models-oracle

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.

Se explica a continuación el valor que se debe informar de los parámetros que requiere el chart, tomando como ejemplo el módulo OHSSO. Esta configuración deberá repetirse para el resto de módulos de MDM, utilizando en cada caso el usuario y esquema de base de datos correspondientes.

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 u oracle
- **global.database.host**: host o IP de la base de datos
- **global.database.port:** puerto de la base de datos
- **global.database.oracle_sid:** Oracle SID, sólo es necesario informarlo en el caso de que el tipo de BD sea oracle
- **global.database.oracle_partitioning:** true o false, indica si la instalación de oracle soporta particionamiento de tablas o no, sólo es necesario informarlo en el caso de que el tipo de BD sea oracle

- **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 módulos, 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, que contiene los valores por defecto, ajustándolos en función de las particularidades de la instalación.

NOTA: Se tiene que instalar la versión de MDM que viene en el anexo I. Para ello tendremos que modificar el fichero "values_mdm.yaml" poniendo las versiones que se muestran en las columnas Images Values:

Paquete/Charts	Versión Paquete/Charts	Módulo	Images values 1	Images values 2	Images values 3	Images values 4	Otras variables
MDM	4.1.0	OH_SSO	image_tag_oracle:4.13.0.oracle image_tag_mysql:4.13.0.mysql				
		OH_CON	image_tag_front: 4.11.0	image_tag_front_ng15:4.11.0	image_tag_front_webc: 4.11.0	image_tag_back:4.11.0	
		OH_AUT	image_tag_front: 4.13.4	image_tag_back: 4.13.4	image_tag_front_wcint: 4.7.0		
		OH_ONT	image_tag_front: 4.11.2	image_tag_back: 4.11.3	image_tag_ng15_front: 4.11.2		
		OH_MPI	image_tag_front: 4.15.2	image_tag_back: 4.15.2			
		OH_ATN	image_tag_front: 4.4.1	image_tag_back: 4.4.2			
		OH_ALE		image_tag_back: 3.7.0			
		OH_WCFORM	image_tag_front: 4.15.0				
		OH_WCVIEWER	image_tag_front: 4.10.0				
		OH_DSK	image_tag_front: 3.30.0				
		OH_WCWIDGET	image_tag_front: 4.17.0				
		OH_WCHIRDOC	image_tag_front: 4.7.0				
		OH_INTEGRATOR	image_tag_front: 4.10.0	image_tag_webcomponent: 4.10.0			
		OH_DOC	image_tag_front: 4.2.0				

Una vez configurados todos los parámetros, 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 clúster, 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>
```

De forma opcional, puede configurarse el namespace de instalación como contexto por defecto de la sesión de kubectl, evitando así tener que indicar el parámetro -n <namespace> en cada comando.
Para ello puede ejecutarse:

```
kubectl config set-context --current --namespace=<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

MySQL

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

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

Oracle

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

[FTP_SERVER]
/oradata/Versiones_Producto_OH/OH_v4/sistemas_configuraciones_iniciales_v4/mdm_scripts/Oracle/mdm-global-03-post-oracle

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 -n oh-modules scale deployment ohaut-back --replicas 0

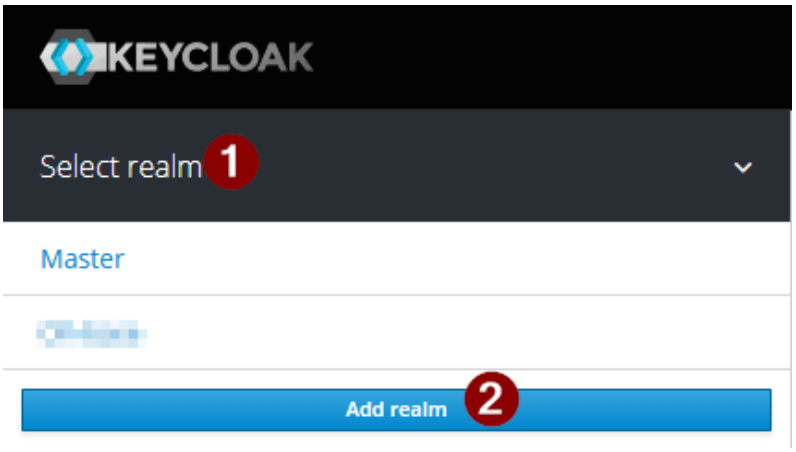
kubectl -n oh-modules scale deployment ohont-back --replicas 0
kubectl -n oh-modules scale deployment ohmpi-back --replicas 0
kubectl -n oh-modules scale deployment ohaut-back --replicas 1
kubectl -n oh-modules scale deployment ohont-back --replicas 1
kubectl -n oh-modules scale deployment ohmpi-back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

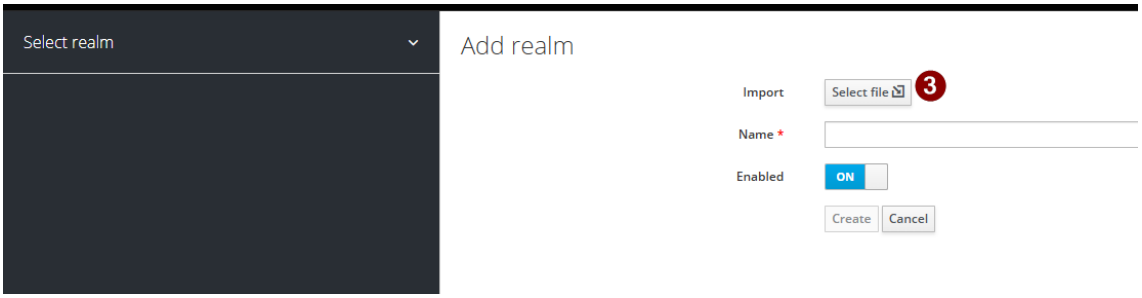
3.5.2. OHSSO

- Acceder a la consola de administración de OHSSO[3] e importar el realm **oh-base**, a partir del fichero:

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

Para importarlo desde la consola de administración pulsamos en “add realm” y posteriormente en “select file”:



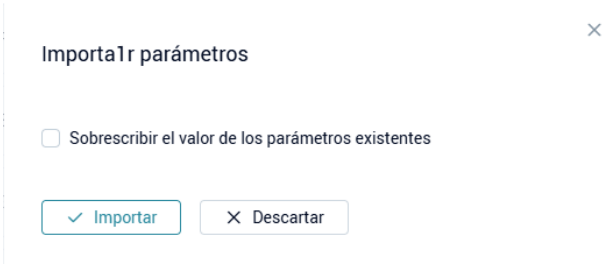


- Acceder a los clients: **hnrole, ruleengine, oh-monitoring y hn-install,** revisar sus **redirectUris** y añadir la del entorno en que se está desplegando (y pulsar **Save** tras añadirlas), es decir, la url pública con la que se expone desde haproxy, por ejemplo, para el clúster de desarrollo sería: 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] El contexto SSO se encuentra en <dominio principal>/auth. A partir de aquí seleccionamos la opción “Administration Console”, utilizando las credenciales configuradas para el usuario admin en la variable **onesaithealthcare_ohsso_chart.ohsso.db.adminpassword**.

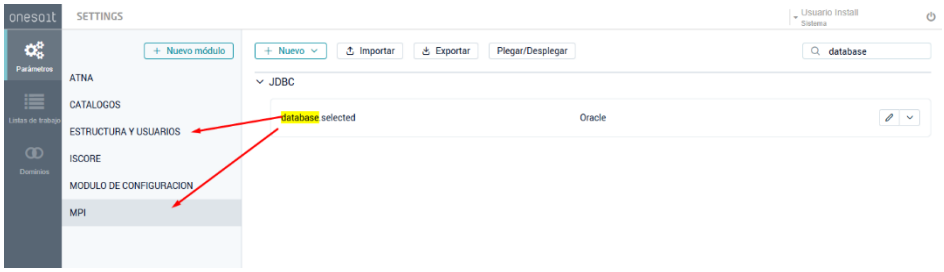
3.5.3. OHCONF

Acceder al módulo OHCON (la ruta sería *<dominio principal>/hnconf*) (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_SERVER) /oradata/Versiones_Producto_OH/OH_v4/OH_CON_v[VERSION]/TOTAL/catálogo/OHCON_HNCONF.csv**
- **(FTP_SERVER) /oradata/Versiones_Producto_OH/OH_v4/OH_CON_v[VERSION]/TOTAL/catálogo/OHCON_ISCORE.csv**
- **(FTP_SERVER) /oradata/Versiones_Producto_OH/OH_v4/OH_AUT_v[VERSION]/TOTAL/config/OHCON/OHCON_HNAUT.csv**
- **(FTP_SERVER) /oradata/Versiones_Producto_OH/OH_v4/OH_MPI_v[VERSION]/TOTAL/config/OHCON_ISPOB.csv**
- **(FTP_SERVER) /oradata/Versiones_Producto_OH/OH_v4/OH_ONT_v[VERSION]/TOTAL/config/OHCON/OHCON_HNCAT.csv**
- **(FTP_SERVER) /oradata/Versiones_Producto_OH/OH_v4/OH_ATN_v[VERSION]/conf/Totales/OHCON/OHCON_HNATNA.csv**

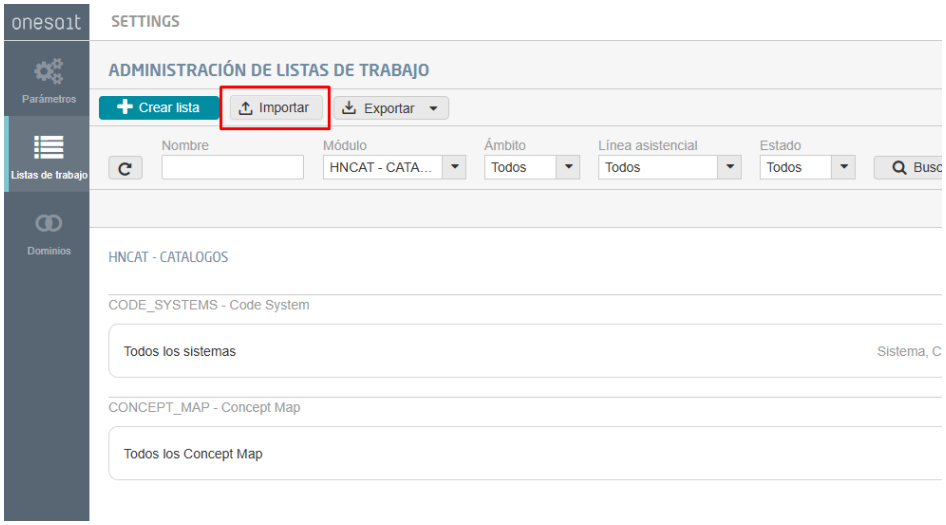
Una vez cargadas las propiedades de los módulos, buscar en MPI y OHAUT la propiedad **"database.selected"**, como valor inicial tiene Oracle, pero según el sistema utilizado podría otro sistema de BD, como por ejemplo MySql.



Deslogarse del módulo OHCON y volver a logarse.

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

- **(FTP_SERVER) /oradata/Versiones_Producto_OH/OH_v4/OH_ONT_v[VERSION]/TOTAL/config/OHCON/HNCAT_WorkList.csv**



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

```
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 (la ruta sería *<dominio principal>/hncat*) e importar los catálogos de los siguientes ficheros:

- **(FTP_SERVER) /oradata/Versiones_Producto_OH/OH_v4/OH_ONT_v[VERSION]/conf/Post/CodeSystem**

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

- **(FTP_SERVER) /oradata/Versiones_Producto_OH/OH_v4/OH_ONT_v[VERSION]/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; los pasos para crear esta estructura funcional se detallan en el documento de manual de configuración y se realizarán cuando se completen todos los pasos de este documento de instalación.

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

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

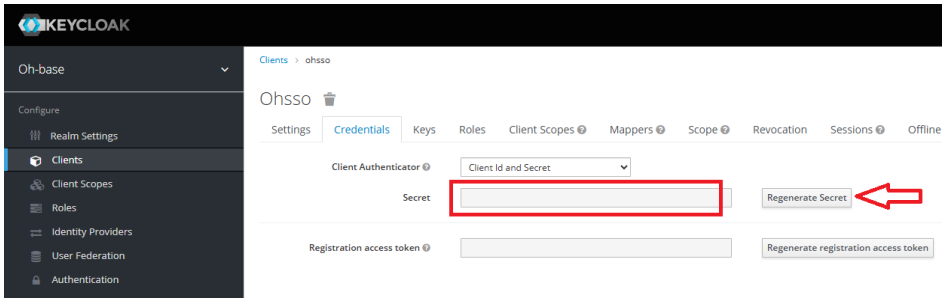
El reinicio de los pods puede realizarse ejecutando los siguientes comandos:

```
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
```

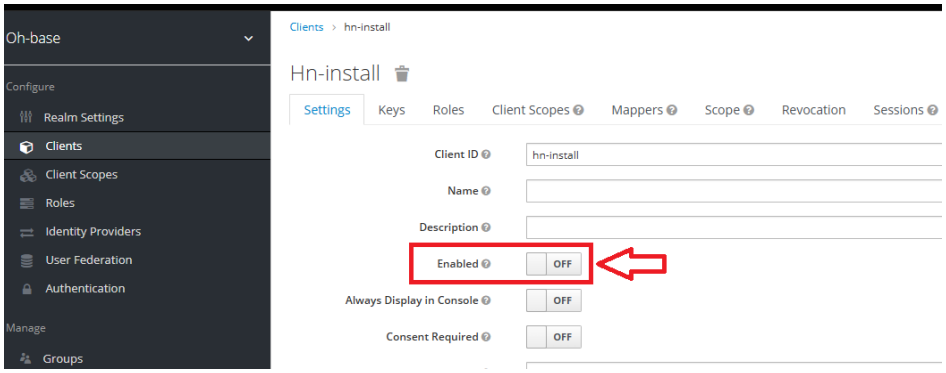
3.5.6. CONFIGURACIÓN DEFINITIVA OHSSO

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.

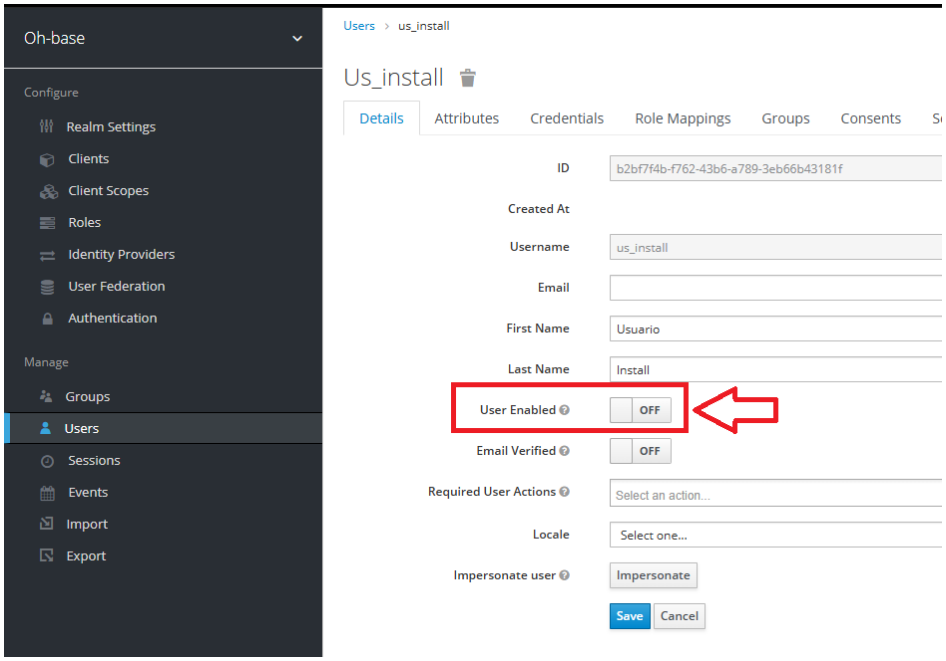
- Acceder a la consola de administración de OHSSO (*<dominio principal>/auth*), al client ohsso, y regenerarle las credenciales (pestaña Credentials → botón Regenerate Secret), anotando el nuevo secret ya que se usará en un paso posterior.



- Deshabilitar el client hn-install, marcando la opción “OFF” en Enabled y salvar los cambios.



- Deshabilitar usuario us_install y salvar los cambios.



- 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
```

- 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
```

- 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[4], creado durante la configuración inicial de OHSSO y configurado por defecto.

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

[4] El usuario se crear con la contraseña por defecto 12345678a El sistema solicitará el cambio de contraseña tras el primer login.

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 requisitos 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 u Oracle 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.

Existen dos escenarios, un primer escenario en el cual se dispone de un usuario administrador de la base de datos con el que crear los diferentes esquemas necesarios y un segundo escenario en el que la gestión de la base de datos es externa y se debe pedir a un externo que se creen los esquemas necesarios y nos proporcionen los usuarios y credenciales para acceder a los mismos.

Para el primer escenario:

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_SERVER]
/oradata/Versiones_Producto_OH/OH_v4/OH_HDR_v[VERSION]/bbdd/mysql/totales/1-ohhdr-user.sql

- Consent Manager
[FTP_SERVER]
/oradata/Versiones_Producto_OH/OH_v4/OH_CSM_v[VERSION]/bbdd/total/mysql/1-ohcsm-user.sql

Oracle 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_SERVER] /oradata/Versiones_Producto_OH/OH_v4/OH_HDR_v[VERSION]/bbdd/oracle/totales/1-ohhdr-tablespace.sql

[FTP_SERVER] /oradata/Versiones_Producto_OH/OH_v4/OH_HDR_v[VERSION]/bbdd/oracle/totales/2-ohhdr-user.sql

- Consent Manager
[FTP_SERVER] /oradata/Versiones_Producto_OH/OH_v4/OH_CSM_v[VERSION]/bbdd/total/oracle/1-ohcsm-tablespace.sql

[FTP_SERVER] /oradata/Versiones_Producto_OH/OH_v4/OH_CSM_v[VERSION]/bbdd/total/oracle/2-ohcsm-user.sql

Para el segundo escenario:

Oracle o MySQL sin acceso de administrador

Si no se dispone de acceso de administrador se tendrá que solicitar al equipo administrador del gestor de base de datos, 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. Usuario de base de datos utilizado por el módulo Global Repository (OH_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. Usuario de base de datos utilizado por el módulo Consent Manager (OH_CSM). 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_SERVER] /oradata/Versiones_Producto_OH/OH_v4/OH_HDR_v[VERSION]/bbdd/mysql/totales/2-ohhdr-create-database.sql

[FTP_SERVER] /oradata/Versiones_Producto_OH/OH_v4/OH_HDR_v[VERSION]/bbdd/mysql/totales/3-ohhdr-tables.sql

- Consent Manager
Con el usuario propio del módulo lanzar los siguientes scripts:

[FTP_SERVER]
/oradata/Versiones_Producto_OH/OH_v4/OH_CSM_v[VERSION]/bbdd/total/mysql/2-ohcsm-create-database.sql

[FTP_SERVER]
/oradata/Versiones_Producto_OH/OH_v4/OH_CSM_v[VERSION]/bbdd/total/mysql/3-ohcsm-tables.ddl.sql

4.3.2. Oracle

- Global Repository

Con el usuario propio del módulo lanzar los siguientes scripts:

[FTP_SERVER]
/oradata/Versiones_Producto_OH/OH_v4/OH_HDR_v[VERSION]/bbdd/oracle/totales/3-ohhdr-tables.sql

- Consent Manager
Con el usuario propio del módulo lanzar los siguientes scripts:

[FTP_SERVER]
/oradata/Versiones_Producto_OH/OH_v4/OH_CSM_v[VERSION]/bbdd/total/oracle/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 descritos a continuación de los campos que se quiera dar a la instalación.

NOTA: Se tiene que instalar la versión de DATA que viene en el anexo I. Para ello tendremos que modificar el fichero "values_data.yaml" poniendo las versiones que se muestran en las columnas Images Values:

Paquete/Charts	Versión Paquete/Charts	Módulo	Images values 1	Images values 2	Images values 3	Images values 4	Otras variables
Data	4.2.0	OH_HDR	onesaithealthcare-ohhdr-chart.ohhdr.version: 4.3.0				
		OH_CSM	onesaithealthcare-ohcsm-chart.ohcsm.version: 4.1.0				dependencies.ohcsm: true
		OH_HDA	onesaithealthcare-ohhda-chart.ohhda.version: 4.19.0				

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 cuando el clúster expone servicios mediante la API estándar de Kubernetes (gateway.networking.k8s.io/v1) o cuando el clúster dispone de Istio instalado y los servicios se exponen mediante recursos 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. (En este caso sería: oh-docker-creds)
- **dependencies.ohhdr/ohcsm/ohhda**: Por defecto se dejan en true
- **dependencies.ohcsm/ohdvm/ohhdaale/ohhda-concerted:** Por defecto se dejan a false
- **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 (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 (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

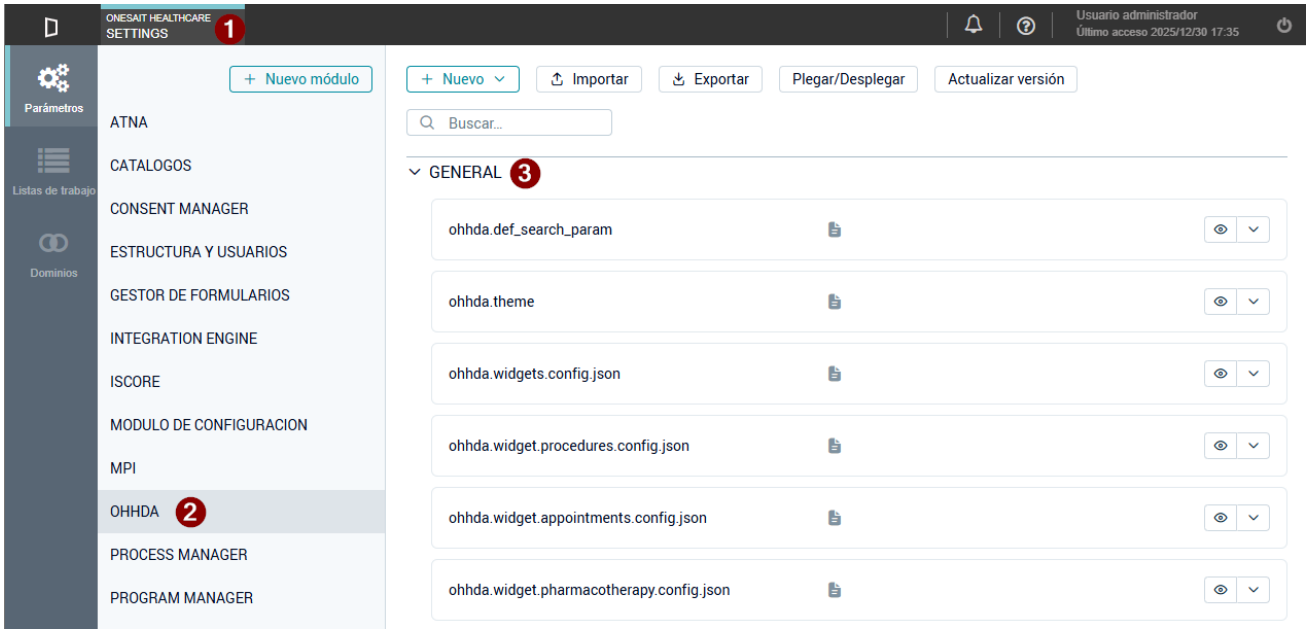
4.5.1. Visor Historia Clínica (OH_HDA)

Setting

Importar los ficheros de la ruta:

[FTP_SERVER]
/oradata/Versiones_Producto_OH/OH_v4/OH_HDA_v[VERSION]/conf/OHCON/OHCON_OHHDA.csv

La configuración cargada obtenida:



4.5.2. Consent Manager (OH_CSM)

Setting

Importar los ficheros de la ruta:

[FTP_SERVER]
/oradata/Versiones_Producto_OH/OH_v4/OH_CSM_v[VERSION]/conf/OHCON /

Ontology

- Comprobar que existe el siguiente catálogo en Ontology:

  Para ello, acceder al módulo Ontology Server (OHONT) con credenciales de administración (la ruta sería *<dominio principal>/hncat*), navegar a la sección Catálogos → CodeSystems y verificar la existencia del catálogo indicado.

  http://hl7.org/fhir/CodeSystem/CONSENT_TYPE
  - Si no existe, crearlos con las siguientes propiedades:
    Title: CONSENT_TYPE
    Name: CONSENT_TYPE
    Hierarchy: Sin especificar
    Content: Complete
    Status: Activo
    Properties:
      Boolean - createConsentManager
      Boolean - editConsentManager
      String - tipo
      Boolean – singleConsent

  - Si ya existe, se pueden eliminar los elementos que vengan por defecto.

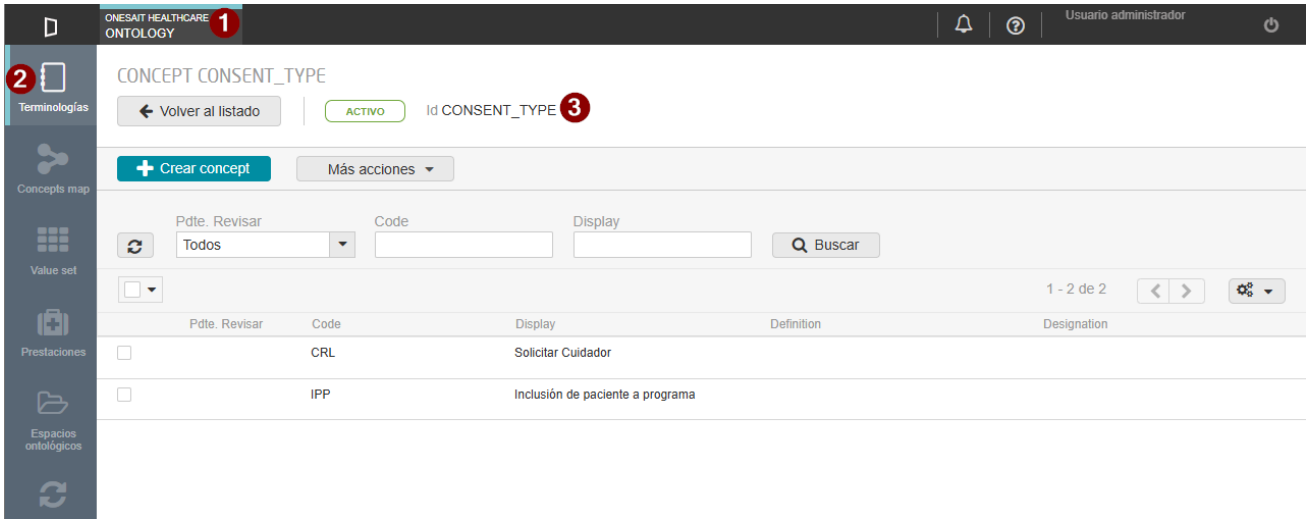
Antes de crear los elementos nuevos, tenemos que comprobar que está creado el catálogo “designation-use.json”; sino lo está importamos el fichero que está en la ruta:

(FTP_SERVER) /oradata/Versiones_Producto_OH/OH_v4/OH_ONT_v[VERSION]/conf/Post/CodeSystem/designation-use.json

- Crear las siguientes entradas en el catálogo CONSENT_TYPE accediendo a la gestión de elementos y dando de alta los códigos con los valores indicados:
  * Code: CRL
    Display: Solicitar Cuidador
    tipo: C
    createConsentManager: true
    editConsentManager: true
    singleConsent: false

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

El catálogo quedará de la siguiente manera:



5. Instalación Paquete Integration & Interoperability

5.1. Módulos que incluye

- **OHIEN (Control Panel)**

Panel centralizado de control para visualizar y administrar el ecosistema Kafka, permitiendo la gestión de topics (canales lógicos de mensajería), grupos de consumidores, integraciones, esquemas y el estado general del clúster. Kafka actúa como plataforma de mensajería distribuida orientada a eventos.

- **Clúster Kafka**

Clúster de Kafka con configuración adaptada para OHIEN en modo KRaft, en el que la gestión de los metadatos y la coordinación del clúster se realiza de forma nativa por Kafka, sin necesidad de ZooKeeper

- **Kafka Connect**

Plataforma de integración que permite conectar Kafka con sistemas externos mediante conectores estándar. La configuración y supervisión de los conectores se realiza desde el módulo OHIEN (Control Panel).

- **Schema Registry**

Servicio encargado de almacenar y gestionar los esquemas de datos utilizados en Kafka (Avro, JSON Schema y Protobuf), garantizando la validación de los mensajes y la compatibilidad entre productores y consumidores a lo largo del ciclo de vida de los eventos.

- **Operador camel-k**

Instalación del operador camel-k necesario para el despliegue y ejecución de integraciones desarrolladas con Apache Camel sobre Kubernetes, permitiendo definir y ejecutar flujos de integración de forma nativa en el clúster.

Importante:

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

Nota: CamelK
Si necesitas usar integraciones desarrolladas con Camel, debes instalar un segundo chart para Camel K (onesaithealthcare-ohien-camelk-chart). 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 (epígrafe 3 del presente documento) 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*.
- No se requiere ninguna base de datos adicional para la instalación y funcionamiento de OHIEN, excepto si se habilita el módulo opcional RTEP. Solo en ese caso será necesario la base de datos PostgreSQL y sus credenciales de acceso.
- 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:

Nota: Kafka Storage Type
Asegúrate de que el StorageClass seleccionado soporte block storage y aprovisionamiento dinámico.

5.3.2. 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 se 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

Instalación OHIEN con helm

Se puede consultar el apartado *2. 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
```

```
NOTA: Se tiene que instalar la versión de OH_IEN que viene en el anexo I. Para
      ello tendremos que modificar el fichero "values_ohien_iengine.yaml"
      poniendo las versiones que se muestran en las columnas Images Values:
```

PaqueteICharts	Versión PaqueteICha	Módulo	Images values 1	Images values 2
Ohien iengine	4.2.0	kafkaien	onesaithealthcare_ohien_kafkaien_chart.image.tag: 4.0.0	
		kafka_connect	onesaithealthcare_ohien_kafka_connect_chart.image.tag: 4.0.1	
		schemaregistry	onesaithealthcare_ohien_schemaregistry_chart.image.tag: 4.0.0	
		control_panel	onesaithealthcare_ohien_control_panel_chart.image.backend.tag: 4.1.0	onesaithealthcare_ohien_control_panel_chart.image.frontend.tag: 4.1.0

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 (Se recomienda `5Gi` en entornos de pruebas y `10Gi` para entornos productivos).
- `**onesaithealthcare_ohien_kafkaien_chart.kafka.storage.storageClass**`: StorageClass de Kubernetes a usar, el configurado cumpliendo los prerrequisitos indicados en el apartado anterior. Se desaconseja dejarlo vacío, es decir, en modo `ephemeral`incluso en entornos preproductivos.
- `**onesaithealthcare_ohien_kafkaien_chart.kafka.storage.deleteClaim**`: Si es `true`, el PVC se elimina al borrar el release de Helm. Para producción, 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 4.0.1 .

onesaithealthcare_ohien_schemaregistry_chart

- `**onesaithealthcare_ohien_schemaregistry_chart.image.tag**`: Versión de la imagen Docker de Schema Registry. Por defecto 4.0.0 .

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 4.1.0 .
- `**onesaithealthcare_ohien_control_panel_chart.image.frontend.tag**`: Versión de la imagen Docker del frontend del Control Panel. Por defecto 4.1.0
- `**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 pudiendo oscilar habitualmente entre varios minutos y decenas de minutos hasta que todos los pods estén disponibles).

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

Con el comando:

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

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_SERVER, descargar el fichero de

(FTP_SERVER) /oradata/Versiones_Producto_OH/OH_v4/OH_IEN_v[VERSION]/hnconf/total/OHCON_OHIEN_20251125.csv

Luego de ingresar en HNCONF, utilizar la opción de "Importar" y seleccionar el fichero: OHCON_OHIEN_20251125.csv.

Importación en HNCONF

1. Ingrese al módulo HNCONF (la ruta sería <*dominio principal*>*/hnconf*).
2. Accede a la opción Settings → Integration Engine.
3. Haga clic en Importar y seleccione el fichero descargado:
   `OHCON_OHIEN_20251125.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

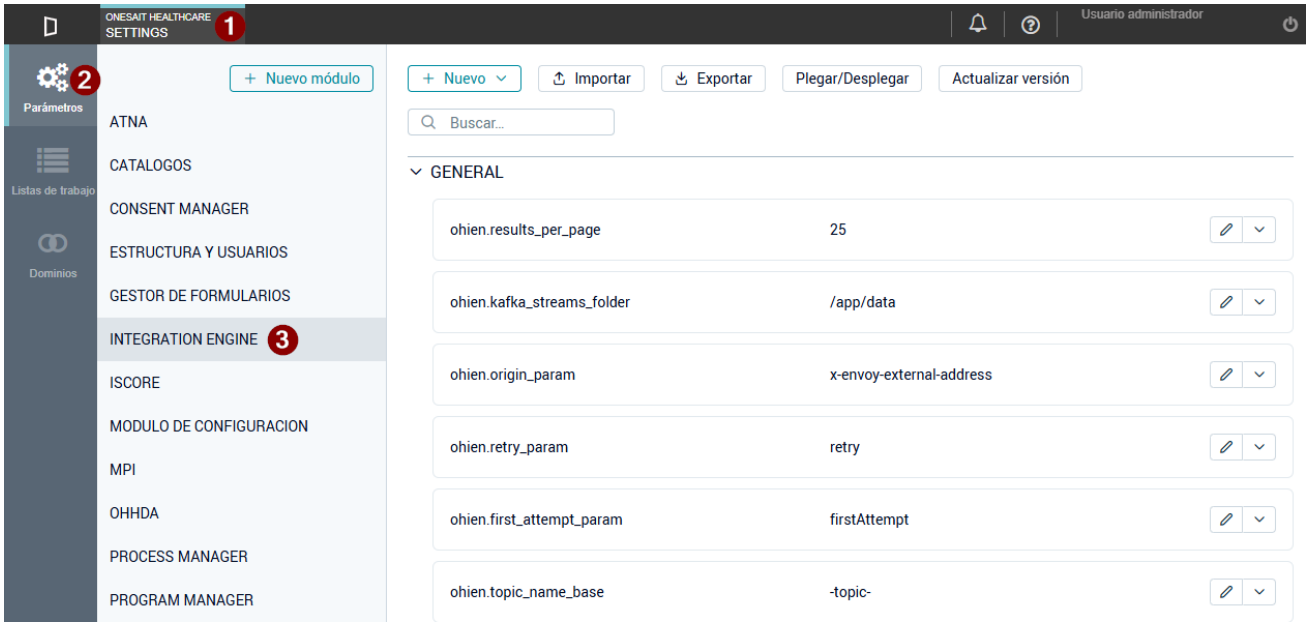
Parámetro	Descripción
`ohien.kafka.config.connect.password`	Contraseña del usuario Kafka utilizado por OHIEN. Obtener desde el `Secret` llamado `ohien` en el namespace correspondiente, revelando su valor.

Diríjase a **Settings > ISCORE

- ISCORE

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

Vista previa de los parámetros de configuración:



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

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

5.4.2. 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 fichero de configuración por defecto.

NOTA: se debe utilizar el paquete Helm que viene dentro de los artefactos de instalación en vez de él que esta subido en el repositorio. La ruta sería la siguiente:

[FTP_SERVER]/oradata/Versiones_Producto_OH/OH_v4/onesaithealthcare-ohien-camelk-chart-4.0.0_fix.tgz

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

Se explica a continuación el valor que se debe informar en los parámetros que requiere el chart. 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 (se aconseja Eclipse Temurin JDK 17).

- `**ohien.toolImage**`: Indica la imagen Docker que se usará para el entorno de herramientas de compilación (Quarkus Mandrel builder con JDK 21).
- `**camel-k-operator.operator.enabled**`: Activa (`true`)
- `**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**`: Se aconseja dejar desactivada esta opción (`false`), aunque el sistema permitiría configurar un repositorio Maven personalizado, a través de los siguientes parámetros:
    - `**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
```

NOTA: Se tiene que instalar la versión de camel-k que viene en el anexo I. Para ello tendremos que modificar el fichero "values_ohien_camelk.yaml" poniendo las versiones que se muestran en las columnas Images Values:

Paquete/Charts	Versión Paquete/Chart	Módulo	Images values 1
Ohien camelk	4.0.0	camel-k	camel-k-operator.operator.image: docker.io/apache/camel-k:2.7.0

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

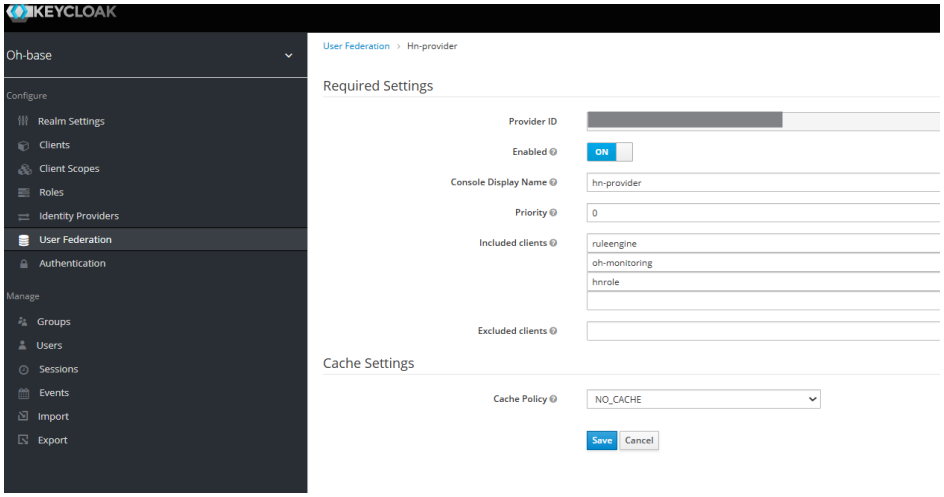
6.2. Prerrequisitos

- 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, a la "Administration Console" (la ruta sería <*dominio principal*>*/auth*)
- Si se tiene instalado el client oh-monitoring, ignorar este punto. Si no se tiene instalado el client oh-monitoring, se crea siguiendo estos pasos:

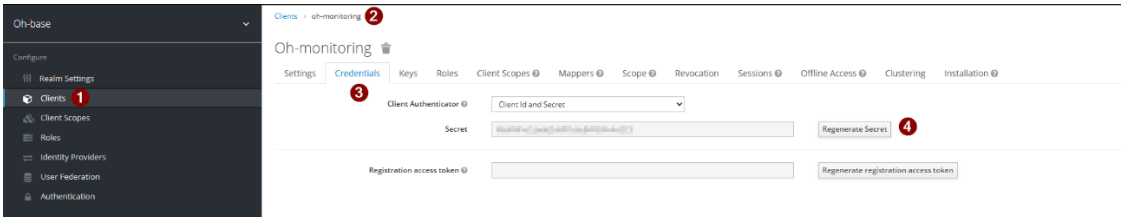
    - Acceder a Client Scopes y añadir los scopes (de tipo openid-connect): openid, groups.
    - Importar el client oh-monitoring con el json:

      (FTP_SERVER) /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 pulsar Edit en hn-provider, para como se muestra a continuación 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 apuntar su valor 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[5].

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.

[5] Podemos comprobarlo usando en la terminal este comando *kubectl get clusternetworkpolicies -A -o yaml.* Tanto si no encuentra nada como si encuentra habilitados los puertos indicados la configuración será correcta.

6.4. Procedimiento de despliegue de Módulos

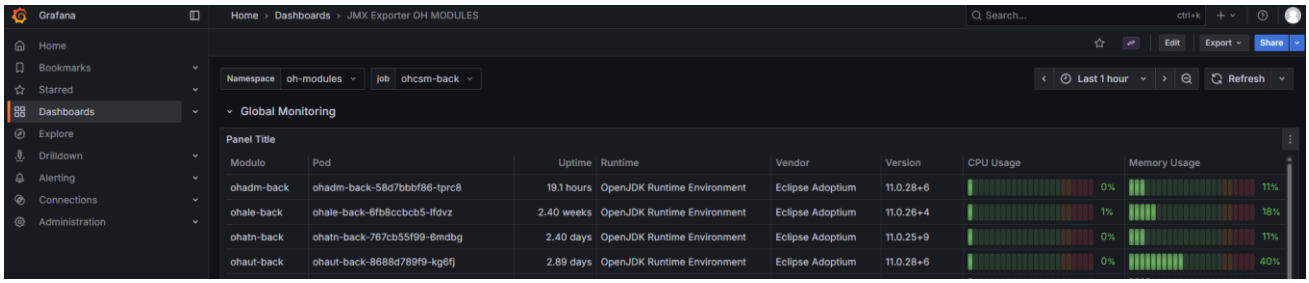
Consultar la guía general de prerrequisitos donde se indican varias alternativas para registrar el repositorio helm e instalar el chart.
El repositorio se encuentra en:
https://nexus.devops.onesait.com/repository/onesait-healthcare-helm-charts
Se deberá disponer de credenciales para poder acceder al 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 <dominio principal>/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 <dominio principal>/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 Framework BI

7.1. Módulos que incluye

- Módulo OHBI

7.2. Prerrequisitos

Se deben cumplir los requisitos 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_SERVER) /oradata/Versiones_Producto_OH/OH_v4/OH_BI_v[VERSION]/bbdd/total/mysql**

Si se utiliza Oracle en la instalación, la persistencia de estos objetos se realizará 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 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 un volumen persistente.

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

Tan solo habrá que cambiar en el PV a crear respecto al de la entrega, lo siguiente (acorde a lo proporcionado por el administrador de sistemas una vez haya creadc 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 el PV de la entrega, creando antes el storageClass mediante el siguiente fichero:

(FTP_SERVER) /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, que contiene los valores por defecto, ajustándolos en función de las particularidades de la instalación.

```
NOTA: Se tiene que instalar la versión del OH_BI que viene en el anexo I. Para
      ello tendremos que modificar el fichero "values_bi.yaml" poniendo las
      versiones que se muestran en las columnas Images Values:
```

PaqueteICharts	Versión PaqueteICha	Módulo	Images values 1	Images values 2	Images values 3
Framework BI	4.1.0	OH_BI	bi.tag.back: 4.4.0	bi.tag.front: 4.1.0	
		OH_DTC	dtc.tag.back: 4.1.0	dtc.tag.front: 4.1.0	dtc.tag.client: 4.4.0

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 del api general de kubernetes (gateway.networking.k8s.io/v1) o del 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. Si la opción es MySQL, añadir los siguientes:
    - **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.

Si se va a usar Derby, la bd embebida, añadir los siguientes:

- **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_SERVER)/oradata/Versiones_Producto_OH/OH_v4/OH_BI_v[VERSION]/bbdd/total/PermisosOHBI_MySql**
- OHAUT_Permisos_OHBI_mysql.sql **(FTP_SERVER)/oradata/Versiones_Producto_OH/OH_v4/OH_BI_v[VERSION]/bbdd/total/PermisosOHBI_MySql**

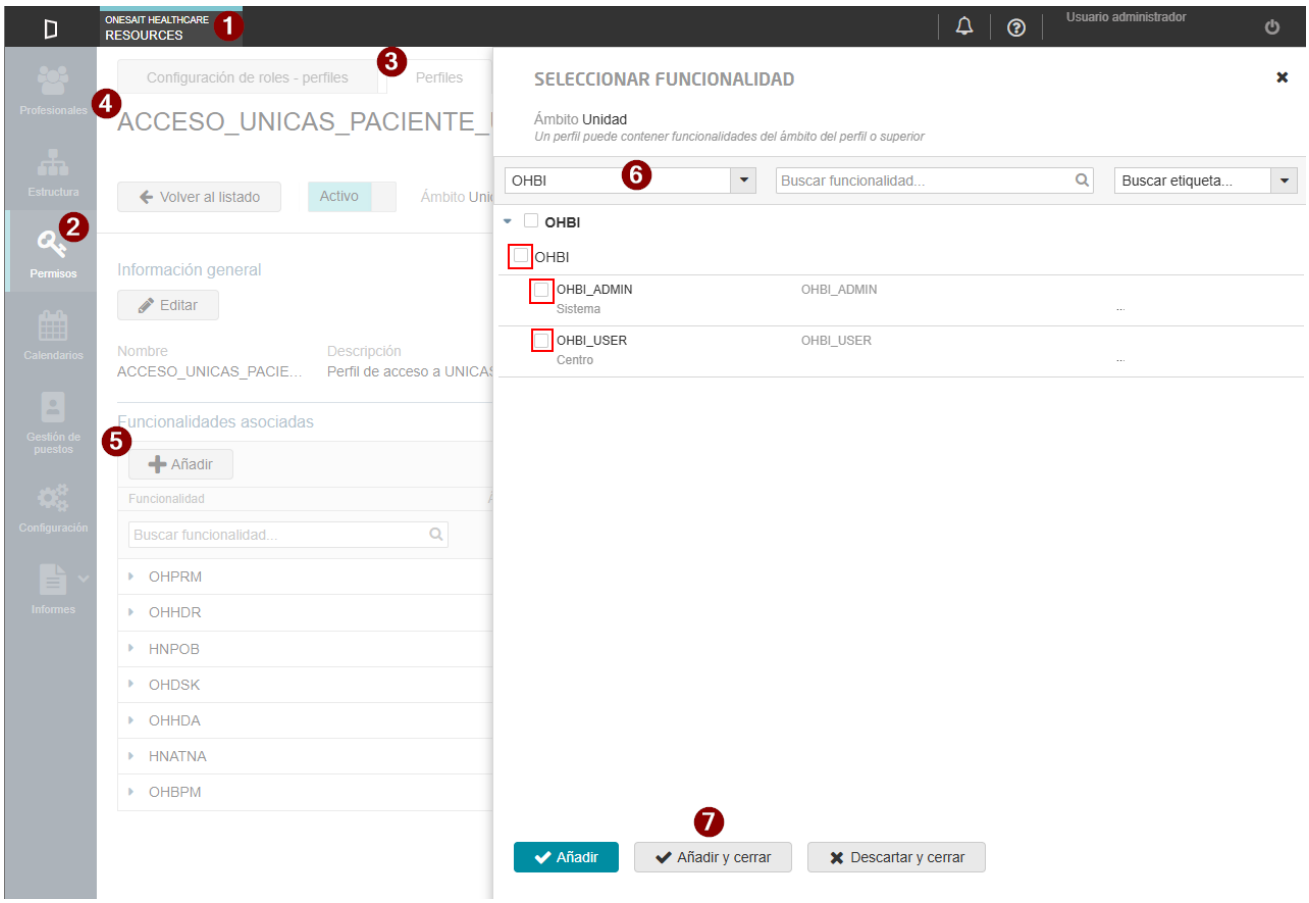
Si el módulo MDM está en una bbdd Oracle:

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

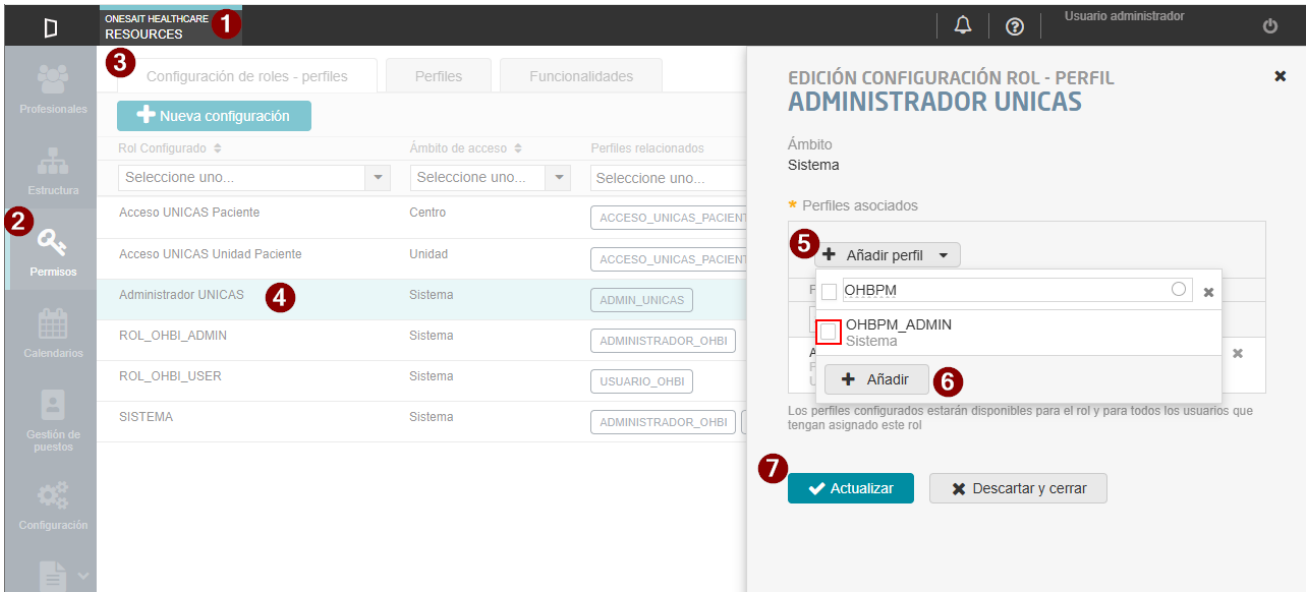
- OHAUT_Permisos_OHBI_oracle.sql **(FTP_SERVER) /oradata/Versiones_Producto_OH/OH_v4/OH_BI_v[VERSION]/bbdd/total/PermisosOHBI_Oracle**

A continuación, desde el módulo OHAUT (la ruta sería *<dominio principal>/hnaut*), 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.

Ejemplo asignación permisos a role:



Ejemplo asignación perfiles a role:



7.02. Instalación DTC

7.1. Módulos que incluye

- Módulo de Configuración de la Ingesta (OHDTC)

7.2. Prerrequisitos

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

7.3. Procedimiento de despliegue de Capa de Persistencia

Si la capa de persistencia va sobre Oracle, se creará el modelo de datos del módulo OH_DTC:
[FTP_SERVER]
/oradata/Versiones_Producto_OH/OH_v4/OH_DTC_v[version]/bbdd/total/oracle

Si la capa de persistencia va sobre MySql, se creará el modelo de datos del módulo OH_DTC:
[FTP_SERVER]
/oradata/Versiones_Producto_OH/OH_v4/OH_DTC_v[version]/bbdd/total/mysql

7.4. Procedimiento de despliegue de Módulos

Para instalar el módulo del DTC se empleará el chart de Helm correspondiente, **onesaithealthcare-dtc-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 DTC:

helm show values

```
helm show values onesait-healthcare-helm-repo/onesaithealthcare-dtc-chart > values_dtc.yaml
```

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

NOTA: Se tiene que instalar la versión del OH_DTC que viene en el anexo I. Para ello tendremos que modificar el fichero "values_bi.yaml" poniendo las versiones que se muestran en las columnas Images Values:

Paquete/Charts	Versión Paquete/Charts	Módulo	Images values 1	Images values 2	Images values 3
Framework BI	4.1.0	OH_BI	bi.tag.back: 4.4.0	bi.tag.front: 4.1.0	
		OH_DTC	dtc.tag.back: 4.1.0	dtc.tag.front: 4.1.0	dtc.tag.client: 4.5.0

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**: nombre del Gateway.
- **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).
- **dtc.module.namespace:** nombre namespace.
- **dtc.module.schedule**: periodicidad de ejecución del cronJob. Se debe poner en formato crontab. Este dato aplica a los yamls de la ingesta que se generen desde este módulo.
- **dtc.module.max_days**: máximo de días a ingestar a partir de la fecha final del último periodo ingestado. Será 0 si se quiere ingestar todo. Este dato aplica a los yamls de la ingesta que se generen desde este módulo.
- **dtc.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 muy antiguos en los que no haya

datos. Este dato aplica a los yamls de la ingesta que se generen desde este módulo. Debe tener el formato dd/MM/yyyy HH:mm:ss.

- **dtc.resources.yamls.limits.cpu**: límite de CPU que se usará en la ingesta posterior (mínimo 400m). Este dato aplica a los yamls de la ingesta que se generen desde este módulo.
- **dtc.resources.yamls.limits.memory**: límite de memoria que se usará en la ingesta posterior (mínimo 700Mi). Este dato aplica a los yamls de la ingesta que se generen desde este módulo.
- **dtc.keycloak.realmname:** nombre del realm de keycloak. Por defecto será oh-base
- **dtc.keycloak.resource**: resource del keycloak. Por defecto será hnrole
- **dtc.database.type.source**: tipo de bbdd origen de la cual se extraerá la información a ingestar en el datastore.
- **dtc.database.type.target**: tipo de bbdd destino de la ingesta (datastore).
- **dtc.database.datasource.host**: host de la base de datos origen de la cual se extraerá la información a ingestar en el datastore.
- **dtc.database.datasource.port**: puerto de la base de datos origen de la cual se extraerá la información a ingestar en el datastore.
- **dtc.database.datasource.sid**: sid o servicio de la base de datos origen (solo para oracle).
- **dts.database.datasource.use_sid**: indica si se usará SID (true) o Service (false) para la conexión a la base de datos origen (solo para oracle).
- **dtc.database.datasource.schema**: esquema de base de datos origen (solo para mysql).
- **dtc.database.datasource.parameters**: parámetros de base de datos origen.
- **dtc.database.datasource.username**: usuario de conexión a la base de datos origen.
- **dtc.database.datasource.password**: contraseña de conexión a la base de datos origen.
- **dtc.database.target.host**: host de la base de datos staging area y destino (datastore).
- **dtc.database.target.port**: puerto de la base de datos staging area y destino (datastore).
- **dtc.database.target.sid**: sid o servicio de la base de datos destino staging area y destino (solo para oracle).
- **dts.database.target.use_sid**: indica si se usará SID (true) o Service (false) para la conexión a la base de datos staging area y destino (solo para oracle).
- **dtc.database.target.parameters**: parámetros de base de datos staging area y destino.
- **dtc.database.target.iengine.schema**: esquema de base de datos staging area (solo para mysql).
- **dtc.database.target.iengine.username**: usuario de conexión a la base de datos staging area.
- **dtc.database.target.iengine.password**: contraseña de conexión a la base de datos staging area.
- **dtc.database.target.datastore.schema**: esquema de base de datos destino (solo para mysql).
- **dtc.database.target.datastore.username**: usuario de conexión a la base de datos destino.
- **dtc.database.target.datastore.password**: contraseña de conexión a la base de datos destino.

- **dtc.persistence.storagesize**: espacio de almacenamiento en disco para la persistencia de la información.
- **dtc.persistence.storageclassname**: nombre del storageclassname que vamos a utilizar. Es un objeto que ya debe existir en el entorno con independencia de este despliegue.

Se instala el chart con el comando:

helm install

```
helm install -f values_dtc.yaml dtchncat onesait-healthcare-helm-repo/onesaithealthcare-dtc-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

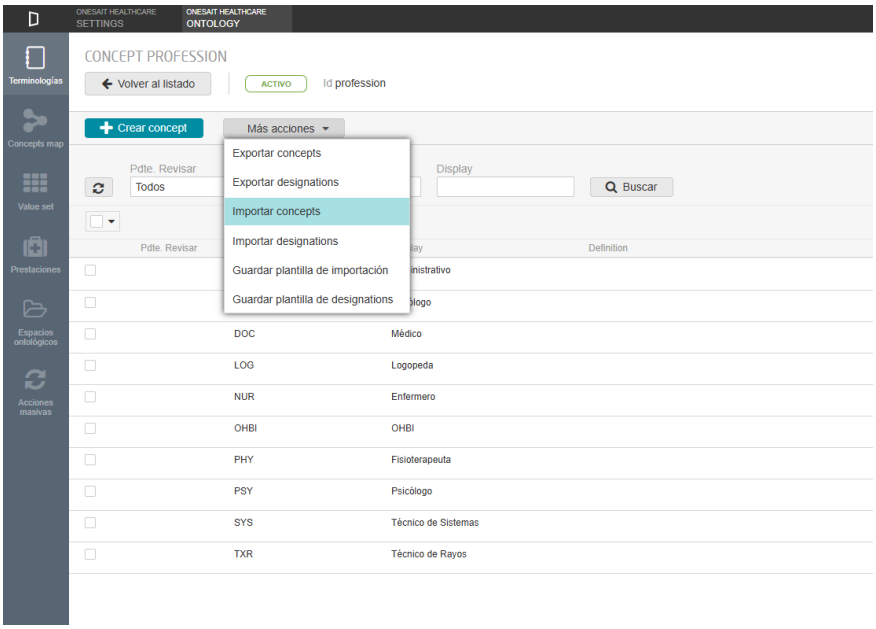
```
kubectl describe pod <pod name> -n <namespace de instalación>
```

7.4. Operaciones post-instalación

Una vez se ha instanciado el helm correspondiente y haya desplegado el módulo correctamente se deben realizar los siguientes pasos en el orden indicado, que habilitará todo el mecanismo necesario (perfiles, roles, etc) para poder acceder a este módulo desde el escritorio.

- Crear un concept en el catálogo 'profession' del módulo OHONT importando el contenido del fichero.

- profession.csv **(FTP_SERVER)/oradata/Versiones_Producto_OH/OH_v4/OH_DTC_v[VERSION]/bbdd/Permiso OHDTC**



- Crear un concept en el catálogo 'tipo_profesional' del módulo ONONT importando el contenido del fichero.
  - tipo_profesional.csv **(FTP_SERVER)/oradata/Versiones_Producto_OH/OH_v4/OH_DTC_v[VERSION]/bbdd/Permiso OHDTC**
- Crear un elemento en el concept map en la relación entre catálogo origen 'profession' y catálogo destino 'tipo_profesional' del módulo OHONT importando el contenido del fichero.
  - Profesion_Roles.csv **(FTP_SERVER)/oradata/Versiones_Producto_OH/OH_v4/OH_DTC_v[VERSION]/bbdd/Permiso OHDTC**
- Si el módulo MDM está en una bbdd MySql, ejecutar el siguiente script:
  - OHAUT_Permisos_OHDTC_mysql.sql **(FTP_SERVER)/oradata/Versiones_Producto_OH/OH_v4/OH_DTC_v[VERSION]/bbdd/Permiso OHDTC/MySql**

Si el módulo MDM está en una bbdd Oracle, ejecutar el siguiente script:

  - OHAUT_Permisos_OHDTC_oracle.sql **(FTP_SERVER) /oradata/Versiones_Producto_OH/OH_v4/OH_DTC_v[VERSION]/bbdd/Permiso OHDTC/Oracle**

- A continuación, desde el módulo OHAUT, habrá que asignar a los usuarios a los que se le quiera dar acceso al módulo, el rol anteriormente creado por scripts 'ROL_OHDTC_ADM' o asignar el perfil anteriormente creado por scripts 'PERF_OHDTC_ADM' a uno de los roles que tengan estos usuarios.

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).

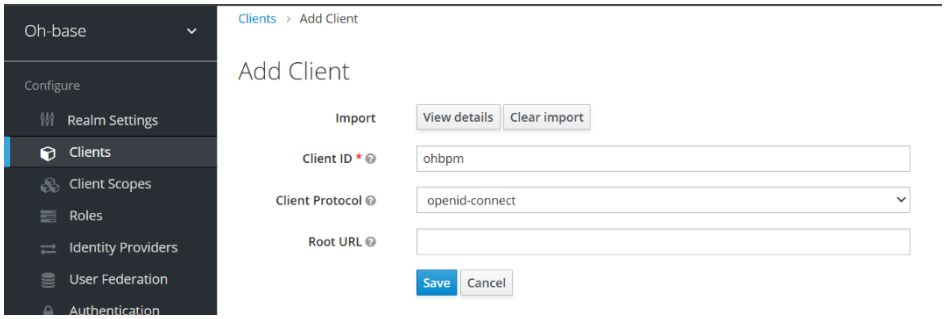
8.2. Prerrequisitos

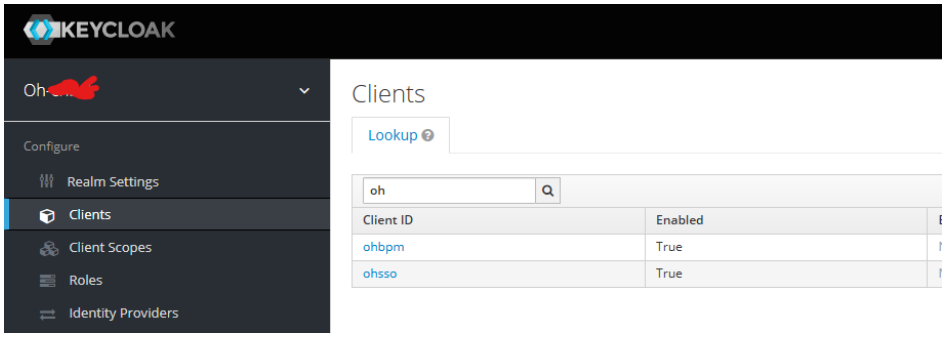
- Se deben cumplir los requisitos generales de la instalación y haber realizado los pasos indicados en la guía correspondiente.
- Deben instalarse previamente los paquetes MDM y DATA (epígrafes 3 y 4 del presente documento).
- Se debe crear client en keycloack para ohbpm para todos los accesos a BPM, como se indica a continuación.

8.2.1. Prerrequisito KEYCLOACK

Instalar cliente ohbpm o usar el ohsso para el uso de acciones offline (creación de actividades dinámicas o temporizadas).

El client **ohbpm** se instala en el realm oh-base de keycloak[6], pulsando "Clients" → "Create" → y ahí importamos el json que se proporciona a continuación.





[6] El contexto SSO se encuentra en <dominio principal>/auth. A partir de aquí seleccionamos la opción "Administration Console", utilizando las credenciales configuradas para el usuario admin en la variable onesaithealthcare_ohsso_chart.ohsso.db.adminpassword.

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_SERVER]
/oradata/Versiones_Producto_OH/OH_v4/OH_BPM_v[VERSION]/bbdd/totales/mysql/1-ohbpm-database.sql

[FTP_SERVER]
/oradata/Versiones_Producto_OH/OH_v4/OH_BPM_v[VERSION]/bbdd/totales/mysql/2-ohbpm-user.sql

- Forms Builder

[FTP_SERVER]
/oradata/Versiones_Producto_OH/OH_v4/OH_GEN_v[VERSION]/bbdd/mysql/totales/01-user_mysql_ddl.sql

Oracle con acceso de administrador

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

- Process Manager

[FTP_SERVER]
/oradata/Versiones_Producto_OH/OH_v4/OH_BPM_v[VERSION]/bbdd/totales/oracle/1-ohbpm-database.sql

[FTP_SERVER]
/oradata/Versiones_Producto_OH/OH_v4/OH_BPM_v[VERSION]/bbdd/totales/oracle/2-ohbpm-user.sql

- Forms Builder

[FTP_SERVER]
/oradata/Versiones_Producto_OH/OH_v4/OH_GEN_v[VERSION]/bbdd/oracle/totales/01-user_oracle_ddl.sql

Oracle o MySQL 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

8.3.1. MySQL

- Process Manager

Con el usuario propio del módulo (us_ohbpm) lanzar los siguientes scripts:

[FTP_SERVER]
/oradata/Versiones_Producto_OH/OH_v4/OH_BPM_v[VERSION]/bbdd/totales/mysql/3-ohbpm-camunda-engine.sql

[FTP_SERVER]
/oradata/Versiones_Producto_OH/OH_v4/OH_BPM_v[VERSION]/bbdd/totales/mysql/4-ohbpm-tables.sql

- Forms Builder

Con el usuario propio del módulo (us_hnfor) lanzar los siguientes scripts:

[FTP_SERVER]
/oradata/Versiones_Producto_OH/OH_v4/OH_GEN_v[VERSION]/bbdd/mysql/totales/02-tablas_mysql_ddl.sql

[FTP_SERVER]
/oradata/Versiones_Producto_OH/OH_v4/OH_GEN_v[VERSION]/bbdd/mysql/totales/03-ini_datos_mysql_dml.sql

[FTP_SERVER]
/oradata/Versiones_Producto_OH/OH_v4/OH_GEN_v[VERSION]/bbdd/mysql/totales/04-funciones_mysql_ddl.sql

8.3.2. Oracle

- Process Manager

Con el usuario propio del módulo lanzar los siguientes scripts:

[FTP_SERVER]
/oradata/Versiones_Producto_OH/OH_v4/OH_BPM_v[VERSION]/bbdd/totales/oracle/3-ohbpm-camunda-engine.sql

[FTP_SERVER]
/oradata/Versiones_Producto_OH/OH_v4/OH_BPM_v[VERSION]/bbdd/totales/oracle/4-ohbpm-tables.sql

- Forms Builder

[FTP_SERVER] /oradata/Versiones_Producto_OH/OH_v4/OH_GEN_v[VERSION]/bbdd/oracle/totales/02-tablas_oracle_ddl.sql

[FTP_SERVER] /oradata/Versiones_Producto_OH/OH_v4/OH_GEN_v[VERSION]/bbdd/oracle/totales/03-ini_datos_oracle_dml.sql

[FTP_SERVER] /oradata/Versiones_Producto_OH/OH_v4/OH_GEN_v[VERSION]/bbdd/oracle/totales/04-_funciones_oracle_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.

NOTA: Se tiene que instalar la versión de OH_BPM que viene en el anexo I. Para ello tendremos que modificar el fichero "values_bpm.yaml" poniendo las versiones que se muestran en las columnas Images Values:

Proccess	4.2.0	OH_BPM	onesaithealthcare-ohbpm-chart.ohbpm.version.image_tag: 4.13.0
		OH_GEN	onesaithealthcare-ohgen-chart.ohgen.version.image_tag: 4.13.0
		OH_PRM	onesaithealthcare-ohprm-chart.ohprm.version.image_tag: 4.13.0

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 u Oracle.
- **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, para Oracle sería: jdbc:oracle:thin:@database_host:port:SID).

  Para Oracle, si no se tiene acceso al SID, se debería de cambiar el valor SID por el SERVICE (jdbc:oracle:thin:@database_host:port/SERVICE**)**

- **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, para Oracle sería: jdbc:oracle:thin:@database_host:port:SID).

  Para Oracle, si no se tiene acceso al SID, se debería de cambiar el valor SID por el SERVICE (jdbc:oracle:thin:@database_host:port/SERVICE**)**

- **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)

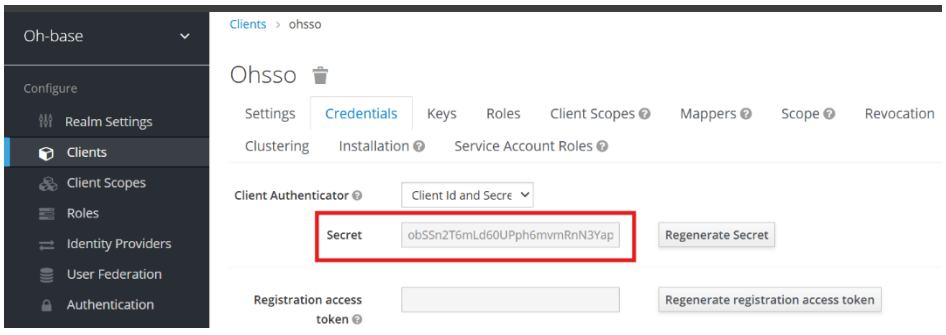
Modificar el configmap **ohbpm (ohbpm-back-cfg)**, cambiar el login y clave del usuario offline creado antes de la instalación en el keycloack. **SE DEBE PEGAR CODIFICADO EN BASE64 (USAR LA PÁGINA** https://www.base64encode.org/ **)**. Se haría con los comandos:

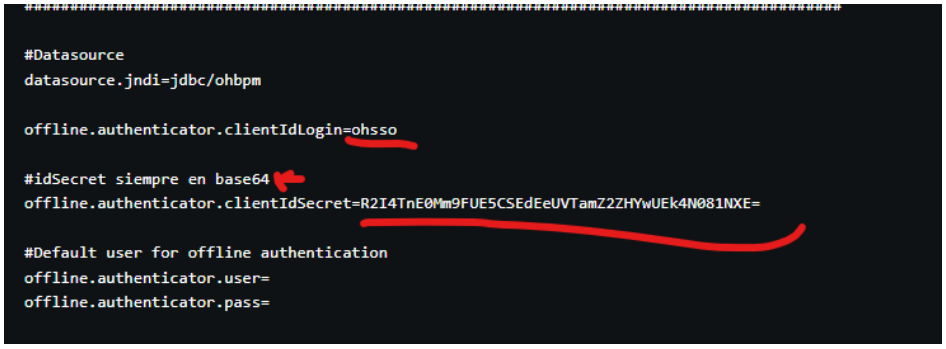
kubectl create secret

```
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_SERVER] /oradata/Versiones_Producto_OH/OH_v4/OH_BPM_v[VERSION]/src/totales** los siguientes archivos de configuración:

- **OHCON_OHBPM_prop.csv** que deberá importarse en la sección de configuración.
- **OHCON_OHBPM_WL_.csv que** deberá importarse en la sección de listas de trabajo.

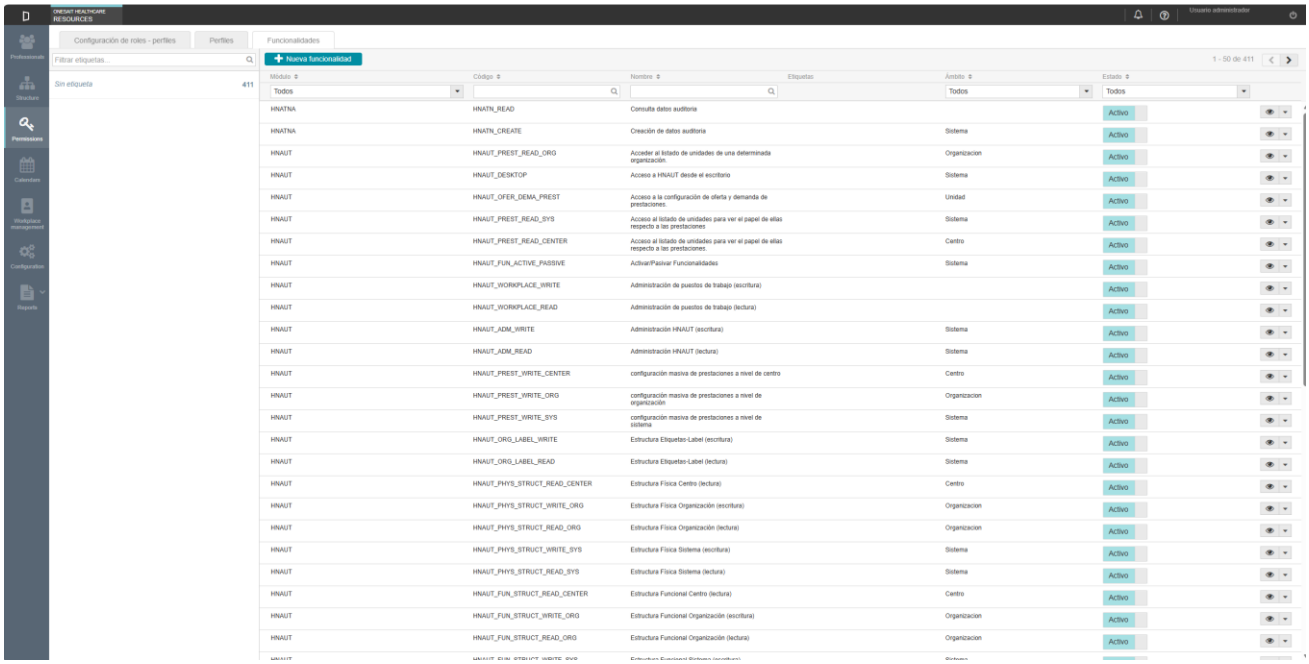
Configuración en módulo Ontology

Se adjunta en la carpeta **/oradata/Versiones_Producto_OH/OH_v4/OH_BPM_v[VERSION]/src/totales** los siguientes archivos de catálogos (si existiera algún fichero json más, importarlo también dado que se tratan de incrementales).

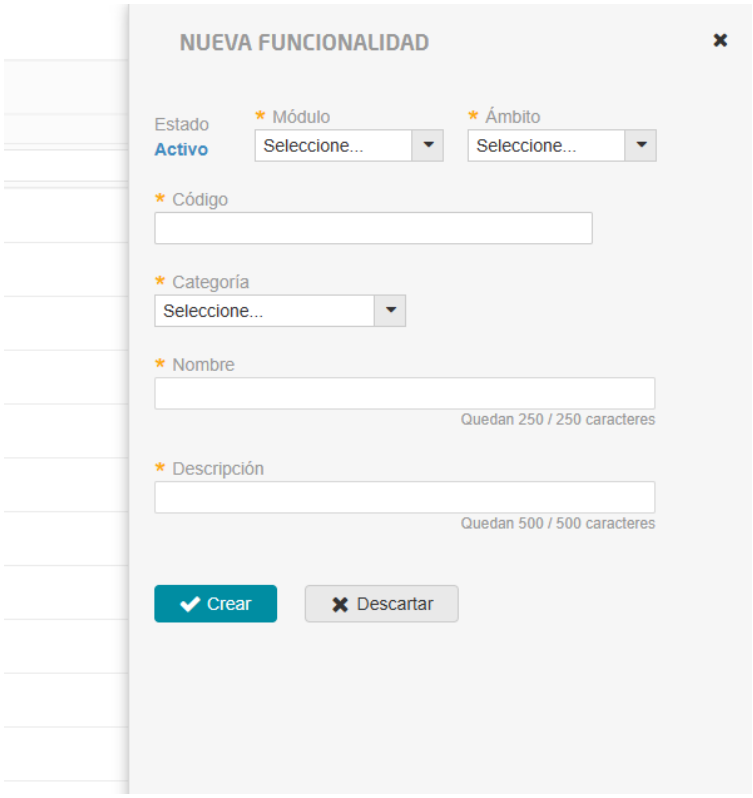
- **OHONT_BPM_ActivityFilters.json** filtros para las actividades de la Smart.
- **OHONT_BPM_Exclusion_Reason.json** motivos de exclusión.
- **OHONT_BPM_Instantiation_Reason.json** motivos de instanciación.
- **OHONT_BPM_KPIs.json** nombre de los kpi a controlar.
- **OHONT_BPM_Pause_Reason.json** motivos 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 instanciación.
- **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.



Pulsando “nueva funcionalidad” aparecerá un formulario en el que se deben introducir los datos que se indican a continuación.



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_SERVER] /oradata/Versiones_Producto_OH/OH_v4/OH_PRM_v[VERSION]/src/totales** los siguientes archivos de configuración:

- **OHCON_OHPRM_porp.csv** que deberá importarse en la sección de configuración.
- **OHCON_OHPRM_wl.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_SERVER] /oradata/Versiones_Producto_OH/OH_v4/OH_GEN_v[VERSION]/src/totales** los siguientes archivos de configuración:

- **OHCON_HNFORM_prop.csv** que deberá importarse en la sección de configuración.
- **OHCON_HNFORM_WL.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_v4/OH_GEN_v[VERSION]/src/totales** los siguientes archivos de catálogos (si existiera algún fichero json más, importarlo también dado que 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, como se ha mostrado en el epígrafe 8.5.1.

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

9. Guías de configuración adicionales

9.1. Configuración de persistencia sobre NFS

Para los módulos que requieran persistencia en sus guías de instalación se indica como prerrequisito disponer de algún Storage Class con un driver con provisión dinámica.

En caso de no disponer de dicho driver o para entornos de desarrollo se puede optar por crear manualmente los Persistent Volumes necesarios. Una opción para realizar esta tarea sería disponiendo de un servidor NFS.

9.1.1. Prerrequisitos

- Disponer de un servidor NFS accesible desde los nodos del cluster kubernetes.
- El sistema operativo de los nodos worker del cluster debe tener instalado soporte para poder montar volúmenes NFS.

9.1.2. Pasos para creación manual de PV sobre NFS

En primer lugar se crea un Storage Class de esta forma (cambiar el nombre test-nfs por el que se requiera):

```
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: test-nfs-sc
provisioner: kubernetes.io/no-provisioner
reclaimPolicy: Retain
volumeBindingMode: WaitForFirstConsumer
```

Code block 1 StorageClass test-nfs-sc

Creamos el PersistentVolume de la siguiente forma:

```
kind: PersistentVolume
apiVersion: v1
metadata:
  name: test-nfs-pv
spec:
  capacity:
    storage: 1Mi
  mountOptions:
    - port=12049
  nfs:
    server: 192.168.146.153
    path: /test_pod
  accessModes:
    - ReadWriteOnce
  persistentVolumeReclaimPolicy: Retain
  storageClassName: test-nfs-sc
  volumeMode: Filesystem
```

Code block 2 PersistentVolume test-nfs-pv

Se cambiarán los siguientes campos:

- metadata.name: cambiar al nombre con el que se prefiera crear el PV (normalmente contendrá el nombre del módulo al que se asocia).
- spec.capacity.storage: tamaño en megabytes (Mi) o gigabytes (Gi) que se limita al PV
- spec.mountOptions.port: este campo sólo se debe indicar si el servidor NFS no está expuesto en el puerto estandar de dicho sevicio (2049).
- spec.nfs.server: dirección ip o host del servidor NFS
- spec.nfs.path: path del servidor NFS (a partir de la raíz que expone) que se asocia a este volumen, el path debe crearse previamente en el servidor NFS
- spec.storageClassName: nombre del StorageClass que creamos en el paso anterior

IMPORTANTE: si el módulo a desplegar va a tener varias instancias (como por ejemplo el cluster kafka) se debe crear un PersistentVolume por cada instancia que se quiera desplegar, y **el path de cada PersistentVolumen tiene que ser distinto** (por ejemplo kafka0, kafka1, kafka2).

Los módulos que despliegan con charts en caso de requerir persistencia ya deberían incluir en sus Deployment o StatefulSet las referencias a StorageClass y deberían solicitar como parámetro de instalación el nombre del StorageClass al que asociarse.

A partir de aquí sería una guía los desarrolladores de módulos que no tuvieran definida la configuración de persistencia en sus charts. Se presentan 2 opciones, crear manualmente los PersistentVolumeClaim y referenciarlos en los volumes del Deployment o definir en el Deployment o StatefulSet un volumeClaimTemplate (en este caso los PVC se crearán automáticamente asignándole los volúmenes que encuentre con su storageClass).

Opción 1: creación manual de PersistentVolumeClaim

Creamos el PersistentVolumeClaim de la siguiente forma:

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: test-nfs-pvc
  namespace: oh-modules
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 1Mi
  storageClassName: test-nfs-sc
  volumeMode: Filesystem
```

Code block 3 PersistentVolumeClaim test-nfs-pvc

Se cambiarán los siguientes campos:

- metadata.name: cambiar al nombre con el que se prefiera crear el PVC (normalmente contendrá el nombre del módulo al que se asocia).
- metadata.namespace: namespace donde se despliega el módulo al que se asociará este PVC
- spec.resources.requests.storage: tamaño en megabytes (Mi) o gigabytes (Gi) que se limita al PVC
- spec.storageClassName: nombre del StorageClass que creamos en el paso anterior

Finalmente se asociaría el PV al Deployment añadiendo lo siguiente en las secciones volumeMounts y volumes (cambiar a los nombres con los que se hayan creado los recursos y el mount path donde se deba montar al módulo):

```
'''

          volumeMounts:
            - mountPath: /tmp/test-nfs
              name: test-nfs-pvc

...

      volumes:
        - name: test-nfs-pvc
          persistentVolumeClaim:
            claimName: test-nfs-pvc

...
```

Code block 4 Deployment

Opción 2: definición de volumeClaimTemplate

En el Deployment o StatefulSet añadiríamos las siguientes secciones (cambiar a los nombres con los que se hayan creado los recursos y el mount path donde se deba montar al módulo):

```
'''

          volumeMounts:
            - mountPath: /tmp/test-nfs
              name: test-nfs

...

  volumeClaimTemplates:
    - apiVersion: v1
      kind: PersistentVolumeClaim
      metadata:
        name: test-nfs
      spec:
        accessModes:
          - ReadWriteOnce
        resources:
          requests:
            storage: 1Gi
        storageClassName: test-nfs-sc
        volumeMode: Filesystem

...
```

Code block 5 volumeClaimTemplates

Se cambiarán los siguientes campos:

- volumeClaimTemplates.metadata.name: cambiar al nombre con el que se prefiera crear el volumen (normalmente contendrá el nombre del módulo al que se asocia).
- volumeClaimTemplates,spec.resources.requests.storage: tamaño en megabytes (Mi) o gigabytes (Gi) que se limita al volumen
- volumeClaimTemplates.spec.storageClassName: nombre del StorageClass que creamos en el paso anterior

Esto creará automáticamente tantos PVCs como instancias se levanten, para el caso anterior se generarían con los nombres: test-nfs-0, test-nfs-1, test-nfs-2, ...

9.2. 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 la guía "00.0 Requisitos de instalación"

Esta configuración NO ESTÁ RECOMENDADA para entornos PRODUCTIVOS.

- Preparación del nuevo namespace
- OPCION 1 - Configuración del Gateway HTTP en el nuevo namespace
- OPCION 2 - Configuración del Gateway HTTPS en el nuevo namespace
- Configuración DNS para el nuevo entorno
- Related articles

9.2.1. 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
```

Code block 6 kubectl create namespace

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

```
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-develop
```

Code block 7 kubectl create secret

9.2.2. 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
```

Code block 8 oh-modules-gateway

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
```

Code block 9 oh-modules-gateway

Creamos el Gateway

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

Code block 10 kubectl apply

Comprobamos que se ha creado el Gateway

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

Code block 11 kubectl get gateways

9.2.3. 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 lde 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
```

Code block 12 oh-modules-gateway

Creamos el Gateway

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

Code block 13 kubectl apply

Comprobamos que se ha creado el Gateway

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

Code block 14 kubectl get gateways

9.2.4. 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.

Anexo I

Las versiones del chart, del módulo y de las diferentes imágenes que se tienen que instalar, se encuentran en el fichero "*Versiones.xlsx*", que se incluye en la ruta:

(FTP_SERVER) /oradata/Versiones_Producto_OH/Versiones.xlsx

Anexo II

Os proporcionamos una tabla con las urls de los módulos que a los cuales se tiene que acceder de manera directa:

MODULO	URL
OHSSO	<dominio principal>/auth
OHCONF	<dominio principal>/hnconf
OHONT	<dominio principal>/hncat
OHAUT	<dominio principal>/hnaut
Prometheus	<dominio principal>/prometheus
Grafana	<dominio principal>/grafana