



Documentación Técnica
GSDPI - AVIB
v 1.0.0

Copyright

This document is Copyright © 2024 by its contributors as listed below. You may distribute it and/or modify it under the terms of either the GNU General Public License (<http://www.gnu.org/licenses/gpl.html>), version 3 or later, or the Creative Commons Attribution License (<http://creativecommons.org/licenses/by/3.0/>), version 3.0 or later.

All trademarks within this guide belong to their legitimate owners.

Colaboradores

Miguel Salinas Gancedo: versión español

Realimentación

Por favor, dirija cualquier comentario o sugerencia sobre este documento a:
salinasmiguel@uniovi.es

Fecha de publicación y versión del software

Publicado 3 Octubre 2024. Basado en AVIB versión 0.0.1.SNAPSHOT.

Contenidos

Copyright.....	2
Introducción.....	4
Análisis.....	5
Dominio del problema.....	6
Metodología Agile: Azure Devops.....	7
Seguridad del sistema.....	9
Despliegue del servicio.....	9
Configuración del servicio.....	9
<u>Integrar servicios</u>	9
Datasets y su estructura.....	12
Encodings: tipos.....	13
Pipeline de los datos.....	14
Arquitectura.....	15
Introducción.....	16
Tech Stack.....	18
TechStack: Backend.....	18
TechStack: Frontend.....	19
<u>TechStack: Infraestructura</u>	21
Microservicios.....	24
Flujo de Proyección.....	26
Diagrama de despliegue.....	30
Código.....	32
Repositorios de código.....	41
Operaciones.....	43
<u>Despliegue infraestructura kubernetes</u>	44
Introducción.....	44
Instalación del cluster de Kubernetes.....	45
Instalación gestor paquetes de Kubernetes.....	53
Introducción.....	53
Instalación del CLI de Helm.....	53
Despliegue servicio Database: MongoDB.....	55
Despliegue servicio IAM: keycloak.....	58
Despliegue servicio Object Storage: Minio.....	62
Configuraciones Kubernetes post-despliegue.....	66
<u>Configuración reverse-proxy: HAProxy</u>	68
Introducción.....	68
Configuración kubernetes ingress.....	69
Reglas de ingress.....	70
Despliegue servicios de negocio.....	74
Introducción.....	74
Pasos a seguir en el despliegue servicios negocio backend Java.....	74
<u>Pasos a seguir en el despliegue servicios negocio frontend Java</u>	79
Acceso al Service Registry de Azure.....	82
Acceso local a Azure.....	82

Introducción

Este es un documento técnico destinado a todo usuario con rol de Arquitectos o Desarrollador que tenga que comprender las diferentes partes en que está diseñado el sistema. Igualmente todo usuario que tenga que mantener y monitorizar el mismo con rol tipo devops también le será de gran ayuda.

Este documento está dividido en cuatro grandes secciones:

1. **Análisis:** en donde se intenta explicar el dominio del problema que quiere resolver este sistema.
2. **Arquitectura:** en esta sección se describen con texto y diagramas los frameworks y librerías escogidos así como están entre si relacionados y se comunican entre sí, para resolver el dominio del problema que nos ocupa.
3. **Código:** en esta sección se enumeran los repositorios de código, buenas prácticas a la hora de desarrollar cada una de las piezas que forman los servicios de negocio que resuelven el dominio de problema antes indicado.
4. **Despliegue:** por último en esta sección se explica en detalle, como se debe de desplegar un sistema distribuido como este, indicando los pasos a seguir en caso de tener que desplegarlo desde cero.

Análisis

Vamos a explicar en esta sección que dominio de problema queremos solventar y que metodología herramientas hemos utilizado para implementar esta solución

Dominio del problema

Antes de empezar a analizar como queremos implementar esta solución, debemos de comprender que problema queremos resolver.

Actualmente el departamento GSDPI ha desarrollado durante años diferentes proyectos con un denominador común, ser capaces de analizar gran cantidad de datos visualmente utilizando la técnica de **Morphing Projections**. Esta técnica permite poder encontrar patrones de comportamiento en grandes Datasets de alta dimensionalidad, empleando para ello técnica de Inteligencia Artificial que permiten reducir esta alta dimensionalidad a 2 o tres dimensiones capaces de ser dibujadas en un canvas de 2 o tres dimensiones. Estas proyecciones en este canvas junto a técnicas visuales de color y movimiento gracias a esta técnica de Morphing o moviendo de estos puntos en el espacio, permiten que el analista pueda encontrar patrones de comportamiento en donde las técnicas numéricas no son capaces de mostrarnos con tanta claridad este comportamiento.

Por lo tanto una necesidad surgida de este desarrollo era como resolver un problema como este de forma escalar, dinámica y adaptada a cualquier dataset procedente de cualquier dominio: industrial, salud, marketing, etc.

El proceso necesario a la hora de adaptar las aplicaciones individuales de ingestar los datasets y visualizar el resultado de los mismos, es costoso en tiempo. Ademas la solución debiera de poder ser configurable y explotable en entornos Web fácilmente accesible desde cualquier navegador actual. Ademas debiéramos de tener en cuenta la escalabilidad y el uso por diferentes usuarios al mismo tiempo, cada uno de ellos con un problema diferentes de un espacio distinto.

El análisis de todos estos requisitos han hecho posible el desarrollo de este sistema teniendo en mente estos puntos:

- Sistema amigable y fácilmente accesible vía Web
- Sistema configurable a cualquier dominio, creando un standard que se capaz de adaptarse a cualquier dominio.
- Un sistema escalable que pueda crecer según las necesidades
- Un sistema basado en datos, teniendo este concepto como piedra angular al rededor del cual diseñar todo el sistema.
- Un sistema resiliente capaz de mantener la integridad en todo momento.
- Un sistema seguro, por se consumido a través de internet, un canal no seguro a todas luces.
- Siguiendo los standares actuales del mercado, a nivel de diseño, arquitectura e implementación
- Utilizando herramientas OpenSource que sean mantenidas por la comunidad o empresas, dandonos una seguridad de su continuidad y soporte durante años.

Metodología Agile: Azure Devops

A la hora de desarrollar una herramienta como esta de carácter distribuida, debemos en primer lugar seleccionar el entorno y metodología para su análisis, desarrollo y despliegue.

Para resolver este primer dilema, se ha seleccionado la herramienta de Microsoft llamada Azure Devops. Esta plataforma de Microsoft cumple con todas las condiciones necesarias a la hora de abordar un proyecto como este:

1. Herramienta mantenida y soportada por Microsoft, lo que nos da la seguridad de su continuidad y soporte actual y futura.
2. Es una herramienta segura que está perfectamente integrada con el SSO de Microsoft de la Universidad de Oviedo, por lo que podemos utilizar nuestras mismas credenciales que actualmente nos definen en la Universidad de Oviedo, no teniendo que crear nuevas credenciales para ello.
3. Es una herramienta que está formada por varios módulos integrados, que no utilizaremos en su totalidad pero que si permiten crear un seguimiento y control perfecto de un proyecto de la embergadura como este:
 - **Wiki:** Este módulo representa la parte documental del sistema, en donde podemos crear vistas y documentación digital fácilmente distribuible y mantenible. La división de esta documentación sigue la misma distribución que la de este documento.
 - **Board:** este módulo gestor de tareas y seguimiento del proyecto siguiendo la metodología agile. Este módulo permite crear un backlog de tareas, ordenarlas y ejecutarlas de forma controlada. Igualmente permite el seguimiento de las mismas, así como su evolución por medio de listados, kambas, diagramas y KPIs.
 - **Repos:** Esta sección es donde crearemos todos los repositorios de código de todos los servicios de negocio del sistema. Esta sección es un gestor de versiones distribuido como puede ser Github o Gitlab.
 - **Pipelines:** este módulo en principio no está siendo utilizado, pero representa aquella parte del sistema en donde podemos implementar los pipelines CI/CD de integración y despliegue del mismo de forma automática. Actualmente y como se explicará en otros capítulos, las etapas a la hora de integrar y desplegar nuestros servicios será manual siguiendo unos pasos detallados que veremos en próximos capítulos.
 - **Test Plan:** es otro módulo que no utilizaremos y sirve para diseñar pruebas de integración de nuestro sistema, para poder mantener un nivel de calidad continua. En nuestro caso estas pruebas unitarias y de integración se hacen manualmente en tiempo de desarrollo

- **Artefactos:** este es una módulo que no utilizaremos tampoco, pero que a futuro podría ser interesante a la hora de mantener una cache controlada para todas las dependencias, que son muchas, utilizadas por todos nuestros servicios de backend y frontend.

A todas estas herramientas debemos de añadir aquellas que son ofrecidas por la infraestructura de Azure, no confundir con Azure Devops. Actualmente la infraestructura de Azure es muy grande ofreciendo todo tipo de servicios, orientados a la ejecución, monitoreo, redes y mucho mas. De todos los servicios ofrecidos por Azure solamente utilizaremos uno, que es el **gestor privado de imágenes de Docker**, pues como iremos viendo en otros capítulos todos los servicios del sistema tienen en común ser contenedores de docker, por lo que todos ellos están empaquetados como imágenes de Docker que deben de ser almacenadas y gestionadas por repositorios especiales. Azure ofrece este tipo de repositorios y será el que utilizaremos para almacenar nuestras imágenes de docker.

Seguridad del sistema

La seguridad como es lógico, es una apartado muy importante a la hora de implementar cualquier solución software. Nuestro caso no es menos y por ello debemos de tener muy en cuenta este apartado, mas si cabe, ya que la solución va a estar servida pública servida desde Internet.

En nuestro caso hemos querido seguir los estándares de seguridad y especificaciones seguidas en el mercado actualmente.

Actualmente la especificación seguida por el mercado es Oauth 2.1, implementada y mantenida por los grandes del mercado tecnológico como: Microsoft, Meta, X, Redhat, Amazon o Google entre otros.

Actualmente el sistema OpenSource mejor del mercado mantenido por Red-Hat se llama **Keycloak**, en su vertiente comercial y RH-SSO en su vertiente comercial disponible en OpenShift.

Esta herramienta implementa todo la especificación Oauth y al mismo tiempo existen varias módulos y librerías que se pueden integrar en tus herramientas. En nuestro caso con el Portal implementado en Angular.

Lo primero que tenemos que hacer es lógicamente desplegar el servicio de Keycloak dentro del cluster de Kubernetes, para después configurarlo correctamente para poder trabajar con nuestros servicios y recursos particulares.

La herramienta Keycloak es compleja y extensa, pero nosotros solo utilizaremos algunos recursos para poder trabajar de forma segura con nuestros servicios.

Despliegue del servicio

Para saber más como desplegar la herramienta, se puede consultar los puntos relacionados con el despliegue y los README.md dentro del repositorio de deployment del proyecto, en donde se especifican los comandos a ejecutar.

Configuración del servicio

Igualmente en el capítulo de despliegue se explica con detalle como configurar el servicio de Keycloak una vez este ha sido desplegado.

Integrar servicios

Una vez ya tenemos nuestro servicio Keycloak corriendo en el Cluster y configurado correctamente. Solo debemos de integrar el servicio de portal para que este se integre con Keycloak: Las recomendaciones de Keycloak es que no implementemos por nosotros mismo el login del sistema, que será aquella vista que los usuarios utilizarán para autenticarse en el sistema. Nosotros hemos caso a estas

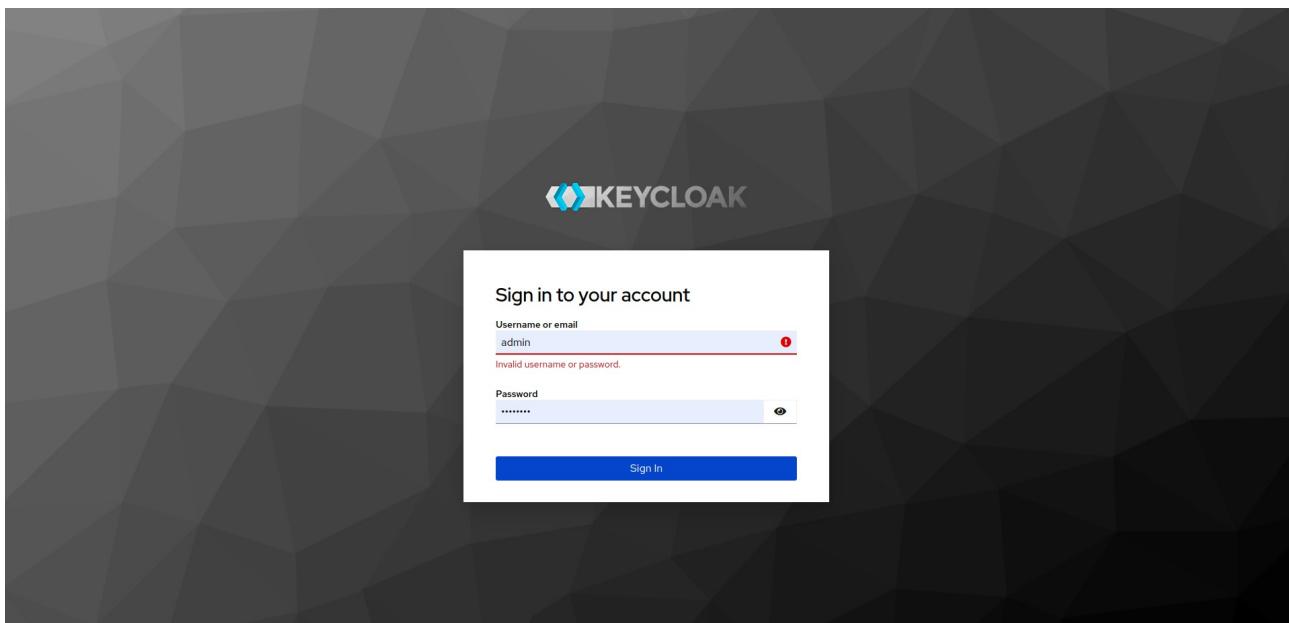
recomendaciones y hemos delegado el uso de esta vista a Keycloak. Por lo que la vista de login que nos aparecerá cada vez que tengamos que logearnos en el sistema será la que ofrezca Keycloak. Esta vista sigue los más altos niveles de calidad de código respecto a la seguridad, implementado y controlado por Red-hat.

Actualmente Keycloak está configurado para:

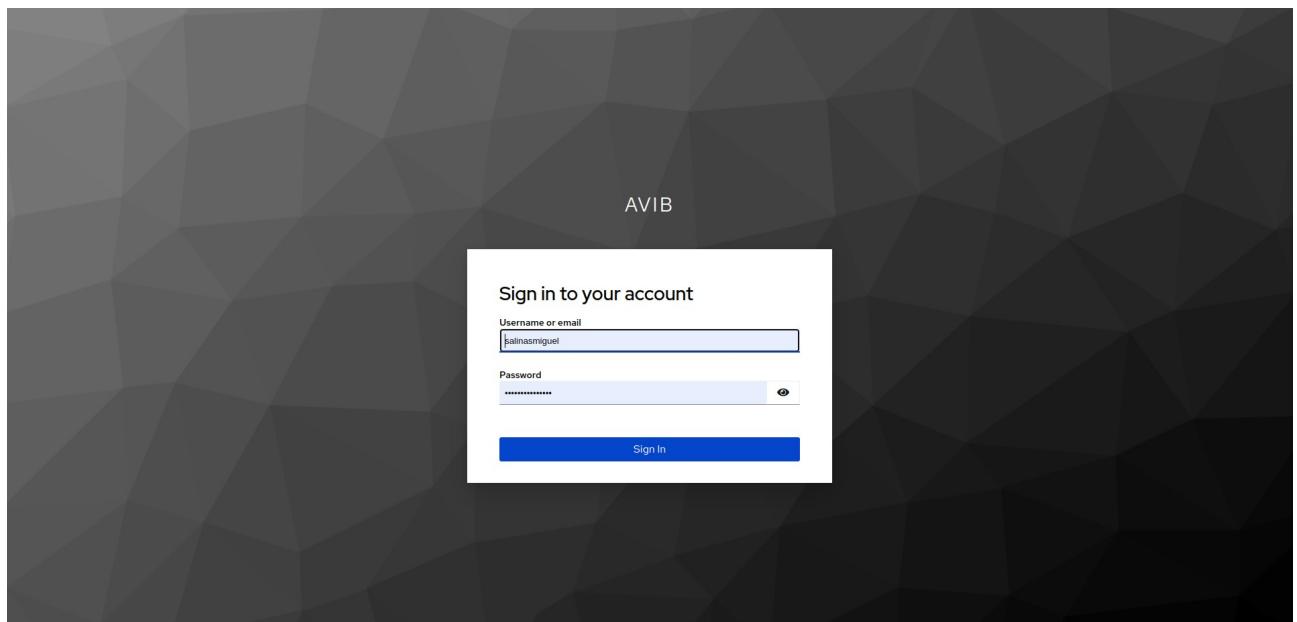
- Guardas nuestras cuentas en PostgreSQL.
- Se ha creado un realm propio para el sistema con el fin de agrupar todos los recursos de seguridad propios del mismo: usuarios, roles y clientes y políticas de seguridad: password, CORS, frame access, etc.
- Política de hashing de password sha256, de alta seguridad.
- CORS configurado para solo dar acceso desde el dominio avispe.uniovi.es
- Paquete de acceso desde eframes solamente al dominio avispe.uniovi.es

Para acceder al Admin Console de Keycloak, que es el portal de administración del mismo, la URI es: <https://avise.uniovi.es>

- Pantalla de acceso a Admin Console de keycloak:



- Pantalla de acceso a AVIB Portal



Datasets y su estructura

Introducción

Vamos a explicar y a desarrollar el análisis de la estructura que deben de tener todos los datasets de ingesta, para que podamos proyectar cualquiera de ellos independientemente del contexto asociado: industria, salud, finanzas, etc:

Este diagrama expone la arquitectura de estos datasets de forma normalizada:

		ATTRIBUTES				SAMPLE ANNOTATIONS		
		A1	A2	Am	SA1	SAq
SAMPLES	S1	e11	e21	em1	s11	sq1
	S2	e12	e22	em2	s12	sq2

	Sn	e1n	e2n	emn	s1n	sqn
	AA1	a11	a21	am1			
ATTRIBUTE ANNOTATIONS		
	AAp	a1p	a2p	amp			

Podemos decir, que como máximo pueden existir 5 ficheros de ingesta que definan por completo nuestro caso:

- **Datamatrix:** representa una **matriz de n Muestras y m Atributos de valores numéricos** (separador punto) que representan el estado de cada uno de estos atributos para cada una de las muestras con las que contemos. **Estas matrices deben de tener muestras identificadas de forma única a través de un atributo llamado sample_id. Igualmente cada atributo de cada muestra debe ser identificado de forma única.**

- **Sample Annotation:** es una **matriz de n Muestras y q Anotaciones**, que representa todos los metadatos asociados a cada una de nuestras muestras. Estas anotaciones pueden ser de tipo:
 - **string:** código alfanumerico
 - **enumerado:** lista de string acotada
 - **numérico:** enteros o flotantes (separador punto)
 - **fecha:** dd/mm/yyyy
 - **booleano:** true, false
- **Attribute Annotations:** es una **matriz de m Atributos y p Anotaciones**, que representa todos los metadatos asociados a cada uno de nuestros atributos. Igualmente estas anotaciones pueden ser de tipo:
 - **string:** código alfanumerico
 - **enumerado:** lista de string acotada
 - **numérico:** enteros o flotantes (separador punto)
 - **fecha:** dd/mm/yyyy
 - **booleano:** true, false
- **Sample Annotations Precalculated:** es una **matriz de n Muestras y 2r (x,y) proyecciones precalculadas**, para cada una de las muestras, lógicamente deben de ser números (separador punto)
- **Attribute Annotations Precalculated:** es una **matriz de m Atributos y 2s (x,y) proyecciones precalculadas** para cada uno de los atributos, para cada una de las muestras, lógicamente deben de ser números (separador punto)

Estas últimas matrices las precalculadas no aparecen, pues son opcionales y solo serán ingestadas si las proyecciones del sistema ya han sido calculadas externamente al sistema, no teniendo que ser proyectadas por ningún Job, sino que el resultado ya se da por bueno y puede ser visualizado ya por el sistema.

Actualmente el sistema cuenta con repositorio capaz de crear ejemplos aleatorios con un número de muestras, atributos y anotaciones aleatorio y configurable. Este repositorio se llama: **uniovi-avib-morphingprojections-dataset-generator**. Es un script en Python al que se le pueden pasar como argumentos estos valores. Vamos a ver como ejemplo de ficheros este ejemplo.

Un ejemplo para entender la especificación:

Este es un ejemplo con un datamatrix de 100 muestras y 5 atributos cada una de ellas. Estos atributos son:

A1	A	B	C	D	E	F	G
1	sample_id	A1	A2	A3	A4	A5	
2	S1	0.8276283698788387	0.019057011136434987	0.46585279762165444	0.9236208505778929	0.11818480043985657	
3	S2	0.2811067810554566	0.5065895353926222	0.8624519701506427	0.6589115811251005	0.6352519973898256	
4	S3	0.08419974777018902	0.9723497325397383	0.7703346817569983	0.23367282989114624	0.1458469755275239	
5	S4	0.5848580389952914	0.8045370236421836	0.1962043527331443	0.5293367793415696	0.42384105115702886	
6	S5	0.6387694725983045	0.14009401342670136	0.5264733066862978	0.2644547729684438	0.8306248597698743	
7	S6	0.5473811123760557	0.29072007260340427	0.5805340833902929	0.3379883047293244	0.26739547773836825	
8	S7	0.27230002450933466	0.9073628763346864	0.6369271450848922	0.0218634933867784	0.34734235960729265	
9	S8	0.5822496532202024	0.874243491004269	0.007512511649650722	0.78626479368616	0.02842934256554941	
10	S9	0.6648456042845119	0.6003599668994563	0.9874296165990819	0.47067222077788706	0.4340175389871527	
11	S10	0.6514706697069692	0.5081329302598018	0.27994516521467927	0.18051297607918504	0.5585043661893486	
12	S11	0.22120112390722846	0.17156304362266317	0.5675216817045298	0.7599034211675749	0.7126193078825264	
13	S12	0.48259955427410106	0.6779587432200155	0.6760557757631843	0.5601013995190497	0.3472963396528058	
14	S13	0.8184078774220603	0.301487351863183	0.5102342915885675	0.40927305842909234	0.06471515593581201	
15	S14	0.09346802677582011	0.6155901913751622	0.05367702371242977	0.4281554170636891	0.35525013874863187	

Este ejemplo cuenta con un colección de 100 muestras y 4 anotaciones por cada una de ellas una para el identificador de cada muestra, sample_id y cuatro restantes una por muestra con estos tipos:

- **SA1:** enumerado
- **SA2:** string
- **SA3:** enumerado
- **SA4:** numérico

A1	A	B	C	D	E	F	G
1	sample_id	SA1	SA2	SA3	SA4		
2	S1	V3_SA1	jsMWIIIfY	V1_SA3	0.022449564178882886		
3	S2	V3_SA1	IYdgHdCl	V2_SA3	0.05995981417635299		
4	S3	V3_SA1	VRzMtAhJ	V2_SA3	0.6052806806526674		
5	S4	V2_SA1	TEjYTMxJ	V2_SA3	0.5942079166543317		
6	S5	V1_SA1	BlzhTEHx	V1_SA3	0.548938008867065		
7	S6	V3_SA1	ZBaMdvg	V2_SA3	0.7916437479773124		
8	S7	V3_SA1	tdReLhLZ	V2_SA3	0.4641359301233092		
9	S8	V3_SA1	dgSamfKr	V1_SA3	0.2624218191381503		
10	S9	V3_SA1	MFvQKszZ	V1_SA3	0.12466342478528558		
11	S10	V2_SA1	tGxiEZMt	V1_SA3	0.5241411057131129		
12	S11	V2_SA1	gFmskRfJ	V2_SA3	0.2984583520192535		
13	S12	V2_SA1	vEYtmISV	V2_SA3	0.31784513883484544		
14	S13	V2_SA1	FTxpKgfL	V1_SA3	0.8032517325806493		
15	S14	V3_SA1	YioZmiOj	V2_SA3	0.05698554660105093		
16	S15	V1_SA1	NrQIKxML	V1_SA3	0.10411331909872956		
17	S16	V3_SA1	vZcovCWD	V1_SA3	0.34708075723010845		
18	S17	V2_SA1	YqUkfKGj	V1_SA3	0.08957410769959473		
19	S18	V3_SA1	cQOVcPdl	V1_SA3	0.14371349357691632		
20	S19	V2_SA1	nAvcrSft	V2_SA3	0.8242456763875635		

Igualmente este ejemplo cuenta con un fichero de anotaciones de atributos como sigue:

A1	A	B	C	D
1	attribute_id	AA1	AA2	
2	A1	V1_AA1	V1_AA2	
3	A2	V2_AA1	V3_AA2	
4	A3	V2_AA1	V2_AA2	
5	A4	V1_AA1	V1_AA2	
6	A5	V2_AA1	V4_AA2	
7				
8				

En este caso como solo contamos con 5 atributos por muestra tenemos cinco filas y en concreto cada atributo cuenta con 2 anotaciones en este ejemplo:

- AA1: de tipo enumerado
- AA2: de tipo enumerado

Este ejemplo también cuenta con valores precalculados, contamos con un fichero de valores precalculados para las muestras, por lo que el fichero será de este tipo. 100 filas una por muestra precalculada y solamente una proyección (encoding) 2 columnas una para la x y la otra para la y llamada SA_PRECAL. Todos los valores deben ser numéricos como es lógico.

A1	Bx	Cy	D	E	F
sample_id	x_SA_PRECAL	y_SA_PRECAL			
1 S1	0.946842470995339	0.624951489005514			
2 S2	0.265208611432022	0.695583075283825			
3 S3	0.278565487518169	0.129967080581966			
4 S4	0.584493803979731	0.171890600559406			
5 S5	0.138612277442548	0.420732590635496			
6 S6	0.320088171893749	0.034531889049786			
7 S7	0.014974959355702	0.019004343648884			
8 S8	0.318900332656997	0.925999070325055			
9 S9	0.193662421018199	0.400474519842865			
10 S10	0.181167813876028	0.246226399651079			
11 S11	0.321404876425768	0.882013200846849			
12 S12	0.313698712991031	0.662791050307102			
13 S13	0.542384976891982	0.792745997904896			
14 S14	0.994980818608406	0.057084871556268			
15 S15	0.47278495726037	0.573903659612536			
16 S16	0.641902563791262	0.224029506208642			
17 S17	0.00162777000107	0.000507001010176			

Por último contamos con un fichero de valores precalculados para los atributos, con este aspecto. Cinco filas una por atributo y una sola proyección precalculada como en el caso anterior, uno para la x y otro para la y con nombre llamada AA_PRECAL. Valores numéricos igualmente.

A1	Bx	Cy
attribute_id	x_AA_PRECAL	y_AA_PRECAL
1 A1	0.730905679530352	0.069199394683096
2 A2	0.929771580830262	0.852936333891561
3 A3	0.855647111379026	0.483148617055294
4 A4	0.285449436363672	0.229667818310741
5 A5	0.487258700586798	0.92660136126474
7		
8		
9		

Este ejemplo es muy instructivo, pues muestra claramente, la estructura de estos ficheros y la relación de cada uno de ellos a nivel de filas y columnas. Todos tienen en común:

- Los ficheros deben de seguir un formato tipo **csv**, separador por **comas**.
- Separados de decimales para valores numéricos ha de ser el **punto**.
- El nombre de cualquier anotación de cada muestra puede ser cualquier **alfanumérico**, evitar espacios y caracteres especiales.

- El nombre de cualquier anotación de cada atributo puede ser cualquier **alfanumérico**, evitar espacios y caracteres especiales.
- Nombre de la anotación para definir las muestras se ha de llamar **sample_id** y el valor de cada uno de ellas a de ser **único** para cada muestra.
- Nombre de la anotación para definir los atributos se ha de llamar **attribute_id** y el valor de cada uno de ellas a de ser **único** para cada atributo.

Los nombres escogidos para cada anotación, son muy importantes, pues este nombre deberá de coincidir a la hora de configurar nuestro caso. Igualmente el valor contenido en cada uno de estas anotaciones a de ser del tipo configurado. Si es numérico, un numero, si es un enumerado, la lista debe de contener los posibles valores existentes en los ficheros de ingesta, etc.

Encodings:

Arquitectura

TODO

Introducción

Antes de empezar a detallar los frameworks, herramientas y la forma en que estas están relacionadas para resolver nuestro sistema, y aunque ya hemos esbozado el dominio del problema, vamos a intentar definir cual es el problema que queremos resolver más en detalle, para ya después poder definir como resolverlo y con que.

Actualmente contamos con infinidad de datasets, pues ahora mas que nunca es sencillo recolectar datos en gran cantidad y en un corto periodo de tiempo de cualquier problema que nos podamos plantear, en cualquier sector: industrial, médico o financiero entre otros.

Tambien es cierto que estos datos suelen tener un denominador en común:

- La cantidad de datos de la que podemos disponer es muy **grande**
- El **formato** de los mismos suele ser **heterogéneo**, procedente de muchas fuentes
- **No sueles estar normalizado**, pues no existe ningún standard a la hora de ser recogidos
- Suele proceder de **muchas fuentes diferentes**, por lo que suele esta fuente de información suele estar distribuido.
- La **dimensionalidad** de estos datos suele ser grande, difícil de analizar por medios visuales, siendo complicado el encontrar patrones de comportamiento en los mismos.
- Relacionado con el primer punto la gran cantidad de **información a visualizar, ha de ser lo suficientemente ágil**, con el fin de poder sacar conclusiones de forma certera

Todas estas características an inspirado el desarrollo de esta solución, intentando resolver cada uno de estos puntos:

- Se ha creado un entorno en donde la escalabilidad sea una característica a tener siempre en cuenta. Actualmente los problemas de gran tamaño se resuelven escalando los sistemas horizontalmente, creando clusters de nodos que comparten sus recursos con el fin de poder hacer frente a problemas que consumen gran cantidad de estos recursos. Sistemas basados en **contenedores** que abstraen el uso de la infraestructura subyacente: memoria, CPU o disco, así como gestores inteligentes de estos artefactos como es **Kubernetes**, hacen posible esta escalabilidad horizontal del sistema a futuro.

- Para resolver el problema del formato, se ha escogido el **formato csv** aquel formato que es mas utilizado en el mercado y fácil uso dentro de cualquier lenguaje, a la hora de almacenar esta cantidad de información.
- Igualmente se ha creado un standard a la hora de presentar esta información al sistema, con el fin de poder abstraerse de cualquier dominio de problema, sea cual sea su origen. Se hablará de este formato en detalle en futuros capítulos.
- Para intentar resolver este problema se ha escogido sistemas que puedan escalar horizontalmente a la hora de manejar este tipo de documentos que como ya hemos relatado, antes será el formato csv. Estas herramientas se llaman **Object Storage** y son sistemas que han demostrado en el mercado su capacidad de resolver este tipo de problemas: Google Drive o Dropbox son ejemplos por todos conocidos. En nuestro caso, hemos adoptado por soluciones Open source como es **Minio**. De la que hablaremos de ella en futuros capítulos.
- Para resolver la **dimensionalidad** de estos datos y poder ser visualizados es un espacio mas simple como es el 2D, se ha adoptado por utilizar algoritmos de Machine Learning, especializados en reducir esta dimensionalidad sin que el carácter e información contendida en estos datasets se vea mermada en lo mínimo. Actualmente el sistema utiliza el algoritmo **t-SNE** para resolver este tipo de problemas, pero es sistema se ha diseñado lo suficiente abierto como para integrar otros algoritmos a futuro.
- Por último estos datasets son grandes verticalmente muchas muestras así como horizontalmente mucha dimensión, el proyectar y reducir la dimensión de estos datasets, permite reducir una dimensión, pero la otra seguirá siendo grande, por lo que visualizar esta de forma ágil es un reto a resolver. Actualmente las herramientas gráficas para visualizar información no siempre aprovechan la capacidad hardware de nuestros equipos, especialmente a nivel visual con GPUs en ocasiones muy rápidas. Se ha escogido para esto un framework de visualización que si aproveche esta capacidad de cómputo y permita visualizar el resultado de estos grandes dataset de forma ágil en pantalla. En concreto se ha escogido el framework **regl-scatterplot**, por ser Open Source y aprovechar la potencia de las GPUs en todo momento.

Tech Stack

Tras describir el dominio de problemas al que nos enfrentamos, debemos de seleccionar las mejores herramientas que se adapten a nuestras necesidades, siempre partiendo de la premisa que todas serán Open Source. Algunas de ellas ya se han nombrado pero aquí vamos a listarlas junto al resto de tecnologías que vamos a usar a la hora de desarrollar el producto.

A la hora de describir la pila de tecnologías o lista de tecnologías podemos hacer tres grupos para que sea mas sencillo y poder encajarlas dentro del contexto del grupo al que pertenecen.

Estos grupos son:

- **Backend**: tecnologías que van a dar soporte al desarrollo de todos los microservicios que implementan la lógica de negocio del sistema.
- **Frontend**: tecnologías que van a dar soporte al desarrollo de todos los microservicios que implementan la interfaz de usuario del sistema
- **Infraestructura**: tecnologías que van a dar soporte y van a crear un entorno seguro en donde todos los servicios de negocio y aquellos que podemos llamar de infraestructura van a convivir juntos implementando como es lógico el dominio del problema que nos ocupa.

TechStack: Backend

Este primer grupo de tecnologías están dentro del contexto del backend del sistema, esta tecnología correrá en el host, dando soporte a la implementación de todos nuestros microservicios. Vamos a crear una tabla donde aparezcan todas ellas y la misión que tienen en concreto.

Nombre	Lenguaje	Descripción
SpringBoot	Java	Framework Lider en el desarrollo de microservicios en Backend en Java OpenSource y mantenido por VMWare. Será utilizado a la hora de desarrollar todos los microservicios de negocio a excepción de aquellos que necesiten modelos matemáticos. Tiene una documentación y comunidad enorme y es utilizado por los actores mas grandes del mercado en sectores como banca, marketing, industria o salud entre otros.
Flask	Python	Framework ligero utilizado para desarrollar aquellos microservicios que deben soportar el desarrollo y publicación de algoritmos o análisis matemáticos
scikit learn	Python	Si bien esta librería es muy amplia relacionada con el Machine Learning, es listada aquí, por ser la que ofrece el algoritmo de t-SNE de proyección que actualmente implementa el sistema

pyscaffold Python

Scaffolding para crear mis microservicios de Python. Es una herramienta OpenSource que te crea un arbol de proyectos muy útil con muchas herramientas integradas y listas para crear tus proyectos: documentación, testing y desarrollo.

Tanto en este grupo como en el resto no voy a describir todas las dependencias que he utilizado para cada uno de los microservicios, pues excedería a este documento. Estas se pueden consultar directamente en el código para cada uno de los microservicios y el detalle de las mismas ir a cada uno de los portales oficiales que corresponda.

TechStack: Frontend

Este segundo grupo de tecnologías están dentro del contexto del frontend del sistema, dando soporte a la implementación del interfaz o entorno gráfico que será utilizado por los usuarios para interactuar con el sistema.

Nombre	Lenguaje	Descripción
Angular	TypeScript css html	Primer Framework, líder en el desarrollo de microservicios en frontend, permitiendo implementar soluciones reactivas SPA, creado y soportado por Google con una comunidad inmensa.
PrimeNG	TypeScript css html	Si bien Angular nos ofrece el framework necesario para crear estas aplicaciones, todos los componentes necesarios para crear estas interfaces no son ofrecidos por Angular, en nuestro caso hemos optado por utilizar esta paleta de componentes, con una comunidad y documentación muy grande, ampliamente utilizado por la comunidad. Opensource. Esta librería también existen para otras tecnologías como React o Vue nosotros utilizaremos la opción de Angular por eso la coletilla de NG al final
regl-scatterplot	TypeScript css html	Si bien PrimeNG ofrece todos los componentes que necesitamos, solamente este ha sido integrado de forma separada. Este componente representa el scatter plot utilizado a la hora de visualizar el resultado proyectado de nuestros datasets. Es de destacar que siendo OpenSource, es capaz de manejar la GPU del equipo, pudiendo pintar millones de puntos en el canvas de una forma ágil. Igualmente es integrable en Angular. Es cierto que el soporte y documentación no es comparable a las dos anteriores librerías, pero tiene su comunidad y es un proyecto activo utilizado en entornos de BigData como el que nos ocupa

TechStack: Infraestructura

Por último este grupo representa todos las demás tecnologías que dan soporte a las anteriores, en forma de implementaciones de lógica de negocio que nosotros no desarrollamos, sino que nos integramos con ellas como puede ser: persistencia o seguridad, así como creando un entorno controlado y seguro para que todos los servicios puedan ejecutarse de forma ordenada, monitorizable y resiliente.

Nombre	Descripción
MongoDB	Base datos no relacional, sin esquema que se integra perfectamente con backend y que permite mantener el estado del sistema en todo momento. Sistema escalable, OpenSource y ampliamente utilizado por la comunidad. Tambien instalaremos la herramienta MongoDB Compass que es un gestor de bases de datos mongo creado y mantenido por MongoDB
Minio	Sistema OpenSource lider en la gestión de Object Storage. Se utiliza para gestionar los recursos (datasets) en forma de ficheros csv tanto en la ingesta como para los resultados proyectados. Tambien instalaremos el CLI de Minio llamado mc
Keycloak	Sistema OpenSoure lider en IAM o gestores de la Autenticación y Autorización. Mantenido por Red-hat y facilmente integrable con el backend y frontend. Se utilizará como es lógico para gestionar toda la seguridad del sistema siguiendo la especificación Oauth, ampliamente utilizado por todas las empresas como AWS, Google, Facebook, X o Microsoft entre otras.
Docker	Se ha comentado ya alguna vez, que todos los servicios utilizados por el sistema sean estos de frontend, backen o infraestructura siempre serán empaquetados como imagenes de Docker e instanciados como contenedores, esta tecnología permite abstraerse el SO que subyace en el host, y permite que es estos microservicios puedan escalarse y ser resilentes a un nivel que no es posible conseguir con tecnologías nativas o de máquinas virtuales
Minikube	Todos estos contenedores han de ser orquestados o controlados de forma dinámica, Kubernetes es sin duda el orquestador lider el mercado creado por Google originalmente pero mantenido actualmente por la comunidad convirtiéndose en un standard dentro de la franja que el ocupa. Este sistema permite implementar la escalabilidad y resiliencia antes ya comentada. En concreto y dada la infraestructura con la que contamos se ha elegido a Minikube como implementación de Kubernetes, por se la primera opción desarrollada para poder desplegar cluster de Kubernetes en entornos reducidos y acotados como es el nuestro. Cuenta con una amplia comunidad y documentación clara que se puede consultar en cualquier momento.
HAProxy	Como se ha comentado nuestra solución AVIB convive con otras soluciones como pueden ser aplicaciones de Bokeh. Por ello necesitamos de un proxy que haga de frontal y redirecciones las peticiones hacia un servicio u otro. HAProxy es una solución Open Source ampliamente utilizada por la comunidad, con una documentación muy clara y muchos ejemplos, en comparación con NGInx o Apache
VirtualBox	Si bien Minikube aconseja utilizar el driver de docker para ser instalado en el host. Nosotros hemos optado por utilizar este otro driver el de VirtualBox que es el Hypervisor o gestor de Mañquinas Virtuales lider OpenSource mantenido por Oracle. Esta herramienta controlará la máquina virtual en donde desplegaremos Minikube, creando un entorno mucho mas aislado que Docker al tener que compartir el espacio del Host con otros servicios como ya hemos indicado.
az CLI	Este es el CLI que se utiliza para comunicarse con el único servicio externalizado, el registro privado de contenedores mantenido por Azure. Ese servicio se encarga de mantener de manera segura todas las

Azure Devops

imágenes de Docker de todos los microservicios de backend o frontend que hemos desarrollado.

Es la herramienta elegida para implementar la gestión del proyecto bajo la filosofía Agile. Herramienta creada por Microsoft integrada con SSO de la Universidad de Oviedo y que ofrece todos los servicios que podemos necesitar a la hora de gestionar nuestro proyecto

Azure

Este el nombre que Microsoft le da la infraestructura Cloud creada por él. Como ya se ha indicado de todos los servicios que ofrece esta infraestructura, solamente utilizaremos el Registro Privado de Contenedores para mantener seguras todas las imágenes de todos nuestros microservicios de negocio, tanto backend, como frontend.

Microservicios

Podemos decir que el sistema es una solución distribuida y descentralizada, que sigue el patrón de microservicios a la hora de ser implementada. Este patrón aboga por dividir el dominio del problema en pequeños subsistemas que implementen un a pequeña parte de este dominio y que sean al mismo tiempo autosuficientes dentro de su subdominio pero que puedan comunicarse con otros microservicios para que todo este dominio de problema pueda ser resuelto. Esta comunicación ha de ser segura y siguiendo estandares del mercado, en nuestro caso HTTP.

Por esta razón hemos dividido este dominio de problema en los siguientes subsistemas implementados por un microservicio en cada caso.

Cada uno de los microservicios tiene un link al repositorio de código donde se encuentra la implementación del mismo bajo el gestor de repos de Azure Devops.

Nombre	Tecnología	Entorno	Descripción
uniovi-avib-morphingprojections-backend-security	SpringBoot	Backend	Implementa la seguridad del sistema, se integra con Keycloak, y mantiene el acceso de usuarios y sus roles
uniovi-avib-morphingprojections-backend-organization	SpringBoot	Backend	Implementa la gestión de la Organización sus Proyectos y Casos. Se integra con MongoDB
uniovi-avib-morphingprojections-backend-annotation	SpringBoot	Backend	Implementa la configuración de los casos y de sus anotaciones. Se integra con MongoDB
uniovi-avib-morphingprojections-backend-storage	SpringBoot	Backend	Implementa el acceso y gestión de los recursos de un caso. Se integra con Minio
uniovi-avib-morphingprojections-backend-job	SpringBoot	Backend	Implementa el manejador de Jobs para proyectar y obtener resultados apartir de los recursos ingestados para los casos
uniovi-avib-morphingprojections-backend	SpringBoot	Backend	Implementa el gateway del sistema que redirecciona el tráfico desde el exterior hacia todos los servicios corriendo en el cluster
uniovi-avib-morphingprojections-backend-analytics	Python	Backend	Implementa los algoritmos de análisis como histogramas y refresion lineal
uniovi-avib-morphingprojections-job-projection	Python	Backend	Implmenta el Job encargado de proyectar nuestros recursos ingestados, bien utililzando el algoritmo t-SNE o bien ya creados esternamente, siempre siguiendo la configuración de cada caso. Este no es un microservicio como tal, pero es creado y destruido bajo demanda, no como los otros servicios que siempre

uniovi-avib-morphingprojections-portal	Angular (Typescript, SCSS, HTML)	Frontend	Este es el microservicio que implementa la interfaz de usuario y que será utilizada por todos los usuarios para interactuar con el sistema
---	---	----------	--

Caso de Uso: Proyección

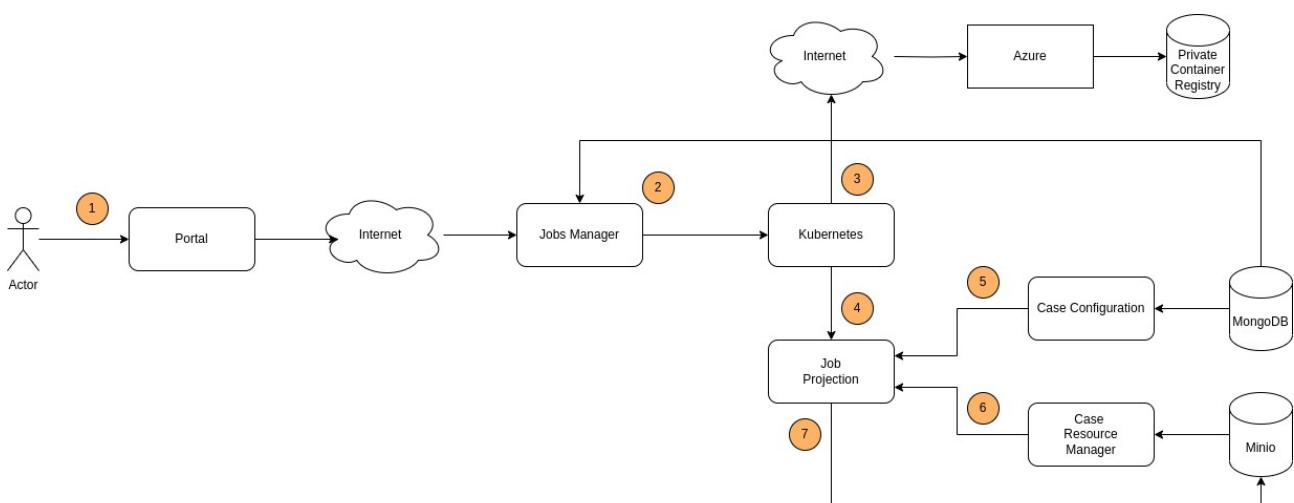
De todos los proceso de negocio cabe destacar el relacionado con la creación, ejecución y monitorización de la proyección utilizando los recursos escalables de Kubernetes llamados Jobs. Estos recursos están cargados de proyectar nuestros recursos en información útil fácilmente visualizable en un canvas 2D, que puedan ser explotado posteriormente con técnicas visuales como el Morphing Projections, grupos de color, filtros o encodings que aporten información útil a la hora de descubrir patrones de comportamiento en los datos que estamos estudiando.

Aunque inicialmente el sistema está destinado a proyectar un dataset de alta dimensionalidad en un canvas 2D utilizando algoritmos de ML como puede ser el t-SNE, el sistema se ha implementado siguiendo una arquitectura abierta que permite ampliar el sistema con nuevos algoritmos, incluso no relacionados con la reducción de la dimensionalidad.

Vamos a resumir esta arquitectura y a explicar como esta implementada así como podría ser posible ampliar el sistema a futuro.

Flujo de Proyección

En este diagrama se puede ver todos los artefactos que intervienen en el proceso de proyección



1. Un usuario escoge un caso que quiere proyectar, estando este ya configurado. Debemos de tener en cuenta que un caso se considera configurado cuando:
 - Tiene recursos de ingesta asociados, siendo estos datasets con los datos de origen de nuestro dominio de problema a proyectar, tanto a nivel de

Datamatrix, como a nivel de annotations (metadatos asociados a Muestras y Atributos)

- Se han configurado todas las anotaciones a nivel de muestra y atributos
 - Se ha configurado el algoritmo que queremos sea utilizado por el Job Manager cuando este sea ejecutados
2. El Job Manager busca el algoritmo asociado al caso y la imagen en donde este se encuentra implementado
 3. Lanza una petición al Clúster de Kubernetes para que esta imagen sea instanciada, si existe cacheada la ejecutará en ese momento, sino la bajará del Registro Privado de Contenedores de Azure
 4. Una vez la imagen de docker ha sido encontrada, esta es instanciada como un contenedor de Docker.

Ahora ya el contenedor instanciado, este comienza a ejecutar el algoritmo y lo primero que hace es recoger todos los recursos asociados al caso: Datamatrix, Sample Annotations, Attribute Annotations, Sample Precalculated Annotations y Attribute Precalculated Annotations. Como mínimo necesitamos subir el Datamatrix y al menos un recurso de un espacio, en caso de ser el primal el Sample Annotations y/o el Sample Precalculated Annotations y en el caso de proyectar el dual el Attribute Annotations y/o Attribute Precalculated Annotations.

A continuación el job recupera la configuración de todas las anotaciones del caso de tipo Sample y/o Attribute

Ahora ya el Job tiene todos los inputs necesarios para proyectar y aplicar la configuración sobre los recursos de ingestión. El resultado de la proyección será otro recurso que será guardado de nuevo en el gestor de recursos Minio a la espera de ser Analizado por el mismo actor u otro diferente.

Ampliar el sistema

Como se ha comentado se ha implementado una arquitectura desacoplada del tipo algoritmo y de su ejecución. Lo primero permite implementar nuevos algoritmos que cumplen el API de comunicación con los mismos y lo segundo permite que la ejecución de los mismos sea escalable y seguro, pues cada uno de ellos se ejecuta como un Job independiente en forma de contenedor controlado por Kubernetes.

Vamos a ver el API de este Job y lo que se le tiene que pasar para que este funcione.

Actualmente solo existe una implementación de este API llamada **uniovi-avib-morphingprojections-job-projection**. Como ya se ha explicado esta implementación desarrolla el algoritmo t-SNE.

El API de este servicio es:

- **case-id:** representa el identificador único del caso que queremos ejecutar, es un string único asociado a cada caso y guardado en MongoDB. Es un atributo obligatorio.
- **spaces:** este es un atributo opcional y es una lista de strings, que pueden tomar el valor de: primal y/o dual. Esta colección le dice al algoritmo si queremos proyectar el primal y/o dual bajo la misma ejecución, claro está para poder proyectar en un espacio debemos de tener los correspondientes recursos asociados al caso para cada espacio así como las configuraciones correspondientes para el mismo

Estos servicios están implementados como servicios de Python implementando en este caso concreto la proyección t-SNE, pero pudieramos utilizar otros algoritmos para implementar esta proyección. He incluso calcular predicciones o clasificaciones con modelos de machine learning o redes neuronales. Debemos de tener en cuenta que el resultado final en el caso de ser una proyección sigue un modelo preparado para ser explotado visualmente por la herramienta, en caso de que el resultado creado por la herramienta no cumpliera este standard, no podría ser visualizado en el módulo preparado a tal efecto en el Portal teniendo que extender el mismo con otros módulos preparados para poder gestionar estos nuevos tipos de resultados. En cualquier caso el resultado en formato csv, se podría bajar de la herramienta y utilizar herramientas externas de análisis si así lo quisieramos.

Análitica: Algoritmos

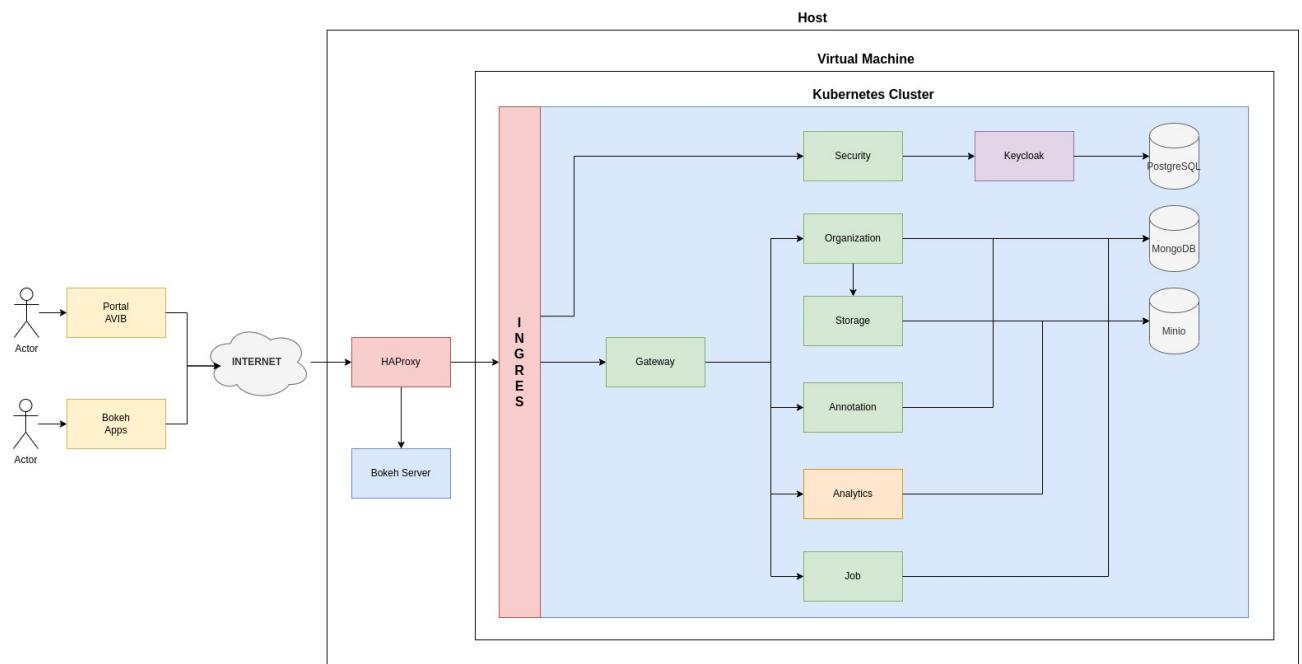
Este servicio está implementado bajo el nombre: **uniovi-avib-morphingprojections-analysis** y es un servicio como ya se ha visto en apartados anteriores de tipo Python y a diferencia del anterior que implementa la proyección de un caso, este es un servicio que está siempre corriendo en el cluster y a la espera de peticiones por parte del Portal, con el fin de obtener resultados analíticos de los datos de un caso. Actualmente este servicio implementa dos analíticas:

- **Histogramas** sobre un conjunto de puntos agrupados desde la interfaz aplicado sobre cualquier anotación creada sobre estos datos
- **Regresión lineal** sobre un conjunto de datos igualmente seleccionados desde la interfaz.

Al igual, que en el caso anterior, este servicio podría ser ampliado con nuevos algoritmos y diseñando un API que debería de ser utilizado gráficamente desde el Portal. El Portal está diseñado para que pueda ser ampliado igualmente con estos nuevos algoritmos fácilmente, desde donde recoger estos datos que puedan ser pasados al nuevo algoritmo implementado en el servicio **uniovi-avib-morphingprojections-analysis** y expuestos como un Web Service de tipo POST con API utilizado por ambas capas: la backend y la frontend.

Diagrama de despliegue

En este diagrama vamos a ver a vista de 1000 pies todo los artefactos que participan en el sistema así como su relación entre los mismos:



Aquí se pueden ver todos los servicios que componen el sistema, tanto de negocio:

- **Seguridad** (Security)
- **Organización** (Organization)
- **Anotaciones** (Annotations)
- **Almacenamiento** (Storage)
- **Job Manager** (Job)
- **Analítica** (Analysis)
- **Gateway**
- **Interfaz de Usuario** (Portal)

Como los servicios de infraestructura:

- **Proxy Kubernetes** (Ingress NGInx Controller)
- **Servicio de Autenticación y Autorización** (Keycloak, PostgreSQL)
- **Gestor de recursos** (Minio)

- **HAProxy**: proxy de entrada a cualquier servicio que corre en el host, tanto al sistema AVIB como a cualquier otro servicio externo como puede ser una aplicación Bokeh

Código

En este capítulo vamos a resumir brevemente como se han desarrollado los microservicios tanto de backend en java o Python, como el de frontend o Portal en Typescript, scss y html.

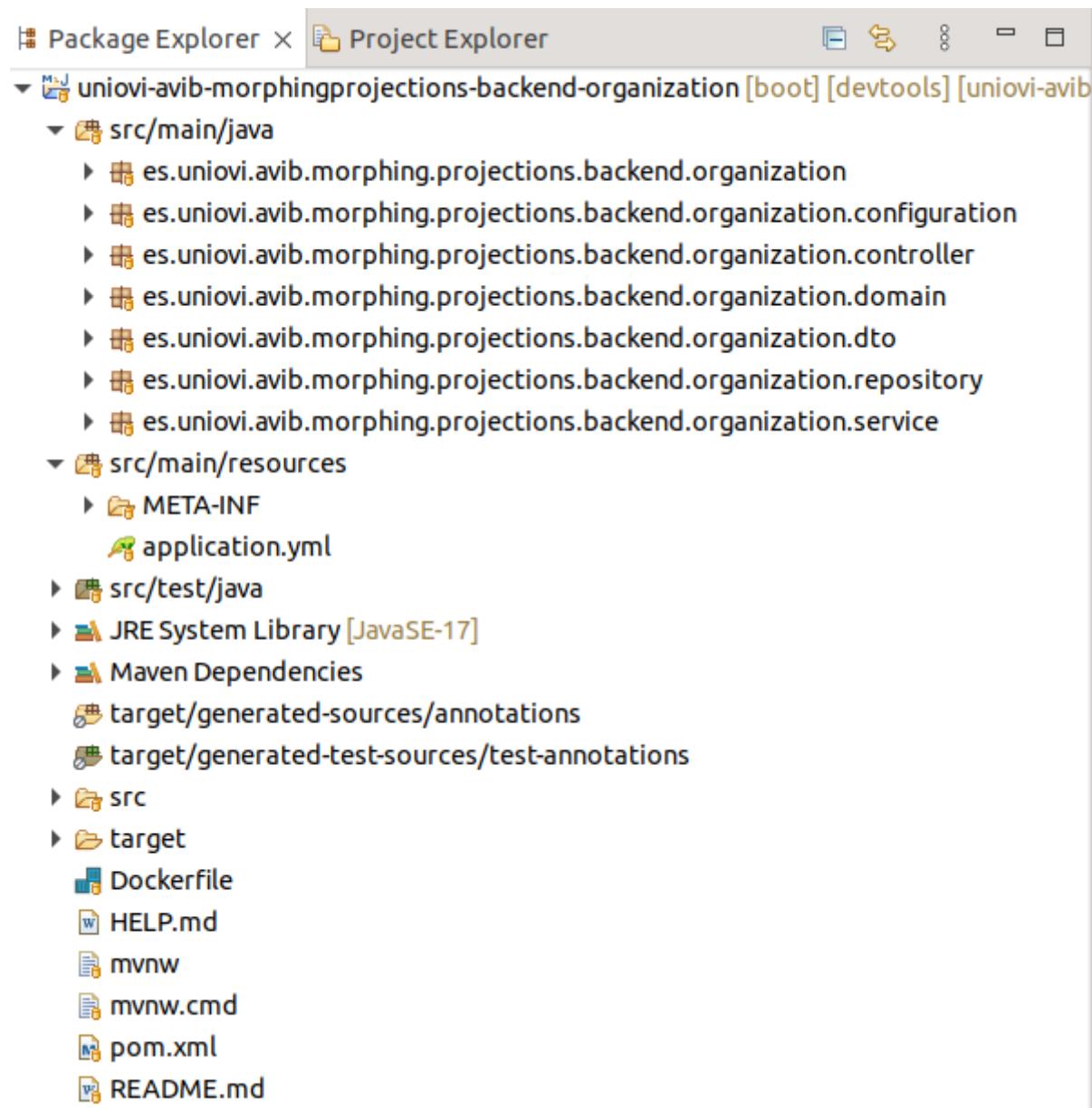
Microservicios de backend java:

Como se ha comentado anteriormente se ha escogido para el desarrollo de los microservicios de backend en Java, el framework Springboot. Todos los servicios siguen el patron de capas:

- **Controller:** capa de API en donde se exponen la funcionalidad del microservicio en forma de Web Services consumibles por otros microservicios.
- **Model y DTOs:** es la capa de objetos de negocio que van a ser persistidos en bases de datos o Object Storage y los DTOs o Transfer Data Objest son aquellos servicios que permiten intercambiar datos a través de las APIs del controller.
- **Repository:** es la capa de objetos que está conectados directamente a los servicios de infraestructura: bases de datos. No siempre es necesaria pues hay microservicios que no interactúan con bases de datos.
- **Services:** es la implementación de la lógica de negocio del propio microservicio. Esta capa hace uso del modelo para poder interactuar con otros microservicios de negocio o infraestructura en caso de ser necesario: bases de datos, object storage o servicios de seguridad en nuestro caso.
- A estas tres capas clásicas debemos de añadir la **configuración** del mismo, en donde se describe los atributos necesarios para que otros micros se puedan comunicar con el y los parámetros que este necesita: credenciales, urls, parámetros de conexión, etc para que este se pueda comunicar con otros microservicios igualmente. En lenguaje utilizado para configura todos estos parámtros es el yaml y estan agrupados por entornos de despliegue. En nuestro casos solo dos:
 - **Desarrollo:** entorno local asociado a nuestro equipo de trabajo, en donde debemos de tener todos los servicios arrancados necesarios para poder desarrollar y depurar el microservicio.

- **Producción:** representa el entorno en donde vamos a desplegar nuestro microservicio dentro del host.

Vamos a ver un ejemplo, utilizando el microservicio de Organización:



En esta captura se puede ver el árbol de carpetas del microservicio. Este es un ejemplo de un microservicio que interactúa con MongoDB gestionando el CRUD de organizaciones, proyectos, casos, recursos y usuarios del sistema.

Si vemos el paquete de controller veremos todas las APIs del mismo, expuesta para ser utilizada por otros micros, especialmente por el Portal. Para poder ver estas APIs se ha integrado a todos los micros con OpenAPI. Esta es una especificación de Java orientada a documentar las APIs y ser accesibles via portales. Este portal solo es accesible en modo desarrollo y no en producción por temas de seguridad. En local podremos acceder al API por ejemplo en este caso de organización a través de esta URI:

<http://localhost:8082/swagger-ui/index.html#/>

The screenshot shows the Swagger UI interface for the 'Organizations API'. At the top, there's a header with the Swagger logo, the URL '/v3/api-docs', and a 'Explore' button. Below the header, the title 'Organizations API' is displayed with a version of '1.0.0' and 'OAS 3.0'. A note says 'This API exposes endpoints to manage organizations.' Below this, there are links to 'Miguel Salinas Ganeco - Website', 'Send email to Miguel Salinas Ganeco', and 'MIT License'. A 'Servers' dropdown is set to 'http://localhost:8082 - Server URL in Development environment'. The main content area shows the 'user-controller' section with several API endpoints listed: 'GET /users', 'POST /users', 'POST /users/{userId}/resetPassword', 'POST /users/inviteUser', 'GET /users/{userId}', and 'DELETE /users/{userId}'. The 'DELETE' endpoint is highlighted with a red background.

Desde esta página podemos ejecutar cualquier API introduciendo los datos necesarios como por ejemplo este endpoint del API de usuarios desde donde podemos recoger los datos de un usuario dado el identificador único creado en el gestor de autenticación de Keycloak. El resultado como es lógico es un JSON con toda la información existente para este usuario

The screenshot shows the Swagger UI interface for a GET request to the endpoint `/users/{externalId}/external`. The `externalId` parameter is required and has a value of `5014fcc1-62e4-4676-8887-0bc007124bef`. Below the parameters, there are sections for Responses, Curl command, Request URL, and Server response. The Response body is displayed as JSON:

```
{
  "userId": "66a08d7fb5b24be6ab0211",
  "externalId": "5014fcc1-62e4-4676-8887-0bc007124bef",
  "firstName": "Miguel Angel",
  "lastName": "Salinas Gancedo",
  "username": "salinasangel",
  "email": "salinasangel@uniovi.es",
  "language": "es",
  "address": ""
}
```

En el caso de tener acceso temporal a estas APIs en producción podemos crear proxies temporales en el host y acceder a la misma URL como en local, por ejemplo para hacer esto para este mismo microservicio ejecutamos este comando desde la consola del host:

The screenshot shows the Swagger UI for the **Organizations API** version 1.0.0, OAS 3.0. It displays the API's purpose, contact information, and a note about exposing endpoints to manage organizations. Below this, it shows the **Servers** section with the URL `http://localhost:8082`. The main area shows the **user-controller** with its methods:

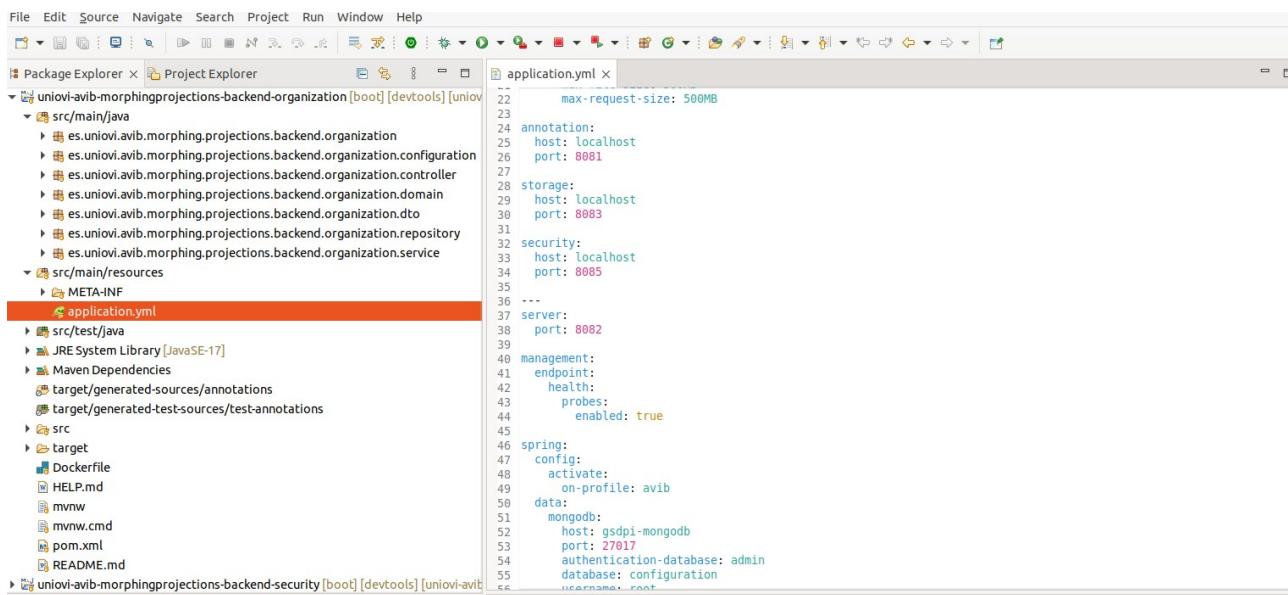
- `GET /users`
- `POST /users`
- `POST /users/{userId}/resetPassword`
- `POST /users/inviteUser`
- `GET /users/{userId}`
- `DELETE /users/{userId}` (highlighted in red)

Para acceder a otra API debemos de buscar como se llama el servicio en Kubernetes y substituir el puerto utilizado por el mismo dentro del cluster.

NOTA: esta herramienta solo se ha implementado en todos los microservicios

de negocio en Java, en el caso del microservicio de Python de Análisis, no se ha hecho.

Se ha comentado antes que una parte importante de los microservicios es la configuración de los mismos. Por ejemplo si nos fijamos en este ejemplo de Organización:



The screenshot shows the Eclipse IDE interface with the Project Explorer and Package Explorer tabs. The Project Explorer shows a Maven project named 'uniovi-avib-morphingprojections-backend-organization'. The application.yml file is selected in the Project Explorer, and its contents are displayed in the editor tab. The configuration file includes sections for max-request-size, annotation, storage, security, server, management, spring, and mongodb.

```
max-request-size: 500MB
annotation:
  host: localhost
  port: 8081
storage:
  host: localhost
  port: 8083
security:
  host: localhost
  port: 8085
server:
  port: 8082
management:
  endpoint:
    health:
      probes:
        enabled: true
spring:
  config:
    activate:
      on-profile: avib
  data:
    mongodb:
      host: gsdpi-mongodb
      port: 27017
      authentication-database: admin
      database: configuration
      username: root
```

Vemos que dentro del fichero **application.yml**, esta toda la configuración necesaria para que este micro pueda publicar sus APIs y al mismo tiempo se pueda comunicar con otros micros externos. Igualmente debemos notar que este fichero está dividido en dos partes separadas por estos caracteres **----**. Estos caracteres agrupan las configuraciones para cada uno de los contextos de despliegue en donde estos microservicios van a correr. El grupo situado por encima es el entorno de desarrollo, y todos los parámetros: urls, puertos, credenciales, etc están asociados a microservicios que corren en local, en nuestro equipo de desarrollo. Por contra todos los parámetros situados por debajo hacen referencia a parámetros propios de microservicios que corren en producción. Este último entorno tiene un nombre asociado llamado en todos los micros de igual forma: **avib**. Este nombre es importante, porque cuando vayamos a compilar y/o empaquetar estos servicios como imágenes de docker deberemos de añadir este nombre de contexto para incluir en tiempo de compilación y empaquetado los parámetros correctos correspondiente al entorno donde queremos que esta imagen de docker sea desplegada.

Estos parámetros sensibles están protegidos a nivel de código por los repositorios privados y en tiempo de ejecución por el mismo cluster, o accesible al menos que tengamos las credenciales para ello.

Microservicios de backend Python:

El sistema aunque cuenta con dos microservicios en Python solamente el de Análisis es desplegado como los microservicios de Java, pues el otro el que implementa la proyección es desplegado bajo demanda por el Job Manager del sistema.

En ese caso la estructura de carpetas de este micro es un poco diferentes al de los de Java aunque arquitectónicamente intenta mantener la misma estructura de capas antes explicada:

```
└─ uniovi-avib-morphingprojections-backend-analytics
    ├─ __pycache__
    ├─ .tox
    ├─ .venv
    ├─ .vscode
    ├─ dist
    ├─ docs
    └─ src
        └─ morphingprojections_backend_analytics
            ├─ __pycache__
            ├─ .cache
            ├─ environment
            ├─ __init__.py
            ├─ service.py
            └─ uniovi_avib_morphingprojections_backend_analytics.egg-info
                ├─ tests
                └─ .coveragerc
            └─ .gitignore
            └─ .readthedocs.yml
        └─ AUTHORS.md
        └─ CHANGELOG.md
        └─ CONTRIBUTING.md
        └─ Dockerfile
        └─ gunicorn_config.py
        └─ LICENSE.txt
        └─ pyproject.toml
        └─ README.md
        └─ requirements.txt
        └─ setup.cfg
        └─ setup.py
        └─ tox.ini
```

En este caso la capa de controller repository, model y service se ha fundido en un solo paquete llamado service. En el caso de la configurar Python no cuenta con ningún sistema tan sofisticado como si tiene SpringBoot, en nuestro caso lo hemos diseñado nosotros mismos bajo la carpeta de environment, donde residen para cada entorno un fichero diferente:

- **environment-avib.yaml**: configuraciones del entorno produccion
- **environment.yaml**: configuraciones entorno desarrollo.

En este caso el servicio utiliza el prefijo despues de la palabra environment junto a un parámetro de empaquetado llamado **ARG_PYTHON_PROFILE** que indica el entorno que queremos empaquetar en nuestra imagen, en este caso es producción:

Microservicios de frontend Angular

El sistema cuenta solamente con un microservicio de frontend llamado portal. Este microservicio utilza el framework de Angular y las dependencias de PrimeNG entre otras. Logicamente el arbol de carpetas es diferente al de backend, aunque la forma de configurar los entornos es semejante al micro de backend de Python:

```
uniovi-avib-morphingprojections-portal
├── .angular
├── dist
├── node_modules
└── src
    ├── app
    │   ├── demo
    │   ├── layout
    │   ├── shared
    │   ├── views
    │   ├── app-routing.module.ts
    │   ├── app.component.html
    │   ├── app.component.scss
    │   ├── app.component.spec.ts
    │   ├── app.component.ts
    │   ├── app.module.ts
    │   └── assets
    └── environments
        ├── environment.avib.ts
        ├── environment.minikube.ts
        ├── environment.prod.ts
        ├── environment.ts
        └── favicon.ico
            ├── index.html
            ├── main.ts
            ├── styles.scss
            └── upload.php
        ├── .editorconfig
        ├── .gitignore
        ├── angular.json
        ├── CHANGELOG.md
        ├── Dockerfile
        ├── package-lock.json
        ├── package.json
        ├── README.md
        ├── tsconfig.app.json
        ├── tsconfig.json
        └── tsconfig.spec.json
```

En esta captura podemos ver el árbol de carpetas del micro de Angular, con

subcarpetas en donde se implementan las vistas, servicios de autenticacion, el menu y la parte de configuración de los entornos bajo la carpeta llamada, environment. Al igual que en el caso del micro de Python existe un fichero por cada entorno y cuyo sufijo indica el nombre del entorno en cuestión. Para ser utilizado en tiempo de compilacion y empaquetado tenemos el mismo parámtro llamado **ARG_ANGLAR_PROFILES** como se puede ver en el ejemplo inferior

```
$ docker build --build-arg ARG_PYTHON_PROFILES_ACTIVE=minikube -t uniovi-avib-morphingprojections-backend-analytics:1.0.0 .
```

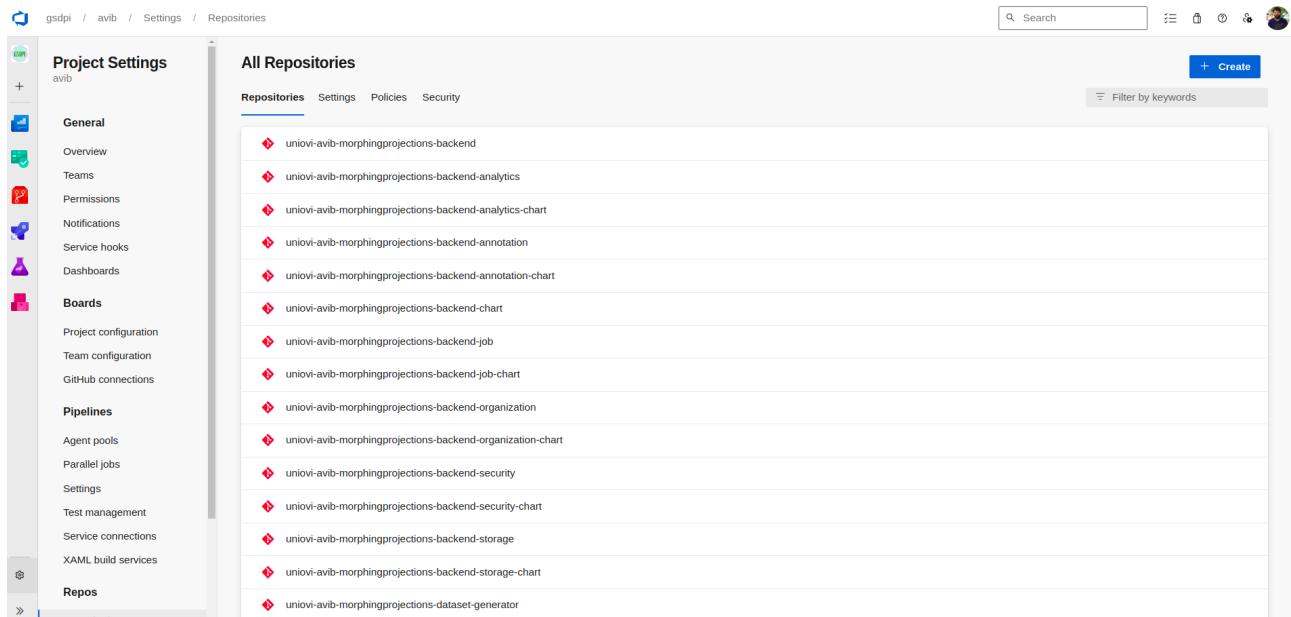
Repositorios de código

Como ya se ha comentado actualmente todos los repositorios están bajo el control de versiones de Azure Devops, siendo estos privados, claro está.

Todos los repositorios siguen una misma política de nombres, con un prefijo que hace referencia a la organización, uniovi, en nuestro caso y al proyecto avib-morphingprojections, más un sufijo propio y único para cada uno de los repos, como se puede ver en esta fórmula:

uniovi-avib-morphingprojections-<SUFIGO_DEL_REPO>

También podemos afirmar que cada repo hace referencia a un único servicio o contexto.



The screenshot shows the 'All Repositories' page in the Azure DevOps interface. The left sidebar is titled 'Project Settings' and lists various sections: General, Overview, Teams, Permissions, Notifications, Service hooks, Dashboards, Boards, Project configuration, Team configuration, GitHub connections, Pipelines, Agent pools, Parallel jobs, Settings, Test management, Service connections, XAML build services, and Repos. The 'Repos' section is currently selected. The main area is titled 'All Repositories' and contains a table with columns: Repository, Settings, Policies, and Security. The table lists 17 repositories, all starting with 'uniovi-avib-morphingprojections-' followed by a specific suffix: backend, backend-analytics, backend-analytics-chart, backend-annotation, backend-annotation-chart, backend-chart, backend-job, backend-job-chart, backend-organization, backend-organization-chart, backend-security, backend-security-chart, backend-storage, backend-storage-chart, and dataset-generator.

Repository	Settings	Policies	Security
uniovi-avib-morphingprojections-backend			
uniovi-avib-morphingprojections-backend-analytics			
uniovi-avib-morphingprojections-backend-analytics-chart			
uniovi-avib-morphingprojections-backend-annotation			
uniovi-avib-morphingprojections-backend-annotation-chart			
uniovi-avib-morphingprojections-backend-chart			
uniovi-avib-morphingprojections-backend-job			
uniovi-avib-morphingprojections-backend-job-chart			
uniovi-avib-morphingprojections-backend-organization			
uniovi-avib-morphingprojections-backend-organization-chart			
uniovi-avib-morphingprojections-backend-security			
uniovi-avib-morphingprojections-backend-security-chart			
uniovi-avib-morphingprojections-backend-storage			
uniovi-avib-morphingprojections-backend-storage-chart			
uniovi-avib-morphingprojections-dataset-generator			

Estos repositorios los podemos agrupar en:

1. Repositorios de servicios de negocio

- uniovi-avib-morphinprojections-backend
- uniovi-avib-morphinprojections-backend-organization
- uniovi-avib-morphinprojections-backend-annotation
- uniovi-avib-morphinprojections-backend-security
- uniovi-avib-morphinprojections-backend-storage

- uniovi-avib-morphinprojections-backend-job
 - uniovi-avib-morphinprojections-backend-job-projection
 - uniovi-avib-morphinprojections-backend-analytics
 - uniovi-avib-morphinprojections-portal
2. Repositorios de servicios de paquetes de heml
- uniovi-avib-morphinprojections-backend-chart
 - uniovi-avib-morphinprojections-backend-organization-chart
 - uniovi-avib-morphinprojections-backend-annotation-chart
 - uniovi-avib-morphinprojections-backend-security-chart
 - uniovi-avib-morphinprojections-backend-storage-chart
 - uniovi-avib-morphinprojections-backend-job-chart
 - uniovi-avib-morphinprojections-backend-analytics-chart
 - uniovi-avib-morphinprojections-portal-chart
3. Repositorios despliegues de servicios de infraestructura
- uniovi-avib-morphinprojection-deployment
4. Repositorios despliegues de documentación del sistema
- uniovi-avib-morphinprojection-documentation
5. Repositorios extras relacionados con herramientas auxiliares o servicios externos al proyecto AVIB, pero que conviven con el cluster en el host.
- uniovi-avib-morphinprojection-dataset-generator
 - uniovi-gsdpi-bokeh-template
 - uniovi-gsdpi-bokeh-mp-traccion

Para mayor detalle sobre que es cada uno se puede consultar el capítulo destinado a tal efecto.

Operaciones

En esta sección vamos a explicar en detalle todos los pasos y configuraciones que debemos de tener en cuenta a la hora de desplegar, configurar y mantener un sistema como es AVIB. Sabiendo que desde el punto arquitectónico es un sistema basado en microservicios que necesitan del apoyo de otros servicios no desarrollados internamente como pueden ser:

- **Dashboard:** esta herramienta es muy útil para visualizar y gestionar casi todos los recursos del cluster: Pods, contenedores, servicios, secrets, etc de forma visual. De todas formas siempre contaremos con el CLI de Kubernetes llamado kubectl que ya hablaremos de el posteriormente.
- **Base de datos** para gestionar los metadatos del sistema.
- **Gestores de ficheros** Object Storage para gestionar los datasources del sistema.
- **Servicio de Autenticación y Autorización** para implementar la seguridad.
- **Reverse Proxies** para redireccionar el tráfico procedente del exterior. En este apartado hablaremos de un servicio que aun no corriendo dentro del cluster de Kubernetes es de vital importancia pues estará corriendo en el Host dando entrada a todas las peticiones externas, tanto al sistema AVIB desplegado en un cluster de Kubernetes como a los servicios que actualmente ya están corriendo en el Host.

Despliegue infraestructura kubernetes

Introducción

En este capítulo se va a explicar como se debe debemos desplegar un cluster de Kubernetes. Como sabemos Kubernetes se define como un orquestador de contenedores, y por ellos su despliegue y correcta configuración es de vital importancia pues todos los servicios de infraestructura y servicios de negocio van a ser empaquetados como imágenes de Docker desplegadas en este entorno, que va a encargarse de gestionarlos correctamente en todo momento.

Debemos tener presente que el cluster de Kubernetes va a ser desplegado en una infraestructura hardware manejada por el departamento AVIB de la Universidad de Oviedo con sus limitaciones a nivel de recursos: memoria, CPU y disco, que la mismo tiempo va a ser un recurso compartido por otros otros servicios ya existentes que no van a correr dentro del espacio de Kubernetes. Dada esta situación se ha adoptado por seleccionar un despliegue de Kubernetes basado en un cluster mono nodo, pues el hardware no es escalable horizontalmente y por ello se ha escogido el servicio de [Minikube](#) por:

- Es el primer servicio desarrollado para desplegar Kubernetes en entornos no escalables como el que nos ocupa.
- Servicio mantenido actualmente por la comunidad, con un roadmap muy activo.
- Permite desplegar Kubernetes en dos modos utilizando drivers: modo contenedor o modo Maquina Virtual utilizando un Hypervisor.
- Documentación amplia y comunidad activa, que permite consultar errores, casos de uso o enviar tickets en caso de dudas o errores.
- Gran cantidad de extensiones faciles de ser desplegadas a través de su consola.
- Poner en marcha un cluster de Kubernetes es rápido y sencillo desde cero.

Por todas estas razones se ha escogido Minikube. Debemos de hacer imcapié en el punto que habla sobre la forma de ser desplegado Minikube, en nuestro caso y viendo que el nodo hardware en donde va a ser desplegado Kubernetes ha de ser compartido por otros servicios se ha adoptado desplegar Kubernetes en modo Máquina Virtual que aísla por completo el cluster y cualquier servicio que corra dentro de el, evitando cualquier problema colateral a nivel de seguridad con el host y los servicios que puedan correr en paralelo en el mismo.

Instalación del cluster de Kubernetes

Vamos a listar todos los pasos y consideraciones que debemos de tener en cuenta a la hora de desplegar un cluster de Kubernetes con las consideraciones antes expuestas:

- **Paso 01: instalar el CLI de minikube**

Como ya se ha comentado hemos escogido la herramienta Minikube para desplegar y gestionar nuestro cluster de Kubernetes, por lo que lo primero que debemos hacer es instalar el CLI (Command Line Interface) de Minikube. Este CLI es un simple binario que debemos de bajarlo a nuestro Host e instalarlo en la carpeta compartida de binarios para poder acceder al mismo desde cualquier cuenta. En nuestro caso escogemos la arquitectura de nuestro Host de tipo amd64:

```
$ curl -LO https://storage.googleapis.com/minikube/releases/latest/minikube-linux-amd64  
$ sudo install minikube-linux-amd64 /usr/local/bin/minikube && rm minikube-linux-amd64
```

Este comando instalar el CLI de Minikube bajo el nombre minikube, siendo accesible desde nuestra cuenta. Para chequearlo podemos ejecutar simplemente este comando para ver que funciona correctamente:

```
$ minikube version  
minikube version: v1.34.0  
commit: 210b148df93a80eb872ecbeb7e35281b3c582c61
```

Este comando muestra la versión de minikube y mas adelante cuando despleguemos un cluster la versión del cluster desplegado en el Host

- **Paso 02: desplegar el cluster de kubernetes**

Antes de ejecutar el comando del CLI de Minikube que instalará el cluster debemos de tener algunas cosideraciones respecto al Host en donde correrá el cluster:

- Vamos a utilizar el modo Máquina Virtual, y por ello necesitamos que el Host tenga instalado un Hypervisor que va a encargarse de gestionar las Máquinas Virtuales que correrán en el Host. Aunque Minikube soporta varias máquinas virtuales la que utilizar por defecto es la De VirtualBox de Oracle, por lo que será esta la que deba de estar instalada en el host antes de desplegar el cluster. La instalación de esta herramienta queda fuera del ámbito de este documento. Pero para más detalle puede consultar el link oficial de Oracle sobre [VirtualBox](#)

- El Host debe de tener los recursos suficientes para albergar el cluster de Kubernetes a nivel de: memoria, CPU y disco. Claro está este debe de tener acceso a Internet en todo momento.

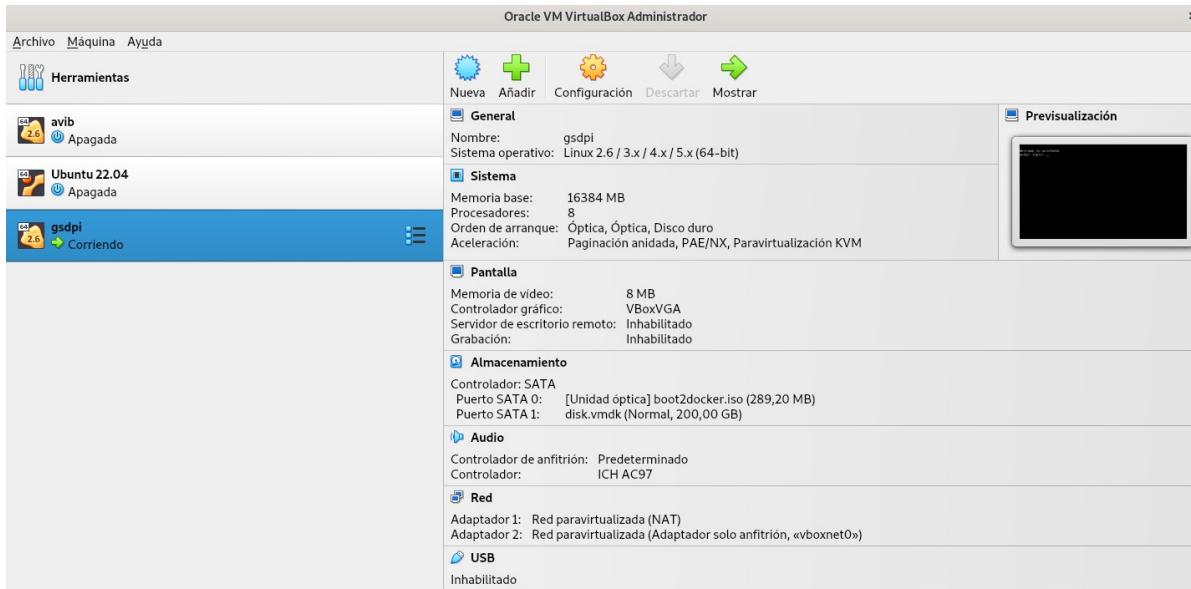
Con estos datos presentes vamos a describir los recurso mínimos y consideraciones que hemos escogido para nuestro cluster:

- **Memoria:** 16 gigabytes de memoria RAM, pues trataremos con datasources que en muchos casos van a requerir de memoria RAM para ser ejecutados correctamente. Este recurso es fácilmente escalable, teniendo en cuenta claro está las limitaciones del nodo hardware que cuenta con hasta 32 Gigas si fuera necesario ampliar la misma.
- **CPU:** máximo cores para que los proceso de machine learning y proyecciones puedan correr de forma ágil. Al igual que en el caso anterior este recurso es fácilmente escalable desde el gestor de VirtualBox si fuera necesario ceder mas potencia de cálculo al cluster.
- **Disco:** 200 gigas para poder almacenar de forma agil ahora y en el futuro los datasources que utilizaremos en nuestros análisis. Este recurso aunque escalable posteriormente se ha definido la Máquina Virtual, no es tan sencillo de ser modificado a posteriori, con peligro de dejar la máquina Virtual no accesible si no se hacen los pasos correctos, por ello se ha escogido suficiente espacio de disco para que no nos quedamos cortos a medio plazo. En este [link](#) publico explico de todas formas como realizar los pasos si nos viéramos en la necesidad de ceder mas espacio de disco a la Maquina Virtual si fuera necesario.
- Como sabemos el acceso a cualquier servicio desde el Host se hace de forma segura utilizando el protocolo TLS. Como ya sabemos para ellos Minikube ofrece la opción de generar **certificados autofirmados** por el sistema, haciendo que el proceso de instalación sea mas sencillo. En este caso debemos tener en cuenta es la expiración del certificado auto-firmado que originalmente crea Minikube para nosotros. Por defecto Minikube utilizar 3 años para la misma, pero nosotros vamos a ampliarla hasta los 10 años para no tener que modificarla posteriormente que no es una tarea nada sencilla

```
$ minikube start --profile=gsdipi --driver=virtualbox --memory=16384 --cpus=max --disk-size=200g --cert-expiration=87600h0m0s:
```

Con estas consideraciones podemos ya ejecutar el siguiente comando de minikube desde la consola del Host. Podemos el driver escogido, los recursos antes listados y los 10 años de vida para los certificados creados por el cluster:

Tras unos segundos y no ha habido ningún problema veremos nuestro cluster corriendo dentro de una máquina virtual con los recurso antes descritos:



Una nota a destacar es que aunque el cluster es un servicio vital, pues dentro de él correrán todos los servicios del sistema, este pueda que tenga que ser parado por mantenimiento. Aunque este es una máquina virtual listada como cualquier otra máquina virtual box, no debemos de manejarla utilizando la herramienta de Virtual Box, sino el CLI de Kubernetes, **siempre deberemos de arrancar o para el cluster desde el CLI y nunca desde Virtual Box, esto puede dejar la instancia corrupta o en estado no consistentes**

Para mayor detalle de como utilizar el CLI podemos consulta la ayuda del mismo desde la consola del host como cualquier comando de Linux:

```
$ minikube --help
```

- **Paso 03: Instalar el CLI de Kubernetes**

Tras desplegar nuestro cluster de Kubernetes, ahora debemos de instalar algunas herramientas para interactuar con el a la hora de desplegar y monitorear el estado de nuestros servicios desplegados dentro de el. Para ellos Kubernetes ofrece el CLI oficial llamado **kubectl**. Si es cierto que minikube en las últimas versiones ya viene preparado con esta herramienta para no tener que instalarla posteriormente, vamos a explicar como instalar el CLI por nuestra cuenta. Un dato que debemos de tener en cuenta antes de instalar este CLI es la versión de kubernetes que Minikube ha instalado en nuestro Host, pues el cliente debe de estar alineado con esta versión para no sufrir efectos desagradables a la hora de ejecutar este comando contra nuestro cluster.

```
$ minikube version --components
minikube version: v1.31.2
commit: fd7ecd9c4599bef9f04c0986c4a0187f98a4396e

buildctl:
buildctl github.com/moby/buildkit v0.11.6 2951a28cd7085eb18979b1f710678623d94ed578

containerd:
containerd g
```

Este comando nos da las versiones de todos los componentes de los que esta formado un cluste de Kubenetes, el que nos interesa el de minikube version que esta alineado con la versión de kubernetes en nuestro caso la 1.31.2, por lo tanto deberemos de instalar esta versión de CLI llamada kubectl, como sigue:

```
$ curl -LO "https://dl.k8s.io/release/1.31.2/bin/linux/amd64/kubectl"
$ sudo install -o root -g root -m 0755 kubectl /usr/local/bin/kubectl
$ kubectl version
WARNING: This version information is deprecated and will be replaced with the output from
kubectl version --short. Use --output=yaml|json to get the full version.
Client Version: version.Info{Major:"1", Minor:"27", GitVersion:"v1.27.4",
GitCommit:"fa3d7990104d7c1f16943a67f11b154b71f6a132", GitTreeState:"clean",
BuildDate:"2023-07-19T12:20:54Z", GoVersion:"go1.20.6", Compiler:"gc",
Platform:"linux/amd64"}
Kustomize Version: v5.0.1
Server Version: version.Info{Major:"1", Minor:"27", GitVersion:"v1.27.4",
GitCommit:"fa3d7990104d7c1f16943a67f11b154b71f6a132", GitTreeState:"clean",
BuildDate:"2023-07-19T12:14:49Z", GoVersion:"go1.20.6", Compiler:"gc",
Platform:"linux/amd64"}
```

Aquí vemos como el comando de kubectl no solo nos da la versión suya sino también la versión del cluster de kubernetes ya funcionando. Esto es así, porque cuando minikube inicia el cluster también crea una configuración de acceso al cluster por defecto, con todas las credenciales necesarias para que un CLI como kubectl pueda lanzar comandos al cluster. Esta configuración se guarda por defecto en este fichero:

`~/.kube/config`

Comentar que si paramos el cluster estas credenciales se borraran del fichero config, volviendo a recreares en su arranque.

Activar addons kubernetes

Una vez el cluster esté funcionando deberemos de instalar un par de extensiones que minikube las llama addons.

Ahora que ya tenemos nuestro cluster corriendo y el CLI instalado para interactuar con él, vamos a terminar esta primera etapa de instalación del cluster desplegando un par de servicios necesarios para acceder externamente a nuestros servicios y poder monitorizar y configurar el mismo visualmente:

Instalación de addons en el cluster Kubernetes

- **Paso 01: Instalar las extensión Kubernetes Dashboard**

Dashboard: este es el servicio que por defecto ofrece Kubernetes para poder visualizar y gestionar los recursos de un cluster de Kubernetes de forma visual, aunque como ya se ha comentado anteriormente, el CLI de Kubernetes llamado kubectl ofrece todo lo necesario para interactuar con el cluster desde linea de comandos, es mas, el Dashboard no ofrece todos los recursos para poder ser gestionados visualmente, pero el CLI si. De todas formas el Dashboard casi todos los recursos que manearemos nosotros en nuestro sistema siendo mucho mas ágil utilizar esta herramienta en muchos casos frente a la linea de comandos ofrecida por kubectl. Para instalar este servicio Minikube lo ofrece bajo la figura de addon, es decir Minikube tiene muchos addons que pueden ser desplegados fácilmente utilizando su CLI de minikube. Para listar todos estos addons podemos ejecutar el comando. Este comando nos da una lista muy extensa con muchos addons entre ellos el llamado dashboard

ADDON NAME	PROFILE	STATUS	MAINTAINER
ambassador	gsdipi	disabled	3rd party (Ambassador)
auto-pause	gsdipi	disabled	minikube
cloud-spanner	gsdipi	disabled	Google
csi-hostpath-driver	gsdipi	disabled	Kubernetes
dashboard	gsdipi	enabled	Kubernetes
default-storageclass	gsdipi	enabled	Kubernetes
efk	gsdipi	disabled	3rd party (Elastic)
....			

Ahora simplemente para instalar el addon llamado dashboard ejecutamos este comando:

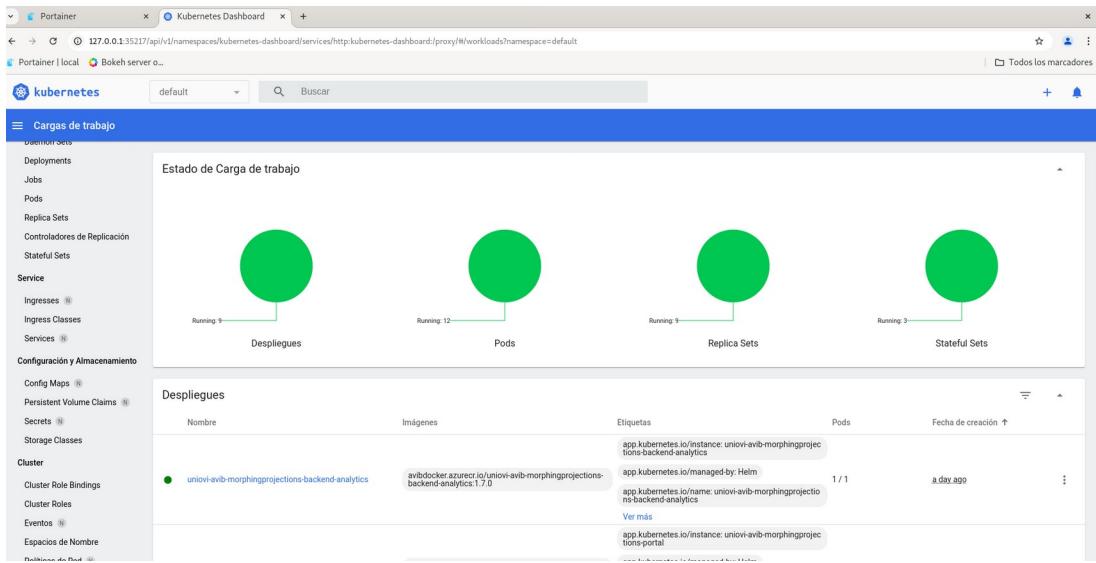
```
$ minikube addon install dashboard
```

Tras unos minutos este addon será instalado, para poder acceder a el deberemos de crear un reverse proxy temporal. Esto lo haremos así porque por defecto el Dashboard viene sin seguridad alguna, y el acceso al mismo será temporal durante el despliegue del sistema, por ello utilizaremos un proxy temporal creado por el CLI minikube dará acceso al servicio del Dashboard, solamente desde el navegador del Host y nunca desde el exterior del mismo. En cuanto terminemos de monitorizar nuestro cluster podemos parar este proxy temporal en cuanto queramos. Vamos a crear este proxy temporal y ver un poco por encima la herramienta de Dashboard:

```
$ minikube dashboard
  Verifying dashboard health ...
  Launching proxy ...
  Verifying proxy health ...
  Opening http://127.0.0.1:35217/api/v1/namespaces/kubernetes-dashboard/services/
http:kubernetes-dashboard:/proxy/ in your default browser...
```

Esta es una herramienta Web visual por lo tanto deberemos de tener en nuestro Host un navegador Web, que abrirá automáticamente una pestaña con acceso al Dashboard desde localhost y puerto aleatorio 35217. Este puerto es aleatorio como comenté por lo que si paramos el proxy y lo arrancamos posteriormente este puerto será diferente. Pero como comentamos esta es una herramienta temporal que nos ayuda a ver los recursos y estado de los recursos de nuestro cluster, pero como comenté, el CLI kubectl ofrece lo mismo y mas pero desde la consola de Linux.

En la siguiente captura podemos ver el Dashboard con todos los recursos y sus estados arrancados por defecto en el sistema. En este documento no se va a hablar de como manejar Kubernetes por estar fuera del ámbito del documento, pero para mayor detalle existe mucha documentación sobre esta herramienta, empezando por la oficial de [Kubernetes](#) a infinito de links y libros sobre el tema.



- **Paso 02: Instalar las extensión Kubernetes Ingress:**

No voy a extenderme demasiado a la hora de definir que es Ingress, pero podemos resumir que Ingress es una especificación creada para Kubernetes en donde se explica como debe de implementarse un proxy dentro de Kubernetes. Como toda especificación esta puede ser implementada por muchas empresas, en nuestro caso escogeremos la de referencia implementada por NGINX, por ser una de las primeras opciones ofrecidas como addon por Minikube y la mas madura de otras soluciones como Istio o HAProxy. Como en el caso anterior por ser un addon sencillito instalarla ejecutando este comando

```
$ minikube addons install ingress
```

Tras uno segundos el servicio del ingress estará desplegado en el cluster. De el hablaremos mas adelante cuando definamos las reglas de redireccionamiento a los servicios que el frontend debe de tener acceso. Por ahora podemos decir que Kubernetes ya cuenta con reverse proxy interno que podrá ser configurado por medio de una serie de reglas con un lenguaje específico del ingress.

Instalación gestor paquetes de Kubernetes

Introducción

Ahora que ya tenemos el cluster de Kubernetes funcionando con los addons ya instalados y los clientes de gestión de Minikube y Kubernetes también instalado, vamos a instalar otro CLI más llamado **helm**. Helm es una herramienta que sirve para instalar paquetes de Kubernetes, entiendo por paquetes como un conjunto de recursos de Kubernetes relacionados entre si y necesarios para que un servicio pueda correr correctamente dentro del entorno de Kubernetes.

Dependiendo del servicio que vayamos a instalar este necesitará unos recursos u otros como por ejemplo: deployments, Pods, containers, services, volume claims, configmaps, secrets, service accounts o roles entre otros muchos. La necesidad de tener que instalar todos estos recursos de forma ordenada y de tener que ser actualizados igualmente si fuera necesario, hace que el uso de herramientas como Helm sean muy útiles. Esta herramienta no solo facilita el despliegue de estos paquetes sino que maneja el estado de las mismas llamadas releases, que son la instancia de un paquete de Helm llamados charts. En este documento no se va a explicar con detalle como crear paquetes chart para ser manejados por la herramienta Helm, pues está fuera del ámbito de este documento, pero para mayor detalle recomiendo leer la documentación oficial de la herramienta [Helm](#).

En este apartado voy a explicar brevemente como instalar el CLI de Helm con el cual podremos interactuar con Kubernetes a la hora de manejar estos paquetes Charts desplegándolos y actualizándolos si fuera necesario.

Como sabemos el sistema esta basado en el patrón de microservicios, donde cada uno de ellos está empaquetado como un paquete chart, preparados para ser desplegado en Kubernetes. Por lo que no solo utilizaremos esta herramienta inicialmente a la hora de desplegar los servicios de infraestructura del sistema ya comentados:

- Bases de Datos.
- Object Storage.
- Servicio de Autenticación y Autorización.

Sino que también será utilizado a la hora de desplegar todos los servicios de negocio propios del sistema tanto de backend como de frontend.

Instalación del CLI de Helm

Para instalar el CLI de Helm será tan sencillo como los otros CLI ya instalados de Minikube o el de Kubernetes. En este caso nos bajamos el CLI para nuestro entorno y

```
$ curl -fsSL -o get_helm.sh https://raw.githubusercontent.com/helm/helm/main/scripts/get-helm-3  
$ chmod 700 get_helm.sh  
$ ./get_helm.sh
```

lo instalamos en el sistema con estos comandos:
Ahora podemos probar que funciona correctamente

Este comando lista todas la releases deplegadas en el sistema, es decir todos los paquetes tipo Chart desplegados, en nuestro caso no hay ninguno, pero el resultado

```
$ helm list
NAME          NAMESPACE  REVISION  UPDATED
STATUS        CHART      APP VERSION
```

indica que funciona correctamente, pues este CLI se alimenta la igual que el CLI de Kubernetes kubectl de la configuracion activa antes descrita y por ellos Kubernetes le esta respondiendo correctamente sin ninguna release desplegada.

Al final cuando despleguemos todos los microservicios veremos que aparecerán muchas releases unas pertenecientes a nuestros servicios de negocio y otras pertenecientes a los servicios de infraestructura antes citados.

Despliegue servicio Database: MongoDB

Como se ha comentado podemos agrupar los servicios dentro del cluster en dos grupos:

- **Servicios de infraestructura:** son todos los servicios que no implementan lógica de negocio de nuestro dominio, pero que son necesarios o de apoyo a la hora de hacer posible esta implementación, por ejemplo implementar el estado de los servicios que lo requieran por medio de bases de datos, o object storages, o sistemas que implementan la autenticación Oauth como es el caso del servicio Keycloak
- **Servicios de negocio:** son por contra aquellos servicios que si implementan la lógica de negocio de nuestro dominio. En nuestro caso son todos los microservicios de backend y frontend que listaremos posteriormente en próximos capítulos, cuando hablemos de como desplegar esta lógica de negocio.

En este caso vamos a centrarnos en uno de estos servicios de infraestructura encargado de manejar el estado del sistema. Como es sabido el patrón microservicios aconseja utilizar una base de datos independiente para cada uno de estos microservicios, por temas relacionados con la escalabilidad y desacoplamiento dentro del modelo de datos. Nosotros no hemos seguido estrictamente este patrón, pues estamos utilizando en este caso una sola base de datos utilizada por dos microservicios implementada por la base de datos no relacional llamada MongoDB

Ala hora de desplegar este servicio hemos utilizado el paquete de heml standar mantenido por MongoDB, para que el despliegue sea lo mas sencillo posible, pues el número de recursos de Kubernetes necesarios para poner en marcha una base de datos como esta en Kubernetes en amplio y complejo. El uso de estos paquetes hacen que este despliegue sea mucho mas sencillo y mantenible. El comando que debemos ejecutar tratándose de un paquete helm es este:

```
$ helm install avib-mongodb oci://registry-1.docker.io/bitnamicharts/mongodb
```

Como podemos ver no hemos modificado ninguno de los parámetros que por defecto define el paquete. Normalmente estos paquetes son altamente configurables y se peude consultar todos estos parámtrios desde el link oficial del paquete helm en este [link](#). Este paquete esta gestionado por el repositorio público de paquetes de helm llamado bitnami. Por defecto este repositorio no viene configurado con el CLI de helm, por lo que debemos añadirlo antes de ejecutar el anterior comando:

```
$ helm repo add bitnami https://charts.bitnami.com/bitnami
```

Por defecto no hemos escogido el namespace en donde desplegar estos recursos, pero escogera el de por defecto en Kubernetes llamado default. Tras unos segundos todos los recursos de la base de datos estarán desplegados en el cluster como se puede ver en la imagen siguiente:

Configuración y Almacenamiento					
Nombre	Imagenes	Etiquetas	Pods	Fecha de creación	
gsdipi-mongodb	docker.io/bitnami/mongodb:7.0.14-debian-12-r3	app.kubernetes.io/component: mongodb app.kubernetes.io/instance: gsdipi-mongodb app.kubernetes.io/managed-by: Helm	1 / 1	a.month.ago	

Pods							
Nombre	Imagenes	Etiquetas	Nodo	Estado	Reinicios	Utilización de CPU (núcleos)	Utilización de memoria (octetos)
gsdipi-mongodb-6fcc8c88f6-b8q8b	docker.io/bitnami/mongodb:7.0.14-debian-12-r3	app.kubernetes.io/component: mongodb app.kubernetes.io/instance: gsdipi-mongodb app.kubernetes.io/managed-by: Helm	gsdipi	Running	0	-	-

Replica Sets					
Nombre	Imagenes	Etiquetas	Pods	Fecha de creación	
gsdipi-mongodb-6fcc8c88f6	docker.io/bitnami/mongodb:7.0.14-debian-12-r3	app.kubernetes.io/component: mongodb app.kubernetes.io/instance: gsdipi-mongodb app.kubernetes.io/managed-by: Helm	1 / 1	a.month.ago	

Una vez desplegada la base de datos, debemos de recuperar las credenciales y el nombre del servicio, pues estos dos datos serán los que deberemos de utilizar a la hora de configurar los microservicios que tengan que acceder a este recurso.

Las credenciales se pueden obtener del secreto creado en el proceso de despliegue llamado **gsdipi-mongodb** como se puede ver en la siguiente imagen, este es la clave del usuario de root con su clave, y esta será la cuenta que utilizaremos para nuestros microservicios:

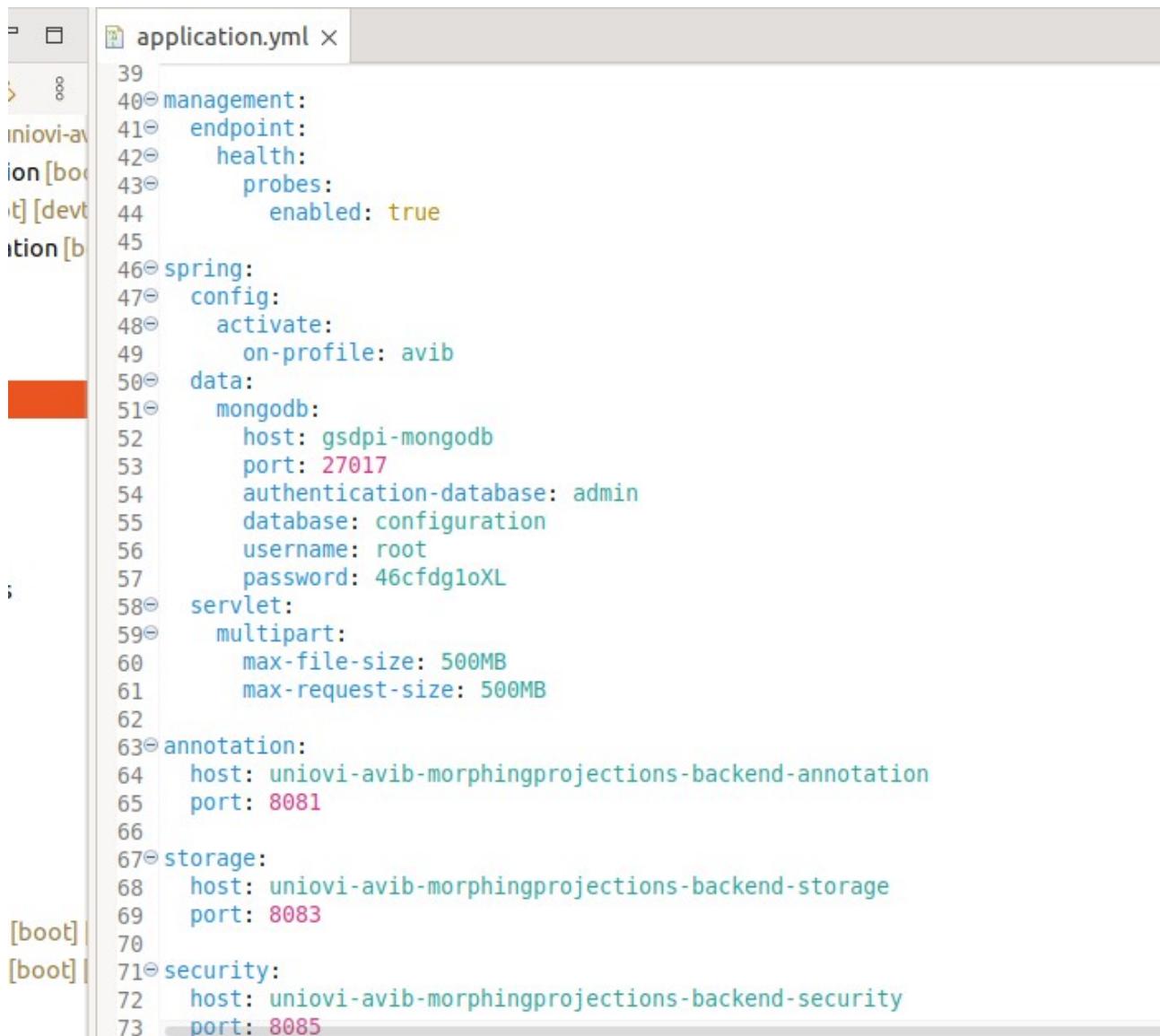
Configuración y Almacenamiento					
Nombre	Espacio de nombre	Fecha de creación	Edad	UID	
gsdipi-mongodb	default	2.oct.2024	a.month.ago	a6397fb5-e152-42ed-bcb1-04c0eeb4fcfd	

Datos					
mongodb-root-password	46cf910XL				

Por último el servicio representa el nombre DNS interno que estos microservicios utilizaran para comunicarse con la base de datos, lo podemos consultar bajo el

servicio creado por el paquete helm llamado **gsdpi-mongodb**. Este nombre es el que utilizaremos como hemos dicho a la hora de configurar los micros de Java y Python que tengan que conectarse a la base de datos para manejar su estado.

Por ejemplo aquí podemos ver la configuración del microservicio de organización que gestiona su estado gracias a esta base de datos como se puede ver en la imagen inferior para el entorno de producción:



The screenshot shows a code editor with the file "application.yml" open. The code is color-coded to highlight different sections and values. The configuration includes management, spring, data (with a mongoDB section), servlet, annotation, storage, and security sections. The mongoDB section specifies host as "gsdpi-mongodb", port as "27017", authentication database as "admin", database as "configuration", username as "root", and password as "46cfdgloXL". The servlet section sets max-file-size and max-request-size both to "500MB". The annotation section binds the service to host "uniovi-avib-morphingprojections-backend-annotation" and port "8081". The storage section binds it to host "uniovi-avib-morphingprojections-backend-storage" and port "8083". The security section binds it to host "uniovi-avib-morphingprojections-backend-security" and port "8085". The code editor interface shows a sidebar with project navigation and a status bar at the bottom.

```
39
40 management:
41   endpoint:
42     health:
43       probes:
44         enabled: true
45
46 spring:
47   config:
48     activate:
49       on-profile: avib
50   data:
51     mongodb:
52       host: gsdpi-mongodb
53       port: 27017
54       authentication-database: admin
55       database: configuration
56       username: root
57       password: 46cfdgloXL
58   servlet:
59     multipart:
60       max-file-size: 500MB
61       max-request-size: 500MB
62
63 annotation:
64   host: uniovi-avib-morphingprojections-backend-annotation
65   port: 8081
66
67 storage:
68   host: uniovi-avib-morphingprojections-backend-storage
69   port: 8083
70
71 security:
72   host: uniovi-avib-morphingprojections-backend-security
73   port: 8085
```

Despliegue servicio IAM: keycloak

En este apartado vamos a explicar como desplegar este otro servicio de vital importancia desde el punto de vista de la seguridad del sistema, pues es el que implementa la especificación OAuth 2.1. Esta herramienta desarrollada y mantenida por Red-Hat es la versión Open Souce de su hermano comercial llamado RH-SSO desplegable en Openshift.

Al igual que en el caso anterior, el despliegue de este tipo de servicios es complejo y por ellos nos hemos apoyado nuevamente en los paquetes de helm mantenido en este caso por Red-Hat. Al igual que en el caso anterior este paquete está mantenido por el repositorio privado de Bitnami, por lo que el repo ya lo tendremos dado de alta, como hemos explicado en el punto anterior. En este caso el comando que ejecutaremos será este:

```
$ helm install avib-keycloak --values values.yaml oci://registry-1.docker.io/bitnamicharts/keycloak
```

En este caso si hemos configurado algunos atributos que por defecto no existen y son de vital importancia, pues en este caso Keycloak debe de ser accesible através de un proxy corriendo en el host implementado por HAProxy. Estos parámetros se definen en un fichero que por defecto se llama values.yaml, en donde se pueden configurar estos parámetros extras necesarios en nuestro caso. El contenido de este fichero es este:

```
auth:  
  adminUser: admin  
  adminPassword: password  
  
proxyHeaders: forwarded
```

En este caso hemos definido:

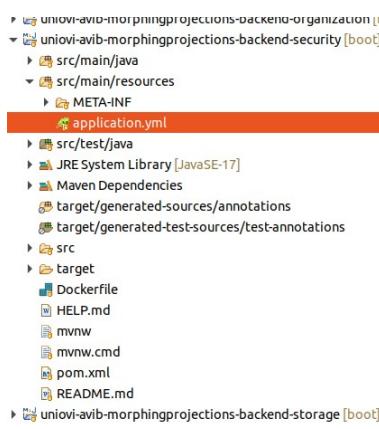
- Las credenciales del usuario admin con privilegios al Admin Console de Keycloak. Esta herramienta Web es la encargada de gestionar keycloak. La utilizaremos inicialmente para:
 - Crear nuestro Tenant, llamados por Keycloak como realms,
 - Crear los roles por defecto que el sistema maneja: ADMIN, USER y GUEST.
 - Crear el cliente que representa al microservicio que tiene acceder a Keycloak para poder autenticar a los usuarios, es decir, el microservicio Portal.
 - Existe otro microservicio que también necesita autenticarse actuando tambien como cliente, este microservicio es el de Security, pues es el encargado de gestionar los usuarios de nuestras organizaciones en el sistema, creando, actualizando o borrando los mismos. También este micro se encarga de modificar nuestra clave si fuera necesario. En este caso utilizaremos la misma cuenta de admin con privilegios suficientes a la hora de manejar estos recursos dentro del servicio de Keycloak.

Para poder acceder al Admin Console hemos creado una regla de ingress para que su acceso sea directo desde internet. El acceso a esta herramienta es importante, en caso de que tengamos que borrar crear cuentas de emergencia, aunque como ya se ha comentado anteriormente el propio sistema AVIB cuenta con un módulo encargado de manejar estos recursos: usuarios y claves.

El acceso al Admin Console de Keycloak es a través de esta URI
<http://avispe.edv.uniovi.es/>

La cuenta de admin es la configurada previamente en el fichero de values.yaml. Mucho cuidado con cambiar esta clave por defecto, pues ya se ha comentado que el microservicio de Security la utiliza, y el cambio de la misma provocará que este micro ya no pueda desarrollar su trabajo. En caso de querer modificarla deberemos de modificar este dato en el micro y redesplegarlo.

Aquí se puede ver esta configuración en el microservicio de Security:



```

values:
  8    enabled: true
  9
10  keycloak:
11    server-url: http://localhost:8088
12    admin-username: admin
13    admin-password: password
14
15  ...
16  server:
17    port: 8085
18
19  management:
20    endpoint:
21      health:
22        probes:
23          enabled: true
24
25  spring:
26    config:
27      activate:
28        on-profile: avib
29
30  keycloak:
31    server-url: http://gsdipi-keycloak:80
32    admin-username: admin
33    admin-password: password

```

Una vez desplegado Keycloak debemos de configurarlo utilizando la herramienta llamada **Admin Console** debemos de crear estos recursos para nuestro tenant

- Primero creamos un realm llamado avib, como se puede ver en la imagen inferior. Este dato es importante pues será utilizado por el microservicio Portal para autenticar cualquier usuarios

- Una vez creado este realm, ya podremos crear los siguientes recursos. Primero los roles del realm, que como sabemos son tres: ADMIN, USER y GUEST.

- Y por último el cliente, que utilizará este realm para poder autenticar a los usuarios y gestionar sus AccessTokenes de seguridad. Este cliente se llamará **portal-cli** y será otro dato importante a la hora de configurar el micro de portal. Debemos de configurar la URI que representa este cliente en nuestro caso **avispe.uniovi.es** como se puede ver en la captura inferior

- Por último vamos a crear un solo usuario administrador llamado **administrator** con clave **password**, con role ADMIN. Este usuario será único en el sistema y es el único que puede crear organizaciones. El sistema solo permite crear usuarios de tipo USER y GUEST que pueden crear todo tipo de recursos a excepción de Organizaciones. De esto se hablará mas en detalle en el documento funcional que acompaña este documento técnico.

Despliegue servicio Object Storage: Minio

Ya por último será necesario instalar el Object Storage implementado por Minio. Este servicio se encarga de manejar todos los datasets de nuestros casos en formato csv. Igualmente el resultado de la proyección de los mismo utilizando el algoritmo t-SNE o siendo previamente ya creado externamente a la herramienta, serán también ficheros csv manejados por esta herramienta de forma segura y consumidos por los clientes Portal, servicios de analítica y por los procesos Jobs encargados de proyectar los datasets de entrada. Para mayor detalle sobre estos datasets y su forma, se aconseja leer los primeros capítulos de este documento.

Como en casos anteriores este servicio consta de varios recursos de Kubernetes, difíciles de ser desplegados de forma individual, por lo que buscaremos el paquete de helm que nos facilite las cosas. Este caso es un poco diferente al anterior, pues a la hora de desplegar el servicio de Minio utilizaremos otra especificación de Kubernetes llamada Operadores. Estas aplicaciones son constructores de aplicaciones, en nuestro caso este operador se encarga de crear tenants de Minio, concepto semejante al hablado en el capítulo anterior, y al mismo tiempo cuenta con una herramienta capaz de monitorizar y gestionar visualmente todos los buckets (carpetas) y keys (ficheros) almacenados por Minio. Es una herramienta útil para poder ver estos recursos, aunque el sistema AVIB cuenta con su propio módulo de gestión de recursos integrados con los casos. **Por lo que no se recomienda ni crear ni borrar estos recursos directamente desde Minio**, pues esta herramienta como es lógico no está integrada con AVIB y no sabe nada relacionado con los casos. Se puede utilizar para monitorizar y visualizar los recursos, pero nunca modificarlos.

Para despegar Minio ejecutaremos este comando de helm:

```
$ helm install avib-minio-operator --namespace minio-operator --create-namespace minio-operator/operator
```

En este caso el repositorio utilizado no será el de Bitnami sino el oficial de Minio por lo que deremos de registrar este repositorio antes de ejecutar el comando anterior:

```
$ helm repo add minio-operator https://operator.min.io
```

Tras unos segundos todos los recursos del operador serán instalados en este caso bajo el namespace de **minio-operator**. Se ha escogido este namespace pues esta herramienta no interviene directamente en la implementación del sistema, sino que es una herramienta auxiliar necesaria para crear el Tenant. Al igual que en caso anterior tras desplegar el operador debemos de crear el tenant para AVIB con uno recursos concretos. A diferencia del caso anterior esta herramienta no será accesible externamente, evitando puntos de acceso no necesarios, por lo que para poder acceder temporalmente para crear este Tenant, crearemos en el host un reverse proxy temporal como se ve en este comando:

```
$ kubectl --namespace minio-operator port-forward svc/console 9090:9090
```

El puerto de acceso al operador es 9090, para poder acceder a esta herramienta Web necesitamos la credenciales de admin, a diferencia al caso anterior hemos dejado que el paquete de helm cree un token aleatorio en este despliegue que puede ser consultado como secreto en el namespace de minio-operator. Este secreto se llama **console-sa-secret** y si lo abrimos utilizando el Dashboard veremos el valor de este token como se en la captura inferior:

The screenshot shows the Kubernetes Dashboard interface. In the top navigation bar, 'minio-operator' is selected from the dropdown. Below the navigation, the path 'Config And Storage > Secrets > console-sa-secret' is visible. The main content area is titled 'Secrets' and displays the details for the 'console-sa-secret' entry. The 'Metadata' section shows the secret's name ('console-sa-secret'), namespace ('minio-operator'), creation date ('2 oct. 2024'), and age ('a month ago'). It also lists annotations and labels. The 'Data' section shows two entries: 'ca.crt' (1111 bytes) and 'token' (14 bytes). The 'token' value is displayed as a long string of characters.

Nombre	Espacio de nombre	Fecha de creación	Edad	UID
console-sa-secret	minio-operator	2 oct. 2024	a month ago	49fee968-989d-437a-afe0-3b271863e820

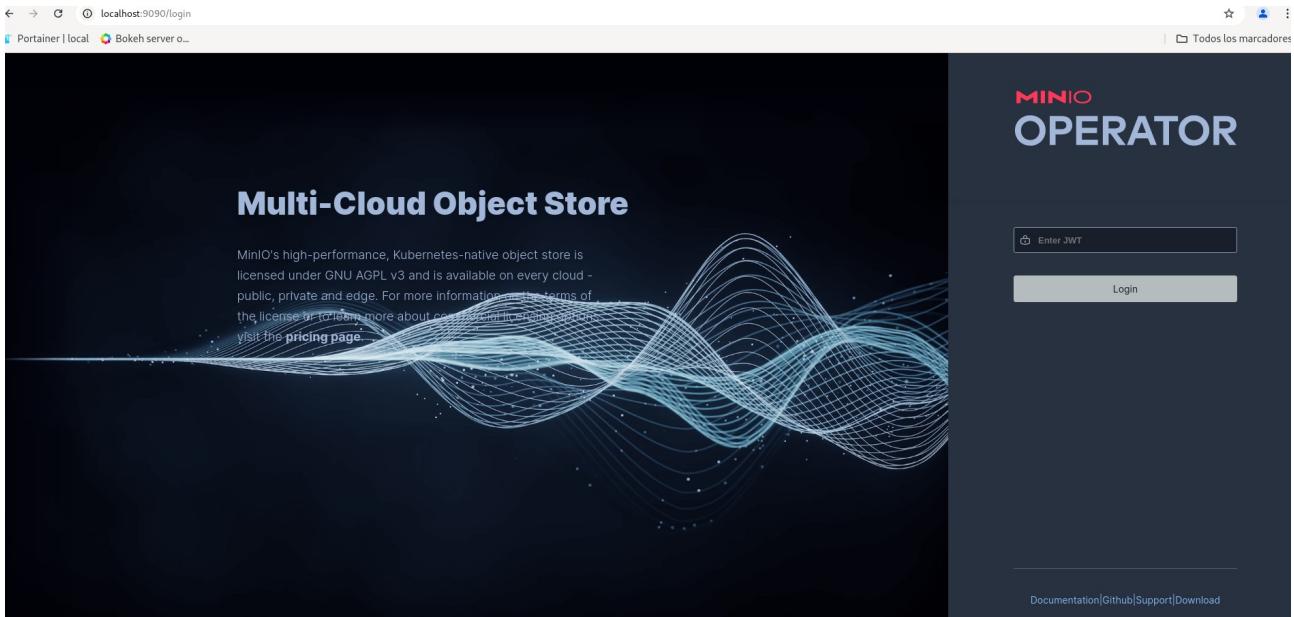
Annotations: kubernetes.io/service-account.name: console-sa, kubernetes.io/service-account.uid: 6eedd2e6-9da4-44ff-b244-9c93cf851142, meta.helm.sh/release-name: gspi-minio-operator, meta.helm.sh/release-namespace: minio-operator

Data:

- ca.crt: 1111 bytes
- token: 14 bytes

```
eyjhGc1DjSUzT1nLltIotpZCI0t1Fcc314UFRlU1R8ImpXTUd]P02w0EDK2kg2vneUuUnZB188dcJGV8dFMtk1f0 eyJp-3M01J7dw1caes1d0v2L3H[ln2jy2uNvY2Nv0f501Iw13V1Z2Q01cvppby9zXZ38Mn1vnu0v3u01c3 Bhv2U0101J1taW5pby1vc0lyXXrvcl1c1mt1Ym1bvn0Z2Muaw0vC2ydm1j2nfj7z91bn0vC2V1cmv0lmshhu101j)h52zbx1LXNhNXL1y1j1dc1st1nt1myVbni02Xeaaw8v-c2ydnij2wfj)yc291brndyC2Yyml1Z51h12hvvdsl0mshhu101j j125zbx2LXLXm11w1a3v1Z2X01ZXR1c5pb92ZxJ2wN1Wn1j53Vidc91Z2J2awN1lWFjY291bn0udwlk1j1oinV1Z60yZTy0WmNCNC00NG2m1W1yN0t0tWm5K2m0d0UxHTQ11w1c3V11jol1c31zdgvt0n1lcn2p1Zhy2Nwd500m1pbmvlV1w9w ZXJhdG9yOmNvbnlvbGUc2E1f0_AXrGePU1RpLx2Kyv050rXW2pFpxz7Nlp0z51Ayeb06c14q7x1Yd1bJd006c21Mvn1qe96n7mdC1psFp4x8zvPct74nBtxtvCvk0d0umUNDt1HbN9nRoFb28WpL04naayryvyl0PEz1dsRw9gWYMo DLK0XWMwM114x8qbDvqHP44MIVZUC_ofv1WxwhUkvuMtbf5GLpcat7h6fjY1vNRCMzHb1nk0BHZZLjKKqaFwmG5RLoopvUrwsZzxA97d0cn0312VgkgEHdx34-Y3R5r0dWnfh2d2Zr5fDU0rA5cNScE]F4CtmVjvts3-MgPBhXEFPatuu5Ef20g
```

Este valor será el utilizado a la hora de acceder a la consola de Administrador del Operador de Minio a través de la URI <http://localhost:9090>



Introducimos este token y ya podremos crear el Tenant para nuestro sistema:

Aquí podemos ver los valores escogidos a la hora de crear ese Tenant:

Estos son los datos del tenant:

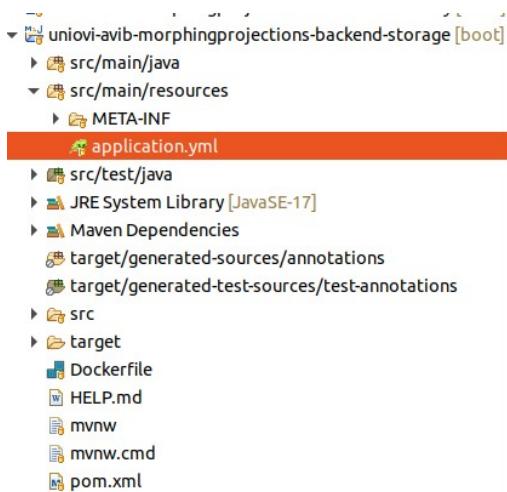
- Name: uniovi
- Namespace: default
- Number of Servers: 1
- Drives per Server (Volumes): 4

- Total Size (Sixe per drive): 30Gb
- Erasure Code Priority: EC:2 (Default).
- CPU Request: 2.
- Memory Request: 2.

Cuando guardemos el operador va a crear otros recursos de Kubernetes que representan al tenant, bajo el puerto por defecto 9000 en el namespace default, junto a los otros recursos. Al igual que en el caso de MongoDB el acceso a este recurso se guarda en forma de secretos que podemos recuperar, pues estos serán las credenciales utilizadas por el microservicios que se comunican con Minio. El microservicio de storage utilizado por:

- El Portal para gestionar la ingesta, gestión de los recursos de los casos y visualización del resultado proyectados
- El microservicio que implementa el algoritmo de proyección t-SNE, pues debe recuperar estos recursos y proyectarlos en 2D siguiendo la configuración previamente creada para nuestros casos.
- El microservicio que implementa algoritmos de analítica tipo histogramas y regresiones lineales sobre las muestras de los datasets ingestados previamente.

Para recuperar estas credenciales en el momento de crear el tenant el sistema te avisa que va a crear unas credenciales (access_key, secret_key) en formato json y que las guardes, pues estas serán las utilizadas a la hora de configurar el acceso al tenant recién creado para el sistema AVIB. Por ejemplo en esta captura se puede ver como se configura el microservicio de storage con esta credenciales :



```

30
31 management:
32   endpoint:
33   health:
34   probes:
35     enabled: true
36
37 spring:
38   config:
39   activate:
40     on-profile: avib
41
42 servlet:
43   multipart:
44     max-file-size: 500MB
45     max-request-size: 500MB
46
47 minio:
48   host: uniovi-hl
49   port: 9000
50   disable-tls: true
51   access-key: RS6P1ApnSrDf9MaS
52   secret-key: weQJEUUEVfNZV1Zs0MnA0e9CT2d71jSK
53

```

Configuraciones Kubernetes post-despliegue

Una vez desplegado el cluster de Kubernetes, y desplegados los addons necesarios para monitorizar los despliegues y permitir el acceso externo al cluster a nuestros servicios por medio del Ingress. Solamente nos queda por configurar un solo recurso muy importante y este es el acceso al Registro Privado de Contenedores de Azure.

Como explicaremos con mas detalles en capitulos posteriores, todos nuestros servicios de negocio, están empaquetados como imágenes de docker que pueden ser desplegadas en el cluster de Kubernetes. Pero estas imágenes residen en un repositorio espacial capaz de manejar y servir estas imágenes. Pero este repositorio es privado como es lógico, controlado por la infraestructura de Azure bajo una cuenta o service account con sus credenciales. En el momento que queramos desplegar una de estas imágenes utilizando un paquete de helm, este servicio encargado de desplegar todos los recursos necesarios, no solo la imagen sino otros mas, que se hablaron de ellos posteriormente, tiene que conectarse al registro privado e Azure con estas credenciales.

Estas credenciales se guardan como un recurso especial dentro del cluster llamado **Image Pull Secret**, que representa las credenciales de acceso a la infraestructura de Azure y en concreto al servicio de Container Registry donde residen todas la paquetes de helm y las imágenes de docker que maneja cada uno de ellos. Este recurso ha de ser desplegado manualmente desde la consola utilizando el CLI de Kubernetes ejecutando este comando:

```
$ kubectl create secret docker-registry acr-avib-secret --docker-server=avibdocker.azurecr.io --  
docker-username=avibdocker --docker-  
password=BAqBdHVbrSmPOxH96IGHlcze7gx8lclsWJNxUFyx/c+ACRB1+L5M
```

Este comando indica el nombre que le hemos dado a estas credenciales, en nuestro caso **acr-avib-secret**, este nombre es importante, pues este dato deberá de ser configurado en todos los paquetes de helm, para indicar a este servicio donde se encuentran las credenciales que debe utilizar para validarse contra Azure y bajar los paquetes de helm y las imágenes de docker que maneja cada uno de ellos.

Igualmente se indica el nombre del host asociado con el espacio dentro del servicio privado del Container Registry que hemos creado en Azure para gestionar nuestros artefactos: imágenes y paquetes de helm llamado **aviddocker.azurecr.io** y por último las credenciales como es de esperar el username **avibdocker** y la password de este servicio que nos da acceso al mismo.

Tenemos que tener en cuenta que este secreto con estas credenciales debe de existir en todos los namespace en donde despleguemos nuestros servicios. Como el cluster es solamente utilizado por los servicios del sistema AVIB, todos los servicios son desplegados en el namespace por defecto llamado default. Es aquí en este namespace donde debemos de crear este recurso como se puede ver en la imagen siguiente:

The screenshot shows the Kubernetes Dashboard interface. The left sidebar has a tree view with 'Config And Storage' expanded, showing 'Secrets' selected. The main area shows the details for a secret named 'acr-avib-secret' in the 'default' namespace.

Metadatos

Nombre	Espacio de nombre	Fecha de creación	Edad	UID
acr-avib-secret	default	2.oct.2024	a month ago	b8615ab7-3162-4f9d-b50e-ed2a6e1ed7c9

Datos

```
dockerconfig.json ✓
{
  "auths": {
    "avibdocker.azurecr.io": {
      "username": "avibdocker",
      "password": "BAqBdHVbr5mP0xH96lGhlcz7gx8lcIsWJNxUFyx/c+ACRB1+L5M",
      "auth": "YXZpYmRvY2tlcjpcQXFCEhWYnJTbVBPeEg5NmxHSGxjemU3Zg4b6NJc1dKTnhVRn14L2MrQUNSOjErTDVN"
    }
  }
}
```

Configuración reverse-proxy: HAProxy

Introducción

Como ya se ha comentado la herramienta AVIB esta corriendo en un nodo compartido por otras herramientas. Igualmente el acceso al cluster debe ser direccionado por un proxy, en nuestro caso hemos escogido HAProxy por estas razones:

- Es una herramienta OpenSource ampliamente utilizada en el mercado, con un soporte duradero.
- Tiene una documentación excepcional, si la comparamos con otras opciones como son Nginx o Apache Web.
- Tiene una comunidad amplia con muchos ejemplos, que pueden ser consultados fácilmente
- Tiene una sintaxis y configuración, mucho mas amigable sin la comparamos con las otras herramientas.
- Es fácilmente empaquetable como contenedor Docker
- A nivel de performance es muy parecido a NGInx y superior a Apache

Configuración kubernetes ingress

A la hora de configurar las reglas del ingress debemos de tener muy presente que servicio vamos a configurar y que conexión tiene con el proxy del Host implementado por HAProxy.

En principio de todos los servicios que corren en kubernetes solamente vamos a exponer tres servicios, aunque realmente podríamos hacer solo dos. Vamos a listar estos servicios:

- **Portal del sistema:** este servicio de negocio de frontend implementado bajo el nombre **uniovi-avib-morphingprojections-portal:<VERSION>** representa el frontend del sistema, es decir, la interfaz gráfica utilizada por los usuarios para interactuar con el sistema.
- **Gateway del sistema:** este servicio de negocio de backend implementado bajo el nombre **uniovi-avib-morphingprojections-backend<VERSION>** representa la puerta de entrada a todos los servicios de negocio del sistema, utilizado por portal para poder interactuar con el backend en todo momento y el resto de servicios de infraestructura del mismo
- **Gestor de autenticación y autorización del sistema:** este servicio de negocio de seguridad implementado por la herramienta Keycloak, bajo el nombre **gsdpi-keycloak-0<VERSION>** es el servicio encargado de gestionar las cuentas de usuario y la autenticación y autorización de las mismas implementando la especificación OAuth 2.1

Deberemos de crear una regla de redirección para cada uno de estos servicios tanto para cargar el portal, como para que este pueda autenticarse y posteriormente pueda acceder a todos los recursos del sistema implementados por todos los microservicios de negocio a través del gateway.

Aunque Ingress es una especificación como ya se ha comentado la forma y implementación de la misma puede diferenciarse un poco de un producto a otro. Nosotros utilizaremos el ingres de NGInx por ser la de referencia y la ofrecida por Minikube como addon.

Reglas de ingress

- **Paso 01: Regla de ingress para servicio Keycloak**

En este caso el servicio de Keycloak está desplegado en el Webcontext / (raiz), por lo que en este caso la regla para este servicio se puede expresar como esto bajo el fichero llamado avispe-keycloak.ingress.yaml

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: avib-keycloak-ingress
spec:
  rules:
  - host: avispe.edv.uniovi.es
    http:
      paths:
      - path: /
        pathType: Prefix
        backend:
          service:
            name: gsdpi-keycloak
            port:
              name: http
```

Lo más destacado de esta regla es:

- El nombre del **host** que debe de coincidir con el DNS ofrecido por el departamento a la hora de identificar nuestros servicios
- El **path** absoluto / (raíz) bajo el cual el servicio de Keycloak esta desplegado en Kubernetes
- El **nombre y puerto del servicio** de kubernetes representando a la herramienta Keycloak desplegada, en nuestro caso se llama **gsdpi-keycloak** y el puerto es el representado por el nombre **http** del mismo, en concreto se puede ver en kubernetes que es el 80, pero utilizando el nombre de forma indirecta es mas sencillo desacoplándonos del numero en concreto.

- **Paso 02: Regla de ingress para servicio Portal**

En este caso el servicio de Portal está desplegado en el Webcontext **/morphingprojections-portal**, por lo que en este caso la regla para este servicio se puede expresar como esto bajo el fichero llamado avispe-portal.ingress.yaml:

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: avispe-portal-ingress
  annotations:
    nginx.ingress.kubernetes.io/use-regex: "true"
    nginx.ingress.kubernetes.io/rewrite-target: /$2
spec:
  rules:
    - host: avispe.edv.uniovi.es
      http:
        paths:
          - path: /morphingprojections-portal(/|$(."))
            pathType: ImplementationSpecific
        backend:
          service:
            name: uniovi-avib-morphingprojections-portal
            port:
              name: http
```

Lo mas destacado de esta regla es:

- El nombre del **host** que debe de coincidir con el DNS ofrecido por el departamento a la hora de identificar nuestros servicios al igual que antes.
- El **path relativo morphingprojections-portal** bajo el cual el servicio de Keycloak esta desplegado en Kubernetes y todas las páginas ofrecidas por el mismo. De ahí el utilizar reglas de tipo regex para el redireccionamiento
- El **nombre y puerto del servicio** de kubernetes representando a la herramienta Keycloak desplegada, en nuestro caso se llama **uniovi-avib-morphingprojections-portal** y el puerto es el representado por el nombre **http** del mismo.

- **Paso 03: Regla de ingress para servicio Gateway**

En este caso el servicio de Portal está desplegado en el Webcontext /morphingprojections-portal, por lo que en este caso la regla para este servicio se puede expresar como esto bajo el fichero llamado avispe-backend.ingress.yaml

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: avispe-backend-ingress
  annotations:
    nginx.ingress.kubernetes.io/use-regex: "true"
    nginx.ingress.kubernetes.io/rewrite-target: /$2
    nginx.ingress.kubernetes.io/proxy-body-size: 500M
spec:
  rules:
    - host: avispe.edv.uniovi.es
      http:
        paths:
          - path: /morphingprojections-backend(/|$(.))
            pathType: ImplementationSpecific
        backend:
          service:
            name: uniovi-avib-morphingprojections-backend
          port:
            name: http
```

Lo más destacado de esta regla es:

- El nombre del **host** que debe de coincidir con el DNS ofrecido por el departamento a la hora de identificar nuestros servicios al igual que antes.
- El **path relativo morphingprojections-backend** bajo el cual el servicio de Keycloak esta desplegado en Kubernetes y todas las páginas ofrecidas por el mismo. De ahí el utilizar reglas de tipo regex para el redireccionamiento
- El **nombre y puerto del servicio** de kubernetes representando a la herramienta Keycloak desplegada, en nuestro caso se llama **uniovi-avib-morphingprojections-backend** y el puerto es el representado por el nombre **http** del mismo.
- En este caso particular hay un **atributo llamado proxy-body-size** extra importante que son los megas máximo permitos a procesar en una petición a

través del ingress en este caso se ha puesto a 500Mb para permitir subir ficheros de entrada grandes.

Todas estas reglas deberán de ser desplegada en kubernetes utilizando el CLI de kubectl con este comando:

```
$ kubectl apply -f avispe-keycloak.ingress.yaml  
$ kubectl apply -f avispe-portal.ingress.yaml  
$ kubectl apply -f avispe-backend.ingress.yaml
```

Una vez desplegadas se pueden consultar utilizando el CLI kubectl o desde el Dashboard de Kubernetes en el namespace default:

The screenshot shows the Kubernetes Dashboard interface in a Google Chrome browser. The title bar indicates the page is at 127.0.0.1:35217/api/v1/namespaces/kubernetes-dashboard/services/http:kubernetes-dashboard/proxy/#/ingress?namespace=default. The left sidebar has a 'Service' section with 'Ingresses' selected, and a 'Cluster' section with various items like Cluster Role Bindings, Cluster Roles, and Events. The main content area is titled 'Ingresses' and lists six entries:

Nombre	Etiquetas	Endpoints	Hosts	Fecha de creación
avib-backend-ingress	-	192.168.59.103	avispe.edv.uniovi.es	10.days.ago
avib-portal-ingress	-	192.168.59.103	avispe.edv.uniovi.es	10.days.ago
avib-keycloak-ingress	-	192.168.59.103	avispe.edv.uniovi.es	16.days.ago
backend-ingress	-	192.168.59.103	minikube.io	22.days.ago
keycloak-ingress	-	192.168.59.103	minikube.io	22.days.ago
portal-ingress	-	192.168.59.103	minikube.io	24.days.ago

Despliegue servicios de negocio

Introducción

Como ya se ha comentado todos los servicios de negocio se han empaquetado como paquetes de tipo Chart para ser desplegados por el CLI de Helm. Todos los servicios están formados por tres recursos:

- **Deployment:** gestión de la replicas del Pod
- **POD:** gestión del contenedor de Docker que empaqueta el servicio de negocio
- **Service:** acceso al servicio desde dentro del cluster y por las reglas del ingres en caso de ser necesario

Solamente el microservicio de Job tiene un recurso extra, llamado **service account**, pues este servicio debe de interactuar con Kubernetes bajo demanda para crear los jobs que el frontend le solicita, por ellos este service account tiene los roles y permisos necesarios para crear estos recursos de Kubernetes llamados Jobs.

Todos estos recursos deberán de ser definidos en un paquete de tipo Chart y empaquetados y publicado como si de una imagen de Docker se tratara en Azure. Es mas todos estos principalmente son responsables de crear el contenedor asociado al mismo y por lo tanto deberán de bajarse la imagen de Docker correspondiente de Azure en el momento de Despliegue.

Pasos a seguir en el despliegue servicios negocio backend Java

Actualmente el sistema no cuenta con una infraestructura CI/CD, por lo que deberemos de desarrollar estos pasos manuales u no gracias a ningún pipeline que automatice estas estapas. Por ellos vamos a listar cada una de estas estapas a la hora de tener que actualizar un microservicio, bien porque tengamos que corregir un error del mismo, o hallamos introducido una mejora. Aunque las estapas a seguir son muy parecidas en backend o frontend sean estas en Java o Python vamos a crear tres listas por separado para mas claridad:

Estas etapas que son 7, se pueden dividir en dos grupos. Este primer grupo de etapas corresponden a la parte del pipeline llamado CI: Continuos Integration (Empaquetado del servicio)

- **STEP01:** Compilación del microservicio de backend bien através de la línea de comandos o bien desde la interfaz de desarrollo que usemos en mi caso Eclipse STS 4.X. En este aso vamos a escoger la línea de comandos, teniendo el cliente de maven ya instalado y configurado como es lógico:

Tras unos segundos ya tendremos nuestro artefacto jar compilado listo para ser empaquetado

- **STEP02:** Compilación y tageo de la imagen de Docker con nuestro jar, para ello cada repositorio ya cuenta con un Dockerfile, por lo que solamente deberemos de añadir el argumento propio del entorno en donde queremos desplegar nuestra image, esta será siempre producción con el argumento llamado avib y un tag con una versión que será la siguiente que le corresponda. Actualmente no se sigue una política de version estricta, pues es un proceso manual, yo actualmente estoy sumando uno al minor de la version:

```
$ docker build --build-arg ARG_SPRING_PROFILES_ACTIVE=avib -t uniovi-avib-morphingprojections-backend-organization:1.5.0 .
```

- **STEP03:** Publicar la imagen en el Registro privado de imágenes de Azure. Calro está deberemos de tener configurado el acceso a Azure correctamente, como se explica en el último capítulo de este documento:

```
$ docker push avibdocker.azurecr.io/uniovi-avib-morphingprojections-backend-organization:1.5.0
```

- **STEP04:** Ahora que ya tenemos nuestra imagen subida en el registro privado de Azure, vamos a crear el paquete de helm que nos permite desplegar el mismo en un cluster de Kubernetes. En este caso debemos de buscar el repositorio con el mismo nombre que el microservicio que hemos publicado anteriormente pero con el sufijo chart, que indica que es un repositorio con los scripts de helm. Dentro de el ejecutamos

Borramos el anterior paquete creado en el repositorio

```
$ rm uniovi-avib-morphingprojections-backend-organization-1.4.0.tgz
```

Modificamos la version de la imagen y del paquete que vamos a crear. Estas versiones se encuentran en el fichero **values.yaml**, etiqueta llamada **imagen.tag** como se puede ver en la captura

```
... ! values.yaml X  
uniovi-avib-morphingprojections-backend-storage-chart > ! values.yaml  
You, 3 weeks ago | 1 author (You)  
1 # Default values for uniovi-avib-morphingprojections-backend-storage. You, 8 months  
2 # This is a YAML-formatted file.  
3 # Declare variables to be passed into your templates.  
4  
5 replicaCount: 1  
6  
7 image:  
8   repository: avibdocker.azurecr.io/uniovi-avib-morphingprojections-backend-storage  
9   pullPolicy: IfNotPresent  
10  # Overrides the image tag whose default is the chart appVersion.  
11  tag: "1.3.0"  
12  
13  # get Private Container Registry credentials to have access previous pull any container  
14  imagePullSecrets:
```

Esta versión corresponde a la versión que previamente hemos escogido cuando hemos creado nuestra imagen de docker en el apartado 2. Y por ultimo cambiamos la version del paquete helm a publicar editando ahora el fichero llamado **chart.yaml**, modificando los argumentos llamados **version** y **appVersion**, con una versión que en mi caso coinciden con la version de la imagen que hacen referencia, por simplicidad, como se puede ver en la captura inferior.

```
! Chart.yaml X  
uniovi-avib-morphingprojections-backend-storage-chart > ! Chart.yaml  
You, 3 weeks ago | 1 author (You)  
1 apiVersion: v2 You, 8 months ago • last commit ...  
2 name: uniovi-avib-morphingprojections-backend-storage  
3 description: A Helm chart for uniovi-avib-morphingprojections-backend-storage microservice in Kubernetes  
4  
5 # A chart can be either an 'application' or a 'library' chart.  
6 #  
7 # Application charts are a collection of templates that can be packaged into versioned archives  
8 # to be deployed.  
9 #  
10 # Library charts provide useful utilities or functions for the chart developer. They're included as  
11 # a dependency of application charts to inject those utilities and functions into the rendering  
12 # pipeline. Library charts do not define any templates and therefore cannot be deployed.  
13 type: application  
14  
15 # This is the chart version. This version number should be incremented each time you make changes  
16 # to the chart and its templates, including the app version.  
17 # Versions are expected to follow Semantic Versioning (https://semver.org/)  
18 version: 1.3.0  
19  
20 # This is the version number of the application being deployed. This version number should be  
21 # incremented each time you make changes to the application. Versions are not expected to  
22 # follow Semantic Versioning. They should reflect the version the application is using.  
23 # It is recommended to use it with quotes.  
24 appVersion: "1.3.0"  
25
```

Una vez modificado estos argumentos ya podemos crear nuestro nuevo paquete de helm con este comando

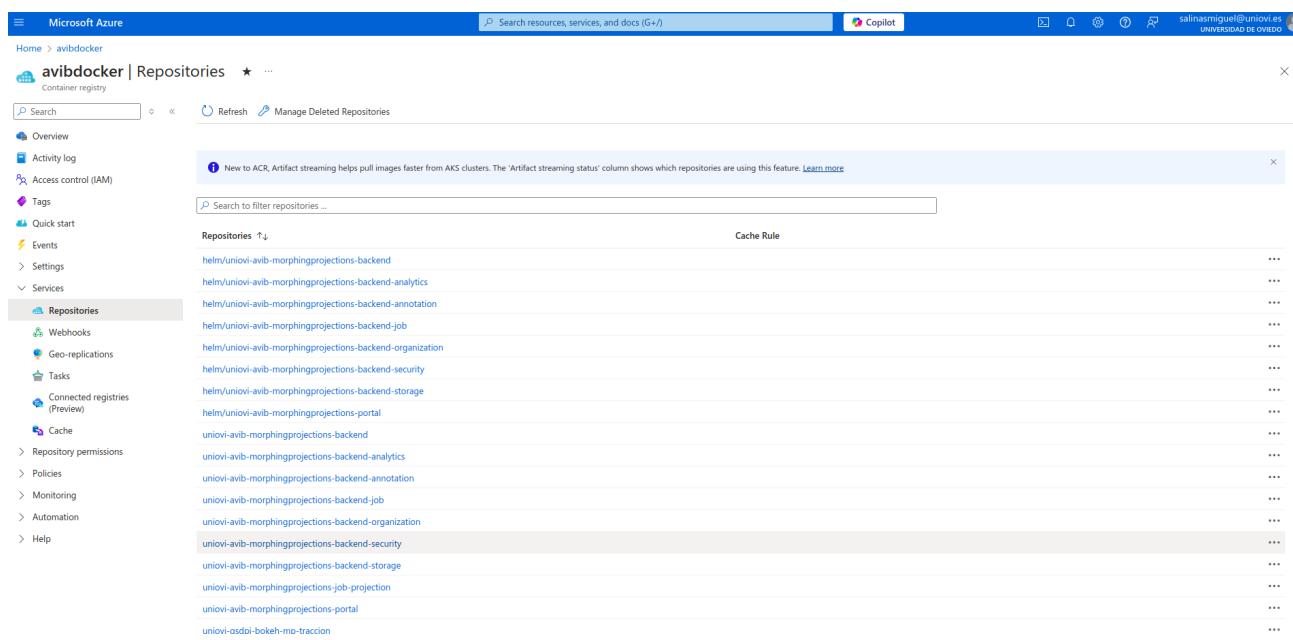
```
$ helm package .
```

Esto creara un nuevo fichero zip con el nombre del paquete que hayamos escogido previamente en los ficheros antes descritos.

- **STEP05:** Ahora nos queda publicar tambien este paquete de helm en Azure como si se tratara de una imagen de Docker:

```
$ helm push uniovi-avib-morphingprojections-backend-organization-1.5.0.tgz  
oci://avibdocker.azurecr.io/helm
```

Después de estos comandos, ya contamos en el registro de Azure tanto con la imagen de nuestro servicio con la nueva implementación del mismo, como con el paquete de helm, que será el que utilicemos para desplegarlo en el entorno de Kubernetes.



Ahora faltaría las otras 3 etapas que corresponden a la parte del pipeline llamado CD: Continuos Delivery (Disponibilidad del servicio): Hasta ahora todos estas etapas han sido ejecutadas **en el entorno de desarrollo**, donde residen los repos de código tanto del microservicio como del paquete helm, pero ahora el despliegue de estos artefactos creados y publicados en Azure **serán ejecutados dentro del nodo host** donde el cluster se está ejecutando:

- **STEP06:** aunque esta etapa es opcional, vamos a borrar el anterior despliegue llamada release en el lenguaje utilizado por helm dentro de nuestro cluster.

Claro está que el CLI de kubernetes estará instalado y configurado correctamente, como ya se ha explicado en apartados anteriores. Ejecutamos

```
$ helm delete uniovi-avib-morphingprojections-backend-organization
```

Este comando desinstalar la ultima release, del microservicio escogido y esto implica que todos los recursos asociados al mismo, serán borrados: deployments, pods y servicios en el menor de los casos, en otros casos, service accounts también.

- **STEP07:** despliegue del nuevo paquete de helm, y creación de la nueva release y de todos los nuevos recursos asociados. Podemos fijarnos, como al desplegar el nuevo paquete, se creara un nuevo deployment con la versión que justamente hemos escogido en etapas, anteriores, fijarse en ello, pues sino puede que estemos desplegando algo incorrecto. Tras unos segundos y a través del Dashboard de Kubernetes, podemos chequear todos estos nuevos recurso creados, que deben de estar en verde indicando que kubernetes ha podido levantar el contenedor asociado y este funciona, con un nivel de salud correcto.

```
$ helm install uniovi-avib-morphingprojections-backend-organization  
oci://avibdocker.azurecr.io/helm/uniovi-avib-morphingprojections-backend-organization
```

Aquí podemos ver una lista completa de todos los deployments de todos los microservicios del sistema:

Nombre	Imagenes	Etiquetas	Pods	Fecha de creación
uniovi-avib-morphingprojections-backend-analytics	avibdocker.azurecr.io/uniovi-avib-morphingprojections-backend-analytics:1.8.0	app.kubernetes.io/instance: uniovi-avib-morphingprojections-backend-analytics app.kubernetes.io/managed-by: Helm app.kubernetes.io/name: uniovi-avib-morphingprojections-backend-analytics	1 / 1	2 days ago
uniovi-avib-morphingprojections-portal	avibdocker.azurecr.io/uniovi-avib-morphingprojections-portal:1.8.0	app.kubernetes.io/instance: uniovi-avib-morphingprojections-portal app.kubernetes.io/managed-by: Helm app.kubernetes.io/name: uniovi-avib-morphingprojections-portal	1 / 1	17 days ago
uniovi-avib-morphingprojections-backend-annotation	avibdocker.azurecr.io/uniovi-avib-morphingprojections-backend-annotation:1.7.0	app.kubernetes.io/instance: uniovi-avib-morphingprojections-backend-annotation app.kubernetes.io/managed-by: Helm app.kubernetes.io/name: uniovi-avib-morphingprojections-backend-annotation	1 / 1	21 days ago
uniovi-avib-morphingprojections-backend	avibdocker.azurecr.io/uniovi-avib-morphingprojections-backend:19.0	app.kubernetes.io/instance: uniovi-avib-morphingprojections-backend app.kubernetes.io/managed-by: Helm app.kubernetes.io/name: uniovi-avib-morphingprojections-backend	1 / 1	21 days ago

Si todas las estapas han ido bien, podemos dar por finalizada la actualización del microservicio, que claro está deberá de ofrecer lo esperado cuando este sea utilizado

por el sistema

Pasos a seguir en el despliegue servicios negocio frontend Java

En este caso el número de microservicios involucrados es uno solo el del portal, que representa, como sabemos el frontend del sistema, por lo que estos pasos solo se aplican cuando queramos actualizar este microservicio. Los pasos a seguir son los mismos que los anteriores, para el caso de backend, solamente difiere en la primera etapa. En este caso la compilación y empaquetado en una imagen de docker se hace en un solo paso, pues es el propio Dockerfile quien hace esta tarea en dos pasos, como se puede observar viendo este ficheros.

- **STEP01:** compilación y empaquetado (transpiling) del portales. Como en el caso anterior debemos de escoger el argumento del contexto en donde correr el microservicio con un valor igual a avib y un tag con la versión que le corresponda.

```
$ docker build --build-arg ARG_ANGULAR_PROFILES_ACTIVE=build-avib -t uniovi-avib-morphingprojections-portal:1.8.0 .
```

- **STEP02:** Tageo de la imagen para ser subida a Azure. No debemos olvidar también como el tag en este caso, lleva el prefijo del registro privado de Azure creado por nosotros para almacenar estas imágenes:

```
$ docker tag uniovi-avib-morphingprojections-portal:1.8.0 avibdocker.azurecr.io/uniovi-avib-morphingprojections-portal:1.8.0
```

- **STEP03:** publicación de la imagen en Azure:

```
$ docker push avibdocker.azurecr.io/uniovi-avib-morphingprojections-portal:1.8.0
```

- **STEP04:** borramos el paquete de helm existente

```
$ rm uniovi-avib-morphingprojections-portal-1.7.0.tgz
```

- **STEP05:** configuracion y empaquetado de un nuevo paquete de helm. Los ficheros a editar y argumentos a cambiar son los mismos que en caso anterior, por lo que no voy a repetirlo.

```
$ helm package .
```

- **STEP06:** Publicación del paquete de helm en Azure igualmente:

```
$ helm push uniovi-avib-morphingprojections-portal-1.8.0.tgz
oci://avibdocker.azurecr.io/helm
```

- **STEP07:** Ahora y al igual que antes el resto de etapas serán ejecutadas en el mismo orden que en el caso del backend. Borramos la anterior release

```
$ helm delete uniovi-avib-morphingprojections-portal
```

- **STEP08:** Desplegamos la nueva release y esperamos igualmente a que los recursos se hayan ejecutados por kubernetes de forma correcta, color verde:

```
$ helm install uniovi-avib-morphingprojections-portal
oci://avibdocker.azurecr.io/helm/uniovi-avib-morphingprojections-portal
```

Pasos a seguir en el despliegue servicios negocio backend Python

Al igual que en el caso anterior, solamente contamos con un servicio de backend de Python, que es el de analítica. Los pasos son los mismos que los narrados para el backend de Java, con la misma circunstancia que el frontend, pues Python no es compilado como si lo es Java, por lo que la primera etapa será la de empaquetar la imagen de Docker como en el caso anterior.

Pasos a seguir en el despliegue servicios Job proyección

Este servicio es especial, pues a diferencia de los anteriores no es desplegado por nosotros, sino que es desplegado bajo demanda por el gestor de Jobs cuando queremos proyectar un caso. Por esto solo existen etapas de tipo CI y nunca de CD, pues repito, es el gestor quien se encarga de ello bajo demandas del usuario.

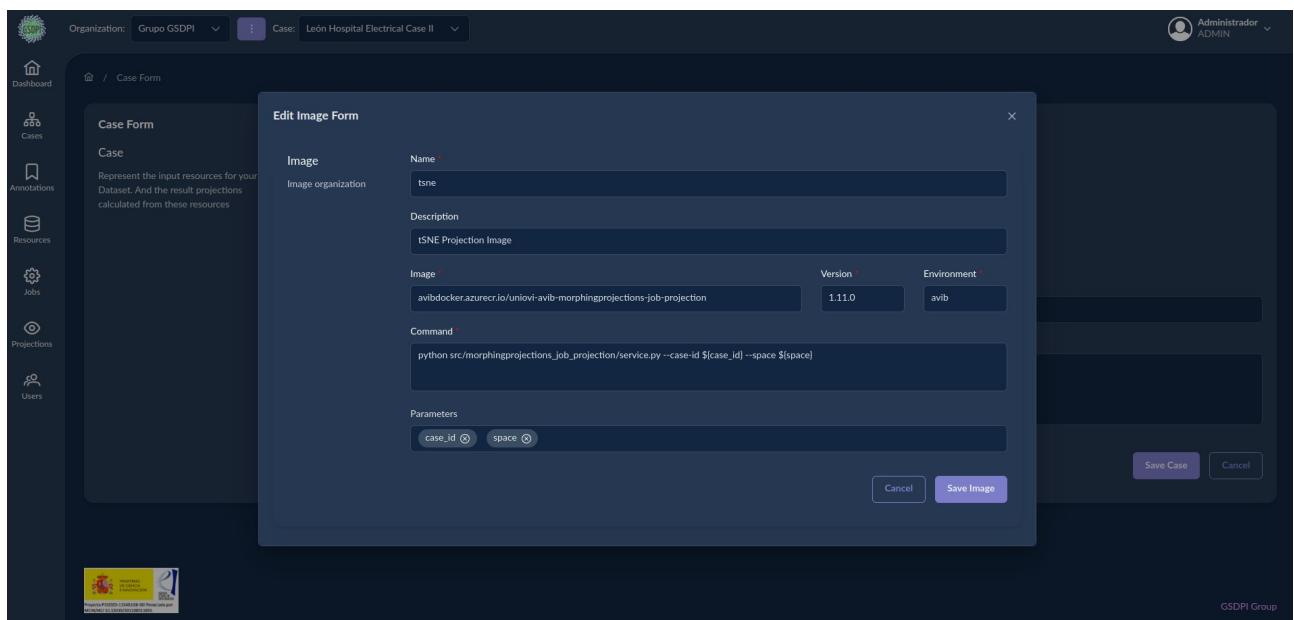
Las etapas de CI en este podemos resumirlas sin mas detalle en estas tres: compilacion, empaquetado y publicación de la imagen.

```
$ docker build --build-arg ARG_PYTHON_PROFILES_ACTIVE=avib -t uniovi-avib-
morphingprojections-job-projection:1.1.0 .
```

```
$ docker tag uniovi-avib-morphingprojections-job-projection:1.1.0
avibdocker.azurecr.io/uniovi-avib-morphingprojections-job-projection:1.1.0
```

```
$ docker push avibdocker.azurecr.io/uniovi-avib-morphingprojections-job-projection:1.1.0
```

Solamente debemos tener en cuenta que al subir la versión de la imagen deberemos de tenerlo en cuenta los casos que la utilizan, teniendo que actualizar la versión del job. Para ellos como usuario, podemos editar la imagen asociado a nuestro caso y dentro del formulario de la imagen poner la versión del job correspondiente, como se puede ver en la imagen inferior:



Acceso al Service Registry de Azure

Actualmente el número de recursos y servicios es enorme, pero como ya se ha comentado en apartados anteriores, solamente vamos a utilizar un recurso controlado por Azure y es el **Azure Container Registry**, que es el servicio en donde vamos a alojar todas las imágenes correspondientes a los microservicios de negocio del sistema. El resto de servicios de infraestructura formado por Bases de Datos, Gestores de Objetos o Servicios de Autenticación y Autorización, sus imágenes estarán gestionadas por registros de imágenes públicos y no por nosotros.

Con todo esto deberemos de tener acceso al portal de Azure para:

1. Toda imagen desarrollada localmente deberá de ser publicada en este registro privado de Azure
2. El cluster en el momento de desplegar una de estas imágenes deberá tener también acceso al registro privado, por ello deberemos de crear dentro del cluster en el namespace donde despleguemos nuestras imágenes un recurso de tipo secreto en donde almacenemos las credenciales de acceso a nuestra cuenta de Azure

Acceso local a Azure

Como hemos comentado en el punto 1, localmente el equipo que utilicemos para compilar y publicar la imagen resultante de nuestro microservicio deberá de tener acceso a Azure con las credenciales correctas, para que en el momento de la publicar la imagen Azure nos de acceso al Container Registry.

Para poder autenticarnos contra Azure utilizaremos el CLI de azure llamado az, que podrá instalarse fácilmente, como podemos ver en la página de Microsoft, en este caso para un entorno [Linux](#)

```
$ curl -sL https://aka.ms/InstallAzureCLIDeb | sudo bash
```

Esto nos instalará en el entorno un CLI llamado az, como se puede ver viendo la versión del mismo:

```
$ az --version
azure-cli          2.23.0 *
core                2.23.0 *
telemetry           1.0.6
Extensions:
azure-devops        0.18.0
```

```
Python location '/opt/az/bin/python3'
Extensions directory '/home/miguel/.azure/cliextensions'
```

```
Python (Linux) 3.6.10 (default, Apr 29 2021, 12:10:04)
[GCC 9.3.0]
```

Legal docs and information: [aka.ms/AzureCliLegal](#)

You have 2 updates available. Consider updating your CLI installation with 'az upgrade'

Please let us know how we are doing: [https://aka.ms/azureclihats](#)
and let us know if you're interested in trying out our newest features:
[https://aka.ms/CLIUXstudy](#)

Ahora solamente nos toca logearnos en la plataforma

```
$ az login
```

Esto nos abrirá un Login Web desde donde podremos meter nuestras credenciales para acceder a la suscripción de nuestro Directorio (Tenant). Como utilizaremos nuestras mismas cuentas de la Universidad de Oviedo, nuestro Directorio se llama **Universidad de Oviedo** y dentro de él hemos creado una suscripción propia para nuestro Sistema llamada **Azure subscription 1** como se puede ver en la siguiente captura:

The screenshot shows the Azure Subscription Overview page. Key details include:

- Subscription ID:** c224798d-014e-41ba-b44a-f284169a40e8
- Directory:** Universidad de Oviedo (unioviode.onmicrosoft.com)
- Status:** Active
- Parent management group:** 05ea74a3-92c5-4c31-978a-925c3c799cd0
- Subscription name:** Azure subscription 1
- My role:** Owner
- Plan:** Azure Plan
- Secure Score:** Not available

Spending rate and forecast: Current cost €2.62, Forecast €18.52.

Costs by resource: Shows a donut chart for 'avibdocker' with a value of €2.62.

Top products by number of resources: Shows a bar chart with 1 registry and 1 dashboard.

Azure Defender coverage: Shows a chart indicating that Azure Defender is not enabled for this subscription. An 'Upgrade coverage' button is present.

Esta suscripción sirve para agrupar de forma lógica todos los recursos Azure que estamos utilizando para nuestro proyecto, que como se ha comentado solamente es el Registro Privado de Contenedores de Docker. Igualmente esta suscripción tiene asociado un owner, con un método de pago (tarjeta de crédito) que asume el gasto de los recursos utilizados.

Actualmente no se está cobrando nada pues el gasto incurrido hasta ahora solamente procede del Registro de Contenedores de Docker donde el espacio ocupado se está manteniendo al mínimo, sabiendo que estos gastos acumulados no deben de superar los 100 euros. En todo momento este owner puede ser cambiado por otra persona que asuma el método de pago y los gastos correspondiente si se superase ese límite de 100 euros.

Para ver todas las imágenes de Docker y gestionarlas, poder borrar versiones anteriores, para liberar espacio, por ejemplo, se ha creado dentro de nuestra suscripción, bajo el servicio Registro Privado de Contenedores un espacio llamado **avibdocker**, donde residen todas nuestras imágenes, como se puede ver en la siguiente captura:

Actualmente estas son las imágenes que manejamos. Podemos crear tres grupos:

- Grupo de imágenes para cada uno de nuestros microservicios:
 - **uniovi-avib-morphingprojections-backend:** Microservicio en Java Gateway
 - **uniovi-avib-morphingprojections-backend-analytics:** Microservicio en Python Analítica: histogramas, regresión lineal
 - **uniovi-avib-morphingprojections-backend-annotation:** Microservicio en Java configuración de anotaciones de casos

- **uniovi-avib-morphingprojections-backend-job:** Microservicio en Java que actua como cliente de Kubernetes para lanzar y monitorizar las proyecciones de nuestros casos
- **uniovi-avib-morphingprojections-backend-organization:** Microservicio en Java que gestiona la organización, proyectos y casos
- **uniovi-avib-morphingprojections-backend-security:** Microservicio en Java que se integra con Keycloak como cliente, manjenando usuarios y roles. Cliente del IAM Keycloak
- **uniovi-avib-morphingprojections-backend-storage:** Microservicio en Java que gestiona el acceso al Object Storage (Minio)
- **uniovi-avib-morphingprojections-job-projection:** este servicio implementa el algoritmo t-SNE y la lógica configurada a nivel de caso. Este proceso es disparado por el microservicio de **backend-job** en el momento que ejecutamos un caso.
- **Uniovi-avib-morphingprojections-portal:** este servicio implementa el Portal o interfaz de usuario con el cual el usuario interactua a la hora de ingestar, ejecutar y explotar visualmente un caso.

Este otro grupo son los paquetes utilizados para desplegar cualquier servicio listado en el grupo anterior, podemos fijarnos como todos ellos están agrupados bajo la carpeta de helm, indicando que son precisamente paquetes de helm preparados para ser desplegados en el cluster de Kubernetes. Cada uno de ellos tiene encapsulado las configuraciones propias de cada contenedor.

- helm/uniovi-avib-morphingprojections-backend
- helm/uniovi-avib-morphingprojections-backend-analytics
- helm/uniovi-avib-morphingprojections-backend-annotation
- helm/uniovi-avib-morphingprojections-backend-job
- helm/uniovi-avib-morphingprojections-backend-organization
- helm/uniovi-avib-morphingprojections-backend-security
- helm/uniovi-avib-morphingprojections-backend-storage
- helm/uniovi-avib-morphingprojections-portal

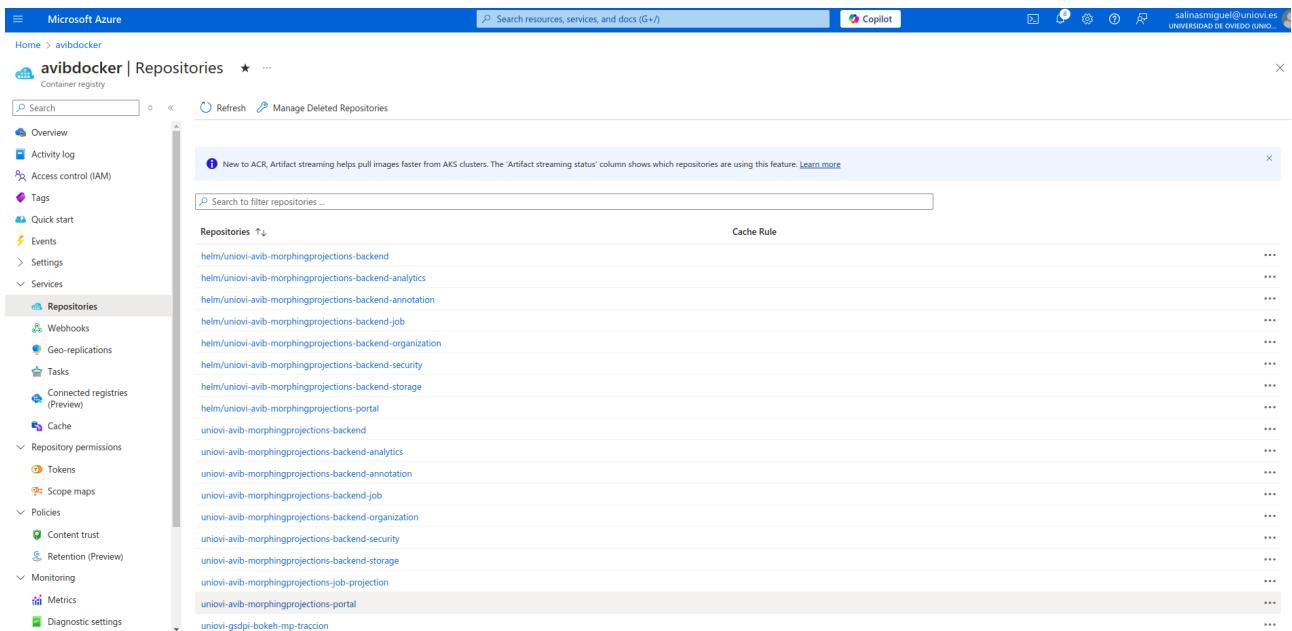
Por ultimo podemos citar este ultimo grupo que representan todos las imágenes que no son propias del sistema AVIB. Como ya hemos dicho varias veces, este sistema convive con otros servicios, en concreto con aplicaciones de tipo Bokeh, y esta imagen representa precisamente, la aplicación Bokeh llamada MP Tracción que sin correr dentro del cluster de Kubernetes, si corre como contenedor dentro del mismo host que el cluster:

- uniovi-gsdpi-bokeh-mp-traccion

Todos estos paquetes de Helm tienen en común el número de recursos creados por cada uno de ellos:

- **Deployment**: recurso encargado de crear replicas para los PODs que maneja. En nuestro caso hemos limitado la replica a uno, por ser suficiente para la carga de trabajo estimada para cada uno de ellos. El escalado de estos PODs y de sus contenedores internos es extramadamente sencillo hacerlo en caso de tener cargas de trabajo altas si fuera este el caso.
- **POD**: unidad mínima de despliegue de Kubernetes encargada de desplegar el contenedor propio manejado por el POD.
- **Service**: recurso de Kubernetes, encargado de balancear el acceso interno al contenedor que maneja

Solamente el paquete de helm llamado helm/uniovi-avib-morphingprojections-backend-job crea un recurso extra que los demás no hacen, y esto es un **service account** con los permisos necesarios, para que el microservicio que contiene pueda actuar de cliente de Kubernetes a la hora de crear y monitorizar los jobs encargados de ejecutar los algoritmos de proyección, como el t-SNE.



Acceso local a Docker Hub

Finalmente nos decantaremos por utilizar Docker Hub, pues las imágenes serán de acceso público, y Docker Hub ofrece acceso público ilimitado. Por ello hemos migrado todas las imágenes de docker y todos los paquetes de Helm de Kubernetes a

esta infraestructura. La migración supone crear una nuevo repositorio en donde vamos a publicar imágenes y paquetes. En este caso el paquete no puede llamarse igual que la imagen que representa como si ocurría antes en Azure, pues Azure añadía un prefijo llamado helm a nodos estos paquetes, ahora esto no ocurre en Docker Hub y por ello ha habido algunos cambios:

- Hemos renombrado todos los paquetes de helm añadiendo al final del nombre el sufijo chart como ya hacíamos con los repositorios de código.
- Ahora al ser repositorios públicos no es necesario configurar ningun pullSecret en los paquetes de helm como antes si lo hera por ser el registro de Azure privado