Dibujo en blanco y negro

Descripción generada automáticamente con confianza media

**UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO**

DIRECCIÓN GENERAL DE TECNOLOGÍAS DE LA INFORMACIÓN Y COMUNICACIÓN.

FACULTAD DE ESTUDIOS SUPERIORES ARAGÓN.

PARTICIPACIÓN DE PROYECTO

*APUNTES Y BITACORA*

Alumno.

*Rodríguez Pichardo Adrián Leonardo.*

Asesor.

Cruz Santiago Pedro Damián

SEPTIEMBRE de 2025 – MARZO de 2026

[APUNTES 4](#_Toc212540876)

[Base de Datos de Series Temporales (TSDB) 4](#_Toc212540877)

[Características de las bases de datos de series temporales 5](#_Toc212540878)

[Optimizado para datos con fecha y hora 5](#_Toc212540879)

[Tasas de ingestión elevadas 5](#_Toc212540880)

[Consultas eficientes para intervalos de tiempo 5](#_Toc212540881)

[Políticas de compresión y conservación de datos 5](#_Toc212540882)

[Software OpenSource 6](#_Toc212540883)

[InfluxDB 6](#_Toc212540884)

[TimescaleDB 6](#_Toc212540885)

[Prometehus 7](#_Toc212540886)

[Agentes a Configurar 7](#_Toc212540887)

[Propios de la TSDB 7](#_Toc212540888)

[Independientes de la TSDB 8](#_Toc212540889)

[Graficadores 8](#_Toc212540890)

[Propios de la TSDB 8](#_Toc212540891)

[Independientes de la TSDB 9](#_Toc212540892)

[Contenedores 9](#_Toc212540893)

[Glances 9](#_Toc212540894)

[OAuth2.0 10](#_Toc212540895)

[Principios de OAuth2.0 10](#_Toc212540896)

[Roles de OAuth2.0 10](#_Toc212540897)

[Tokens de acceso y código de autorización de OAuth 2.0 10](#_Toc212540898)

[¿Cómo funciona OAuth 2.0? 11](#_Toc212540899)

[Tipos de concesión en OAuth 2.0 11](#_Toc212540900)

[Grid UNAM 12](#_Toc212540901)

[Antecedentes 13](#_Toc212540902)

[OIDC 13](#_Toc212540903)

[Bearer Authorization 13](#_Toc212540904)

[Bearer Token 13](#_Toc212540905)

[HTCondor 13](#_Toc212540906)

[Entendiendo los Scripts 14](#_Toc212540907)

[1) set-function \_\_check\_account\_oidc { 14](#_Toc212540908)

[2) set-function \_\_check\_agent { 14](#_Toc212540909)

[3) set-function \_\_check\_dir\_gridunam { 14](#_Toc212540910)

[4) set-function \_\_check\_dir\_oidc-agent { 14](#_Toc212540911)

[5) set-function \_\_check\_load\_account\_oidc { 15](#_Toc212540912)

[6) set-function \_\_check\_pass\_account\_oidc { 15](#_Toc212540913)

[7) set-function \_\_check\_token { 15](#_Toc212540914)

[8) set-function \_\_create\_account\_oidc { 16](#_Toc212540915)

[9) set-function \_create\_agent { 16](#_Toc212540916)

[10) set-function \_jwtd { 17](#_Toc212540917)

[11) set-function \_jwtd\_epoch { 17](#_Toc212540918)

[12) set-function \_use\_agent { 17](#_Toc212540919)

[13) set-function activar\_acceso\_gridunam { 18](#_Toc212540920)

[14) set-function consultar\_jobs { 18](#_Toc212540921)

[15) set-function crear\_cuenta\_oidc { 18](#_Toc212540922)

[16) set-function descargar\_archivos { 18](#_Toc212540923)

[17) set-function eliminar\_mis\_agentes { 18](#_Toc212540924)

[18) set-function enviar\_jobs { 19](#_Toc212540925)

[19) set-function estado\_cluster { 19](#_Toc212540926)

[20) set-function generar\_token { 19](#_Toc212540927)

[21) set-function ping\_cluster { 19](#_Toc212540928)

[22) set-function run\_test\_cluster { 19](#_Toc212540929)

[23) set-function show\_cpus\_all\_cluster { 20](#_Toc212540930)

[24) set-function show\_cpus\_cluster { 21](#_Toc212540931)

[Flujo 21](#_Toc212540932)

[Propuestas de migración a Python. 22](#_Toc212540933)

[1. Traducción directa 22](#_Toc212540934)

[2. Reimplementación con librerías de Python 22](#_Toc212540935)

[3. Integración nativa con APIs OIDC y HTCondor 23](#_Toc212540936)

[BITACORA 23](#_Toc212540937)

# APUNTES

## Base de Datos de Series Temporales (TSDB)

Una base de datos de series temporales proporciona aplicaciones en tiempo real con conjuntos de registros basados en un marco temporal. Una base de datos de serie temporal está optimizada para conjuntos de registros en los que los datos cuentan una historia en orden cronológico. La duración de un conjunto de datos puede ser de unos pocos milisegundos entre sí, o una base de datos de series temporales puede usarse para puntos de datos por hora, diarios, mensuales o anuales.

Una base de datos de serie temporal (TSDB) se ha diseñado específicamente para los conjuntos de registros centrados en un marco temporal. Por ejemplo, es posible que necesite actualizaciones minuto a minuto en una aplicación meteorológica. Una base de datos de series temporales puede almacenar datos por minuto para mostrarlos a los usuarios. Una TSDB puede ser una opción cuando se necesitan consultas más refinadas basadas en el tiempo.

Estas bases de datos suelen estar optimizadas para las lecturas y la creación de nuevos registros. Las TSDB se han creado para recuperar mejor los registros basándose en una marca de tiempo en comparación con otras bases de datos comunes. Los TSDB también son mejores para las aplicaciones en tiempo real en las que los usuarios reciben un conjunto lineal de información basado en la marca de tiempo de cada registro.

El motor de almacenamiento TSDB funciona optimizando las consultas para grandes conjuntos de datos y recuperando datos basándose en marcas de tiempo récord.

Como los registros suelen contener un pequeño número de campos, las consultas se simplifican para los desarrolladores. Un TSDB puede devolver miles de registros, pero el conjunto de registros suele contener solo unos pocos campos. El número limitado de campos y las consultas simplificadas aceleran el rendimiento de las aplicaciones en tiempo real y reducen la sobrecarga en el servidor de bases de datos.

**Link:**

* <https://www.purestorage.com/es/knowledge/what-is-a-time-series-database.html>

-----------------------------------------------------------------------------------------------------------------------------

Las bases de datos de series temporales son bases de datos especializadas diseñadas para gestionar datos organizados e indexados por tiempo. A diferencia de las bases de datos tradicionales, que están optimizadas para el almacenamiento de datos de uso general, las TSDB se centran en el almacenamiento, la consulta y el análisis eficientes de secuencias de puntos de datos con marca temporal.

### Características de las bases de datos de series temporales

#### Optimizado para datos con fecha y hora

En esencia, las TSDB están construidas para manejar datos con marcas de tiempo como atributo fundamental. Cada punto de datos de una TSDB incluye una marca de tiempo, que sirve como índice primario. Esto permite a estas bases de datos almacenar y recuperar eficazmente secuencias ordenadas en el tiempo y proporcionar un acceso rápido a tendencias históricas o acontecimientos recientes.

#### Tasas de ingestión elevadas

Los datos de series temporales suelen generarse a un ritmo rápido (una herramienta de supervisión de servidores que captura métricas del sistema en tiempo real). Las TSDB están optimizadas para estas altas tasas de escritura y pueden ingerir grandes cantidades de datos sin ralentizarse ni perder información.

#### Consultas eficientes para intervalos de tiempo

El análisis de datos de series temporales suele implicar la consulta de intervalos o ventanas temporales concretos, como "las últimas 24 horas" o "este año comparado con el año pasado". Las TSDB se construyen teniendo esto en cuenta, ofreciendo capacidades de consulta especializadas que permiten a los usuarios recuperar rápidamente datos sobre intervalos de tiempo definidos. También admiten agregaciones como medias, sumas o tendencias para ofrecer valiosos análisis sin una lógica de consulta compleja.

#### Políticas de compresión y conservación de datos

Para gestionar la enorme cantidad de datos de series temporales que se generan a lo largo del tiempo, las TSDB utilizan técnicas avanzadas de compresión de datos. Estos métodos reducen los requisitos de almacenamiento al tiempo que preservan el rendimiento de la consulta.

Las TSDB suelen incluir políticas de retención para que los usuarios puedan definir cuánto tiempo deben conservarse los datos. Por ejemplo, un sistema puede conservar los datos detallados del último mes, mientras que reduce el muestreo para los datos más antiguos. El muestreo descendente es el proceso de reducir la granularidad de los datos a lo largo del tiempo. Por ejemplo:

* Las lecturas de temperatura en bruto podrían registrarse cada 10 segundos durante los 7 días más recientes.
* Para los datos más antiguos, el sistema puede reducir la muestra a medias horarias para ahorrar espacio, pero conservando las tendencias históricas.

Link:

- <https://www.datacamp.com/es/blog/time-series-database>

### Software OpenSource

InfluxDB

InfluxDB es una popular base de datos de series temporales de código abierto desarrollada por InfluxData. Se diseñó específicamente para altas tasas de ingestión y consultas eficientes de datos con marca de tiempo, lo que la convierte en una solución habitual para la supervisión del IoT, las métricas DevOps y los análisis en tiempo real.

**Texto

El contenido generado por IA puede ser incorrecto.**

TimescaleDB

TimescaleDB es una extensión de código abierto para PostgreSQL diseñada para combinar la potencia de bases de datos relacionales con la funcionalidad de las series temporales. Te permite aprovechar SQL a la vez que manejas eficazmente los datos con fecha y hora. Esto lo hace especialmente adecuado para casos de uso que requieren integrar datos de series temporales con datos relacionales, como la analítica empresarial o la telemetría IoT.

**Imagen que contiene Interfaz de usuario gráfica

El contenido generado por IA puede ser incorrecto.**

Prometehus

Prometheus es un sistema de supervisión y alerta con una TSDB incorporada, ampliamente adoptado en DevOps para métricas del sistema en tiempo real, seguimiento del rendimiento y gestión de alertas.

**Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

El contenido generado por IA puede ser incorrecto.**

**Link:**

* <https://www.datacamp.com/es/blog/time-series-database>

### Agentes a Configurar

Los agentes en una Base de Datos de Series Temporales (TSDB) son software especializados que se configuran para recopilar, transformar y enviar datos de series temporales al sistema TSDB, monitoreando la fuente de datos (como aplicaciones, servidores o sensores) y adaptando los datos a un formato que la TSDB pueda entender. Los tipos de configuración de agentes varían según la TSDB específica, pero generalmente implican la definición de qué datos se deben recolectar, el formato de los datos y el destino del envío.

#### Propios de la TSDB

Son aquellos que vienen integrados, recomendados o mantenidos oficialmente por la misma TSDB. Están diseñados para funcionar de forma nativa con ella.

**Ejemplos:**

* InfluxDB → usa Telegraf como agente oficial para recolectar métricas y enviarlas directamente a Influx.
* Prometheus → usa Node Exporter, cAdvisor, o los exporters oficiales para recolectar métricas.
* TimescaleDB → se apoya en Timescale Agent (extensiones para PostgreSQL) o integra colectores específicos.
* OpenTSDB → puede usar su propio colector de métricas (tcollector).

**Características:**

* Están optimizados para esa TSDB en particular.
* Generalmente incluyen integración lista para usarse (sin configuraciones extra de compatibilidad).
* Suelen estar mejor documentados y soportados.

#### Independientes de la TSDB

Son agentes que pueden enviar datos a distintas bases o sistemas de almacenamiento de series de tiempo.

**Ejemplos:**

* Collectd → recolecta métricas del sistema y puede enviarlas a InfluxDB, Graphite, Prometheus, etc.
* StatsD → usado para recolectar métricas de aplicaciones y enviarlas a varias TSDB.
* Fluentd / Logstash → recolectan logs/eventos y pueden enviarlos a múltiples TSDBs o motores de búsqueda (Elastic, Loki, etc.).
* Vector (Datadog, Timber.io) → recolecta métricas y logs, con salida a varias bases o plataformas.

**Características:**

* Son más flexibles porque no dependen de una sola TSDB.
* Requieren configuración extra para adaptar el formato de salida a la TSDB destino.
* Útiles en entornos híbridos donde se usan varias TSDBs o diferentes backends.

### Graficadores

Los graficadores para una base de datos de series temporales (TSDB) son herramientas de visualización y monitoreo que extraen datos de la TSDB para presentarlos en formato gráfico e interactivo, permitiendo el análisis de tendencias y el seguimiento de métricas en tiempo real.

#### Propios de la TSDB

Son nativos u oficiales, diseñados por los mismos creadores de la TSDB o pensados para integrarse sin configuración compleja.

**Ejemplos:**

* InfluxDB → trae su propio graficador web llamado Chronograf (y en versiones nuevas, InfluxDB UI).
* Prometheus → incluye un panel web básico integrado para hacer consultas y gráficas simples con PromQL.
* TimescaleDB → ofrece TimescaleDB Dashboard (basado en PostgreSQL) y extensiones para integración directa con visualizadores SQL.
* OpenTSDB → incluye una interfaz web propia para graficar queries.

**Características:**

* Están listos “out of the box” con la TSDB.
* Simples y con menos curva de aprendizaje.
* Limitados en personalización avanzada (comparado con soluciones externas).

#### Independientes de la TSDB

Son herramientas de visualización externas y universales, que se conectan a múltiples TSDBs y otras fuentes de datos.

**Ejemplos:**

* Grafana → el más popular, soporta InfluxDB, Prometheus, TimescaleDB, OpenTSDB, Elasticsearch, Loki, etc.
* Kibana (aunque más de logs con Elasticsearch, puede usarse con métricas).
* Redash → soporta consultas SQL y múltiples fuentes, incluyendo TSDBs.
* Metabase → orientado a SQL, puede integrarse con TimescaleDB (al ser PostgreSQL).

**Características:**

* Flexibles y potentes, con paneles dinámicos y muchas opciones gráficas.
* Funcionan con múltiples TSDBs (y hasta bases relacionales o NoSQL).
* Requieren configuración inicial (conectar datasource, definir queries).

## Contenedores

Un contenedor es un paquete de software que incluye una aplicación y todo lo necesario para que se ejecute, como código, bibliotecas, herramientas y el entorno de ejecución.

Los contenedores virtualizan el sistema operativo y permiten ejecutar la aplicación en cualquier entorno, ya sea local, en la nube o en un centro de datos, de forma aislada y portátil.

A diferencia de una “máquina virtual”, no simula todo un sistema operativo, sino que comparte el mismo núcleo del equipo, lo que los hace más ligeros y rápidos.

## Glances

Glances es una herramienta de monitorización de sistemas de código abierto, multiplataforma (Linux, macOS, Windows) y basada en Python, que permite supervisar en tiempo real diversos aspectos del sistema como la CPU, memoria, disco, red, procesos, temperaturas y más. Presenta la información de manera dinámica y optimizada en un panel fácil de leer, y puede ser usada en terminal o a través de una interfaz web, permitiendo la monitorización remota en modo cliente/servidor.

## OAuth2.0

OAuth 2.0, significa “Open Authorization” (autorización abierta), es un estándar diseñado para permitir que un sitio web o una aplicación accedan a recursos alojados por otras aplicaciones web en nombre de un usuario. Proporciona acceso consentido y restringe las acciones que la aplicación del cliente puede realizar en los recursos en nombre del usuario, sin compartir nunca las credenciales del usuario.

### Principios de OAuth2.0

OAuth 2.0 es un protocolo de autorización y NO un protocolo de autenticación. Está diseñado principalmente como un medio para conceder acceso a un conjunto de recursos, por ejemplo, API’s remotas o datos de usuario.

Auth 2.0 utiliza tokens de acceso. Un Token de acceso es un dato que representa la autorización para acceder a los recursos en nombre del usuario final. OAuth 2.0 no define un formato específico para los tokens de acceso. Sin embargo, en algunos contextos, se suele utilizar el formato JSON Web Token (JWT). Esto permite a los emisores de tokens incluir datos en el propio token. Además, por razones de seguridad, los tokens de acceso pueden tener una fecha de caducidad.

### Roles de OAuth2.0

Los componentes esenciales de un sistema de OAuth 2.0, son los siguientes:

* Propietario del recurso: El usuario o sistema que posee los recursos protegidos y puede conceder acceso a ellos.
* Cliente: El cliente es el sistema que requiere acceso a los recursos protegidos. Para acceder a los recursos, el cliente debe poseer el token de acceso correspondiente.
* Servidor de autorización: Este servidor recibe las solicitudes de tokens de acceso del cliente y las emite una vez que el propietario del recurso se ha autenticado y ha dado su consentimiento. El servidor de autorización expone dos puntos de conexión: el punto de conexión de autorización, que maneja la autenticación interactiva y el consentimiento del usuario, y el punto de conexión de token, que está involucrado en una interacción de máquina a máquina.
* Servidor de recursos: Un servidor que protege los recursos del usuario y recibe las solicitudes de acceso del cliente. Acepta y valida un token de acceso del cliente y le devuelve los recursos adecuados.

### Tokens de acceso y código de autorización de OAuth 2.0

El servidor de autorización de OAuth 2 no puede devolver directamente un token de acceso después de que el propietario del recurso haya autorizado el acceso. En su lugar, y para mayor seguridad, se puede devolver un Código de Autorización, que se cambia por un token de acceso. Además, el servidor de autorización también puede emitir un Token de actualización con el token de acceso. A diferencia de los tokens de acceso, los tokens de actualización suelen tener un largo plazo de caducidad y pueden cambiarse por nuevos tokens de acceso cuando estos caducan. Dado que los tokens de actualización tienen estas propiedades, deben almacenarse de forma segura por los clientes.

### ¿Cómo funciona OAuth 2.0?

En el nivel más básico, antes de poder utilizar OAuth 2.0, el cliente debe adquirir sus propias credenciales, un id de cliente y un client secret, del servidor de autorización para identificarse y autenticarse al solicitar un token de acceso.

Con OAuth 2.0, las solicitudes de acceso son iniciadas por el cliente, por ejemplo, una aplicación móvil, un sitio web, una aplicación de televisión inteligente, una aplicación de escritorio, etc. La solicitud, el intercambio y la respuesta de los tokens siguen el siguiente flujo general:

1. El cliente solicita autorización (solicitud de autorización) al servidor de autorización, proporcionando el id y el client secret como identificación; también proporciona los ámbitos y un URI de extremo (URI de redireccionamiento) al que enviar el token de acceso o el código de autorización.
2. El servidor de autorización autentica al cliente y verifica que los ámbitos solicitados están permitidos.
3. El propietario del recurso interactúa con el servidor de autorización para conceder el acceso.
4. El servidor de autorización redirige de vuelta al cliente con un código de autorización o un token de acceso, según el tipo de concesión. También puede devolverse un token de actualización.
5. Con el token de acceso, el cliente solicita acceso al recurso desde el servidor de recursos.

### Tipos de concesión en OAuth 2.0

Las concesiones son el conjunto de pasos que un cliente tiene que realizar para obtener la autorización de acceso a los recursos. El marco de autorización proporciona varios tipos de concesión para hacer frente a diferentes escenarios:

* Concesión del Código de Autorización: Este flujo, considerado el más seguro, implica un proceso de dos pasos: el servidor de autorización primero devuelve un código de un solo uso al cliente, quien luego lo intercambia por un token de acceso. Está diseñado para aplicaciones web tradicionales con servidor backend, donde el intercambio se puede realizar de forma segura sin exponer el client\_secret.
* Concesión Implícita: En este flujo simplificado y ahora considerado obsoleto, el token de acceso se devuelve directamente al cliente sin un código intermedio. Se desaconseja su uso porque el token a menudo se transmite en la URL, lo que lo hace vulnerable a fugas y robos, representando un riesgo significativo para la seguridad.
* Concesión del Código de Autorización con PKCE: Esta es una evolución más segura del flujo de código de autorización, que incorpora un "desafío" y un "verificador" criptográfico. Es la opción recomendada para aplicaciones que no pueden almacenar un client\_secret de forma segura, como las aplicaciones de página única (SPA) y las aplicaciones móviles nativas, ya que protege contra ataques de interceptación de código.
* Concesión de Credenciales del Propietario del Recurso: Este flujo requiere que el usuario proporcione sus credenciales directas (nombre de usuario y contraseña) a la aplicación cliente, que luego las intercambia por un token. Su uso se limita a clientes de absoluta confianza, como aplicaciones oficiales del mismo proveedor del servicio, y se emplea cuando la redirección al servidor de autorización no es viable.
* Concesión de Credenciales del Cliente: Utilizado para comunicación entre máquinas (M2M), este flujo permite que una aplicación autentique su propia identidad usando su client\_id y client\_secret para obtener un token de acceso. Es ideal para procesos automatizados, microservicios o trabajos en segundo plano donde no hay un usuario final involucrado que interactúe con la aplicación.
* Flujo de Autorización de Dispositivos: Este flujo está diseñado específicamente para dispositivos con capacidades de entrada limitadas, como televisiones inteligentes o consolas. El dispositivo muestra un código y una URL al usuario, quien luego los introduce en un navegador en otro dispositivo (como su teléfono) para autorizar la aplicación de forma segura.
* Concesión del Token de Actualización: Este no es un flujo de autorización inicial, sino un mecanismo para mantener el acceso. Permite que una aplicación utilice un "token de actualización" de larga duración para obtener un nuevo "token de acceso" cuando el anterior caduca, mejorando la experiencia del usuario al evitar que tenga que volver a iniciar sesión constantemente.

## Grid UNAM

Grid UNAM es una infraestructura distribuida de computación de alto rendimiento / “grid computing” dentro de la UNAM.

Su objetivo es agrupar recursos de cómputo y almacenamiento distribuidos en distintos institutos y dependencias universitarias, para que los proyectos de investigación puedan usar esos recursos de forma compartida y eficiente.

Participan entidades como la DGTIC, LAMOD, el Instituto de Ciencias de la Atmósfera y Cambio Climático, el Instituto de Astronomía, etc.

En cuanto al software se usan herramientas como HTCondor-CE (para interoperar entre nodos) y SLURM (gestión de colas en los nodos) como parte de la arquitectura interna del grid.

Integran sistemas de autenticación / autorización (gestión de identidad) con INDIGO IAM, OpenID / SciTokens, para manejar la seguridad del acceso a recursos distribuidos.

En la parte de usuarios, uno de los requisitos es tener una cuenta en la Red Inalámbrica Universitaria (RIU) que se asocia con INDIGO IAM para generar tokens que permitan el envío de trabajos al grid.

En la documentación “Manejo de tokens” se explica que cuando un trabajo es enviado al sistema de colas, hay un token de acceso que se transporta al “host de ejecución” y se coloca en el entorno de ejecución del trabajo; ese token autoriza al trabajo a acceder a datos durante su ejecución.

En pocas palabras Grid UNAM es una infraestructura de cómputo distribuido con control de acceso, diseñada para que investigadores puedan someter trabajos, hacer cálculos intensivos usando recursos distribuidos, y gestionar acceso seguro a los datos y recursos.

### Antecedentes

#### OIDC

OpenID Connect u OIDC es un protocolo de identidad que utiliza los mecanismos de autorización y autenticación de OAuth 2.0. Mientras que OAuth 2.0 es un protocolo de autorización, OIDC es un protocolo de autenticación de identidad y puede utilizarse para verificar la identidad de un usuario ante un servicio de cliente, también llamado Parte de confianza. OIDC utiliza JSON Web Tokens (JWT), flujos HTTP y evita compartir las credenciales del usuario con los servicios.

https://auth0.com/es/intro-to-iam/what-is-openid-connect-oidc

#### Bearer Authorization

La autorización de portador es un esquema de autenticación HTTP comúnmente utilizado con OAuth 2.0 . En este enfoque, el cliente incluye un token de acceso en el encabezado "Authorization" utilizando el esquema "Bearer", lo que otorga permiso para acceder a recursos protegidos. El servidor valida el token para la autorización. Es un método ampliamente utilizado para proteger el acceso a la API, especialmente en escenarios que involucran aplicaciones de terceros.

https://apidog.com/articles/what-is-bearer-token/

#### Bearer Token

Un token de portador es un tipo de token utilizado para autenticación y autorización y se utiliza en aplicaciones web y API para almacenar credenciales de usuario e indicar autorización para solicitudes y acceso.

Generación de tokens de portador basados ​​en protocolos y especificaciones como OAuth y JWT (JSON Web Token). El usuario autenticado obtiene el token de portador emitido por el servidor y lo envía al servidor en el encabezado de la solicitud. El servidor verifica el token de portador recibido y controla el acceso del usuario basándose en él. El token de portador también suele enviarse mediante una conexión cifrada HTTPS. Esto evita el acceso no autorizado de terceros maliciosos, incluso si el token es robado.

https://apidog.com/articles/what-is-bearer-token/

#### HTCondor

HTCSS es un proyecto del Centro de Computación de Alto Rendimiento (CHTC) de la Universidad de Wisconsin-Madison . Este proyecto recibe financiación de la Fundación Nacional de Ciencias (NSF) a través de la Alianza para el Avance de la Computación de Alto Rendimiento (PATh) .

HTCondor es un sistema especializado en la gestión de cargas de trabajo para trabajos con un uso intensivo de recursos. Al igual que otros sistemas de procesamiento por lotes completos, HTCondor ofrece un mecanismo de cola de trabajos, una política de programación, un esquema de prioridad, y la monitorización y gestión de recursos. Los usuarios envían sus trabajos en serie o en paralelo a HTCondor, quien los coloca en una cola, elige cuándo y dónde ejecutarlos según una política, supervisa cuidadosamente su progreso y, finalmente, informa al usuario al finalizar.

https://htcondor.org/description.html

### Entendiendo los Scripts

#### set-function \_\_check\_account\_oidc {

\_\_check\_agent;

oidc-gen --accounts | grep $grid\_account\_oidc &> /dev/null;

if [ $? == 1 ]; then

\_\_create\_account\_oidc;

Else

\_\_check\_dir\_oidc-agent;

fi}

Se llama a la función \_chech\_agent para asegurarque el agente oidc esté o se intente poner, se ejecuta una lista de cuentas registradas por oidc-gen y pasa la salida a $grid\_account\_oidc para buscar el nombre de la cuenta almacenada en la variable grid\_account\_oidc, silencia la salida porque solo interesa el código de salida del grep, se se devuelve 1, es decir que no encontró la cuenta, se llama a una nueva función para crearla, pero si la encontró llama a otra función para fijar la ruta donde está la cuenta.

#### set-function \_\_check\_agent {

oidc-agent --status &> /dev/null;

if [ $? == 1 ]; then

\_use\_agent;

fi}

Se le pregunta al deamon oidc-agent si esta corriendo, silenciando la salida, si el código de salida es 1 (no esta activo) llama a \_use\_agent para iniciar o crear uno

#### set-function \_\_check\_dir\_gridunam {

if [ ! -d $dir\_gridunam ]; then

echo "El directorio $dir\_gridunam no existe";

mkdir $dir\_gridunam;

echo "Fue creado";

fi}

Se comprueba si la ruta $dir\_gridunam no existe como directorio, si no existe imprime un mensaje, crea el directorio y confirma con otro mensaje

#### set-function \_\_check\_dir\_oidc-agent {

if [ -d $HOME/.config/oidc-agent ]; then

export file\_account\_oidc="$HOME/.config/oidc-agent/$grid\_account\_oidc";

else

export file\_account\_oidc="$HOME/.oidc-agent/$grid\_account\_oidc";

fi}

Se verifica si existe la carpeta, si existe define file\_account\_oidc apuntando $HOME/.config/oidc-agent/$grid\_account\_oidc, si no existe asume la ruta alternativa $HOME/.oidc-agent/$grid\_account\_oidc, se determina donde esta el archivo de cuenta OIDC y lo expone como variable de enterno.

#### set-function \_\_check\_load\_account\_oidc {

\_\_check\_account\_oidc;

oidc-agent --status | grep $grid\_account\_oidc &> /dev/null;

if [ $? == 1 ]; then

oidc-add -f ${grid\_account\_oidc} --pw-file=$pass\_account\_oidc;

fi}

Se llama a la función que comprueba si la cuenta OIDC existe, revisa si el agente tiene cargada la cuenta $grid\_account\_oidc filtrando la salida, si no se encontró la cuenta añade la cuenta al agente usando la contraseña que esta en $pass\_account\_oidc, esto carga la cuenta en el agente para poder genera tokens después.

#### set-function \_\_check\_pass\_account\_oidc {

if [ ! -f $pass\_account\_oidc ]; then

echo "El archivo $pass\_account\_oidc no existe";

tr -cd '[:graph:]' < /dev/urandom | fold -w 32 | head -n1 > $pass\_account\_oidc;

check\_passwd=$(cracklib-check < $pass\_account\_oidc | awk '{print $2}');

while [ "$check\_passwd" != 'OK' ]; do

tr -cd '[:graph:]' < /dev/urandom | fold -w 32 | head -n1 > $pass\_account\_oidc;

check\_passwd="cracklib-check < $pass\_account\_oidc | awk '{print $2}'";

done;

chmod 400 $pass\_account\_oidc;

echo "Fue creado";

fi}

Si el archivo que contiene la contraseña no existe imprime el mensaje, genera una contraseña aleatoria lee de /dev/urandom, filtra caracteres gráficos con tr -cd '[:graph:]', corta en líneas de 32 chars (fold -w 32), toma la primera línea (head -n1) y la guarda en $pass\_account\_oidc. Usa cracklib-check, una herramienta para evaluar la fortaleza de contraseñas, leyendo el archivo, con awk extrae la segunda columna (“OK”) y guarda en check\_passwd.

Con el bucle while se intenta crea otra contraseña hasta que el mensaje de check\_passwd sea “OK”, si se creo correctamente, restringue permisos del archivo pass\_account\_oidc a lectura por el propietario y se notifica con el echo.

#### set-function \_\_check\_token {

if [ $BEARER\_TOKEN ]; then

time\_exp=`\_jwtd\_epoch $BEARER\_TOKEN | grep exp | awk '{print $2}' | tr -d ','`;

time\_now=`date +%s`;

if [ $time\_exp -lt $time\_now ]; then

echo "Expiro la validez del token actual";

generar\_token;

echo "Se genero uno nuevo";

else

echo "Existe un token y es vigente";

fi;

else

if [ -n "${BEARER\_TOKEN-unset}" ]; then

echo "No exite un token";

generar\_token;

echo "Se genero uno nuevo";

else

echo "El token no es valido";

generar\_token;

fi;

fi}

Se comprueba si existe la variable BEARER\_TOKEN, llama a \_jwtd\_epoch con el token para obtener campos decodificados, filtra con grep exp, y awk toma el segundo campo, es decir el valor de exp (tiempo de expiración), tr -d ',' elimina comas y time\_now obtiene tiempo actual en segundos con date +%s

#### set-function \_\_create\_account\_oidc {

\_\_check\_agent;

\_\_check\_pass\_account\_oidc;

echo oidc-gen -w device $grid\_account\_oidc --issuer=$URL\_IAM\_INDIGO --client-id=condorcli --client-secret=QM2qzp4DdO48nJ1PyXNvupf0prV0u6wE --scope="$SCOPE" --pw-file $pass\_account\_oidc;

oidc-gen -w device $grid\_account\_oidc --issuer=$URL\_IAM\_INDIGO --client-id=condorcli --client-secret=QM2qzp4DdO48nJ1PyXNvupf0prV0u6wE --scope="$SCOPE" --pw-file $pass\_account\_oidc;

\_\_check\_dir\_oidc-agent}

Se asegura que haya un agente y que exista el archivo con la contraseña de la cuenta, después imprime el comando que se va a ejecutar y muestra parámetros sensibles, se ejecuta oidc-gen para crear la cuenta OIDC en modo device con issuer, client-id, client-secret, scope y la contraseña, para finalizar determinando la ruta del archivo de cuenta con la función correspondiente

#### set-function \_create\_agent {

oidc-agent --status &> /dev/null;

if [ $? == 0 ]; then

echo "Eliminando agente activo";

$(oidc-agent -k 2>&1 /dev/null) &> /dev/null;

echo "Nuevo agente creado";

eval `oidc-agent -a /tmp/oidc-$(tr -cd '[:alnum:]' < /dev/urandom | fold -w 6 | head -n1)/oidc-agent.$$ "$@"`;

else

echo "Nuevo agente creado";

eval `oidc-agent -a /tmp/oidc-$(tr -cd '[:alnum:]' < /dev/urandom | fold -w 6 | head -n1)/oidc-agent.$$ "$@"`;

fi}

Se verifica el estado del agente. Si ya hay uno (0) se intenta eliminar, luego crea un nuevo agente. En ambos casos (0 o 1) el script termina creando un nuevo agente y evalúa las variables de entorno necesarias.

#### set-function \_jwtd {

if [[ -x $(command -v jq) ]]; then

jq -sR 'split(".")[0,1] | gsub("-";"+") | gsub("\_";"/") | @base64d | fromjson

| if has("exp") then .exp |= strflocaltime("%Y-%m-%dT%H:%M:%SZ") else . end

| if has("iat") then .iat |= strflocaltime("%Y-%m-%dT%H:%M:%SZ") else . end

| if has("nbf") then .nbf |= strflocaltime("%Y-%m-%dT%H:%M:%SZ") else . end

| if has("auth\_time") then .auth\_time |= strflocaltime("%Y-%m-%dT%H:%M:%SZ") else . end' <<< ${1};

echo "Signature: $(echo "${1}" | awk -F'.' '{print $3}')";

echo "tiempo actual: `date +%Y-%m-%dT%H:%M:%SZ`";

fi}

Se comprueba si existe jq, con command -v jq se devuelve la ruta si es que existe y con -x se comprueba el ejecutable, después se usa jk -sR para decodificar y mostrar datos legibles de JWT.

#### set-function \_jwtd\_epoch {

if [[ -x $(command -v jq) ]]; then

jq -R 'split(".") | .[0],.[1] | @base64d | fromjson' <<< "${1}";

echo "Signature: $(echo "${1}" | awk -F'.' '{print $3}')";

echo "tiempo epoch actual: `date +%s`";

fi}

Similar al script anterior pero sin formatear fechas.

#### set-function \_use\_agent {

find /tmp -maxdepth 1 -user $USER -name oidc-agent-service -exec chmod 1777 {} \;;

$(oidc-agent -k 2>&1 /dev/null) &> /dev/null;

pkill -u $USER oidc-agent;

find /tmp -maxdepth 1 -user $USER -not \( -name "oidc-agent-\*" \) -name "oidc-\*" -exec rm -rf {} \;;

eval $(oidc-agent-service use 2>&1 /dev/null) &> /dev/null;

if [ $? == 1 ]; then

pkill -u $USER oidc-agent;

echo "No se pudo reusar un agente, se creara uno nuevo";

find /tmp -maxdepth 1 -user $USER -not \( -name "oidc-agent-\*" \) -name "oidc-\*" -exec rm -rf {} \;;

kill -9 $OIDCD\_PID &> /dev/null;

unset OIDC\_SOCK;

unset OIDCD\_PID;

unset OIDCD\_PID\_FILE;

\_create\_agent;

else

echo "Se reuso el agente";

find /tmp -maxdepth 1 -user $USER -name oidc-agent-service -exec chmod 1777 {} \;;

fi}

Busca en /tmp archivos oidc-agent-service propiedad del usuario y aplica chmod 1777 (los usuarios solo pueden eliminar sus propios archivos dentro de él, no los de otros usuarios), intenta matar el agente, luego mata cualquier proceso oidc-agent del usuario y limpia archivos temporales no deseados en /tmp. NOTA: No comprendo en su totalidad el comando eval, pero podría decir viendo los “echos” que se intenta reutilizar un agente ya activo y si no es posible se reinicia o se crea uno nuevo.

#### set-function activar\_acceso\_gridunam {

\_\_check\_token}

Es un alias para iniciar la validación del token antes de realizar acciones que requieran acceso.

#### set-function consultar\_jobs {

\_\_check\_token;

entidad=${1};

shift;

condor\_ce\_q -name ${entidad} -pool ${entidad}:${PORT} $\*}

Se verifica que haya un token, se asigna entidad al primer argumento, shift elimina ese primer argumento para que $\* contenga los demás parámetros opcionales (desplaza los argumentos posicionales una posición a la izquierda) y el comando condor\_ce\_q apunta al -name y -pool de la entidad (host y puerto) para pasar los parámetros adicionales. Nos sirve para consultar la cola de Jobs en HTCondor-CE.

#### set-function crear\_cuenta\_oidc {

\_\_check\_account\_oidc}

Igualmente es un alias que ejecuta la comprobación/creación de una cuenta OIDC

#### set-function descargar\_archivos {

\_\_check\_token;

entidad=${1};

jobid=${2};

shift 2;

condor\_transfer\_data -name ${entidad} -pool ${entidad}:${PORT} ${jobid} $\*}

Se verifica el token, se toma la entidad y jobid de los dos primeros argumentos, los desplaza 2 lugares a la izquierda con shift 2 y llama al comando condor\_transfer\_data para descargar archivos del job identificado en el puerto correspondiente

#### set-function eliminar\_mis\_agentes {

$(oidc-agent-service kill 2>&1 /dev/null) &> /dev/null;

pkill -u $USER oidc-agent;

find /tmp -maxdepth 1 -user $USER -not \( -name 'oidc-agent-\*' \) -name "oidc-\*" -exec rm -rf {} +;

unset BEARER\_TOKEN}

Se intenta usar oidc-agent-service kill para terminar agentes, mata procesos oidc-agent del usuario, borra archivos temporales de oidc en /tmp y al final se borra la variable del entorno local.

#### set-function enviar\_jobs {

\_\_check\_token;

entidad=${2};

script\_submit=${1};

shift 2;

condor\_ce\_submit ${script\_submit} -remote ${entidad} -pool ${entidad}:${PORT} $\*}

Se verifica el token, desplaza script\_submit al primer argumento y entidad al segundo, se eliminan y se ejecuta condor\_ce\_submit para enviar un script de entrega al puerto remoto

#### set-function estado\_cluster {

\_\_check\_token;

entidad=${1};

shift;

condor\_ce\_status -any -pool ${entidad} $\*}

Se llama a condor\_ce\_status para obtener el estado del closter con -any para cualquier recurso apuntando al puerto entidad

#### set-function generar\_token {

\_\_check\_load\_account\_oidc;

export BEARER\_TOKEN=$(oidc-token ${grid\_account\_oidc})}

Primero se asegura que la cuenta OIDC esta creada y cargada y después ejecuta oidc-token <cuenta> para obtener el token OIDC y lo exporta como BEARER\_TOKEN, haciendo posible que otras funciones puedan usarlo.

#### set-function ping\_cluster {

\_\_check\_token;

entidad=${1};

shift;

condor\_ce\_ping -type SCHEDD -name ${entidad} -pool ${entidad}:${PORT} -table WRITE $\*}

Se ejecuta condor\_ce\_ping con tipo SCHEED enviando una tabla WRITE para probar la conectividad con el servicio planificador del puerto

#### set-function run\_test\_cluster {

\_\_check\_token;

entidad=${1};

shift;

condor\_ce\_run -r ${entidad} $dir\_gridunam/script-test-gridunam.sh $\*}

Se ejecuta condor\_ce\_run – r <entidad> para ejecutar remotamente el script de pruebas script-test-gridunam.sh dentro del directorio $dir\_gridunam

#### set-function show\_cpus\_all\_cluster {

color=$(tput blod; tput setaf 3);

normal=$(tput sgr0);

tput clear;

fquery\_all\_cpus=/tmp/.query\_${USER}\_all\_cpu;

rm -rf ${fquery\_all\_cpus};

for i in submit.grid.unam.mx jamatu.astrosen.unam.mx jupyter2.atmosfera.unam.mx condor-grid.lamod.unam.mx;

do

echo "Consultando ${color} $i ${normal}";

show\_cpus\_cluster $i | tee -a ${fquery\_all\_cpus};

echo;

done;

tput setaf 3;

echo "Total de cores CPU en GridUNAM:";

tput sgr0;

grep totales ${fquery\_all\_cpus} | awk '{print $1 "\t" $6}' | sed 's/\x1B[@A-Z\\\]^\_]\|\x1B\[[0-9:;<=>?]\*[-!"#$%&'"'"'()\*+,.\/]\*[][\\@A-Z^\_`a-z{|}~]//g' | awk '

{

tcores\_idle+=$1;

tcores+=$2;

}

END {

if (tcores\_idle == 0)

{

system("tput blink; tput setaf 1");

printf tcores\_idle " cores CPU disponibles ";

system("tput sgr0");

printf "de " tcores " totales\n";

}

else

{

system("tput blink; tput setaf 2");

printf tcores\_idle " cores CPU disponibles ";

system("tput sgr0");

printf "de " tcores " totales\n";

}

}';

rm -rf ${fquery\_all\_cpus}}

Existen algunos comandos que desconozco completamente pero viendo parte de este script puedo intuir que tiene el propósito de reunir la información de todos o algunos de los clústeres disponibles.

#### set-function show\_cpus\_cluster {

\_\_check\_token &> /dev/null;

entidad=${1};

shift;

fquery\_cpus=/tmp/.query\_${USER}\_${entidad}\_cluster\_cpu;

condor\_ce\_run -lr ${entidad} $dir\_gridunam/test-cmds-slurm.sh $\* | tee ${fquery\_cpus} | awk '{print $9}' | grep -v CPU | tr -s "/" "\t" | awk '

{

cores\_idle+=$2;

cores+=$4

}

END {

if (cores\_idle == 0)

{

system("tput blink; tput setaf 1");

printf cores\_idle " cores CPU disponibles ";

system("tput sgr0");

printf "de " cores " totales\n";

}

else

{

system("tput blink; tput setaf 2");

printf cores\_idle " cores CPU disponibles ";

system("tput sgr0");

printf "de " cores " totales\n";

}

}';

cat ${fquery\_cpus};

rm ${fquery\_cpus}}

Igual que el anterior, pero para consultar un cluster en específico.

### Flujo

Al concluir la revisión de los scripts puedo suponer el flujo de trabajo de los cripts:

1. Verificación de agente OIDC
2. Creación o reutilización de un agente
3. Asegurar la existencia de una cuenta y archivo de contraseña
4. Carga de una cuenta en el agente
5. Generación de un token BEARER
6. Utilización del token para ejecutar comandos en HTCondor sobre el grid (como consultas, envíos o descargas)
7. Funciones de monitorización de recursos

### Propuestas de migración a Python.

He revisado los scripts actuales del módulo gridunamUI los cuales implementan una serie de funciones en Bash para automatizar la autenticación OIDC (OpenID Connect) y la interacción con el sistema de colas HTCondor-CE.

Considero que Bash es adecuado para la automatización en terminales Linux, pero al mismo tiempo su capacidad de estructuración, manejo de errores y mantenimiento a largo plazo es limitada.

Por ello, estoy implementando algunas propuestas de migración gradual a Python, un lenguaje con soporte nativo para programación estructurada, control de procesos, manipulación de archivos y comunicación con APIs, además de contar con bibliotecas específicas para OIDC y HTCondor.

La migración que planeo es en tres niveles de complejidad, que permiten avanzar progresivamente sin comprometer la compatibilidad con el entorno actual del clúster, los cuales son los siguientes:

#### Traducción directa

En este primer nivel propongo una traducción literal de los scripts Bash al lenguaje Python, manteniendo la lógica y los comandos originales.

El objetivo principal es preservar la compatibilidad total con el entorno del módulo gridunamUI, evitando romper las dependencias con las herramientas ya instaladas (oidc-agent, condor\_ce\_\*, jq, awk, etc.).

Para ejecutar comandos del sistema, se utilizaría el módulo estándar de Python **subprocess**, que permite invocar procesos externos y capturar su salida o códigos de retorno.

De esta manera, cada set-function definida en Bash se transforma directamente en una función Python con la misma lógica condicional y los mismos flujos de validación.

Hacerlo de esta manera tiene varias ventajas como la reutilización completa de la infraestructura existente del clúster, conserva la lógica de Bash ya existente, permite validar paso a paso cada componente sin modificar el flujo operativo y considero que es el enfoque más seguro para comenzar la migración en un entorno de suma importancia.

De igual manera considero que tiene algunas limitaciones como depender de las mismas herramientas externas y no se aprovecharía las ventajas estructurales ni de manejo de errores de Python.

Recomiendo usar esta propuesta de migración si es la primera vez que se migrara de lenguaje de programación, ya que estaremos sustituyendo progresivamente los scripts sin modificar el comportamiento observable.

#### Reimplementación con librerías de Python

En este nivel propongo que algunas partes del código original se reescriban, modifiquen o se sustituyan para optimizar el funcionamiento o para integrarlos con nuevas herramientas sin la necesidad de hacerlo todo desde cero, reemplazando progresivamente los comandos Bash por librerías de Python especializadas.

El objetivo aquí es mejorar el código y reducir la dependencia del intérprete Bash.

En este caso se podrían utilizar bibliotecas compatibles como por ejemplo:

* python-oidc-agent o requests-oauthlib, para autenticación OIDC y gestión de tokens.
* htcondor, para interactuar directamente con el sistema de colas sin usar comandos externos (condor\_ce\_submit, condor\_ce\_status, etc.).
* Módulos estándar de Python (os, pathlib, json, base64, time) para reemplazar utilidades de shell (grep, awk, jq, date, etc.).

Aquí tendríamos de igual manera varias ventajas, como el hecho de que el código sería completamente portable entre sistemas sin depender de utilidades Unix, se mejoraría la seguridad y la legibilidad del código y se nos permitiría un control más fino de excepciones y errores.

En cuanto a las limitaciones se requeriría validar la equivalencia funcional con las herramientas de Bash y exige instalar y mantener dependencias Python adicionales en dado caso de que no se encuentren disponibles en el clúster.

#### Integración nativa con APIs OIDC y HTCondor

Aquí planteo una reingeniería completa del flujo operativo, utilizando exclusivamente interfaces de programación (APIs) o librerías oficiales en Python.

Las llamadas a herramientas de línea de comandos desaparecerían por completo, siendo reemplazadas por invocaciones directas a las bibliotecas de OIDC y HTCondor.

Por ejemplo, la gestión de tokens se realizaría directamente a través de la API OIDC, mientras que la interacción con el sistema de colas podría ser manejada mediante las Python bindings de HTCondor.

De esta forma, considero que Python dejaría de actuar como intermediario entre el usuario y el shell, convirtiéndose en un cliente nativo del clúster.

Hay muchas ventajas al implementar una migración de esta manera, ya que eliminamos la dependencia total de los scripts y comandos Bash, se facilita la integración con servicios externos, mejora la eficiencia al evitar procesos intermedios y se nos permitirá incorporar pruebas unitarias y control de excepciones más avanzado.

A pesar de lo anterior, se requiere modificar significativamente la arquitectura del módulo, además de que demanda mayor conocimiento técnico sobre las APIs internas de HTCondor y OIDC y sin olvidar que puede requerir permisos o configuraciones especiales en el clúster.

# BITACORA