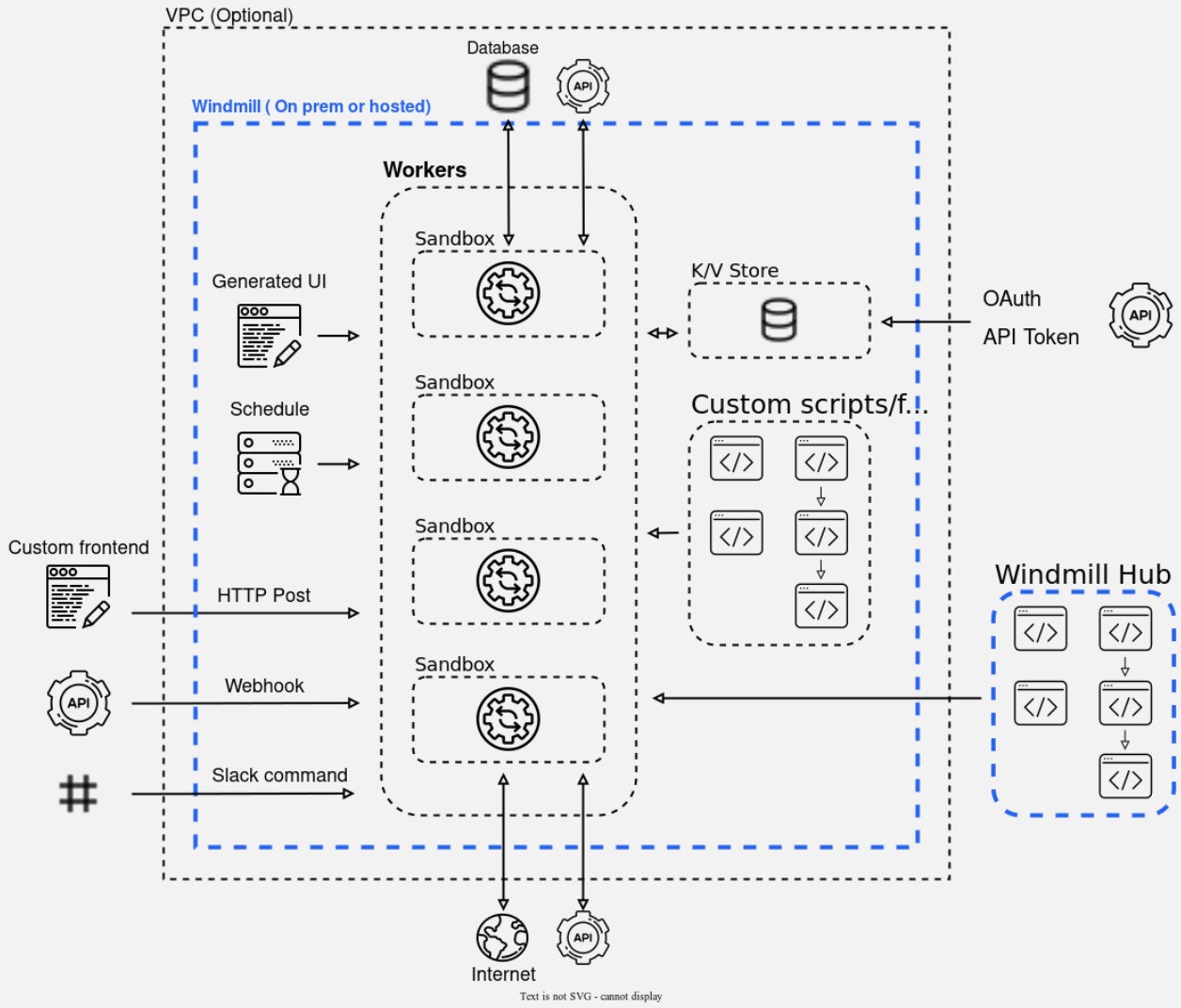
Windmill Open Source Script Orchestration Tool



1. INTRODUCCIÓN

Windmill es una herramienta de código abierto que permite la orquestación de scripts en diversos lenguajes como TypeScript, Go, Python, Bash...

En adición al orquestador, ofrece un entorno de ejecución para cada uno de los lenguajes soportados y la posibilidad de diseñar interfaces personalizadas para cada script o flow a ejecutar, para poder ver de manera más intuitiva las entradas necesarias o los resultados obtenidos. La herramienta puede ser utilizada como app en la nube o en su vertiente self-hosted, y en ambos casos podemos trabajar desde la interfaz web o su CLI indistintamente.

Referencias

Workspace

Scripts

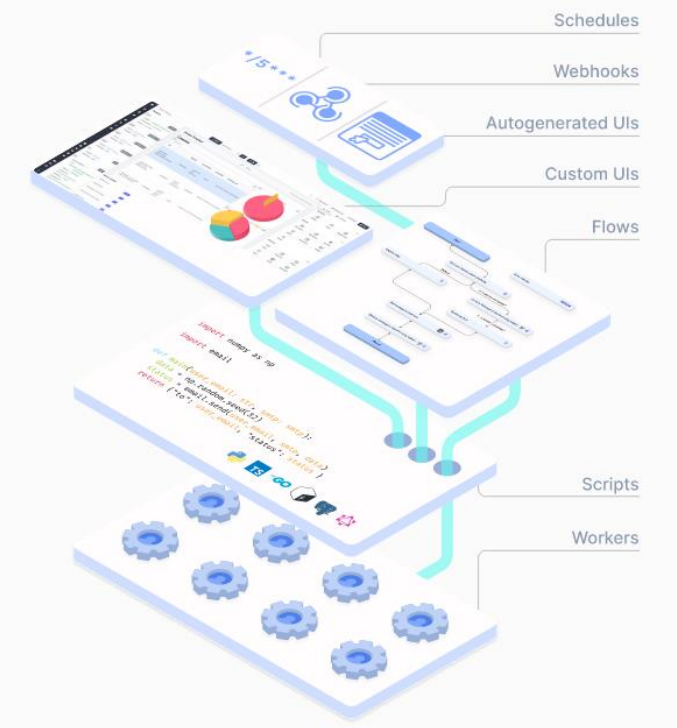
Flows

Apps

Usuarios, grupos y carpetas

Variables contextuales y definidas por el usuario

Recursos

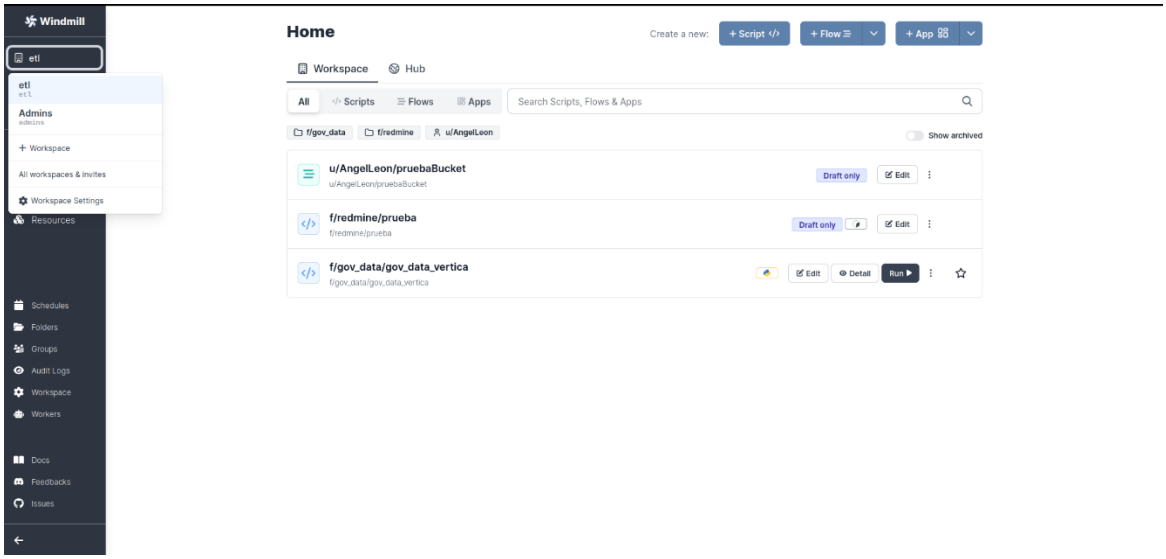


2. CONCEPTOS BÁSICOS

Workspace

Cualquier entidad de Windmill está ligada a un workspace, desde los usuarios hasta los scripts. De hecho, la base de datos en la que se almacenan las diferentes entidades está particionada por workspace, para aislar al máximo entidades presentes en diferentes contextos. Esto permite separar de manera segura objetos pertenecientes a diferentes organizaciones.

- *Workspaces disponibles*



Como se observa en la imagen anterior, podemos desplazarnos entre workspaces clicando en la parte superior izquierda, y dependiendo del workspace en el que nos encontremos tendremos un nombre de usuario u otro (lo único que permanece invariable para el usuario es su correo y contraseña). De igual manera, al seleccionar un workspace sólo serán visibles aquellos objetos ligados a dicho workspace, ya sean scripts, flows, variables...

Scripts

La unidad básica de ejecución y orquestación en Windmill es el script. Como se ha comentado en el primer apartado, estos pueden estar escritos en diversos lenguajes y, en adición, no requieren del empleo de una sintaxis específica para la herramienta. Cabe señalar, sin embargo, que cuando hablamos de script no nos referimos únicamente al código a ejecutar, sino que incluimos también los metadatos asociados al mismo, como podrían ser su nombre, descripción, ruta...

Los scripts pueden ser ejecutados manualmente o de manera automática a través de la definición de un ***schedule,*** empleando para ello una sintaxis similar al cron. Para ejecutar con éxito un script necesitamos añadir una función main() que actúe como punto de entrada y, si necesitamos especificar parámetros de entrada basta con añadirlos a la función main. Dependiendo de los parámetros especificados, Windmill generará una interfaz por defecto para hacer más intuitiva la escritura y especificación de los valores de los parámetros. Un script sencillo de Windmill escrito en Typescript tiene una estructura como la siguiente:

```
// Ctrl+. to cache dependencies on imports hover, Ctrl+S to
format.

// import { toWords } from "npm:number-to-words@1"
// import * as wmill from
"https://deno.land/x/windmill@v1.89.1/mod.ts"

export async function main(
    a: number,
    b: 'my' | 'enum',
    d = 'inferred type string from default arg',
    c = { nested: 'object' }
    //e: wmill.Base64
) {
    let x = await wmill.getVariable('u/user/foo')
    return { foo: a };
}
```

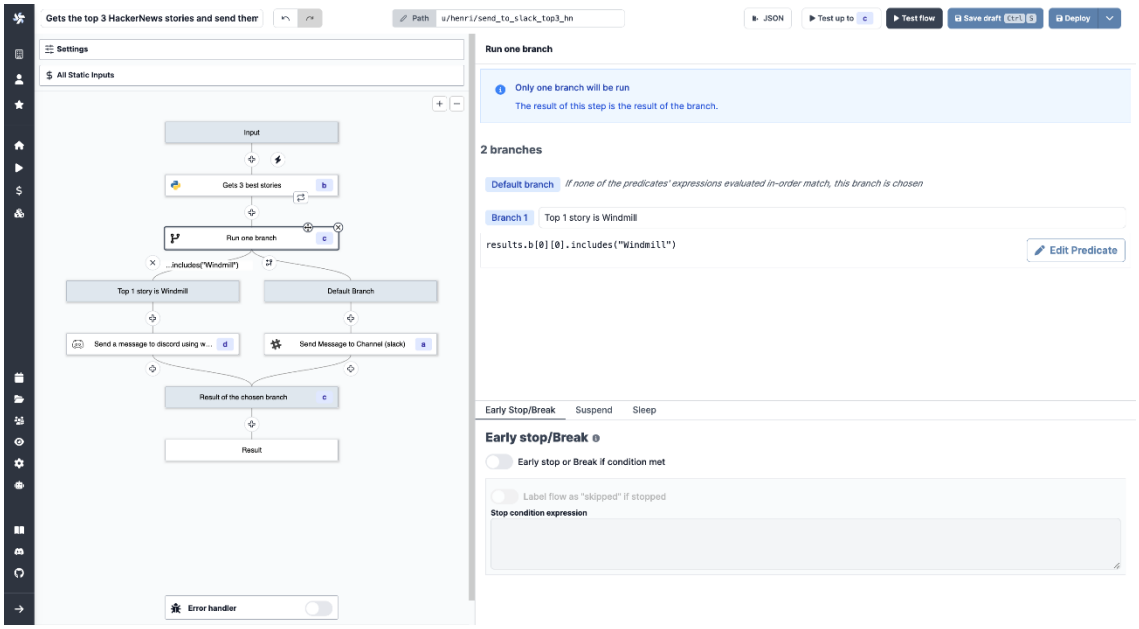
Como se puede observar, hay una línea en la que se llama a una función llamada ***getVariable().*** El concepto de variable lo explicaremos en profundidad en siguientes apartados, pero a grandes rasgos es muy similar a las variables en programación. En el siguiente vídeo de ejemplo se muestra cómo crear y ejecutar un script.

Flows

Como hemos comentado, en Windmill podemos operar directamente con scripts escritos en los diversos lenguajes soportados sin necesidad de añadir sintaxis específica de la herramienta. Sin embargo, en ocasiones nos puede interesar construir un workflow algo más complejo que contenga pasos o scripts adicionales.

En estos casos Windmill nos ofrece los flows, que no son más que una serie de pasos secuenciales en los que es posible realizar gran variedad de tareas predefinidas o personalizadas. Entre estas tareas podemos encontrar algunas de índole más genérica, como podría ser un paso de ejecución de código o un ***if{}else{}*** (llamada branch en Windmill), u otras más orientadas al manejo de servicios concretos, como podría ser Amazon S3.

- *Flow con branch*

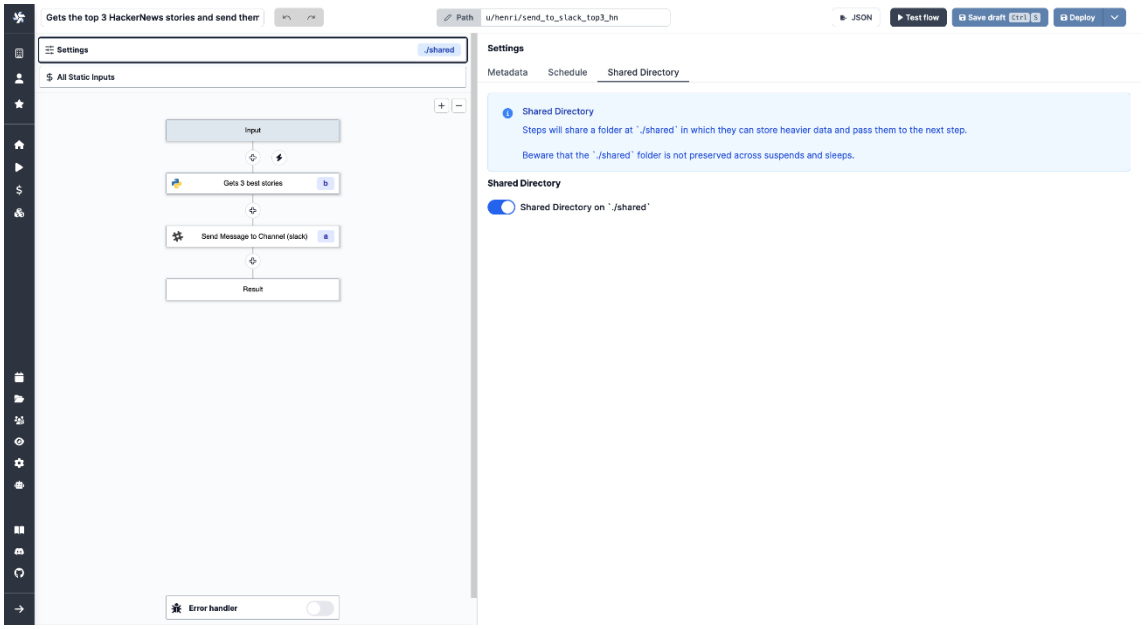


Al igual que los scripts, los flows también disponen de metadatos como el nombre, la descripción o la ruta donde se almacenará el flow, así como la posibilidad de programar su ejecución empleando una sintaxis tipo cron.

A diferencia de los scripts, y como consecuencia de la potencial ejecución de diversos scripts, también se nos permite transmitir los resultados de un paso concreto al siguiente.

Si los objetos a transmitir son relativamente grandes o complejos, en vez de utilizar directamente la salida anterior podemos emplear la carpeta *shared*, que no es más que un espacio conjunto empleado para compartir de manera temporal (se borra después de cada ejecución) determinados archivos entre pasos o scripts.

- *Directorio shared*

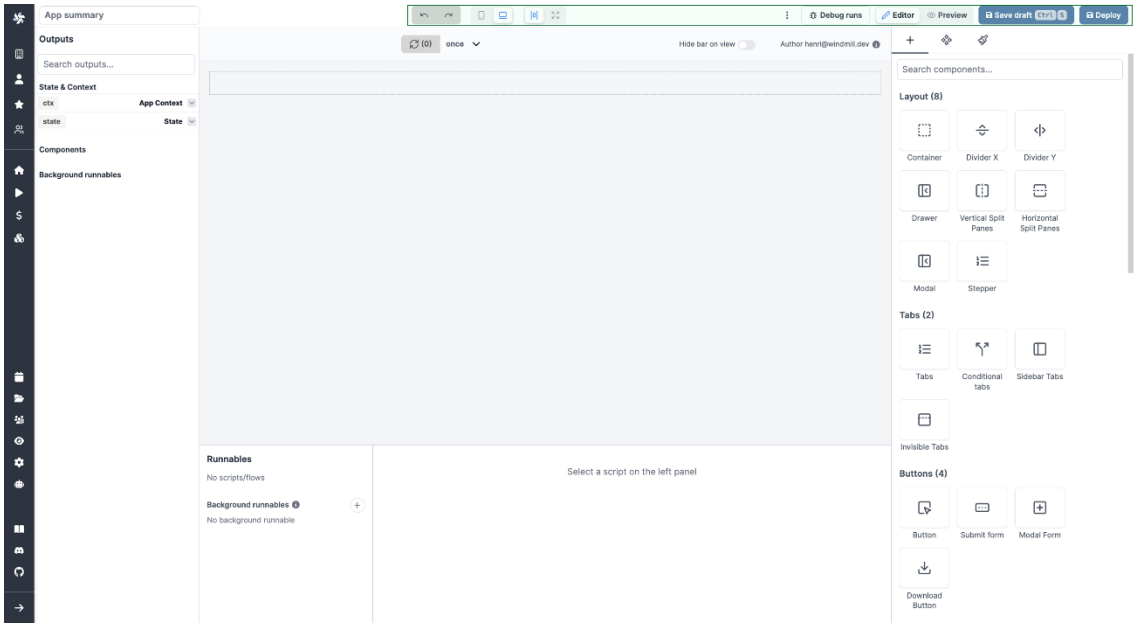


En el siguiente vídeo se muestra cómo crear y ejecutar un flow.

Apps

Como hemos comentado en la introducción, Windmill genera automáticamente UIs personalizadas en función de los parámetros de entrada de un script o flow, pero también nos brinda la posibilidad de desarrollar nuestras propias UIs personalizadas para acercar el entorno del flow/script a un usuario que no tenga conocimientos técnicos. El objetivo principal de las apps es crear un entorno para nuestro flow que nos permita interactuar de manera sencilla e intuitiva con fuentes de datos, los parámetros de entrada y las salidas. Windmill permite crear las apps desde su interfaz web o importar una aplicación existente que emplee Vue, React o Svelte, y, en adición, podemos hacer uso de alguna de las apps presentes en Windmill Hub (proyectos de la comunidad) en caso de ser adecuadas para nuestro caso concreto. Por último, cabe señalar que las apps pueden ser para escritorio o móvil, y es algo que debemos tener en cuenta al iniciar el proceso de desarrollo. A continuación, se muestra un ejemplo de cómo sería el desarrollo de una app a través de la interfaz web de Windmill.

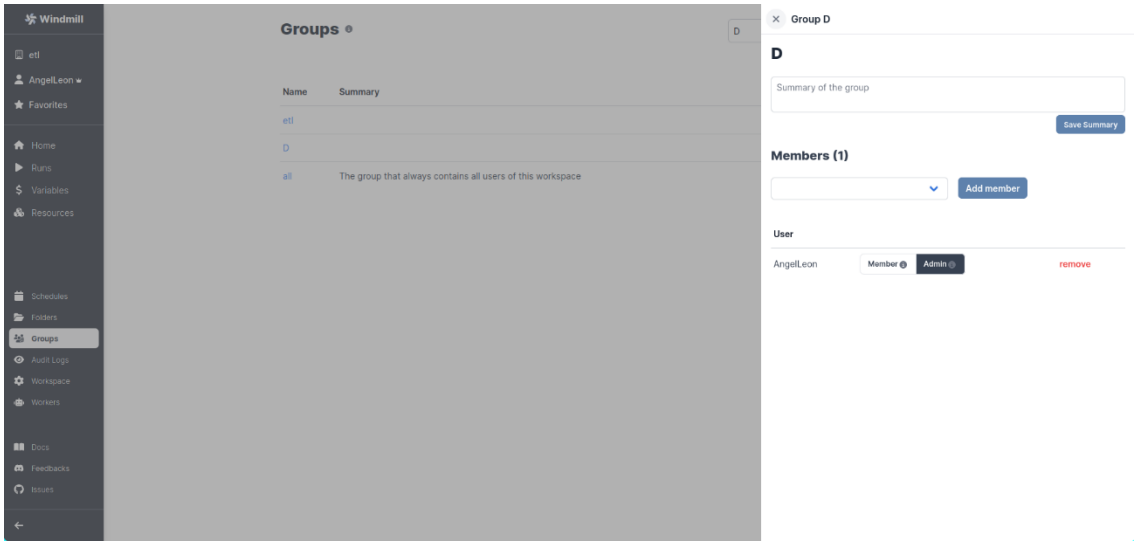
- *Interfaz web para el desarrollo de apps*



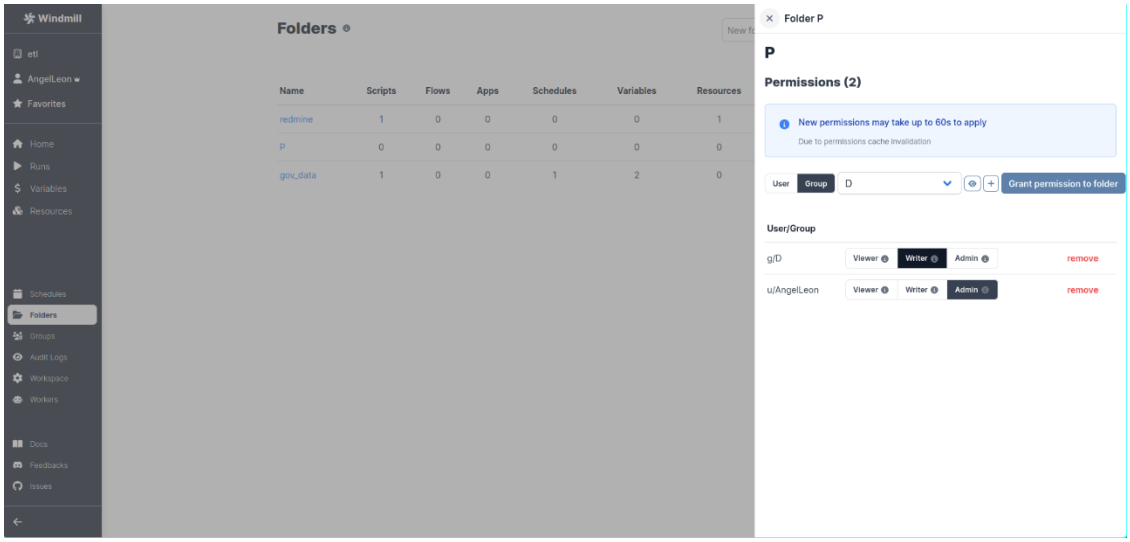
Usuarios, grupos y carpetas

Windmill dispone de manejo de usuarios en conjunto con un sistema de acceso basado en roles (llamados grupos en Windmill), que son básicamente usuarios que poseen permisos similares. Los permisos básicos disponibles son de lectura (viewer), de lectura-escritura (writer) o lectura-escritura con cambio de permisos (admin). Los objetos pueden tratarse de manera individual o en carpetas, que son empleadas para agrupar objetos según los roles/usuarios que tendrán acceso a ellos. Por ejemplo, supongamos que tenemos una serie de scripts pertenecientes a un proyecto llamado P y deseamos que accedan únicamente los empleados del departamento D. Para facilitar el trabajo, podemos crear una carpeta llamada P en la que introduciremos los scripts correspondientes y un grupo llamado D que contendrá los usuarios del departamento D y que tendrán acceso únicamente a los objetos contenidos en la carpeta P. Tras ello sólo quedaría, desde la ventana de la carpeta P, dar acceso al grupo D y fijar los permisos convenientes.

- *Creación del grupo D*

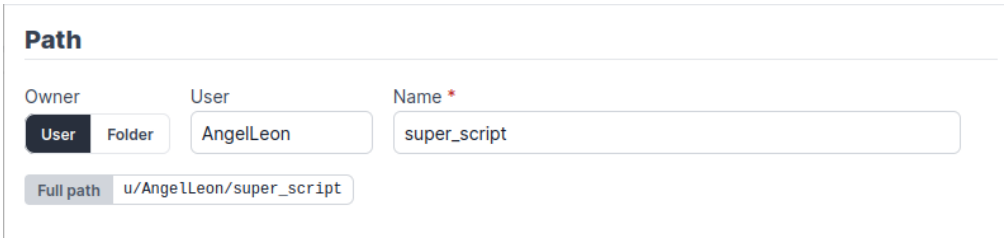


- *Creación de la carpeta P*

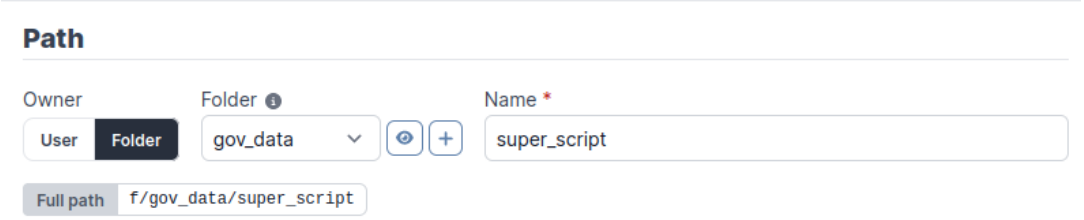


Al crear un script o un flow se nos preguntará el path, que por defecto empieza por u/. El prefijo inicial indica si el propietario del objeto será un usuario concreto o los usuarios/grupos que tengan acceso a la carpeta en la que se almacenará (en este caso emplearemos f/ como prefijo). En lo referente al resto del path, tras escribir el prefijo especificamos el nombre del usuario o carpeta y por último el nombre del script/flow.

- *Path para un usuario*



- *Path para una carpeta*



Variables contextuales y definidas por el usuario

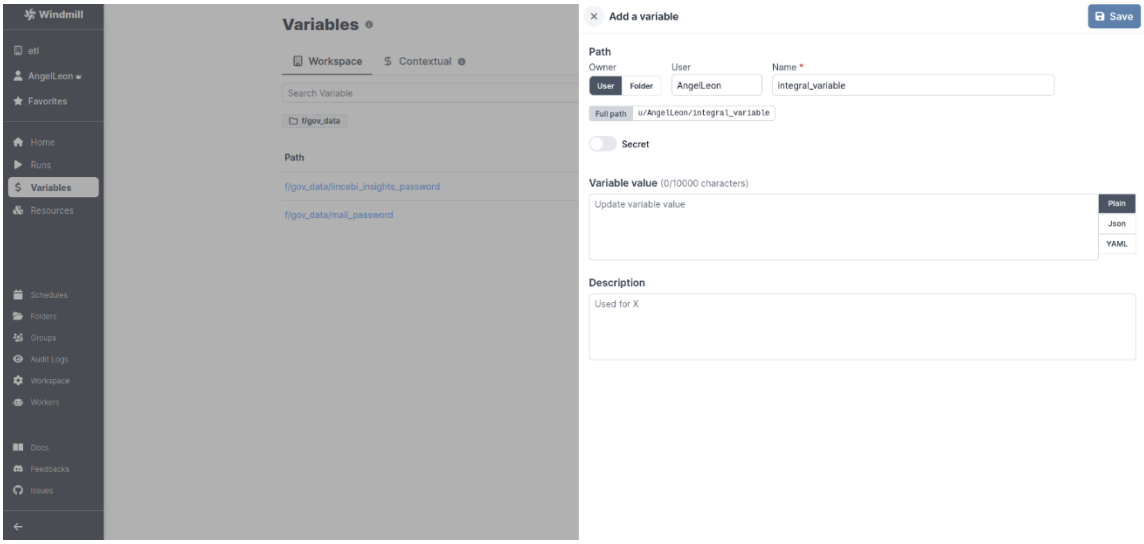
En ocasiones, y con el objetivo de evitar repeticiones innecesarias entre scripts, conviene crear variables que almacenen ciertos valores. Windmill permite la creación de este tipo de variables, a las que llama variables definidas por el usuario.

Dentro de este tipo de variables **tenemos un caso concreto, que es el de los secretos, que a diferencia del resto de variables oculta el valor guardado tras ser introducido por primera vez, y lo almacena cifrado en base de datos**.

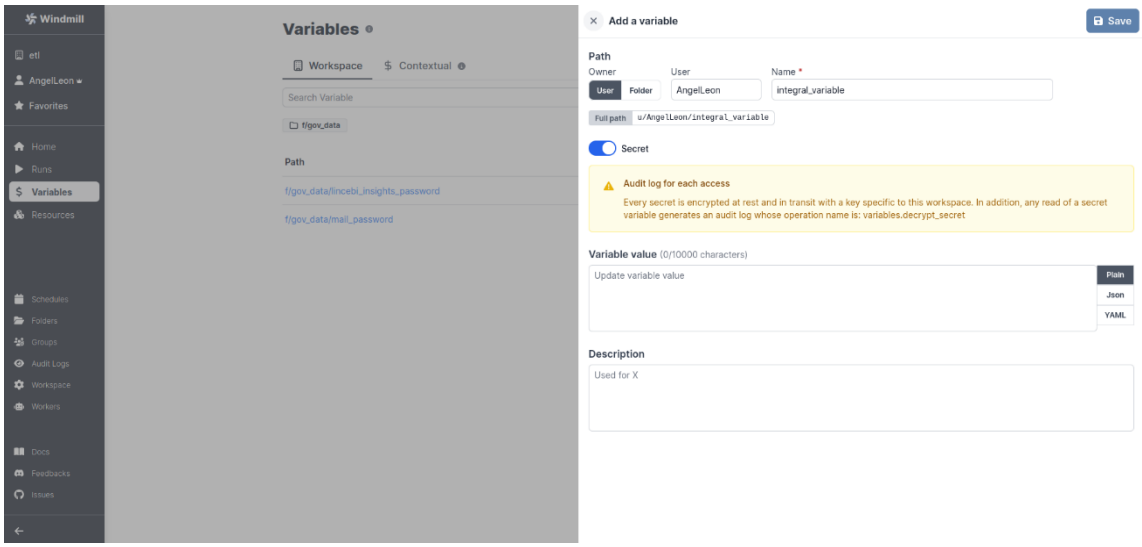
Este caso es bastante útil cuando necesitamos emplear valores especialmente sensibles, como contraseñas, en el código de un script o flow. Por otro lado, tenemos las variables contextuales, que ya vienen definidas por defecto y son básicamente variables de entorno que nos indican diversos aspectos del entorno de ejecución, como podría ser el usuario que ejecuta un script.

No se permite la adición o modificación de este tipo de variables, sino únicamente su consulta. Al igual que los scripts y los flows, las variables tienen un propietario individual o se almacenan en carpetas.

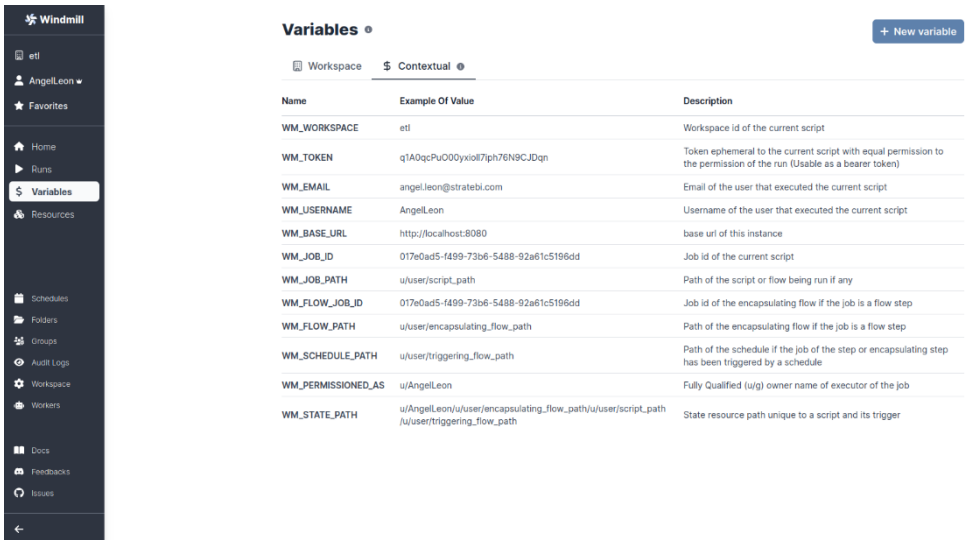
- *Variable común definida por el usuario*



- *Variable secreta definida por el usuario*



- *Variables contextuales*



Como comentamos en el apartado de scripts, podemos acceder fácilmente a estas variables empleando las librerías que nos aporta Windmill en conjunto con el path. Por ejemplo, en Python se haría de la siguiente manera:

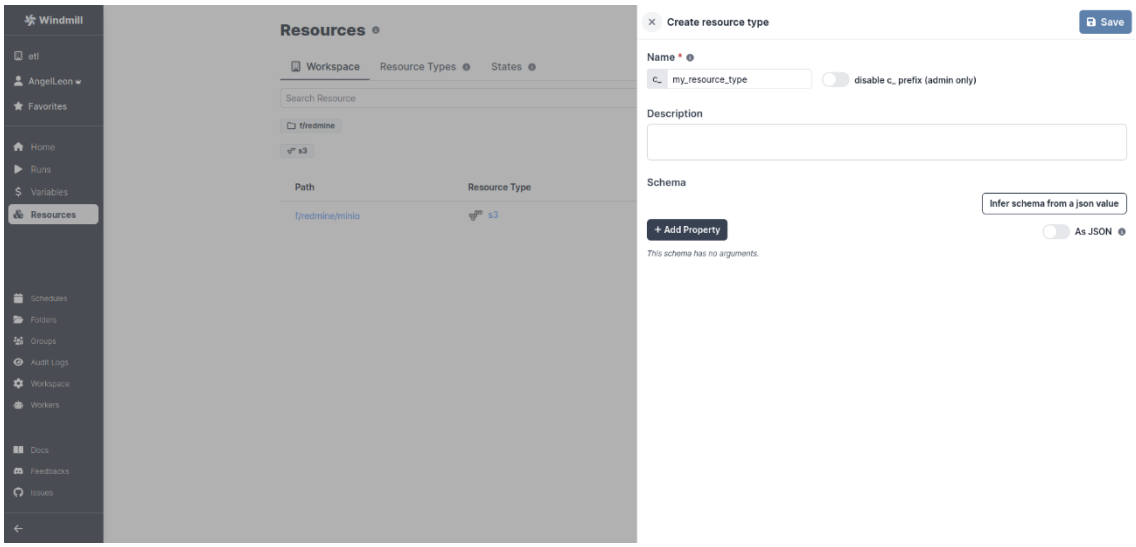
```
import os
import wmill


context_variable = os.environ.get("WM_WORKSPACE")
user_defined_secret = wmill.get_variable("f/examples/secret")
```

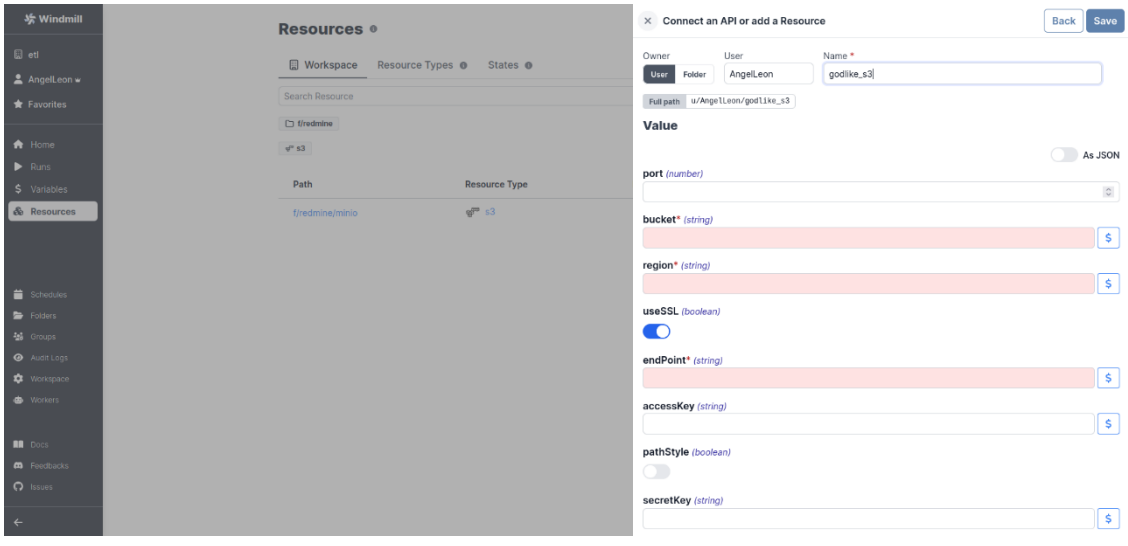
Recursos

En resumen, los recursos son conexiones a servicios externos. Gracias a la amplia gama de servicios compatibles, es posible comunicarse con facilidad con servicios en la nube, bases de datos, redes sociales, plataformas de desarrollo colaborativo... En adición a los tipos de conexión predefinidos también se brinda la opción de definir tipos personalizados, en caso de que el servicio con el que deseemos comunicarnos no disponga de un tipo específico.

- Creación de un nuevo tipo de recurso



- Creación de una nueva conexión con un tipo predefinido



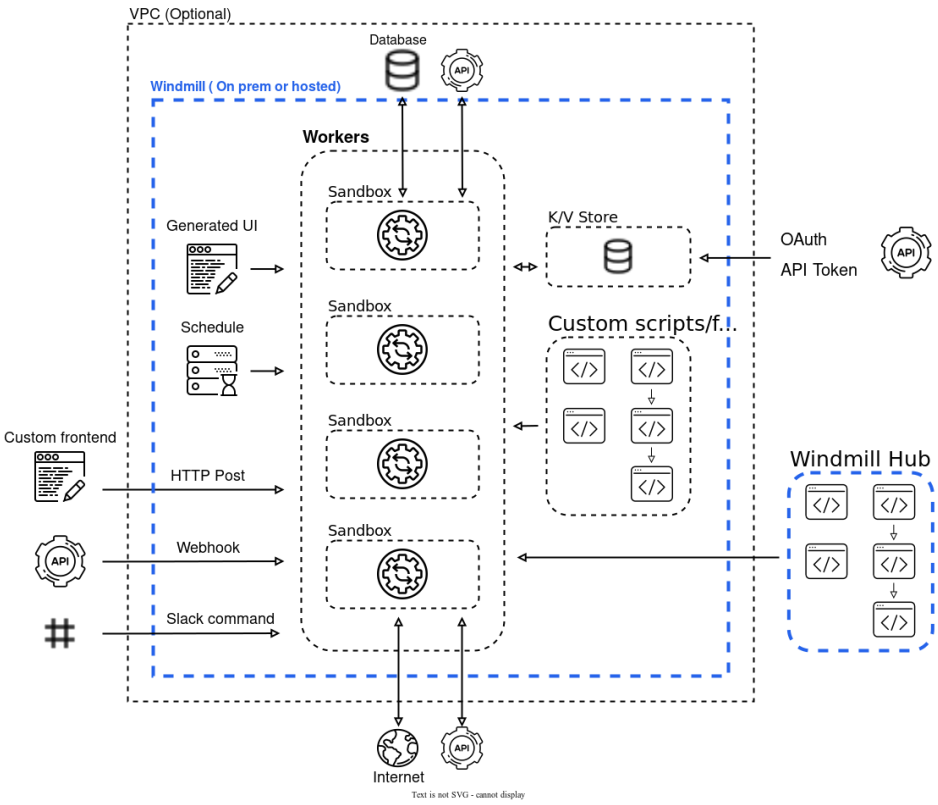
La existencia del apartado de recursos permite mantener de manera centralizada las conexiones a terceros con sus respectivas credenciales asociadas. Estas credenciales podrán ser utilizadas más tarde en scripts o en pasos de un flow.

3. ARQUITECTURA DE WINDMILL

La arquitectura de Windmill es relativamente sencilla si abstraemos un poco. Los componentes más relevantes de la arquitectura son:

- ***Windmill server***: Se trata del servidor principal, y constituye el cerebro de Windmill. Dispone de una interfaz web para operar con los objetos del sistema.
- ***Windmill database***: Se encarga de almacenar los metadatos de los scripts y los flows, información relevante sobre las ejecuciones, variables definidas...
- ***Windmill worker***: Encargado de la ejecución de los scripts y flows, se pone en contacto periódicamente con el servidor principal para consultar si hay trabajo que realizar. Se recomienda tener varios en funcionamiento para aumentar la disponibilidad.
- ***Windmill lsp***: Módulo que permite el uso de intellisense en el editor web del servidor principal.

Aunque sólo he comentado los componentes más importantes, en la siguiente imagen se puede ver con más detalle la arquitectura completa:



4. WINDMILL VS AIRFLOW

Si lo comparamos con otras herramientas del sector como ***Airflow***, es posible afirmar que ambas son bastante competentes, pero desempeñan su trabajo de maneras diferentes. Las diferencias más notables que podemos encontrar entre estas dos herramientas son:

- En Airflow se emplea el término workflow, mientras que el equivalente en Windmill es el flow.

- Airflow se rige rigurosamente por una política de workflows como código, mientras que Windmill es algo más flexible en este aspecto y permite el uso de la interfaz web para la creación y manejo de los distintos objetos presentes en la herramienta

- Airflow fundamenta los workflows en la especificación de DAGs explícitamente a través de scripts de Python, empleando para ello una sintaxis más específica que los flows de Windmill, que como hemos observado anteriormente son básicamente scripts tradicionales. Al definir flows a través de la interfaz web Windmill también emplea DAGs por debajo, pero todo esto es transparente para el usuario.

- La arquitectura y modo de trabajo son distintos en ambos casos. Por ejemplo, en Windmill el servidor principal delega la ejecución y el procesamiento de los workflows en los workers, mientras que Airflow ejecuta el código en el servidor pricipal (aunque realice operaciones en máquinas externas).

- Airflow soporta exclusivamente Python, mientras que Windmill permite la ejecución de scripts en otros lenguajes como TypeScript, Go, SQL o incluso Bash.

- Windmill genera UIs de manera automática dependiendo de los parámetros de entrada, y permite la creación de UIs personalizadas para facilitar la definición de los valores de entrada y la interpretación de las salidas, mientras que Airflow no dispone de esta utilidad.

- Windmill es una herramienta relativamente nueva, por lo que todavía no dispone de una comunidad formada y es posible que presente problemas en alguna característica.

- Airflow ofrece un sólido nivel de integración con diversos servicios externos, mientras que Windmill, a pesar de ofrecer integración con gran cantidad de servicios externos, presenta una menor cantidad de operaciones predefinidas sobre dichos servicios.

- Windmill dispone de un módulo independiente (llamado windmill-lsp) encargado de aportar intellisense al editor web de código del servidor principal.

- La curva de aprendizaje y complejidad general es mayor en Airflow.

Si expresamos estas diferencias de manera resumida:

Airflow	***Windmill***
Término workflow	*Término flow*
Workflows como código	*Mayor uso de interfaz web*
Sintaxis propia	*Scripts tradicionales*
Mayor curva de aprendizaje	*Menor curva de aprendizaje*
Soporte para Python	*Soporte para Typescript, Python, Go, Bash y SQL*
No dispone de generación automática de UIs o creación UIs personalizadas	*Generación automática de UIs para flows y creación de UIs personalizadas*
Herramienta firmemente consolidada y con gran comunidad	*Herramienta relativamente nueva y una comunidad por formar*
Integración de calidad con servicios externos	*Buen nivel de integración, poca diversidad de operaciones predefinidas*
No dispone de intellisense	*Módulo adicional que aporta intellisense*

5. MAS TUTORIALES

1. Integracion SAP - PowerBI
2. PowerBI Trucos (Vol I)
3. PowerBI Trucos (Vol II)
4. PowerBI + Synapse Analytics (paso a paso)
5. 30 Consejos y Buenas Prácticas para hacer un proyecto de Power BI con éxito
6. Cómo crear diseños de Dashboards espectaculares con PowerBI
7. Videotutorial: Trabajando con Python en Power BI
8. Aplicación PowerBi Turismo
9. Aplicación PowerBI Financiera I
10. Aplicación PowerBI Financiera II
11. Aplicación PowerBI eCommerce
12. Aplicación PowerBI Salud
13. Aplicación PowerBi Smart City
14. Aplicación PowerBI Energía
15. Aplicación PowerBI Sports Analytcis
16. Power BI Premium Utilization and Metrics
17. PowerBI Embedded: Funcionamiento y costes
18. Bravo para PowerBI
19. Como integrar Power BI con Microsoft Dynamicssalesfo
20. SQL Server Profiler para Power BI
21. Como usar Report Analyzer en PowerBI, para mejorar el rendimiento
22. Power BI embebido en Jupyter Notebook
23. Tabular Editor para Power BI: Videotutorial y manual en español
24. Personaliza tus gráficas en Power BI con Charticulator y Deneb
25. Comparativa PowerBI vs Amazon QuickSight
26. Como usar emoticonos en PowerBI
27. Buenas prácticas con Dataflows en Power BI
28. Power Automate para Power BI: Cómo funciona
29. ALM Toolkit para Power BI
30. Os presentamos Goals in Power BI para hacer Scorecards
31. Tutorial gratuito en español sobre Power BI Report Builder
32. Conoce PowerBI Diagram View (Visual Data Prep). Paso a paso
33. Futbol Analytics, lo que hay que saber
34. Dashboard de medicion de la calidad del aire en Madrid
35. Como funciona Microsoft Power BI? Videoturial de Introducción
36. Big Data para PowerBI
37. Quieres crear aplicaciones empresariales usando PowerBI, PowerApps y Power Automate de forma conjunta?
38. Power BI tip: Uso de parámetros what-if
39. Como integrar Salesforce y PowerBI
40. Videotutorial: Usando R para Machine Learning con PowerBI

41. Las 50 claves para aprender y conocer PowerBI
42. PowerBI: Arquitectura End to End
43. Usando Python con PowerBI
44. PowerBI + Open Source = Sports Analytics
45. Comparativa de herramientas Business Intelligence
46. Use Case Big Data "Dashboards with Hadoop and Power BI"
47. Todas las presentaciones del Workshop 'El Business Intelligence del Futuro'
48. Descarga Paper gratuito: Zero to beautiful (Data visualization)
49. SAP connection tools for process automation: Microsoft, Pentaho, Talend (User Guide)

stratebi
Analytics and Big Data Consulting