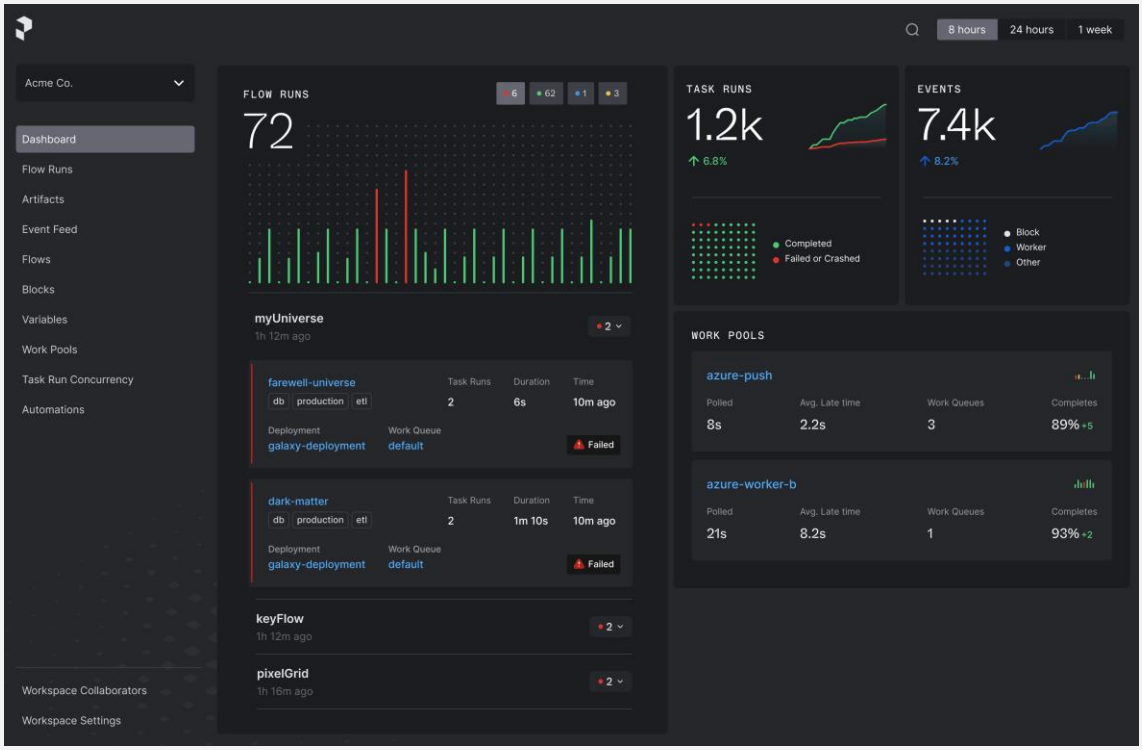
Prefect Data Orchestration Platform



CONTENIDO

1. INTRODUCCIÓN

A grandes rasgos, Prefect es una herramienta de orquestación que permite construir, observar y planificar la ejecución de workflows, que no son más que scripts de Python que constan de una serie de funciones denotadas con una sintaxis determinada.

En los siguientes capítulos trataremos de condensar y explicar los conocimientos básicos necesarios para el manejo y despliegue de Prefect.

Instalación

Se puede instalar fácilmente empleando ***pip:***

```
$ pip install -U prefect
```

Y comprobamos si se ha instalado correctamente:

```
$ prefect version
```

Referencias

Instalación

Flows y Tasks

Opciones en flows y tasks

Task Runners

Deployments

Bloques y bloques en detalle

Bloques de infraestructura en detalle

Bloques de almacenamiento en detalle
Componentes de Prefect y orden de despliegue

Prefect Database en detalle

Work pools y agentes en detalle

2. CONCEPTOS BÁSICOS

Flows y tasks

A grandes rasgos, los workflows de Prefect son scripts de Python que constan de flows y tasks, que son básicamente funciones denotadas con ***@flow*** o ***@task*** en la parte superior.

El flow es la base de los workflows, mientras que las tasks representan partes del trabajo ejecutado en un flow. La definición de tasks no es obligatoria, pero permite visualizar a un nivel más granular qué se está ejecutando en un flow determinado. Un flow puede llamar a otro flow(estos son llamados ***subflows***) o a una tarea, pero una tarea no puede llamar por sí misma a un flow u otra tarea. A continuación, podemos observar un ejemplo sencillo de un workflow:

```
import requests
from prefect import flow, task
@task
def call_api(url):
    response = requests.get(url)
    print(response.status_code)
    return response.json()

@task
def parse_fact(response):
    fact = response["fact"]
    print(fact)
    return fact

@flow
def api_flow(url):
    fact_json = call_api(url)
    fact_text = parse_fact(fact_json)
    return fact_text

api_flow("https://catfact.ninja/fact")
```

La ejecución de workflows sencillos como el anterior puede realizarse manualmente como cualquier otro programa de Python:

```
$ python workflow.py
```

Opciones en flows y tasks

Podemos configurar fácilmente los flows y tasks añadiendo argumentos a las etiquetas ***@flow*** y ***@task***. Como opciones básicas tenemos ***name***, ***description*** o ***version***, cuya

finalidad es simplemente añadir un identificador que perdure entre ejecuciones, documentar para qué sirve cierto flow/task o especificar la versión:

```
from prefect import flow

@flow(name="My Example Flow",
      description="An example flow for a tutorial.",
      version="tutorial_02")
def my_flow():
```

Una buena manera de aprovechar la opción versión es, en caso de utilizar git como controlador de versiones de nuestro código, introducir el hash del último commit. Por defecto Prefect intentará manejar los hashes por su cuenta, pero en ocasiones no puede, por lo que no es mala idea introducirlo explícitamente de la siguiente manera:

```
import os
from prefect import flow

@flow(name="My Example Flow",
      description="An example flow for a tutorial.",
      version=os.getenv("GIT_COMMIT_SHA"))
def my_flow():
    # run tasks and subflows
```

También podemos especificar un nombre que varíe según los parámetros de entrada a nuestro flow empleando la opción ***flow_run_name*** en los flows o, análogamente, ***task_run_name*** en las tasks:

```
@task(name="My Example Task",
      description="An example task for a tutorial.",
      task_run_name="hello-{name}-on-{date:%A}")
def my_task(name, date):
    pass


@flow(flow_run_name="{name}-on-{date:%A}")
def my_flow(name: str, date: datetime.datetime):
    Pass
```

En cuanto a opciones más específicas podemos encontrar los ***tags***, que son únicamente aplicables a las tasks. Esto permite filtrar fácilmente distintas tasks según su utilidad o tipo:

```
@task(name="My Example Task",
      description="An example task for a tutorial.",
      tags=["tutorial","tag-test"])
```

```python
def my_task():
    # do some work
```

Task runners

Los task runners se encargan de ejecutar las tareas siguiendo unas directrices determinadas. Estas no son obligatorias para ejecutar tareas, y sólo se emplearán si lo indicamos explícitamente. Si no indicamos el tipo de task runner a emplear, las tasks serán ejecutadas como funciones ordinarias y no utilizarán ningún task runner. Prefect tiene por defecto dos tipos de task runner:

- ***SequentialTaskRunner*** -> Ejecuta las tareas secuencialmente
- ***ConcurrentTaskRunner*** -> Ejecuta tareas concurrentemente

En adición a las ya nombradas, es posible emplear otros task runners externos que permiten la ejecución paralela o distribuida de las tasks, como ***DaskTaskRunner***.

Para indicar el task runner a utilizar se puede emplear la opción ***task_runner*** en la etiqueta ***@flow*** del flow en el que deseemos emplear el task runner. En adición, hemos de añadir una llamada a la función ***.submit()*** con los argumentos de la task a ejecutar:

```python
def stop_at_floor(floor):
    print(f"elevator moving to floor {floor}")
    time.sleep(floor)
    print(f"elevator stops on floor {floor}")

@flow(task_runner=ConcurrentTaskRunner())
def elevator():
    for floor in range(10, 0, -1):
        stop_at_floor.submit(floor)
```

Deployments

A diferencia de un script de Python con una serie de flows/tasks, un objeto deployment permite transformar un script que se ejecuta únicamente de manera manual y local en una entidad que puede ser manejada, configurada y ejecutada a través del API server de Prefect. En adición, la creación de un deployment permite la planificación de las ejecuciones de los distintos workflows. Para poder construir y ejecutar un deployment necesitamos un workflow con una función con la etiqueta ***@flow*** que sirva como punto de entrada y un Prefect Agent disponible para ejecutar el deployment. Los comandos básicos para el manejo de deployments son los siguientes:

- ***Creación de un deployment*** -> Este comando genera un archivo ***.yaml*** con la configuración del deployment y sube los archivos presentes en el directorio del workflow al almacenamiento remoto que especifiquemos. Si no lo hacemos, simplemente lo guarda de manera local. Si deseamos excluir ciertos archivos para que no sean subidos al almacenamiento remoto hemos de crear un archivo ***.prefectignore*** y añadir los nombres pertinentes.

```
$ prefect deployment build ./<workflow>.py:<flow_entrada> -n
<nombre_deployment> -p <nombre_pool>
#Opciones adicionales
-q <nombre_queue> -> Para un control más granular de las
peticiones de ejecución enviadas a una pool determinada
-o <nombre_yaml> -> Permite especificar un nombre personalizado
para el .yaml
-sb <tipo_almacenamiento>/<nombre_bloque> -> Para emplear
almacenamiento externo en vez del almacenamiento local
-ib <tipo_infra>/<nombre_infra> -> Para especificar en qué
entorno se va a ejecutar el deployment
--path <ruta> -> Empleado para definir, en el recurso de
almacenamiento externo, una carpeta/ruta concreta en la que
almacenar el deployment. Se emplea cuando el almacenamiento es
usado por diversos deployments
--override <opciones> -> Para sobrescribir valores base de un
bloque de infraestructura
```

- ***Modificación de un deployment*** -> Si realizamos modificaciones al `.yaml` generado en la fase anterior, hemos de ejecutar este comando para que los cambios se vean reflejados en la ejecución.

```
$ prefect deployment apply <nombre_yaml>.yaml
```

- ***Ejecución de un deployment*** -> Para ejecutar el deployment que hemos construido/modificado, hacemos un deployment run.

```
$ prefect deployment run '<flow_entrada>/<nombre_deployment>'
```

- ***Visualizar información detallada de un deployment***

```
$ prefect deployment inspect <flow_entrada>/<nombre_deployment>
```

Tras ejecutar el comando ***deployment run*** sólo queda poner en funcionamiento un Prefect agent que escuche en la work pool donde hayamos mandado la ejecución para que así pueda captarlo y ejecutarlo.

Blocks

Los bloques son objetos de Prefect que contienen la configuración necesaria para interactuar con sistemas externos, ya sea a nivel de almacenamiento o infraestructura. Su creación y manejo puede realizarse desde la terminal, código e interfaz web (la documentación oficial recomienda, debido a su sencillez, la interfaz web). Existe una serie de plantillas de bloques predefinidas, pero es posible crear nuevas a través de código. Para crear bloques desde la interfaz web hemos de dirigirnos a la URL que hayamos definido al levantar Prefect API Server y seleccionar el apartado "Blocks" presente en el menú de la izquierda.

Almacenamiento

Los bloques de almacenamiento son empleados para comunicarse con el almacenamiento externo en el que guardaremos el código y otros archivos de los deployments. Los principales proveedores de almacenamiento soportados son los siguientes:

- ***Amazon S3***
- ***Azure***
- ***GCS***
- ***Azure***
- ***Almacenamiento externo personalizado (MinIO, por ejemplo)***

Cabe señalar que para emplear almacenamiento externo en Prefect necesitaremos instalar paquetes adicionales de Python, que varían dependiendo del proveedor o tipo de almacenamiento. Por ejemplo, para Amazon S3 o MinIO:

```
$ pip install s3fs
```

Infraestructura

Los bloques de infraestructura son usados para especificar al Prefect Agent de qué manera ejecutar un determinado deployment. Los bloques de infraestructura más comunes son:

- ***Contenedor de Docker***
- ***Kubernetes Job***
- ***Proceso***

Compartición de bloques

Tanto los bloques de almacenamiento como los bloques de infraestructura pueden ser compartidos entre diversos deployments. En caso de los bloques de almacenamiento simplemente hemos de procurar definir una ruta para cada deployment, para que así no interfieran entre ellos. Esto lo podemos conseguir empleando la opción ***--path*** al ejecutar

un ***deployment build***. En cuanto a la infraestructura, esto ya se consigue por defecto ya que se levanta un contenedor nuevo (si es que deseamos emplear Docker para hacer los ***flow run***) por ejecución. Si deseamos modificar alguno de los valores base del bloque de infraestructura (por ejemplo, la imagen de Docker a utilizar) podemos emplear la opción ***--override*** al ejecutar el comando ***prefect deployment build***.

3. COMPONENTES DE PREFECT

Prefect no es un componente único, sino que está compuesto por diversas partes:

- ***Prefect API Server*** -> Básicamente es la cabeza de toda la estructura, se encarga de la orquestación de los despliegues y ejecuciones
- ***Prefect Database*** -> Guarda metadatos de los flows y tareas en ejecución o que ya se han ejecutado
- ***Prefect UI*** -> Nos permite visualizar, monitorizar, configurar y lanzar ejecuciones a través de una interfaz web sumamente intuitiva
- ***Storage*** -> Permite almacenar el código de los workflows y los resultados provenientes de su ejecución
- ***Agents y Work Pools*** -> Actúa como enlace entre el API server y el entorno de ejecución, los agentes detectan la aparición de workflows a ejecutar en las queues de las work pools y proceden a preparar la ejecución.

Los componentes recién nombrados pueden ser ejecutados en la misma máquina o de manera distribuida (lo ideal).

Prefect API Server

Iniciar este componente es sumamente sencillo:

```
$ prefect server start
```

Por defecto el servidor es levantado en http://localhost:4200, pero si deseamos cambiar estos valores por defecto podemos fijar dichas variables al valor que deseemos empleando:

```
#Para indicar la IP
$ prefect config set PREFECT_SERVER_API_HOST=<customip>
#Para indicar el puerto
$ prefect config set PREFECT_SERVER_API_PORT=<customport>
```

El API server no dispone de un comando ***stop***, por lo que si es lanzado en primer plano se detendrá mediante ***CTRL+c*** o, en caso de lanzarse en segundo plano, empleando el comando ***kill*** para enviar una señal ***SIGINT*** al proceso:

```
$ kill -SIGINT prefectPID
```

Para comunicarte correctamente con el API server para realizar cualquier tarea conviene setear la variable ***PREFECT_API_URL (***si no la seteamos, por defecto la fija a http://127.0.0.1:4200/api), independientemente de si nos encontramos en una máquina externa o en la que se está ejecutando el API server:

```
#Para el servidor empleamos localhost o la ip de la interfaz en
la que estemos ejecutando el server
```

```
$ prefect config set PREFECT_API_URL=http://127.0.0.1:4200/api
#Para una máquina externa
$ prefect config set PREFECT_API_URL=http://<serverIP>:4200/api
```

Prefect Database

Prefect soporta únicamente ***SQLite*** y ***PostgreSQL***. Si no especificamos el tipo, Prefect genera por defecto una instancia en local de SQLite en la ruta ***~/.prefect/prefect.db***. Si se desea emplear Postgre hay que realizar algunos pasos adicionales para su puesta en marcha. Para indicar a Prefect Server una ruta o instancia personalizada, hemos de setear la variable ***PREFECT_API_DATABASE_CONNECTION_URL***:

- ***SQLite***

```
$ prefect config set
PREFECT_API_DATABASE_CONNECTION_URL="sqlite+aiosqlite:////full/p
ath/to/a/location/prefect.db"
```

- ***PostgreSQL***

```
$ prefect config set
PREFECT_API_DATABASE_CONNECTION_URL="postgresql+asyncpg://postgr
es:yourTopSecretPassword@localhost:5432/prefect"
```

Si deseamos resetear la base de datos (borrar los datos y crear nuevamente el esquema) podemos ejecutar lo siguiente:

```
$ prefect server database reset
```

Prefect UI

Como ya he comentado, permite realizar diversas tareas de manera sencilla e intuitiva, así como monitorizar workflows en ejecución, visualizar historial de ejecuciones, comprobar gráficamente el flujo de ejecución de un workflow concreto...

La ruta por defecto para acceder a la interfaz web es http://127.0.0.1:4200 si hemos cambiado el host o el puerto será http://<customip>:<customport>.

Storage

Si no especificamos una unidad de almacenamiento concreta Prefect empleará almacenamiento local de manera temporal, pero esto no nos conviene si deseamos tener Prefect en producción. Si deseamos un almacenamiento presistente podemos hacer uso de los blocks, que no son más que plantillas de declaración de unidades de almacenamiento, ya sea una instancia propia de, por ejemplo, MinIO, o almacenamiento remoto como Amazon S3. La definición de los blocks puede realizarse a través de la interfaz web o empleando código:

```
##Declaración de un block de Amazon S3 o MinIO
from prefect.filesystems import S3

block = S3(bucket_path="my-bucket/a-sub-directory",
           aws_access_key_id="foo",
           aws_secret_access_key="bar"
)
block.save("example-block")
```

Work Pools

Contienen la lógica para el manejo de las queues y las peticiones flow run, y sirven dicha información a los agentes. La creación de work pools, al igual que el resto de elementos, puede realizarse a través del CLI o la web, pero en este caso comentaremos la creación mediante CLI, pues es más rápida:

```
$ prefect work-pool create <poolname>
```

Para listar las work pools disponibles:

```
$ prefect work-pool ls
```

Para mostrar información detallada de una work pool concreta:

```
$ prefect work-pool inspect 'poolname'
```

Agents

Los agentes son los encargados de recoger y analizar la información proveniente de las queues de los workpools y actuar en consecuencia (ejecutando los workflows que corresponda). Para instanciar un agente, ejecutamos:

```
$ prefect agent start -p "my-pool"
```

Como podemos observar, simplemente hemos de indicarle la pool de la que se desea recibir datos. Hay que tener en cuenta que el agente se quedará escuchando indefinidamente en la work pool indicada, por lo que si deseamos detenerlo hemos de emplear, al igual que en Prefect API, ***CTRL+c*** o ***kill***.

4. ORDEN DE DESPLIEGUE

Como hemos podido observar en el apartado anterior, Prefect está compuesto por diversos elementos, y algunos de ellos dependen de que cierto componente ya esté en funcionamiento, por lo que hay que seguir cierto orden a la hora de levantar los componentes. El orden recomendado es el siguiente:

1. ***Base de datos de Prefect***
2. ***Servidor de almacenamiento***
3. ***Prefect Server***
4. ***Work pool***
5. ***Agent***

Una vez los hemos puesto en funcionamiento, para que un cliente pueda ejecutar comandos sobre el servidor sólo le quedaría fijar en su sistema la variable ***PREFECT_API_URL*** con los datos correspondientes al API Server:

```
$ prefect config set
PREFECT_API_URL=http://<serverIP>:<serverPort>/api
```

5. AIRFLOW VS PREFECT

Si lo comparamos con otras herramientas del sector como ***Airflow***, https://todobi.com/que-es-apache-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:

- Airflow se rige rigurosamente por una política de workflows como código, mientras que Prefect es algo más flexible en este aspecto y permite (incluso incentiva) el uso de la interfaz web para la creación y manejo de los distintos objetos presentes en la herramienta.
- Airflow posee una interfaz web mucho más detallada que la de Prefect que, a pesar de su facilidad de uso, se puede quedar algo corta en los que se refiere a opciones.
- Airflow fundamenta los workflows en la especificación de DAGs a través de scripts de Python, empleando para ello una sintaxis mucho más específica que los workflows de Prefect, que son muy similares a un script con la sintaxis tradicional de Python. En Prefect, la funcionalidad presente en los DAGs viene dada por los deployments, cuya configuración se efectúa a través de un ***.yaml***(revisar capítulos anteriores).
- La arquitectura y modo de trabajo son distintos en ambos casos. Por ejemplo, en Airflow el scheduler es un módulo aparte mientras que en Prefect viene integrado en el API Server.
- La curva de aprendizaje y complejidad general es mayor en Airflow.
- Airflow posee una mejor y mayor integración con servicios externos.
- Prefect tiene soporte (en beta) para lenguajes adicionales, como Julia o R.

Si expresamos estas diferencias de manera resumida:

Airflow	*Prefect*
Workflows como código	*Mayor uso de interfaz web*
Interfaz web compleja y confusa	*Interfaz web minimalista y sencilla*
Mayor integración con servicios externos	*Menor integración con servicios externos*
Mayor curva de aprendizaje	*Menor curva de aprendizaje*
Soporte para Python	*Soporte para Python, Julia y R*
Sintaxis propia algo más concreta	*Sintaxis similar a Python tradicional*
Configuración a través de DAGs	*Configuración a través de deployments*

6. CONCLUSIONES

Como hemos podido observar a lo largo de esta guía, Prefect es una herramienta con una sintaxis bastante sencilla y una interfaz web muy intuitiva.

Si sumamos esto a su facilidad de despliegue, nos deja una herramienta con una curva de aprendizaje relativamente baja que, en adición, es muy competente en su campo. Sin embargo, y en base a lo observado en el apartado anterior, Prefect también tiene sus desventajas, por lo que deberemos ponderar, dependiendo del proyecto a desarrollar, qué herramienta nos conviene utilizar.

Por ejemplo, si nuestro proyecto exige la integración de la herramienta con un servicio muy concreto es probable que nos convenga más emplear Airflow, pues, como hemos comentado, tiene una mayor integración con servicios externos.