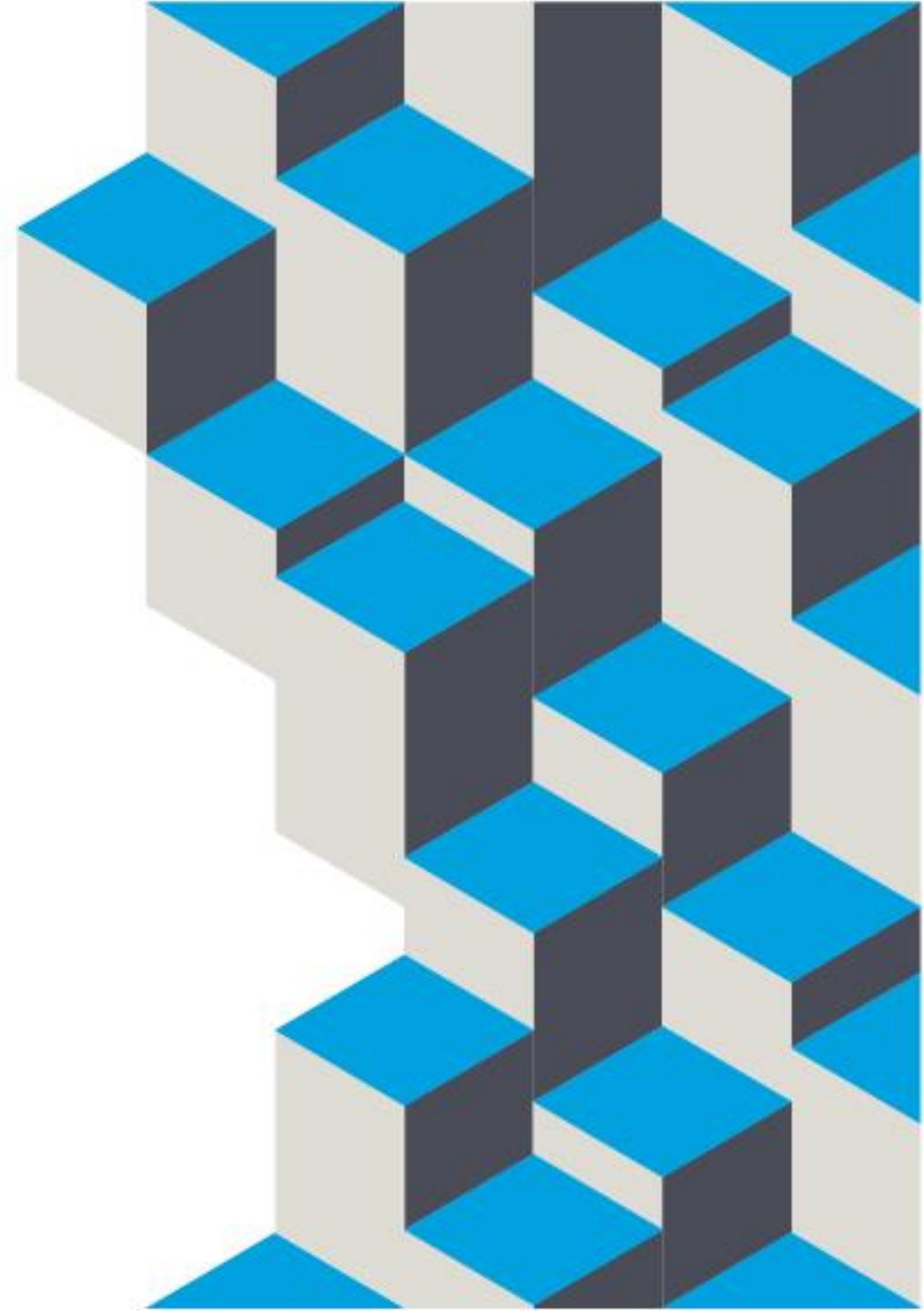


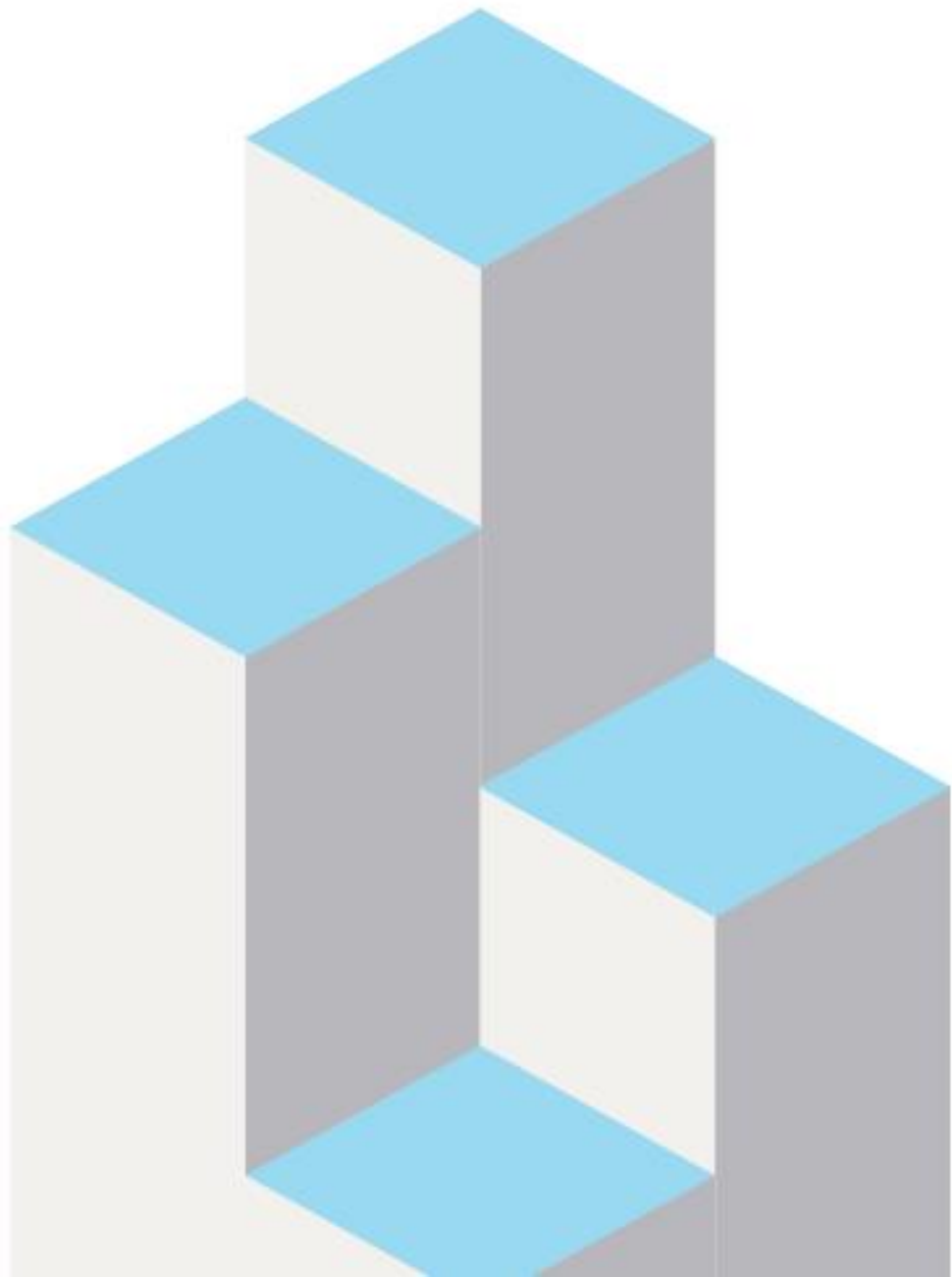
Tus datos en el cloud: Domando tus datos con ETLs en Python

Juan Gabriel Gomila Salas


Albert Gris Cabré

Frogames





Introducción



¿Qué vas a aprender?

- Manejo de datos a escala en el cloud
- Enfrentarte a orígenes de datos reales
- Data Wrangling y ETLs con Python
- Ingeniería de variables
- Poner y marcha y operar con recursos cloud
- Trabajar con Python en VS Code (IDE)



Metodología y desarrollo del curso

- Método del caso:
 1. Introducción: contexto + tecnología + negocio
 2. Exposición del caso: cuál es el reto
 3. Desarrollo del contenido del curso
 4. Resolución del caso
- Enfoque a obtener método, *skillset* y recursos más que aprender una herramienta u otra.
- Problemas y soluciones reales.



whoami



El contexto

- La necesidad de gestionar grandes volúmenes de datos: los datos como activo y su explotación como generador de oportunidades.
- Mover los datos **que** nos interesan, **cuándo** nos interesan, **dónde** nos interesan y **de la forma que precisamos** es signo de control y gobierno de los datos.



Big Data y transformación digital

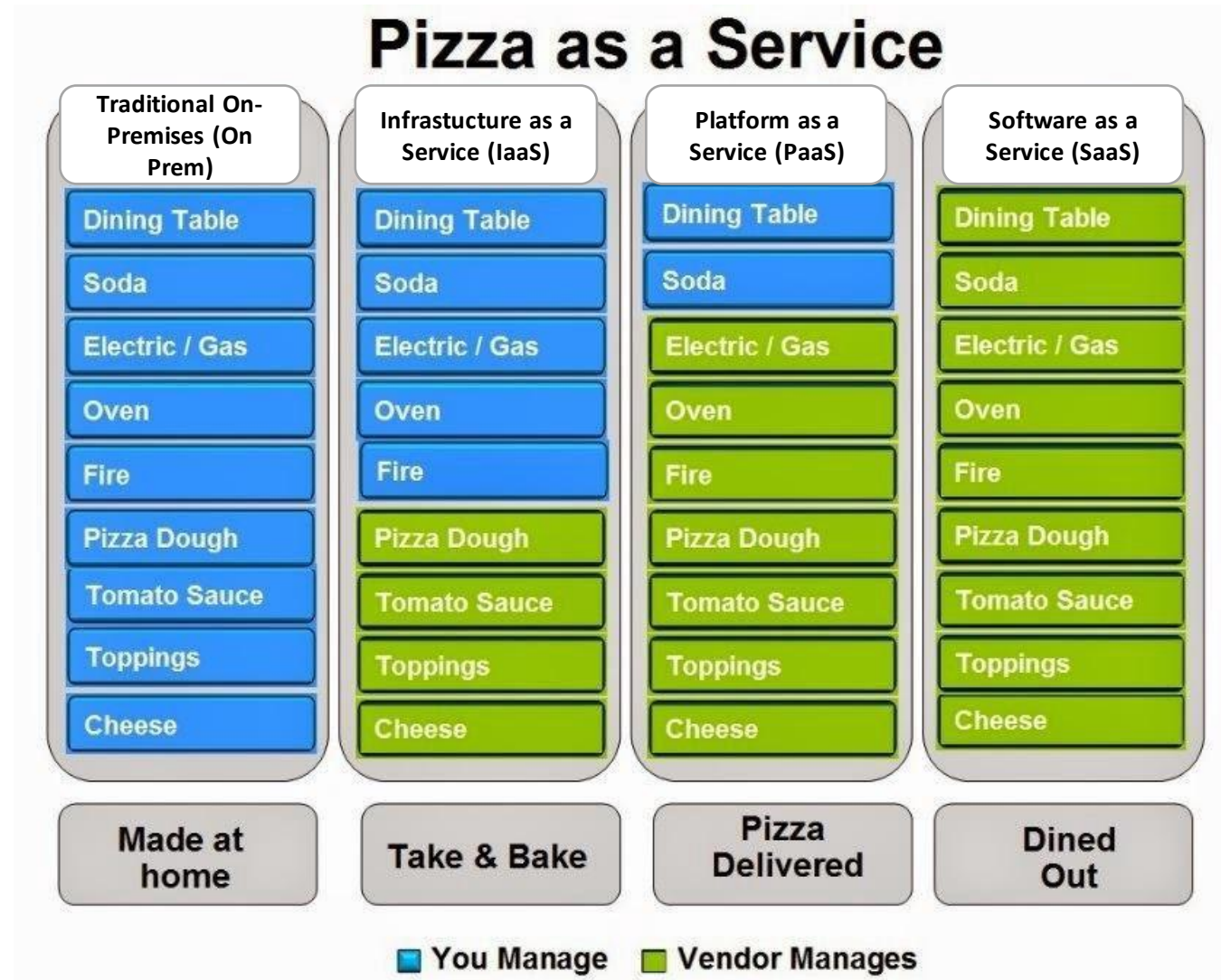
- 3 Vs del nuevo paradigma del análisis de datos:
 - Volumen: 1.1 trillones de MB al día.
 - Variedad: nuevos servicios, Open Data, ...
 - Velocidad: 5G, SSD...
- Internet y la digitalización nos llevan a un nuevo paradigma de negocio: empresas *data-driven*.

Cloud

- ¿Qué hace de un servicio Cloud Computing?
 - Multitenancy
 - Modelo de negocio: ingresos por suscripción (CAPEX v OPEX)
 - On-demand
 - Escalabilidad “infinita”
 - ¿Dónde está?



Cloud: tipos de consumo



Cloud: el mercado

- Hiper-escalares
 - AWS, MS Azure, Google Cloud Platform, AlibabaCloud, IBM...
- CSP: Cloud Service Providers

Figure 1. Magic Quadrant for Cloud Infrastructure and Platform Services



Sistemas de gestión empresariales: perspectiva de negocio

- Procesos de negocio = procesos operativos + procesos estratégicos
- Procesos operativos (venta, almacén, entregas, servicio cliente, ...)



- Procesos estratégicos (marketing, logística, HR, dirección, ...)



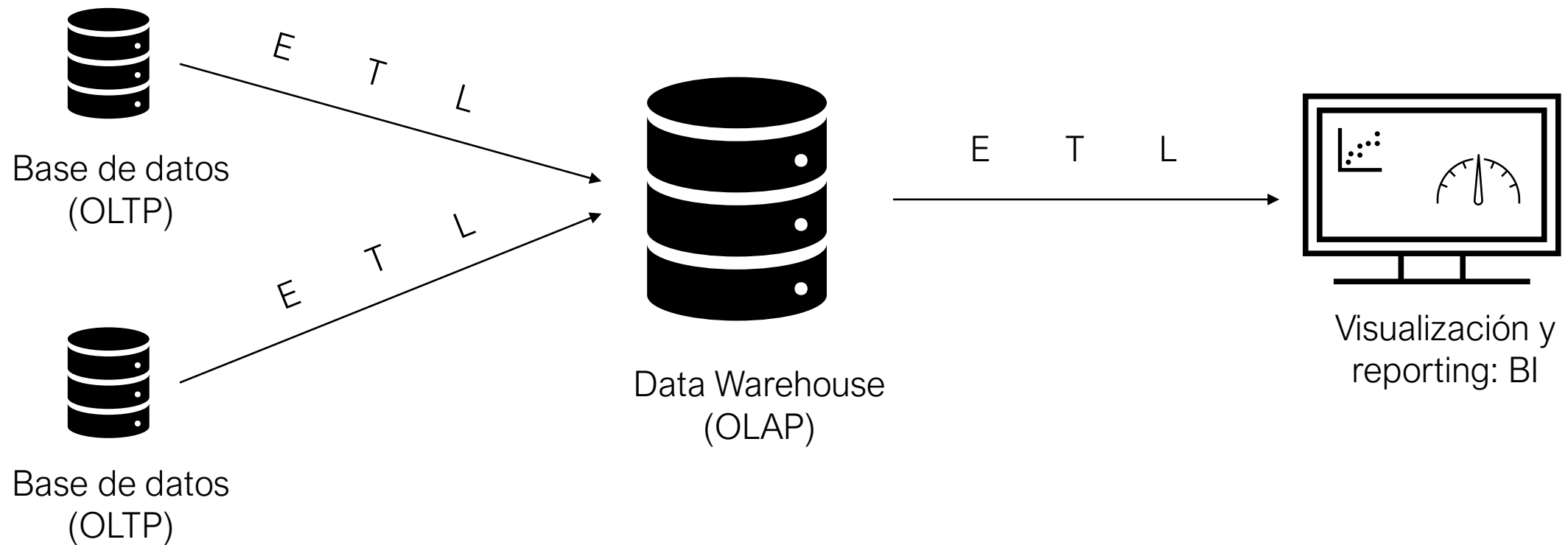


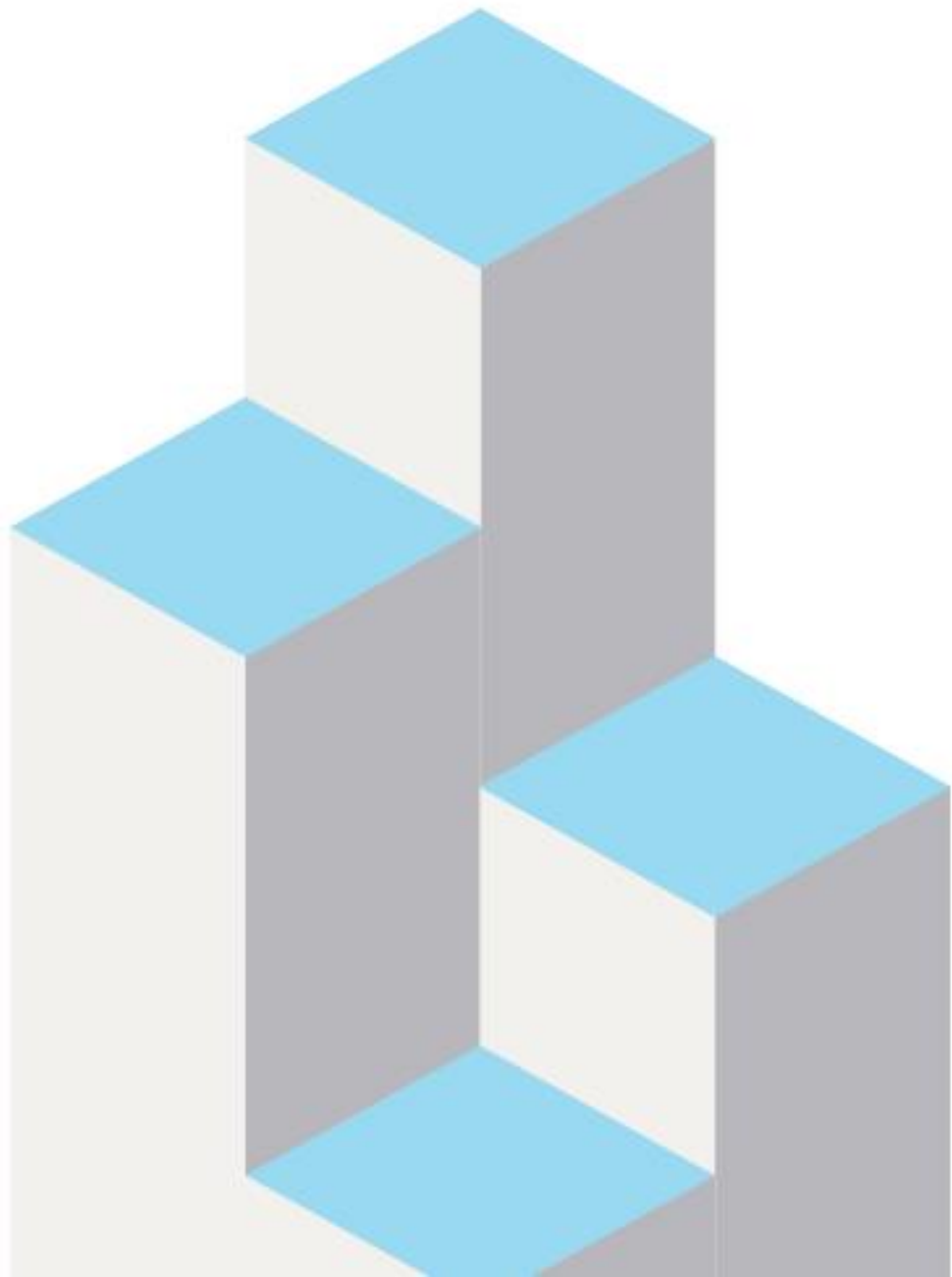
Sistemas de gestión empresariales: perspectiva tecnológica

- Operacional (OLTP)
 - Orientados a transacción.
 - Objetivo negocio: eficiencia operativa.
 - Objetivo tecnológico: escritura.
 - Tecnología: SQL Relacional (3NF)
- Estratégico/Informacional (OLAP)
 - Orientados a análisis.
 - Objetivo de negocio: soporte estratégico.
 - Objetivo tecnológico: lectura + histórico.
 - Tecnología: Relacional (1/2NF), NoSQL
- Naturaleza distinta/objetivos distintos/tecnología cada vez más distinta

Arquitectura BI tradicional

- Batch





Introducción: ETLs



Introducción

- Procesos de obtención (extract), transformación (transform) y carga (load) de los datos.
- Ampliación:
 - Obtener los datos que interese.
 - Transformarlos al estilo, formato y contenido que conviene.
 - Cargarlos en los sistemas de forma óptima.
- Proceso más laborioso de un proyecto de BI/BA



ETL vs ELT

- ETL para entornos data-warehouse
- ELT para entornos data-lake
 - Objetivo es disponer y capturar el dato lo más rápido posible
 - Dato no estructurado
 - *ETLT*



Diseño de la ETL

- Enfoque *top-down*: Dibujar y definir el producto final que deseábamos.
- A partir de ahí, buscamos en el “mundo” interno y externo de qué fuentes de datos podemos abastecernos.
- Comparamos el producto final deseado con los datos que podemos obtener: la transformación.
- Revisamos y adaptamos que el producto pueda almacenarse en el sistema donde dejaremos los datos.



Extract

- Identificación y selección de fuentes disponibles
- Proceso sensible para los sistemas OLTP (recordemos su finalidad...) – óptimo monitorización de infraestructura.
- Estrategia: partición y ventana.
- Pasos:
 - Identificación de las fuentes de datos: ubicación, acceso, modelo y tipos de datos.
 - Definir ventanas y particiones.
 - Definir conexiones para obtener datos.



Transform

- Etapa con mayor valor añadido.
- Incluye:
 - Tratamiento de la calidad de los datos (QA).
 - Integración/unificación estructura (datos y criterios).
 - Reglas de negocio/ingeniería de variables



Transform

- Pasos:
 - QA: nulos, tipos de datos, outliers
 - Validación de reglas específicas (valores incoherentes)
 - Unificación de fuentes
 - Ingeniería de variables

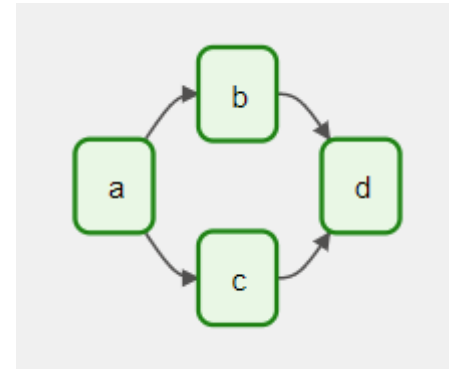


Load

- Cargar los datos de forma eficiente en Sistemas finalistas.
- Estrategia: Insert? Update? Ambas?
- Pasos:
 - Definir conexión al Sistema.
 - Creación del contenedor (base de datos, tablas, ...)
 - Inserción o actualización.

Pipelines, flows y ETLs

- Data pipeline
- ETL
- Flow/DAG/...





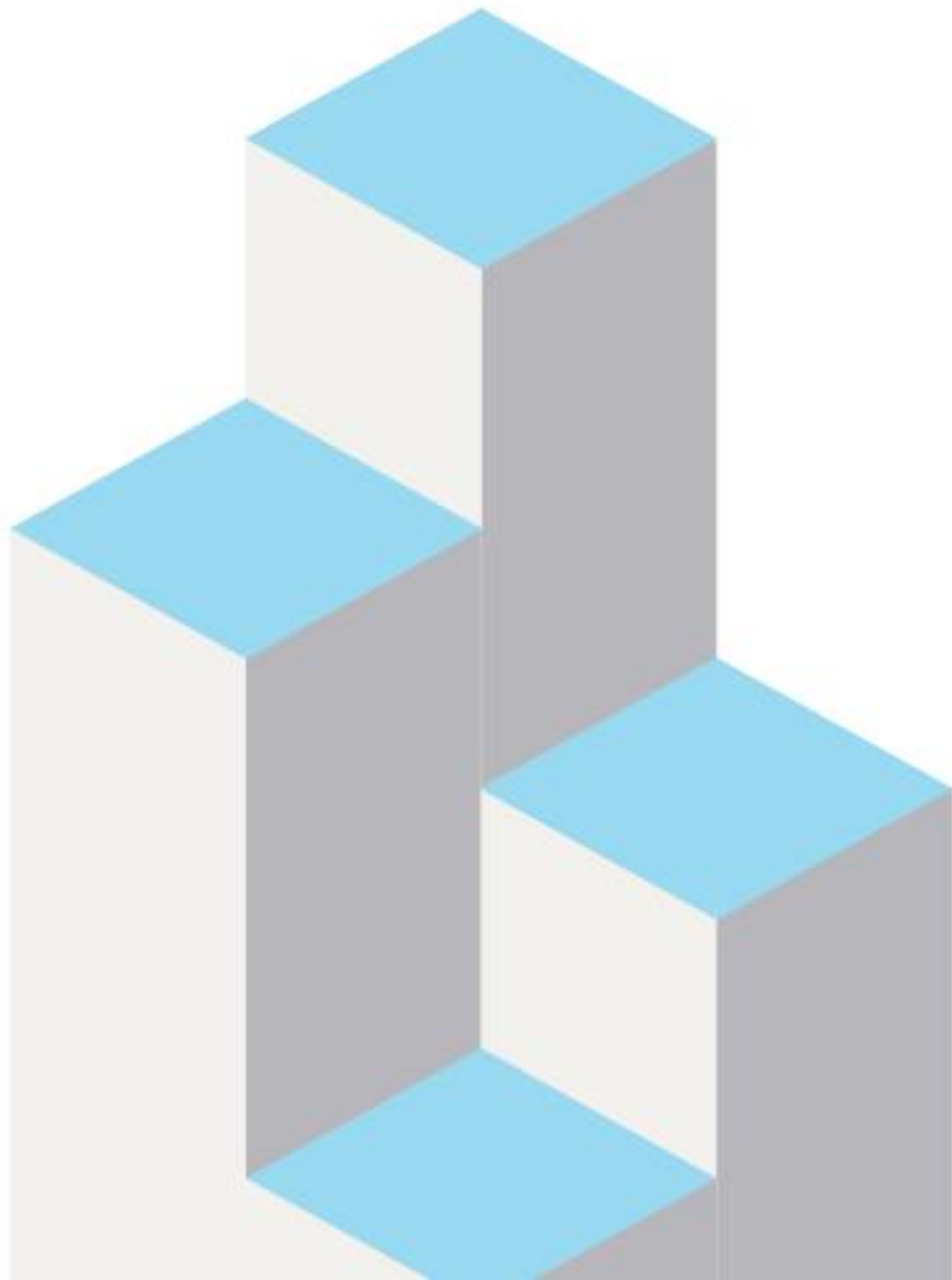
Orquestación y automatización de pipelines

- Software que permite el gobierno de los pipelines.
- Funcionalidades:
 - Gestión centralizada.
 - Monitorización centralizada.
 - Administración de las programaciones.



Casos de uso y mercado

- Herramientas *OpenSource*:
 - Apache Airflow
 - Prefect Core
 - MLflow
- Herramientas comerciales:
 - Pentaho
- Módulos de soluciones BI
- Cloud: Azure data factory



El caso



El objetivo del proyecto

- *Is Bitcoin Price Correlated to the Stock Market?*
<https://decrypt.co/63468/is-bitcoin-price-correlated-to-stock-market>
- → Obtener una tabla de datos que sirva para el análisis.
- *Como debería ser el set de datos?*

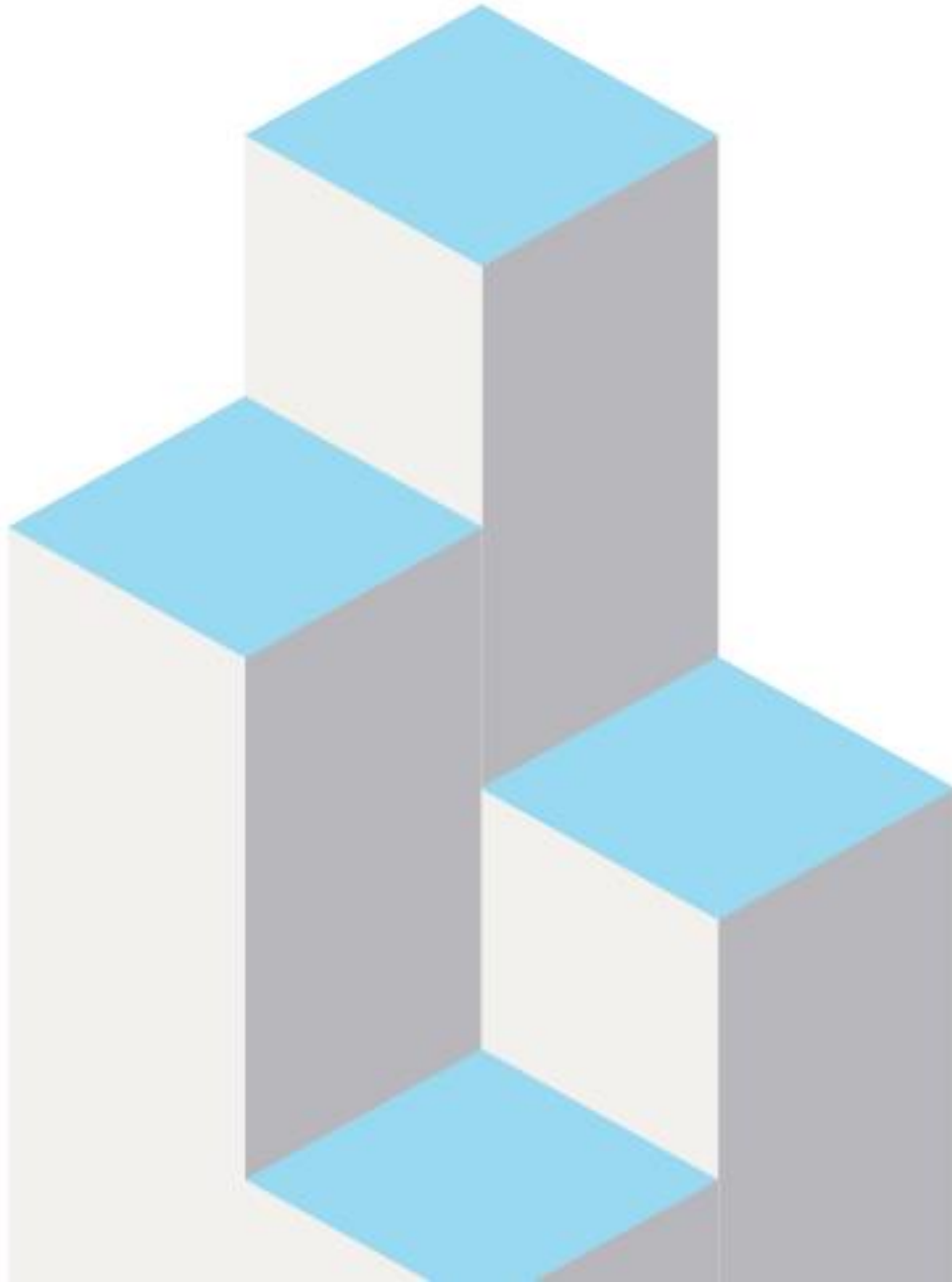


- *ARTICULO*



Los retos

- ¿De dónde obtengo los datos?
- ¿Con qué frecuencia obtengo los datos? ¿Qué partición consulto?
- ¿Dónde puedo almacenar de forma consistente y escalable los datos que obtengo? ¿Uso Cloud? ¿Qué tipo de servicio “XaaS”?
- ¿Cómo automatizo el proceso?



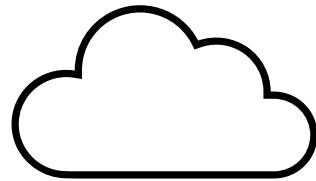
Introducción a Prefect



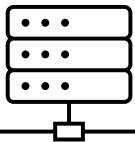
Introducción a Prefect

- Solución completa para el uso de pipelines: “motor”, gestión, automatización y orquestación.
- Modelo híbrido con arquitectura distribuida: *develop local, implement global*
- Está basada en Python.
- Herramienta comercial con hasta 10.000 ejecuciones mensuales gratuitas.
- Soporte y comunidad muy activa.

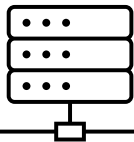
Introducción a prefect: Arquitectura



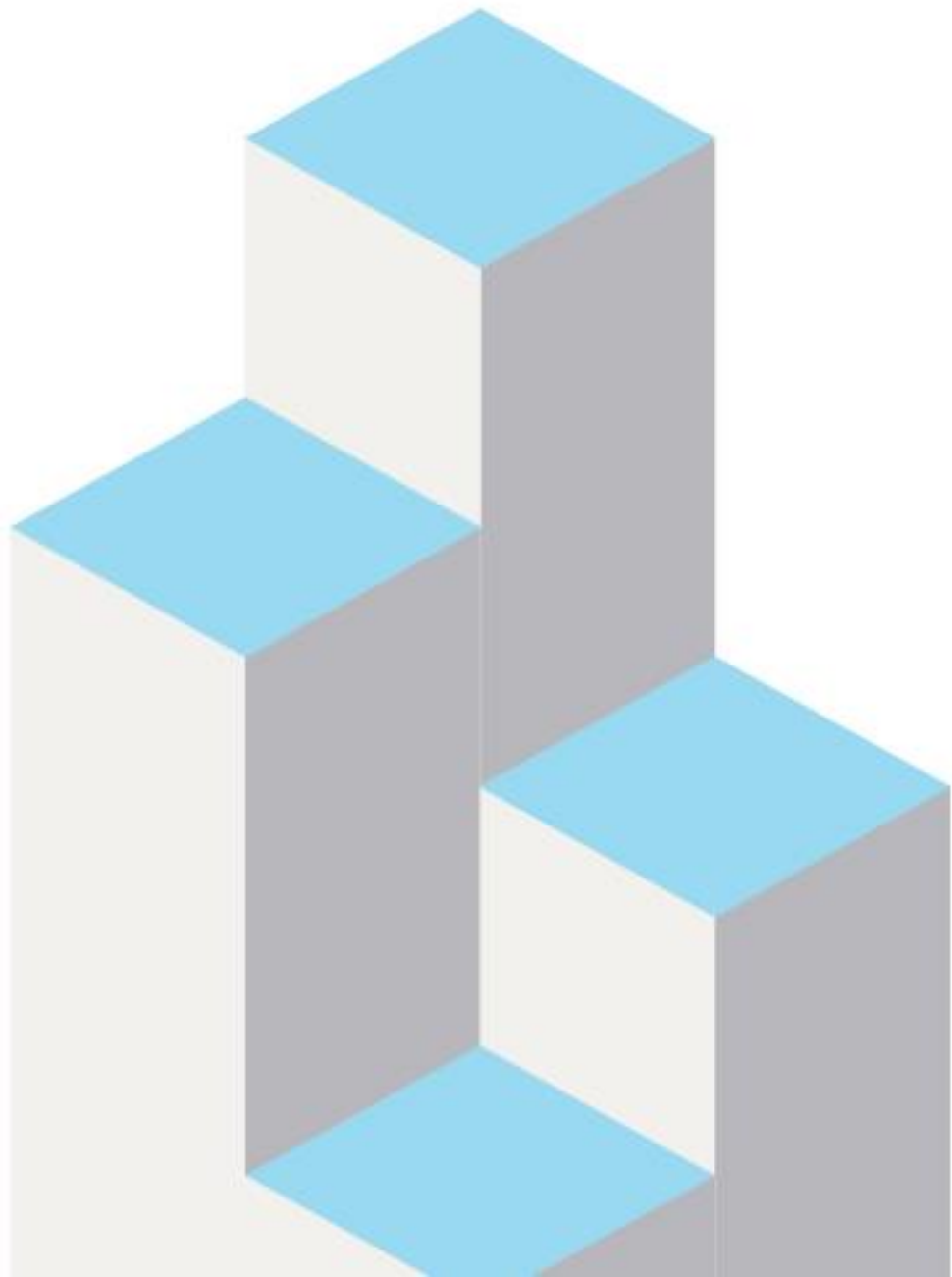
Prefect Cloud



Agente I
(Prefect Core)



Agente I
(Prefect Core)



Prefect Core I



Introducción a Prefect Core

- Paquete de Python Open Source– integrable en scripts que ya tengan otros paquetes.
- Modo framework (propósito general vs propósito específico).
- Se apoya en “Python en paradigma OOP”.
- Crea *objetos* que permiten a Python:
 - Dividir los pipelines en tareas granulares dependientes.
 - Agrupar líneas de código.
- Funcionalidades específicas ya implementados: secretos, conexiones a fuentes y destinos de datos, *scheduling*, ...



Refresco de Python: Python en modo OOP

- Python es un lenguaje multiparadigma. Soporta Orientación a objetos.
- **Clase:** prototipo/plantilla definida por el desarrollador que agrupa datos y funcionalidades. *No contiene datos, solo definiciones.*
 - *Atributos de la clase*
 - *Atributos de la instancia*
 - *Métodos*

} Actualizables



Refresco de Python: Python en modo OOP

```
class coche:
```

```
    num_ruedas = 4 #atributo de la clase - común para todos los objetos
```

```
    def __init__(self, matricula, marca, modelo): #función para instanciar
```

```
        self.matricula = matricula #atributo de la instancia
```

```
        self.marca = marca #atributo de la instancia
```

```
        self.modelo = modelo #atributo de la instancia
```

```
    def identificate(self): #método
```

```
        print("El vehículo con matricula {} es un {}  
        {}".format(self.matricula, self.marca, self.modelo))
```



Refresco de Python: Python en modo OOP

- **Objeto:** cada una de las instancias de una clase.
- **Instanciar:** generar un objeto que hereda el prototipo de estructuras de datos y funcionalidades de la clase.



Refresco de Python: Python en modo OOP

```
coche_1 = coche("9999ABC", "Volkswagen", "Polo")
```

```
coche_1.identificate()
```

```
El vehículo 9999ABC es un Volkswagen Polo
```



Refresco de Python: Pandas & NumPy

- **Numpy:** tratamiento de arrays n-dimensionales.
 - Implementa funciones para tratar arrays de multiples dimensiones.
 - Especialmente útil para trabajar formato matricial.
 - Rendimiento realmente Bueno.
- **Pandas:** tratamiento de DataFrames y Series.
 - Series: ampliación de las listas.
 - DataFrame: permite tratar variables de forma tabular.



Flows y tasks

- Las unidades básicas de un script de Prefect Core.
- Un flow se refiere a un pipeline y una tarea a un conjunto de acciones con un contexto en común dentro de un flow.
- Los flows agregan las tareas explicitando las dependencias que existen entre ellas.
- Las tareas son clases ya definidas que implementan los flows cuando se ejecutan.



Decorando una función con clase

- Desde la perspectiva de Python, una tarea de Prefect es una función decorada con la clase importada *task*.
- En Python, los *decorators* sirven para extender la funcionalidad de una función o una clase. Por ejemplo, anidando una función a una función ya creada.

```
@task
def funcion:
    print("Hello World")
```



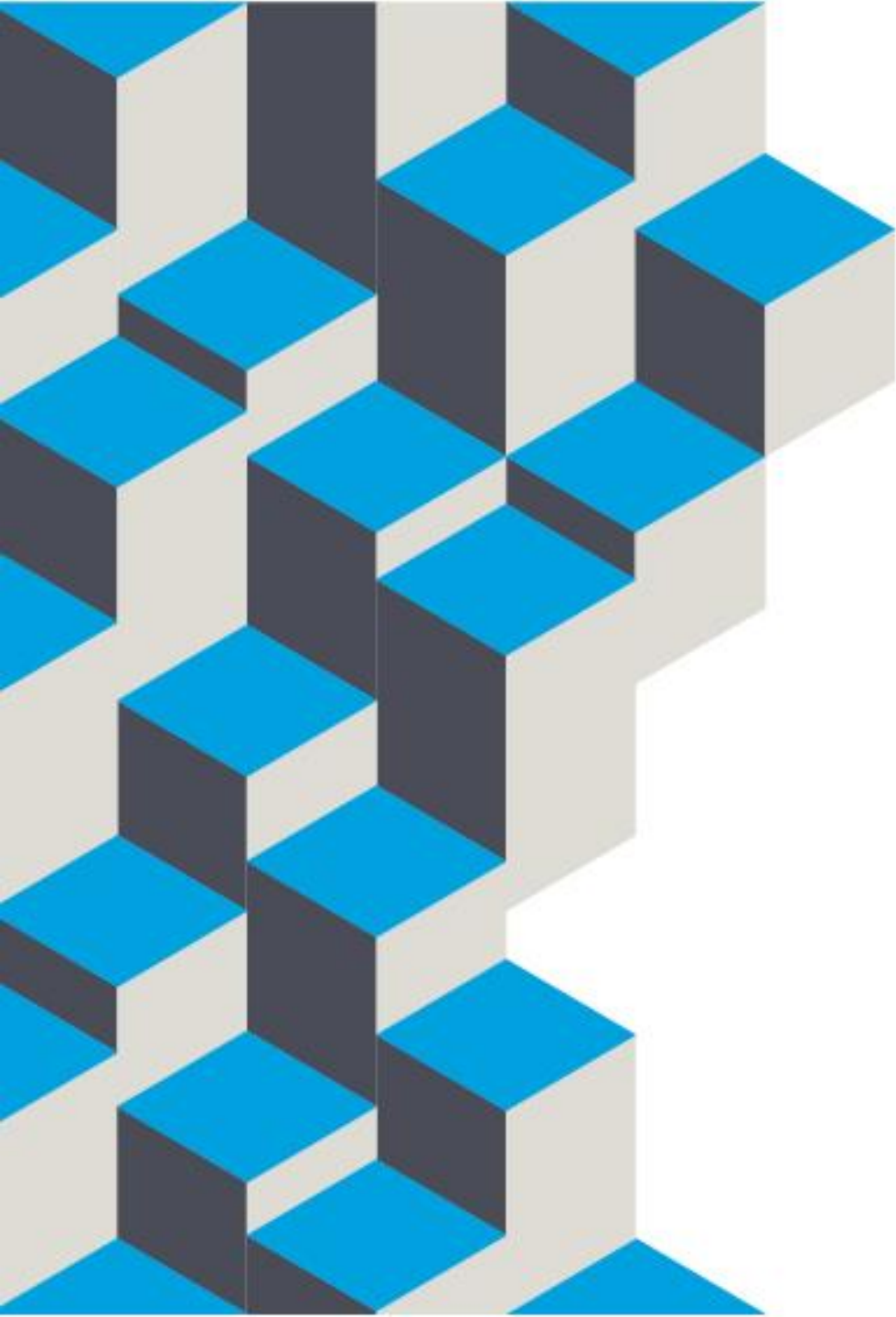

Flows y tasks: scripting

- La secuencia más común para los scripts de Prefect suele ser:
- 1- iniciar el entorno
- 2- definir funciones
- 3- definir las *tasks*
- 4- implementar las *tasks* en un Flow.

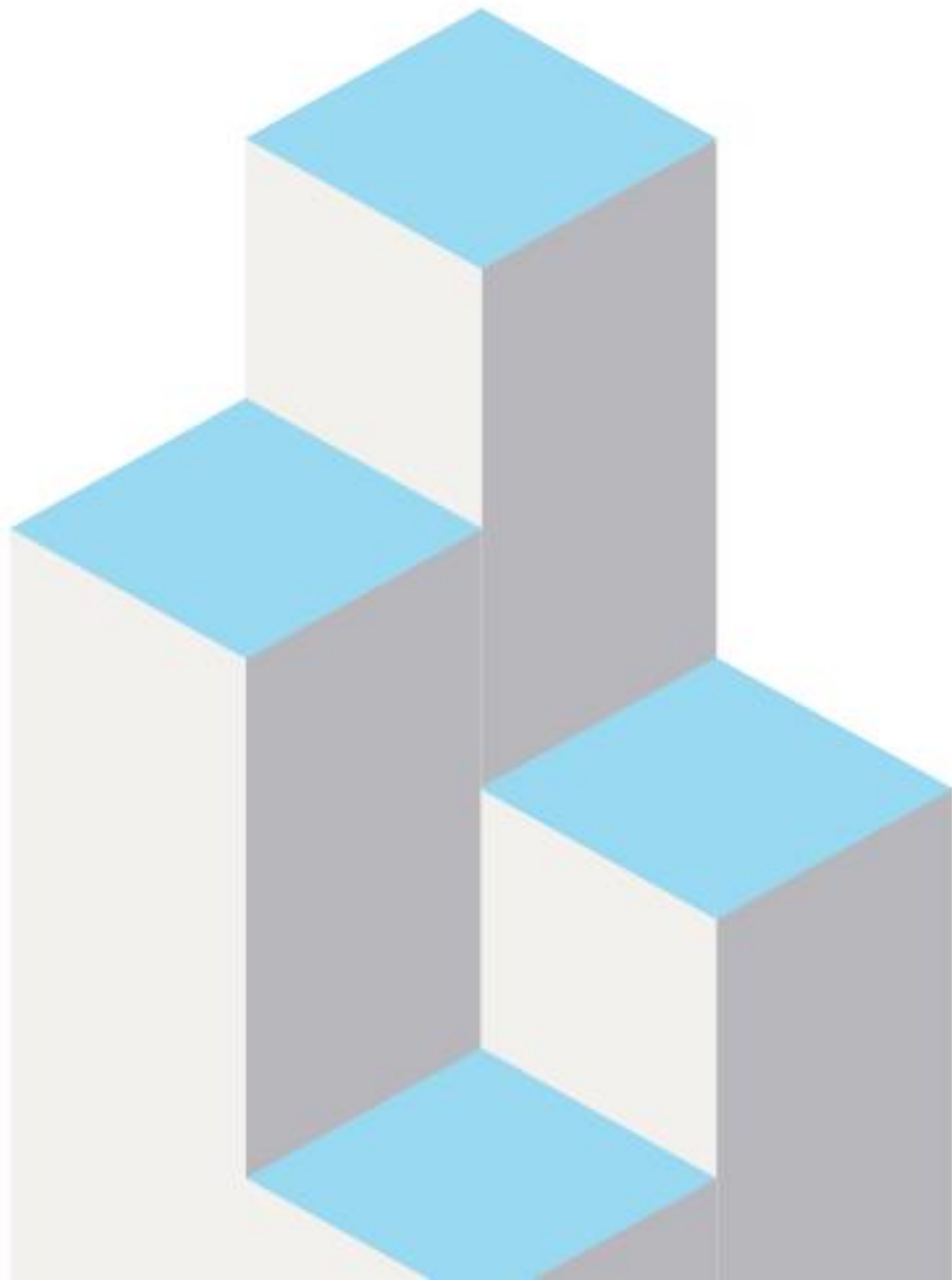


Práctica: Hello World

- Instalar nuestro entorno de Desarrollo:
 - Instalar Python
 - Instalar virtual-env
 - Instalar paquetes
 - Instalar VSCode
- Implementar nuestro primer flow con Prefect Core



Práctica: Hello World



Prefect Core II



Scheduling del flow

- Aspectos a tener en cuenta:
 - 1- Empezar por lo que deseáramos: ¿Qué frecuencia de actualización sería óptima para nuestro análisis?
 - 2- Entender el dataset de origen: cada cuanto se actualizan los datos? Hay nuevos registros?
- Primero desarrollar y testear el Flow → luego programarlo.
- Prefect presenta una amplia flexibilidad con el scheduling, permitiendo adaptarse a prácticamente cualquier necesidad.



Scheduling simple

- El Schedule (sea del tipo que sea), se implementa con la palabra clave `Schedule` en la definición del Flow.

```
from prefect import task, Flow
from datetime import timedelta
from prefect.schedules import IntervalSchedule

@task
def tarea():
    print("Hello world")

schedule = IntervalSchedule(interval=timedelta(minutes=5))

with Flow("Mi flow", schedule) as flow:
    tarea()
```



Scheduling complejo

- Incluye en la programación los siguientes conceptos
 - **clock**: toma el rol de “generar” los disparadores del flow desde una perspectiva temporal. Ejemplos: **IntervalClock** o **CronClock**.
 - **filters**: filtra eventos del *clock* en los que no ejecutar el flow. Por ejemplo, excluir fines de semana.
 - **adjustments**: recogen sub-eventos específicos para los que la condición del filtro es demasiado genérica.



Scheduling complejo

Un Schedule que se ejecute cada 24 h durante los días que no son fin de semana.

```
schedule = Schedule(  
    clocks=[IntervalClock(interval= timedelta(hours=24),  
        start_date=pendulum.datetime(2019, 1, 1,  
            tz="America/New_York"))],  
    filters=[filters.is_weekday]  
)
```




Particionado de datos

- Puedo consultar solo una porción de los datos que me interese, evitando procesarlo en la fase de transformación de la ETL?
- El dispositivo que ejecute el flow (*agent*) no almacena los datasets que gestionemos, pero está limitado a “manejar” datos del tamaño de la memoria que disponga el equipo durante la ejecución.
- Normalmente las APIs permiten filtrar solo un determinado conjunto de datos que cumpla la condición que deseamos.



Particionado de datos

- Ejemplo 1 – APIs REST:
<https://jsonplaceholder.typicode.com/guide/>
- Ejemplo 2 - SQL: cláusulas SELECT, WHERE y HAVING
`SELECT * FROM Ventas WHERE FechaVenta='2011-11-11'`



Calidad de datos

- Nos referimos a calidad de datos al atributo de los mismos por el cual podemos afirmar que se adhieren al uso que pretendemos darles.
- Es importante que nuestra ETL implemente un control de calidad para asegurar que se cumplan ciertos estándares.



Calidad de datos

- Inspección visual / inspección programàtica
- Se implementa como Python al uso, con funciones/funcionalidades como:
 - Forzar un tipo de dato: `str()`, `int()`, ...
 - Cambiar un valor por otro: `df.replace({'suscrito': 1, 'no_suscrito': 0})`
 - Expresiones regulares
 - Nulos:
 - `df.isna().sum()`
 - `df.fillna(<valor>)`
 - Outliers



Manejo de errores

- Logging
- Try-catch
- Retries



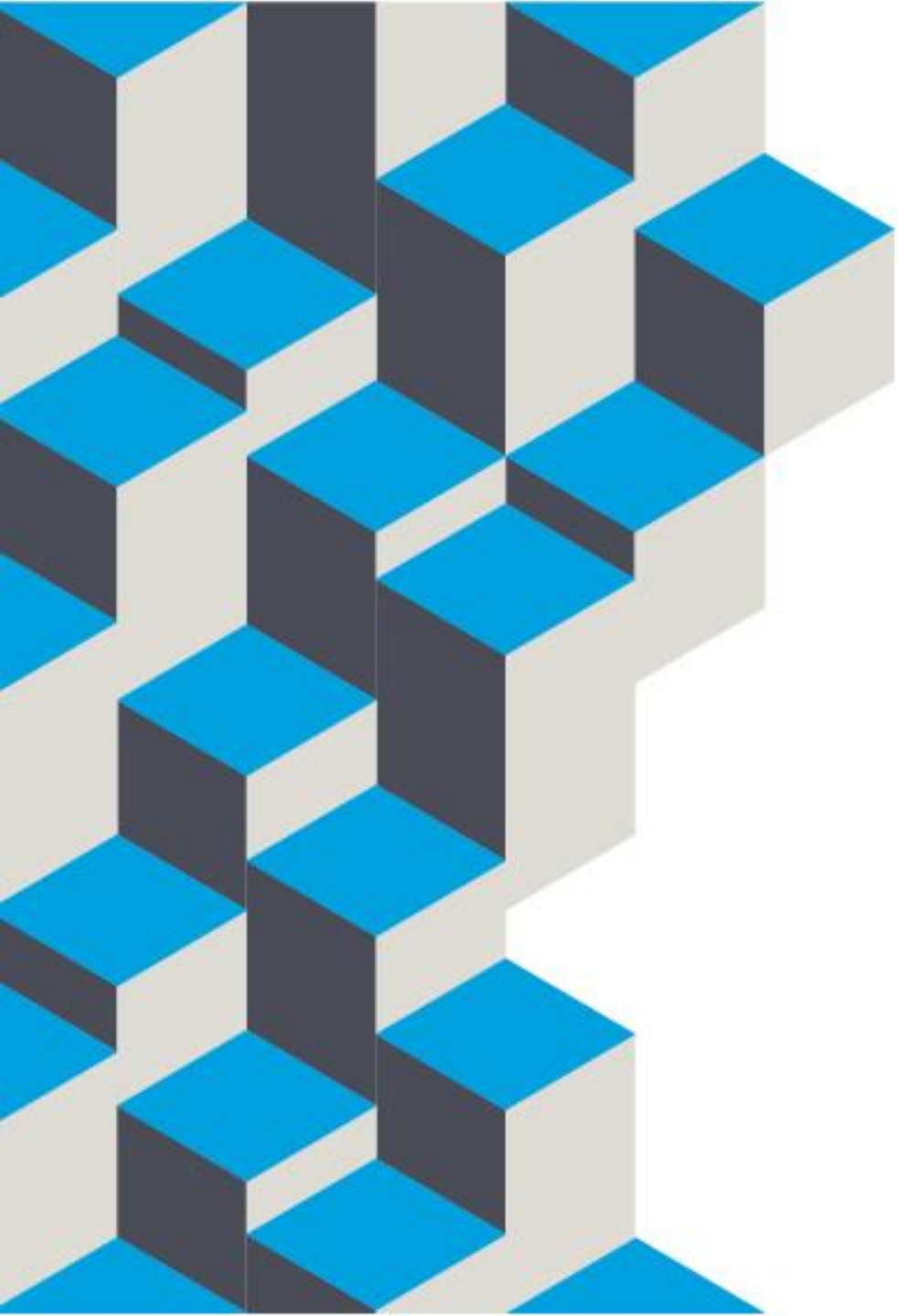
Manejo de errores: ejemplo

```
@task(log_stdout=True, max_retries=3,  
retry_delay=datetime.timedelta(minutes=5))  
def log_my_stdout():  
    print("I will be logged!")
```

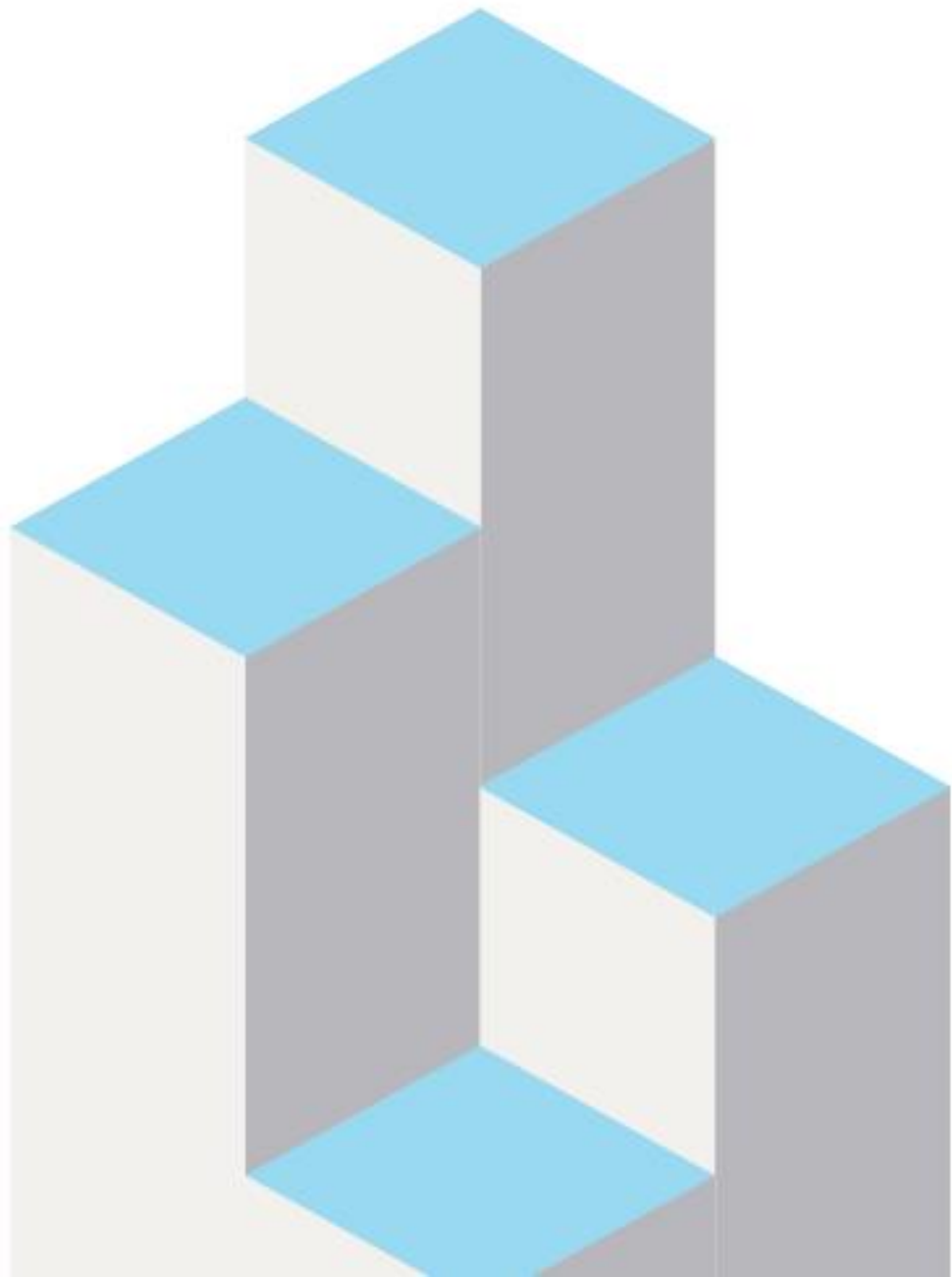


Trazabilidad

- Control de los logs
- Dejar “trazas” de los productos resultados de cada fase del Flow.
- Mejora muchísimo la credibilidad de nuestro Flow de cara a los consumidores de los datos.
 - ¿De dónde viene este resultado en esta agregación?
 - ¿Cómo has calculado esa columna calculada?
- Posible implementación: `df.to_csv(<path>)`



Práctica: Mejorando nuestro Hello World



Prefect Cloud



Orquestración y administración de flows

- Refresco del modelo de arquitectura de Prefect



Orquestración y administración de flows

- Implementa funcionalidades como:
 - Gestión de permisos y roles
 - Centraliza los flows y agentes para:
 - Configuración
 - Monitorización
 - Centraliza e implementa de forma segura parámetros y/o secretos
 - Permite implementar SLAs y gestión de equipos



Agente

- Equipo con un entorno de Python y Prefect Core instalado y funcional vinculado.
- Permite desarrollar e implementar los *runs* de un Flow.
- El equipo debe estar vinculado a nuestro *tenant** de cloud.
- La comunicación agente-cloud es unidireccional.

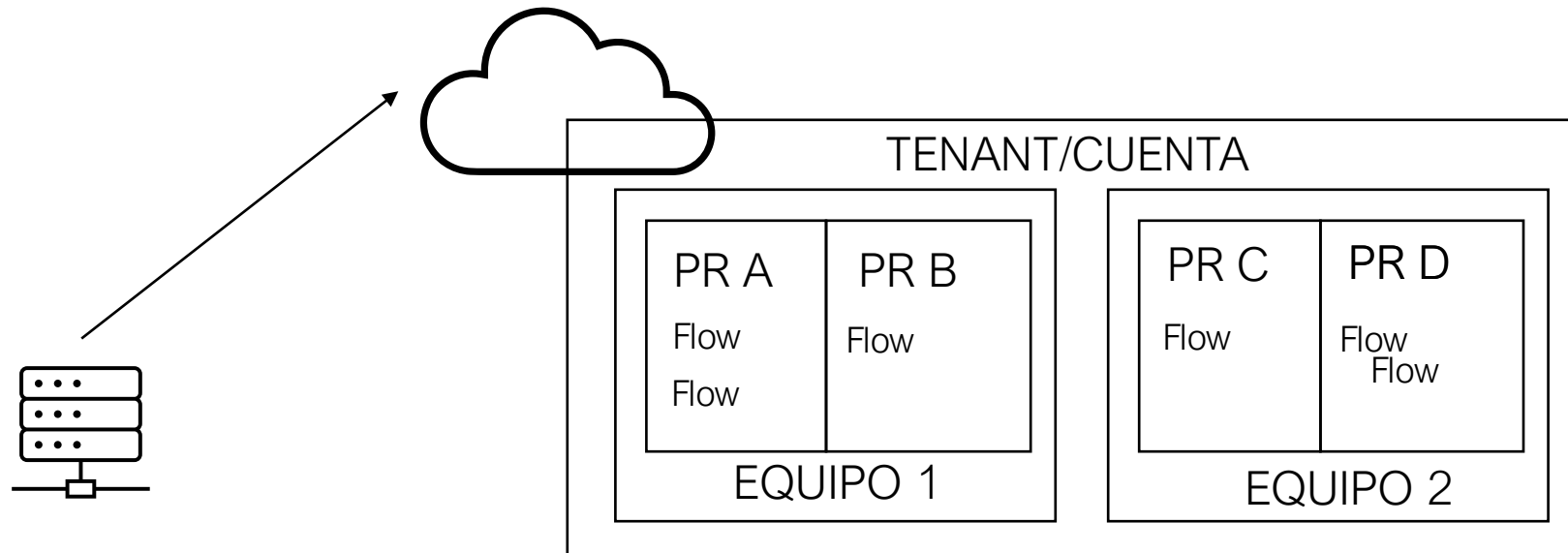


Agente

- Equipo con un entorno de Python y Prefect Core instalado y funcional vinculado.
- Permite desarrollar e implementar los *runs* de un Flow.
- El equipo debe estar vinculado a nuestro *tenant** de cloud.
- La comunicación agente-cloud es unidireccional.

Proyectos y equipos

- Equipo es un conjunto de recursos de Prefect Cloud para una o varias finalidades con accesos regulados por el administrador (cuentas *multi-team* son sólo para premium).
- Proyecto es un contenedor de flows registrados en prefect normalmente para una finalidad o con un contexto en común.





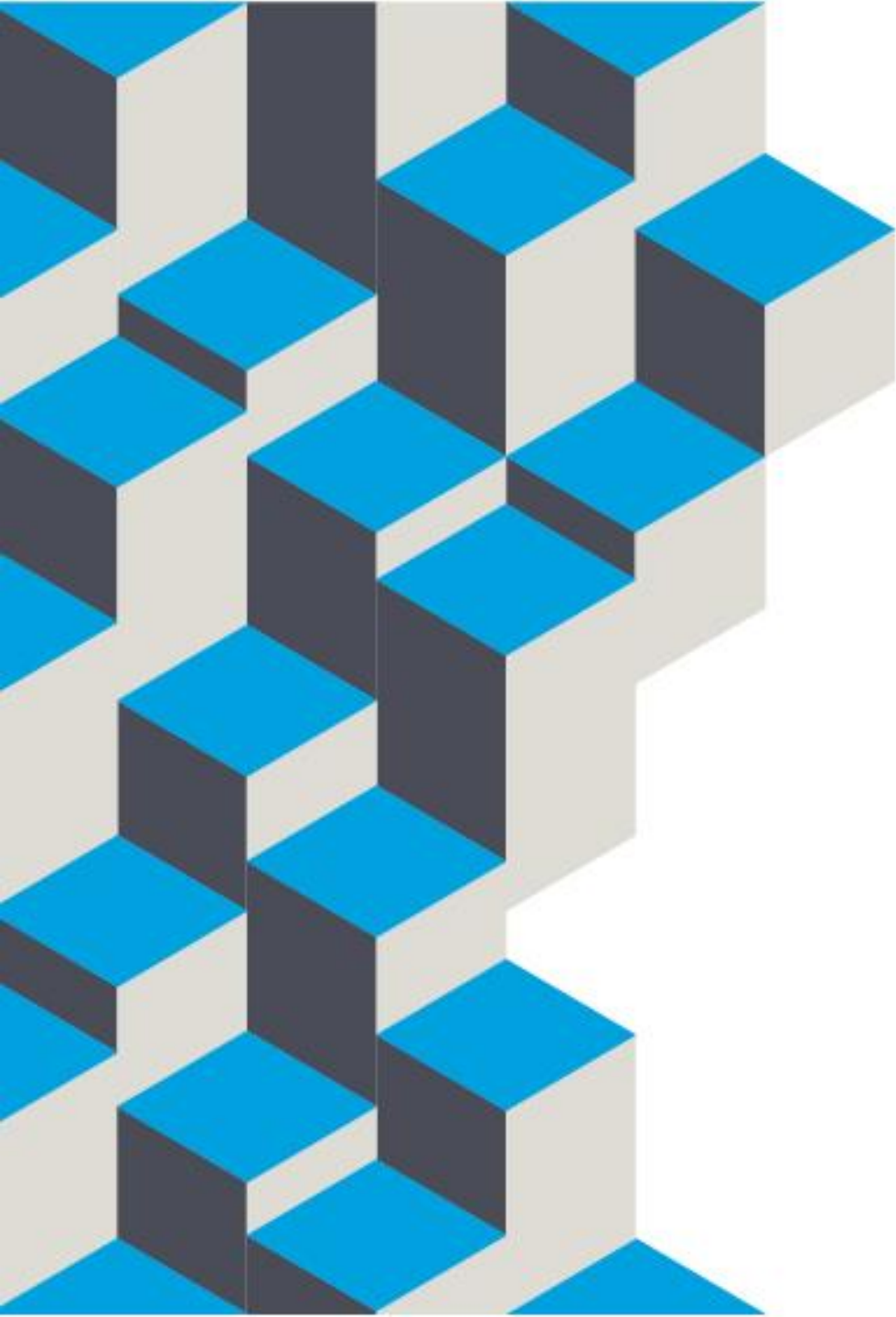
Versionaje

- La manera de desarrollar un Flow en Prefect es la siguiente:
 1. Desarrollar el código para nuestro Flow en un equipo con Prefect Core que esté vinculado en Prefect Cloud.
 2. Testear su funcionamiento en local mediante sentencias `flow.run()` o directamente testeando secciones de código.
 3. Una vez validado el Flow, subirlo/registrarlo a Prefect Cloud.
 4. Monitorizar el Flow en Cloud.
 5. Si es necesario actualizar el Flow para mejorarlo o implementar alguna solución a problemas encontrados, hacerlo.
 6. Si volvemos a subir el Flow con el mismo nombre en el mismo proyecto, Prefect cloud va a registrarlo como una nueva versión de nuestro Flow ya existente.

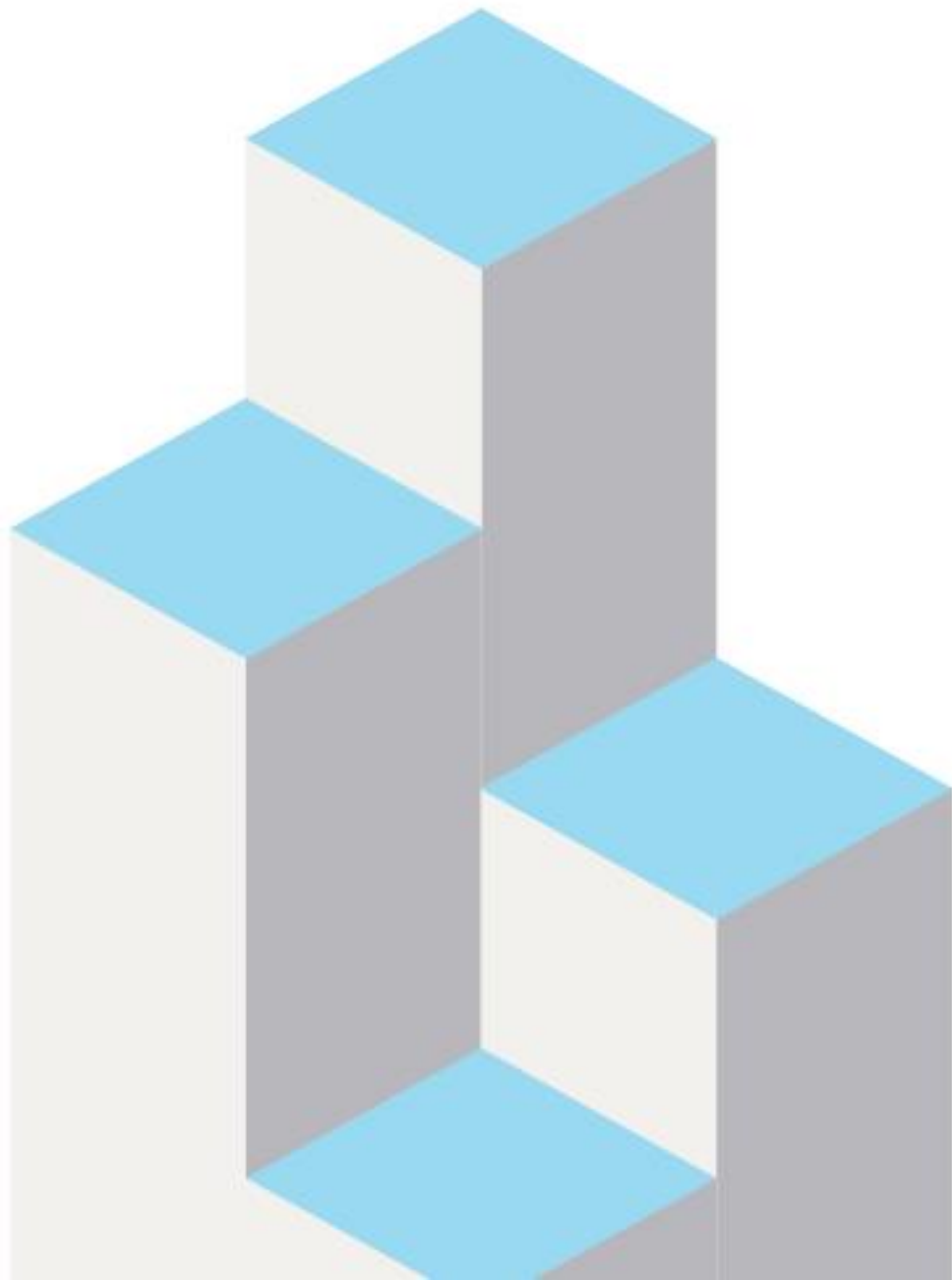


Secretos

- Hay valores/datos sensibles que como mejor práctica no deben quedar plasmados en código o ficheros de configuración.
- Prefect Cloud permite gestionar secretos en modo “clave-valor” de forma segura mediante encriptación.
- A veces hay que especificar al agente que busque el secreto en Prefect Cloud.



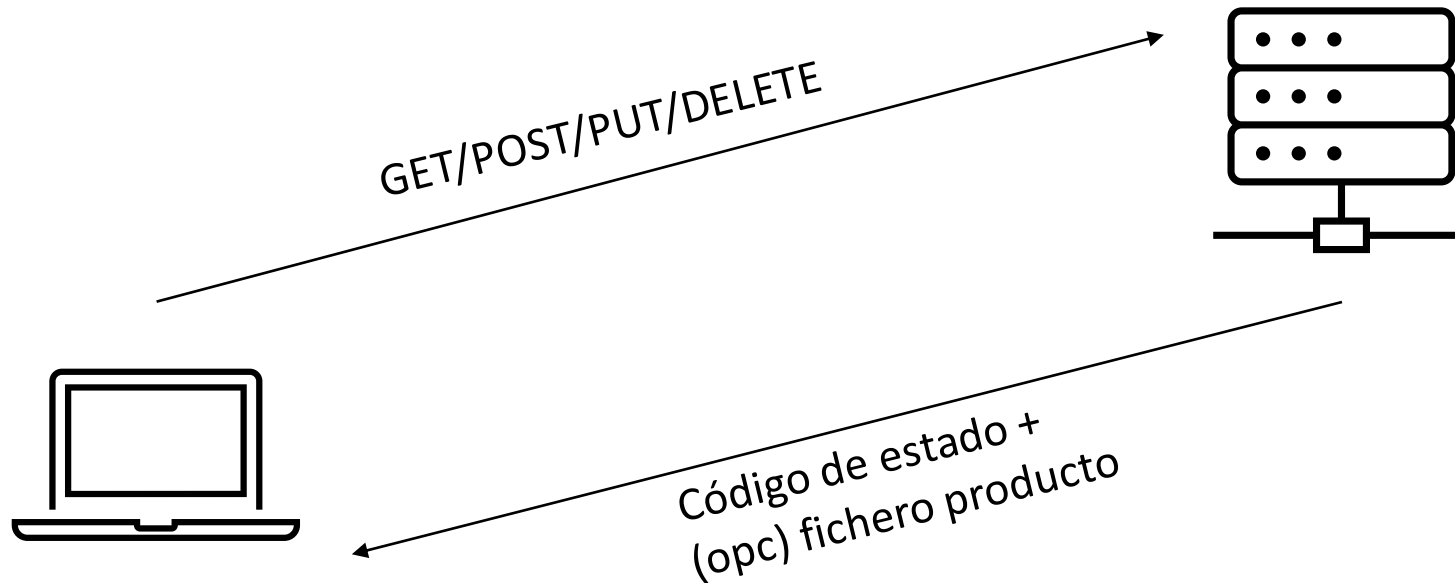
Práctica: Registrando nuestro Hello World en Prefect Cloud



Tu caja de
herramientas para
el Caso

APIs REST

- Una API suele ser una interfaz programática de una aplicación.
- El caso de las APIs REST, son interactuadas a través de HTTP/S.





APIs REST

- “Reglas” de las APIs REST:
 - Cliente-Servidor.
 - Sin persistencia de sesiones: consultas independientes.
 - Cacheables.
 - Permite implementar



APIs REST

- Códigos de estado a destacar HTTP:
 - 1xx: informativos
 - 2xx: correctos
 - 3xx: redirecciones o proceso inacabado
 - 4xx: error del cliente
 - 5xx: error del servidor
- Métodos HTTP:
 - GET → consultar datos. Puede devolver varios tipos de fichero (html, JSON, XML, ...)
 - POST → crear registros.
 - PUT → actualizar registros.
 - DELETE → eliminar registros.



APIs REST



APIs REST

- En Python:

```
import requests
```

```
response = requests.get('<URL_API_REST>')
```

```
response.status_code #devuelve el código de respuesta que ha devuelto
```

```
response.content #devuelve la respuesta del método GET (normalmente fichero)
```



JSON

- JSON es un formato que nace como evolución de XML.



Bases de datos

- RDBMS
- Instancia
- Base de datos
- Esquema
- Tablas



Bases de datos

- Normalización/desnormalización
- SQL vs NoSQL



Recursos cloud

- Azure Virtual Machines
- Azure SQL Server



Creando tu cuenta en Azure

- Recuerda que vas a tener que eliminar lo que hayas creado si no quieres seguir pagándolo.