Clase 21 June

> Fecha y hora: 2025-06-21 07:23:08

Puntos clave

Capítulos y Temas

1. **Levantar la API e Infraestructura en la Nube**

   > El objetivo principal es desplegar la API, conectarla a la base de datos y realizar el lanzamiento, todo en un entorno de nube. Esto implica la configuración de la infraestructura necesaria.

   - Los requisitos mínimos para levantar una UI y una API usando Container Registry y Docker incluyen un Grupo de Recursos (RG) y un Action Container Register.
   - Se pueden crear Grupos de Recursos ilimitados sin afectar el costo.
   - Un Grupo de Recursos, aunque ilimitado, requiere una zona geográfica porque representa una unidad lógica o metadatos que deben almacenarse en un servidor físico en alguna parte del mundo, para restricciones y latencia.

2. **App Service Plan y Web Apps**

   > Un App Service Plan es el servidor web en sí, cuya capacidad se define por el nombre SKU (ej. B1, F1). Las Web Apps se asocian a este plan y son el lugar donde se despliegan los contenedores Docker. Es crucial diferenciar entre el nombre del recurso en la plataforma Azure y el nombre concreto del despliegue.

- Un App Service Plan es el servidor web que proporciona memoria, disco y nodos.
- Las Web Apps son los elementos donde se realiza el despliegue de los contenedores Docker.
- El nombre SKU (ej. B1 por aproximadamente 10-14 dólares mensuales, F1 gratuito) determina la capacidad del servidor.
- Existe una diferencia entre el nombre del recurso que aparece en la plataforma Azure y el nombre concreto del despliegue.

3. **Gestión de Costos y Lecciones Aprendidas**

> Una anécdota real ilustró la importancia de verificar las configuraciones y los costos de los recursos antes de crearlos, ya que la falta de atención resultó en una factura de 15,000 USD. La interfaz de usuario actual de Azure ahora muestra los costos estimados por adelantado, lo cual es una mejora significativa.

- Un ingeniero de datos creó aproximadamente 7 bases de datos sin verificar las configuraciones, lo que generó una factura de 15,000 USD.
- Después de negociaciones, Azure cobró el 20% de la factura (3,000 USD), distribuidos como gasto operativo mensual durante un año.
- La interfaz de usuario actual de Azure muestra los costos estimados (ej. 122 USD mensuales para un servidor Premium B3, P1, B3 con 8 GB de RAM y 2 núcleos) antes de la creación del recurso, una mejora en la experiencia del usuario.

4. **Regiones de Recursos y Planes de Servicio**

> Es posible tener la definición de un Grupo de Recursos en una región geográfica (ej. Central US) y los recursos asociados, como un App Service Plan, en otra región (ej. Central Canadá). Sin embargo, algunos recursos

> específicos (ej. ciertos recursos de Machine Learning) pueden requerir estar en la misma región que el Grupo de Recursos.

- Un Grupo de Recursos puede estar en Central US y un App Service Plan asociado en Central Canadá.
- Algunos recursos (ej. ciertos de Machine Learning) pueden tener restricciones y deben estar en la misma región que el Grupo de Recursos.

5. **Tiers de App Service Plan y Administración de Presupuesto**

> Para optimizar costos, se recomienda a los estudiantes usar el tier F1 (gratuito) o B1 (aproximadamente 13 USD/mes), administrando cuidadosamente su presupuesto de 100 USD. Es fundamental eliminar todos los recursos creados después de las presentaciones del proyecto (antes del 2025-06-28) para evitar cargos continuos. Azure clasifica los tiers en 'Test' y 'Producción', promoviendo una configuración de tres entornos (Desarrollo, QA, Producción).

- Se recomienda usar el tier F1 (gratuito) o B1 (aproximadamente 13 USD mensuales).
- Los estudiantes disponen de un presupuesto de 100 USD.
- Es obligatorio eliminar todos los recursos creados antes del 2025-06-28 (el sábado siguiente a la conferencia) para evitar cargos diarios.
- Azure clasifica los tiers en 'Test' (ej. B1, B2) y 'Producción'.
- La configuración ideal de entornos incluye Desarrollo, QA y Producción, lo que implica multiplicar los recursos por 3.
- La memoria máxima disponible es de 256 GB de RAM.
- El almacenamiento máximo es de 250 GB.

6. **Casos de Uso para Almacenamiento Grande en Servidores Web (250 GB)**

> Aunque los datos no estructurados de usuario (PDFs, imágenes) deben ir a Blob Storage, un gran almacenamiento en el servidor web (ej. 250 GB) se justifica para alojar modelos de lenguaje grandes (LLMs) u otros modelos pesados de IA/ML directamente en el servidor para predicciones en tiempo real. Almacenar estos modelos en Blob Storage y descargarlos por cada solicitud es ineficiente.

   - Los datos no estructurados de usuario (ej. PDFs, imágenes, música) deben almacenarse en Blob Storage.
   - Un servidor web con 250 GB de almacenamiento es útil para alojar modelos de IA/ML, como los LLMs, que pueden pesar entre 30 GB y 200 GB.
   - Estos modelos se utilizan para predicciones o clasificaciones en tiempo real.
   - Es más eficiente alojar los modelos directamente dentro de los activos de la API (ej. con FastAPI, Django) que descargarlos de Blob Storage por cada solicitud.

7. **Uso de APIs y Modelos de Ciencia de Datos**

> Se discute la integración de APIs con sistemas CRM como Microsoft Dynamics para el ingreso de información de nuevos clientes. Se menciona el desarrollo de una API personalizada que consume modelos de ciencia de datos programados por un científico de datos, los cuales discriminan el modelo a consumir según los inputs para evitar sesgos y dar la mejor predicción. La implementación

> con Docker facilita el consumo de estos modelos, que pueden variar en tamaño, desde 6 megabytes hasta 200 o 500 megabytes por modelo.

- Conexión de API con CRM (Microsoft Dynamics) para entrada de información de nuevos clientes.
- Consumo de una API desarrollada internamente.
- Modelos de ciencia de datos programados para discriminar y consumir el modelo adecuado según los inputs, evitando sesgos y mejorando la predicción.
- Uso de Docker para levantar el servidor y consumir los modelos, facilitando el acceso al directorio de archivos del servidor.
- Tamaño de los modelos: algunos son de 6 megabytes, mientras que otros pueden alcanzar los 200 o 500 megabytes.

8. **Creación y Costo de Web Apps en Azure**

> Se explica el proceso de creación de Web Apps en Azure, destacando la importancia de un nombre de host único para el DNS. Se aborda la pregunta sobre cuántas Web Apps se pueden crear sin afectar el costo, concluyendo que el costo se mantiene constante si todas se alojan en el mismo servidor web (ej. 14 dólares), ya que el cobro es por el servidor. Sin embargo, el problema de tener muchas Web Apps (ej. 100) en un servidor con recursos limitados (menos de 2 GB de RAM y un nodo de procesamiento) es que se caerán constantemente. Para pruebas, un servidor pequeño es suficiente, y para un proyecto, se sugiere que dos Web Apps (una para la API y otra para la UI) son suficientes por temas de costo.

- Creación de Web Apps en Azure, utilizando un grupo de recursos existente y asignando un nombre como 'API expertos'.
- La opción 'secure unique default host name' agrega caracteres aleatorios para asegurar que el nombre sea único, funcionando como un DNS.
- Configuración de la Web App para usar un contenedor con Linux, con metadatos en Central US y vinculado a un servidor web existente (menos de 2 GB de RAM y un nodo de procesamiento, suficiente para pruebas).
- El costo de las Web Apps se basa en el servidor web que las aloja; se pueden tener múltiples Web Apps (ej. 100) en el mismo servidor y el costo (ej. 14 dólares) se mantendrá constante.
- El problema de tener muchas Web Apps en un servidor con recursos limitados es la inestabilidad y caídas constantes.
- Para un proyecto, se considera que dos Web Apps (una para la API y otra para la UI) son suficientes por razones de costo.

9. **Política de Feedback**

> El orador no proporciona feedback 24 horas antes de una entrega, pero sí puede hacerlo antes de ese período.

- No se da feedback 24 horas antes de una entrega.
- El feedback se puede dar antes del período de 24 horas previas a la entrega.

10. **Configuración de Contenedores y Servidor Linux**

> Se utilizará un contenedor y un servidor Linux para el despliegue del proyecto.

```
- El proyecto se levantará como contenedor.
Se utilizará Linux como servidor.
```

1. **Operaciones Asíncronas y Colas en la Creación de Recursos**

> La creación de recursos en la nube se maneja de forma asíncrona, utilizando colas y procesos en segundo plano (background). Esto permite al usuario cerrar la interfaz sin esperar la finalización, mejorando la experiencia de usuario. Este tipo de experiencia requiere arquitecturas más complejas como colas, serverless o APIs desacopladas, no una simple API.

- Cerrar la pestaña no detiene la creación del recurso.
La creación de recursos se realiza en segundo plano (background).Se utilizan colas para gestionar las peticiones de creación de recursos.Mejora la experie ncia de usuario al no obligar al usuario a esperar la finalización.Requiere el us o de colas, serverless o APIs desacopladas para lograr esta experiencia.

1. **Azure Container Registry**

> Azure Container Registry es un recurso esencial para levantar aplicaciones con Docker, funcionando como un repositorio de imágenes de contenedor. Se puede encontrar y crear fácilmente en Azure, y su propósito es evitar la creación manual de recursos, que es donde herramientas como Terraform son útiles.

- Es un recurso necesario para levantar aplicaciones con Docker.
También conocido como Container Hub o Registry Hub.Se busca y se crea a t ravés de la opción 'crear recurso' en Azure.Se configura con opciones como 'Dead Central US Standard', considerada más que suficiente.Terraform se utili za para automatizar la creación de estos recursos y evitar el proceso manual.

1. **Opciones de Bases de Datos en Azure**

> Azure ofrece tres métodos principales para configurar bases de datos SQL: SQL Database (Infrastructure as a Service), Managed Instance (ideal para migrar infraestructuras on-premise con SQL Server) y la opción de levantar una Máquina Virtual con el motor de base de datos instalado. Cada opción se adapta a diferentes necesidades y escenarios de migración o despliegue.

- **SQL Database:** Es una opción de Infrastructure as a Service (IaaS) que p roporciona las conexiones necesarias para acceder a la base de datos. Managed Instance: Recomendada para empresas que migran infraestructuras on-premise (ej. Windows Server 2000 con SQL) a la nube, ofreciendo una rut a de migración más directa.Máquina Virtual: Permite levantar una máquina virt ual con el motor de base de datos instalado, replicando un entorno local en la nube. Los sistemas operativos domésticos no son adecuados para proveer se rvicio.Se pueden configurar bases de datos como 'Elastic pools' o 'database s erver'.

1. **Estrategias de Escalado de Bases de Datos: Scale Up vs. Scale Out (Elastic Pools)**

> Se discuten dos estrategias de escalado: 'Scale Up', que implica mejorar las capacidades de un único servidor (ej. aumentar el tier), y 'Scale Out', que distribuye la carga entre múltiples nodos o instancias. Los 'Elastic Pools' en Azure son un ejemplo de 'Scale Out', donde los nodos se añaden y eliminan automáticamente según la demanda, cobrando solo por el tiempo de uso de los nodos adicionales, aunque el precio base es más alto. Se puede establecer un límite máximo de nodos (ej. 10 nodos).

- **Scale Up:** Consiste en mejorar las capacidades de un servidor existente (ej. aumentar el CPU de 90% a un tier superior). Requiere apagar el servidor p ara el upgrade.
Scale Out (Clusterizar): Implica distribuir la carga entre múltiples nodos o inst ancias (ej. de un nodo a tres nodos, reduciendo el consumo de CPU por nodo de 90% a 30%).Elastic Pools: Implementación de Scale Out en Azure.Los Elas tic Pools añaden nodos automáticamente cuando la demanda es alta (ej. de 1 a 3 nodos).Los Elastic Pools eliminan nodos automáticamente cuando la dema nda disminuye (ej. de 3 a 1 nodo si el procesamiento baja a 1% por 20 minuto s).Se cobra solo por el tiempo de uso de los nodos adicionales (ej. 30, 40, 50 minutos por tres nodos).El costo de los Elastic Pools es más elevado que el de las bases de datos no elásticas, pero ofrecen escalado automático.Se puede establecer un tope o límite de nodos a reservar (ej. hasta 10 nodos).

1. **Influencia del Contexto de Negocio en las Decisiones Tecnológicas**

> Las decisiones tecnológicas, como la elección de un servidor elástico, deben basarse en el contexto de negocio y el comportamiento del usuario. Se ejemplifica con el caso de una universidad, donde la demanda es alta solo durante 2-3 semanas de matrícula, y el resto del tiempo es baja. Pagar un servidor elástico más caro (20% o 30% más) por 4 meses para satisfacer solo 20 días de alta demanda puede no ser presupuestariamente eficiente si los usuarios están 'obligados' a usar el sistema. En contraste, en un modelo 'Business to Business' (B2B), los contratos largos aseguran el uso, permitiendo economizar. La diferencia entre 'Front Office' (ej. Amazon, donde la lentitud significa pérdida de clientes) y 'Back Office' (ej. UNAH, donde la pérdida de un estudiante no es crítica si hay miles esperando afuera) también influye en estas decisiones.

- Las decisiones tecnológicas deben considerar el contexto de negocio y el comportamiento del usuario.
Caso Universidad: La demanda es alta solo durante 2-3 semanas de matrícula (incluyendo pre-matrícula, matrícula y adiciones), y baja durante los 5 o 3 meses restantes.Pagar un servidor elástico más caro (20% o 30% más) por 4 meses para satisfacer solo 20 días de alta demanda puede no ser rentable.La UNA puede permitirse la lentitud porque los estudiantes están 'obligados' a usar el sistema y hay muchos otros esperando (200 o 400 estudiantes).Modelo Business to Business (B2B): Los contratos largos (hasta 5 años) aseguran el uso del sistema, permitiendo economizar.Front Office (ej. Amazon): La lentitud o caída del sistema resulta en la pérdida de clientes potenciales.Back Office (ej. UNA): La pérdida de un estudiante no es crítica debido a la alta demanda de cupos.Factores sociales y de negocio influyen significativamente en las decisiones tecnológicas.

1. **Creación de Servidor y Base de Datos en Azure**

> Se procede a crear un servidor de base de datos y una base de datos en Azure, utilizando el grupo de recursos 'expertos'. Se destaca que Azure cobra por base de datos, no por servidor, a diferencia de los App Service Plans. Se configura un usuario administrador ('expertos admin') y se selecciona una configuración de desarrollo con 5 DTU, lo que resulta en un costo de 4 dólares mensuales. El aprovisionamiento de estos recursos puede tomar tiempo.

- Se crea un servidor de base de datos y una base de datos en el grupo de recursos 'expertos'.
Azure cobra por base de datos, no por servidor, a diferencia de los App Service Plans que cobran por servidor y permiten múltiples web apps.Se pueden registrar múltiples DNS sin costo, pero se cobra por la base de datos.Se configura un usuario administrador llamado 'expertos admin'.Se selecciona una conf

iguración de desarrollo: no serverless, DTU básico, 5 DTU, con un costo de 4 dólares mensuales.El aprovisionamiento de estos recursos suele tardar un tie mpo.Los recursos creados hasta el momento incluyen: Container Registry, We b App, Servidor Web, Servidor de Base de Datos y Base de Datos.

1. **Próximos Pasos para el Despliegue del Proyecto**

> Los siguientes pasos para el despliegue del proyecto incluyen el uso de FastAPI para la API, Docker para generar la imagen y contenerizarla, realizar un 'push' al Container Registry, y finalmente vincularlo a la Web App. Esto permitirá configurar un entorno de desarrollo para consumir los productos.

- Levantar el proyecto utilizando FastAPI para la API.
Usar Docker para generar la imagen y contenerizarla.Realizar un 'push' de la i magen al Container Registry.Vincular el Container Registry a la Web App.Confi gurar un ambiente de desarrollo para consumir los productos.

1. **Autenticación y Autorización (Single Sign-On)**

> Se introduce el concepto de Single Sign-On (SSO) con proveedores como Office 365, Google o GitHub. Aunque el SSO funciona bien para uso personal, su publicación requiere configurar una 'aplicación' y someterla a un proceso de aprobación por parte del proveedor (Google, Microsoft, Meta, GitHub), además de tener un dominio propio.

- **SSO:** Single Sign-On.
Se puede implementar con proveedores como Office 365, Google o GitHub.Pa ra que el SSO sea público, se debe configurar una 'aplicación'.La aplicación d

ebe pasar por un proceso de aprobación con el proveedor (Google, Microsoft, Meta, GitHub).Es necesario tener un dominio propio para la publicación.

1. **Azure Active Directory (Entra ID) y Registros de Aplicaciones**

> Azure Active Directory, ahora conocido como Entra ID, es un servicio para gestionar usuarios de sistemas operativos y usuarios de tipo aplicativo. Se detalla el proceso de registro de una nueva aplicación, explicando las opciones 'single tenant' y 'multi tenant' en el contexto de dominios de autenticación. 'Single tenant' restringe la aplicación a un dominio específico (ej. .edu.hn), mientras que 'multi tenant' permite la autenticación de usuarios de diferentes dominios de Office 365, aunque esto conlleva un mayor riesgo de seguridad. Se enfatiza que autenticación y autorización son conceptos distintos. La creación de estas aplicaciones no tiene costo.

- **Active Directory:** Ahora se llama Entra ID.
Permite gestionar dos tipos de usuarios: para sistemas operativos y de tipo ap
licativo.Proceso de Registro de Aplicación: Se realiza en 'Manage App Registr
ations', seleccionando 'New Registration' y asignando un nombre (ej. 'Office 3
65 Login').Opciones de Tenant (en contexto de autenticación):Single Tenant:
La aplicación funciona solo para usuarios de un dominio específico (ej. .edu.h
n).Multi Tenant: La aplicación permite la autenticación de usuarios de diferent
es dominios (ej. distintos clientes de Office 365).La opción 'multi tenant' es m
ás riesgosa, ya que cualquier usuario con Office 365 podría autenticarse (aun
que no necesariamente autorizado).Se distingue entre autenticación y autoriz
ación como dos conceptos diferentes.Se utiliza 'localhost' como ejemplo para
la URL de redirección.La creación de aplicaciones en Entra ID no tiene costo.

1. **Permisos de API y Microsoft Graph**

> Se explica cómo añadir permisos a las aplicaciones registradas para acceder a servicios de Microsoft. Microsoft Graph es el servicio más conocido para obtener información de usuario (nombre, dominio, dirección, correo). Se pueden delimitar los permisos (leer, editar, modificar) para controlar las acciones que la aplicación puede realizar, como actualizar nombres de usuario en Active Directory desde una aplicación móvil. Este concepto de permisos de API es similar en plataformas como Firebase, Google y Meta.

- Los permisos de API se añaden en la sección 'API permissions' de la configu
ración de la aplicación.
Se pueden añadir permisos para acceder a diversos servicios de Microsoft.Mi
crosoft Graph: Es el servicio más famoso para acceder a información de usua
rio (nombre, dominio, dirección, correo).Se pueden delimitar los permisos (ej.
leer, editar, modificar) para controlar las acciones de la aplicación.Un ejemplo
de uso es actualizar nombres de usuario en Active Directory desde una aplica
ción móvil.El concepto de permisos de API es similar en Firebase, Google y M
eta.Un 'usuario aplicativo' es un usuario embebido en la lógica de la aplicació
n (app, proceso batch, serverless).

1. **Autenticación y Autorización de Aplicaciones**

> La autorización de aplicaciones es un proceso crítico para la seguridad y la reputación de las empresas tecnológicas. No se puede permitir que cualquier aplicación se autentique sin un proceso de aprobación riguroso, especialmente para evitar asociaciones con sitios moralmente incorrectos. Empresas como Microsoft y Google requieren acuerdos de licencia, confidencialidad, dominio y sitio web para aprobar una aplicación, probablemente utilizando machine learning para su

> revisión. Este proceso es esencial para proteger su marca. Aunque se puede levantar una aplicación con Google/Firebase para cuentas personales, la adición de otros usuarios para acceso abierto requiere un proceso de aprobación formal.

- Los permisos incorrectos pueden generar vulnerabilidades de seguridad. La autorización de aplicaciones no puede ser a la ligera; no se puede dejar ab ierto para que cualquiera se autentique.Las empresas como Office 365 o Goo gle no vincularían su nombre con sitios moralmente incorrectos (ej. tráfico de órganos).El Single Sign-On (SSO) es una buena práctica.Microsoft requiere au torización para las aplicaciones, incluyendo acuerdos de licencia, confidencia lidad, dominio y sitio web.El proceso de aprobación es para que las empresas sepan el destino de la aplicación.Una vez aprobada, la aplicación puede ser ut ilizada libremente.Para agregar a otras personas a una aplicación con Google/ Firebase, se deben añadir manualmente como usuarios de prueba dentro de l a organización.Para un acceso más abierto, se requiere el proceso de aproba ción formal.El orador ofreció dulces del Día del Estudiante, ordenados por su esposa, ya que no se vieron el sábado pasado.Se propuso un receso de 15 mi nutos.

1. **Configuración de API Gateway y Balanceo de Carga**

> Se discute la conexión de la API al inventario y la base de datos, señalando que el portal con la API está bien, pero la API no debe ir directamente a la base de datos. Se menciona que el ERP maneja los datos de e-commerce (inventario, clientes, transacciones). El concepto de balanceo se conoce como API Gateway, que debería ser el componente a colocar para el balanceo de carga y enrutamiento, no Kubernetes, sino NGINX. Se aconseja evaluar el costo de implementar un API Gateway.

- Conexión de API al inventario.
El portal con API es correcto.La API no debe conectarse directamente a la bas e de datos.El ERP maneja datos de e-commerce (inventario, clientes, transacc iones).El balanceo de carga se refiere a un API Gateway.Se sugiere usar un A PI Gateway para balanceo de carga y enrutamiento.No usar Kubernetes para esto, sino NGINX.Evaluar el costo de implementar un API Gateway.

1. **Optimización de Costos y Grupos de Recursos**

> Se sugiere levantar solo una web app, incluso si está en el mismo App Service, para minimizar costos. También se recomienda pensar en cómo dividir los grupos de recursos.

- Levantar solo una web app para reducir costos.
Considerar el mismo App Service para la web app.Planificar la división de gru pos de recursos.

1. **Configuración de Seguridad para Base de Datos en Azure**

> Para conectarse a la base de datos (Expertus DB), es crucial configurar la seguridad. Esto implica habilitar "Selected Networks" en la sección "Set Server" y agregar IPs permitidas en "Firewall Rules". Además, se debe marcar "Allow Azure Services and Resources to access this server" para permitir que recursos de Azure (como la API) y desarrolladores locales accedan a la base de datos. Esto actúa como una regla de seguridad para evitar conexiones no autorizadas.

- Configurar seguridad para la base de datos Expertus DB.
Habilitar "Selected Networks" en "Set Server".Agregar IPs permitidas en "Fire wall Rules".Marcar "Allow Azure Services and Resources to access this serve

r".Permite acceso a desarrolladores locales y a la API.Evita conexiones de IPs desconocidas.

1. **Diferencia entre Cliente y Motor de Base de Datos**

> Se aclara la distinción entre un cliente de base de datos (como Azure Data Studios o Visual Code con extensión SQL Server) y el motor de la base de datos. El motor de la base de datos está en la nube, y el cliente solo se conecta a ese servidor remoto, sin necesidad de instalar el motor localmente. Esto contrasta con la práctica común en otras clases donde se instala el motor (Postgres, MariaDB, MySQL) localmente y se conecta vía localhost.

- Se usa Azure Data Studios o Visual Code (con extensión) como cliente.
El motor de la base de datos está en la nube.No es necesario instalar el motor de la base de datos localmente.El cliente se conecta al servidor remoto.Contraste con la instalación local de motores como Postgres, MariaDB, MySQL.

1. **Pasos para Conectarse a Azure SQL Database**

> Para conectarse a la base de datos en Azure, se debe agregar un nuevo servidor en el cliente (ej. Azure Data Studios). Se copia el nombre del servidor desde la configuración de la base de datos en Azure, se selecciona "SQL Login" como tipo de autenticación y se ingresa el usuario administrador y la contraseña creados al dar de alta la base de datos. Se menciona un problema potencial con el cambio de IP al usar una conexión de celular.

- Agregar nuevo servidor en el cliente.
Copiar el nombre del servidor de Azure.Seleccionar "SQL Login" para autentic
ación.Ingresar usuario y contraseña de administrador.Posible problema con c
ambio de IP.

1. **Creación de Esquema en la Base de Datos**

> El siguiente paso después de conectarse a la base de datos es crear un esquema (ej. Create schema API). Se recomienda usar un esquema específico para las tablas en lugar del esquema DBO por buenas prácticas y seguridad.

- Crear un esquema después de la conexión.
Ejemplo: Create schema API.Usar un esquema específico en lugar de DBO.Im
portancia para buenas prácticas y seguridad.

1. **Desarrollo de API con FastAPI**

> Para el desarrollo de la API, se utilizará FastAPI. Se menciona que los estudiantes pueden usar otros lenguajes o frameworks si se sienten más cómodos, siempre y cuando sigan las mismas prácticas enseñadas.

- Uso de FastAPI para el desarrollo de la API.
Flexibilidad para usar otros lenguajes/frameworks.Importancia de seguir las p
rácticas enseñadas.

1. **Gestión de Entornos de Desarrollo Python con Pyenv y Virtualenv**

> Se aborda la importancia de configurar entornos de Python (Pyenv y Virtualenv) para manejar proyectos legacy y múltiples

> proyectos, evitando conflictos de versiones y la instalación global de librerías. Pyenv ayuda a gestionar las versiones del lenguaje, mientras que Virtualenv crea entornos aislados para cada proyecto. Se compara con Docker, señalando que Docker consume más memoria y recursos durante el build, lo cual es una razón personal para preferir Pyenv/Virtualenv para el desarrollo local.

Pyenv/Virtualenv

- **Ventajas:**
- Menor consumo de recursos: Como se menciona en tu selección, consume menos memoria y recursos durante el proceso de construcción en comparación con Docker
- Mayor velocidad: Al no tener que construir imágenes completas, el inicio y cambios en el entorno son más rápidos
- Gestión flexible de versiones: Pyenv permite cambiar fácilmente entre diferentes versiones de Python según las necesidades del proyecto
- Entornos aislados por proyecto: Virtualenv crea espacios separados para las dependencias de cada proyecto, evitando conflictos
- Facilidad de uso: No requiere conocimientos de containerización o configuraciones complejas

Docker

- **Ventajas:**
- Aislamiento completo: Incluye no solo versiones de Python y paquetes, sino también el sistema operativo y otras dependencias
- Consistencia garantizada: El mismo entorno exacto en desarrollo y producción
- Reproducibilidad perfecta: Cualquier desarrollador obtiene exactamente el mismo entorno, eliminando el problema de "en mi máquina funciona"

- Portabilidad: Funciona en cualquier sistema que tenga Docker instalado

Consideraciones para elegir

- **Cuando preferir Pyenv/Virtualenv:**
- En máquinas con recursos limitados (menor consumo de RAM)
- Para desarrollo rápido con frecuentes cambios en el código
- Cuando trabajas principalmente en un solo lenguaje (Python)
- Para proyectos más pequeños o prototipos
- **Cuando preferir Docker:**
- Cuando necesitas replicar exactamente el entorno de producción
- En proyectos con múltiples servicios o dependencias externas (bases de datos, colas, etc.)
- Para equipos grandes donde la consistencia entre desarrolladores es crítica
- Cuando el proyecto requiere diferentes versiones de múltiples lenguajes/tecnologías

Como nota adicional, muchos desarrolladores optan por un enfoque híbrido: usar Pyenv/Virtualenv para el desarrollo diario rápido y Docker para pruebas de integración o cuando necesitan replicar exactamente el entorno de producción.

- Necesidad de configurar entornos Python.
Pyenv y Virtualenv para gestionar versiones y entornos.Evitar instalaciones gl obales de librerías.Manejo de proyectos legacy y múltiples proyectos.Compar ación con Docker: Docker consume más memoria durante el build.Pyenv gesti ona versiones del lenguaje.Virtualenv crea entornos aislados.

1. **Comandos Básicos de Pyenv**

> Se explican comandos útiles de Pyenv: pyenv versions para ver las versiones de Python instaladas y los entornos de proyectos, y pyenv install --list para ver todas las versiones de Python

> disponibles para instalar. Para instalar una versión específica, se usa pyenv install <versión>.

- `pyenv versions`: muestra versiones de Python instaladas y entornos de pro
yectos.
pyenv install --list: muestra versiones de Python disponibles para instalar.pye
nv install <versión>: instala una versión específica de Python.

1. **Gestión de Versiones en Node.js (NVM)**

> Se menciona que el equivalente a Pyenv/Virtualenv en Node.js es NVM (Node Version Management), que también permite saltar entre versiones del lenguaje y levantar ambientes particulares de Node.

- NVM (Node Version Management) es el equivalente en Node.js.
Permite cambiar entre versiones de Node.js.Permite levantar ambientes partic
ulares de Node.js.

1. **Versiones de Python y Soporte a Largo Plazo (LTS)**

> Se discuten las versiones de Python, incluyendo las "destiladas" como Anaconda (para notebooks, machine learning). Se enfatiza la importancia de usar versiones LTS (Long Term Support) para proyectos formales, ya que las versiones en desarrollo (como 3.14) no son recomendables para producción. La versión más actual recomendada hace 3 semanas era la 3.13.2.

- Versiones "destiladas" como Anaconda para ML/simulación.
Importancia de usar versiones LTS (Long Term Support) para proyectos forma

les.Versiones en desarrollo (ej. 3.14) no son recomendables para producción.L a versión más actual recomendada hace 3 semanas era la 3.13.2.

1. **Gestión de Entornos Virtuales y Paquetes en Python**

> Se discute la creación y activación de entornos virtuales en Python, utilizando virtualenv con la versión 3.3 y nombrando el entorno 'expertos 202502'. Se compara la idoneidad de diferentes lenguajes de programación para distintas tareas (Go para programación paralela, Node.js/PHP para web, Python/C# para datos/API). Se abordan los desafíos de la gestión de paquetes en Python, como la falta de un equivalente directo a node_modules y los problemas con pip freeze para requirements.txt en builds de Docker debido a conflictos de versiones (ej. FastAPI requiriendo 0.28.1 mientras se instala 0.29). Se menciona Poetry como una alternativa, pero el orador prefiere una gestión manual de las librerías en requirements.txt para asegurar la consistencia en los builds locales y de producción, especialmente con Docker, donde se prefiere un solo contenedor para optimizar el uso de recursos (ej. 2 gigas de RAM).

1. **Riesgos y Vulnerabilidades en el Software de Código Abierto: El Caso XZ Utils**

> Se enfatiza la importancia de evaluar las librerías de código abierto antes de su uso, revisando el repositorio de Git y el estado de los 'issues' (ej. 49 abiertos, 3415 cerrados, el último abierto el 24 de marzo de 2025). Se presenta una anécdota sobre una librería desactualizada (5 años sin commits) que

> causó problemas de configuración. El punto central es la vulnerabilidad de XZ Utils, descubierta por Andrés Freund en San Francisco, Estados Unidos, quien notó un retraso de 500 milisegundos en las conexiones SSH. La causa fue una puerta trasera insertada en liblzma (parte de XZ Utils), que podría haber afectado a la mayoría de los servidores Linux a nivel mundial, incluyendo smartphones, routers y sistemas en la nube. Este incidente se describe como un ataque a la cadena de suministro, donde un atacante (Jia Tan) se infiltró entre los programadores de código abierto, introduciendo código malicioso de forma gradual y explotando la presión sobre los mantenedores (Lace) para fusionar cambios. La versión afectada en Debian Sid fue XZ Utils 5.6.0.