Taller AWS parte dos

Taller - Data Version Control - DVC

La entrega de este taller consiste en un reporte y unos archivos de soporte. Cree el archivo de su reporte como un documento de texto en el que pueda fácilmente incorporar capturas de pantalla, textos y similares. Puede ser un archivo de word, libre office, markdown, entre otros.

DVC es una herramienta complementaria a Git que permite gestionar y versionar datos y modelos de aprendizaje automático en proyectos de ciencia de datos. Es independiente del lenguaje o framework, lo que la hace versátil y adecuada para cualquier proyecto de *Machine Learning*. Con DVC, los equipos pueden mantener un control completo sobre sus datos y modelos, asegurando transparencia, reproducibilidad y colaboración efectiva en el desarrollo de sus proyectos.

Instale DVC

1. En su espacio de AWS Academy lance una instancia t2.micro con sistema operativo Ubuntu Server 22.04 y 10GB de disco.
2. Conéctese a la máquina:
   a) Abra una terminal: En windows, escriba *cmd* y *Enter*. En macOS, abra la aplicación llamada *Terminal*.
   b) En la terminal emita el comando

   ```
   ssh -i /path/to/llave.pem ubuntu@IP
   ```

   donde /path/to/ se refiere a la ubicación del archivo llave.pem que descargó, e IP es la dirección IP de la instancia EC2 que lanzó. Si prefiere, en la terminal puede navegar a la ubicación del archivo llave.pem y emitir el comando

   ```
   ssh -i llave.pem ubuntu@IP
   ```

   Asegúrese de incluir en su reporte un *screenshot* de la conexión a la máquina virtual.
3. Verifique que tiene python 3 instalado y asegúrese que la versión es por lo menos 3.8.

   ```
   python3 --version
   ```

   Debe tener la versión 3.10.

4. Actualice la lista de paquetes del sistema ejecutando el siguiente comando:

   ```
   sudo apt update
   ```

5. Instale pip.

   ```
   sudo apt install python3-pip
   ```

   Verifique su instalación

   ```
   pip --version
   ```

6. Instale vemv para crear ambientes virtuales. En la versión 3.10 para Ubuntu/Debian puede usar el comando

   ```
   sudo apt install python3.10-venv
   ```

7. Cree un ambiente virtual con nombre env-dvc

   ```
   python3 -m venv /home/ubuntu/env-dvc
   ```

8. Active el ambiente con el comando

   ```
   source env-dvc/bin/activate
   ```

   y con el ambiente activo instale dvc con dependencias para administrar el almacenamiento remoto de datos en AWS S3.

   ```
   pip install "dvc[s3]"
   ```

   Otras opciones para el almacenamiento remoto son [s3], [gdrive], [gs], [azure], [ssh], [hdfs], [webdav], [oss] o [all] para instalarlos todos.

9. Verifique que cuenta con Git

   ```
   git --version
   ```

10. Para Windows existe un instalador en https://dvc.org/doc/install/windows. Allí también hay instrucciones para instalarlo con Conda.

Un primer proyecto en DVC

1. En la terminal asegúrese de estar en el home (`cd ~`) y cree una carpeta para su proyecto

   ```
   mkdir dvc-proj
   ```

2. Ingrese a la carpeta del proyecto

   ```
   cd dvc-proj
   ```

e inicie un repositorio de Git

```
git init
```

Asegúrese de que la rama actual quede nombrada como main

```
git branch -m main
```

3. En la misma carpeta inicialice el repositorio de DVC

   ```
   dvc init
   ```

4. Note que se han creado algunos archivos en la carpeta

   ```
   git status
   ```

   Para agregarlos al repositorio de Git realice un commit

   ```
   git commit -m "Inicializacion de DVC"
   ```

5. Agregue un archivo de datos a una subcarpeta data

   ```
   dvc get https://github.com/iterative/dataset-registry get-started/data.xml -o
       data/data.xml
   ```

6. Agregue el archivo de datos descargado al repositorio dvc

   ```
   dvc add data/data.xml
   ```

7. Note que en la subcarpeta data se han creado varios archivos, entre ellos uno con extensión .dvc que permite llevar el registro de los cambios al archivo de datos en git, así como un .gitignore para ignorar el archivo de datos. Explore estos archivos e incluya pantallazos en su **reporte**. Para listar los archivos en la subcarpeta data puede usar

   ```
   ls data
   ```

   o

   ```
   ls -la data
   ```

   Para listar el contenido de un archivo puede usar por ejemplo

   ```
   cat data/.gitignore
   ```

8. Agregue los archivos de registro al repositorio Git con el comando

   ```
   git add data/data.xml.dvc data/.gitignore
   ```

   y haga el commit asociado

   ```
   git commit -m "Add initial data"
   ```

9. Para crear un repositorio de GitHub remoto, siga estos pasos:

   a) Vaya a GitHub (https://github.com/) e inicia sesión en tu cuenta.
   b) Haga clic en el botón “New“ (Nuevo) en la parte superior izquierda de la página de inicio.
   c) Complete el nombre del repositorio, una breve descripción y configure otras opciones según sus preferencias.
   d) Opcionalmente, puede agregar un archivo README o un archivo .gitignore durante la creación del repositorio.
   e) Haga clic en el botón “Create repository“ (Crear repositorio) para crear el repositorio remoto.
   f) Conectar el repositorio local con el remoto:

      1) Copie la URL del repositorio remoto que creó en GitHub.
      2) Ejecute el siguiente comando en la terminal para vincular el repositorio local con el remoto:

```
git remote add origin https://github.com/usuario/nombre_repo.git
```

   g) Enviar los cambios locales al repositorio remoto:

```
git push -u origin main
```

   En consola, recibirás un mensaje solicitando el usuario de tu usuario en github:

```
Username for 'https://github.com':
```

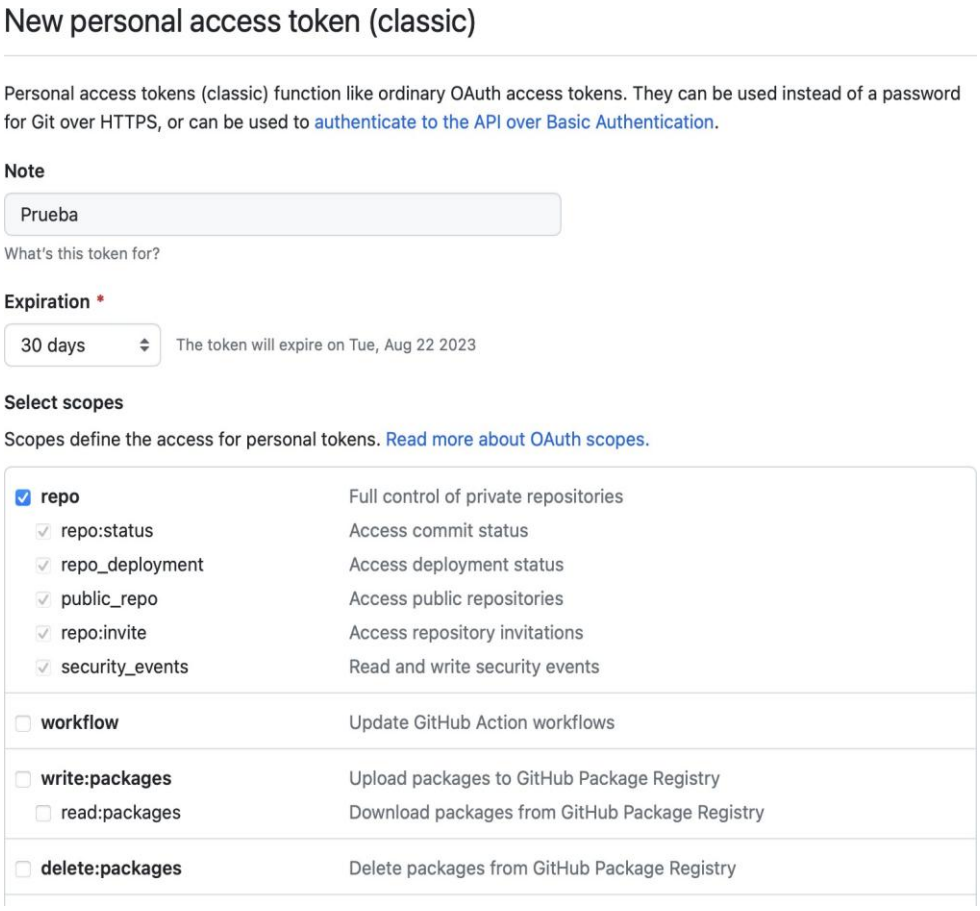
   Este mensaje solicita que ingrese su nombre de usuario de GitHub para autenticar tus operaciones Git cuando interactúa con el repositorio remoto a través del protocolo HTTPS. Posteriormente, le solicitará la contraseña de su cuenta en github:

```
Password for 'https://user@github.com'
```

   Sin embargo, el soporte para la autenticación con contraseña fue eliminado a partir del 13 de agosto de 2021. Ahora, en lugar de usar una contraseña, deberá generar un token de acceso para poder acceder a su cuenta.
   Para esto, debe seguir los siguientes pasos:

      1) En github, vaya a *settings*
      2) *Developer settings*
      3) *Personal access token*
      4) *Tokens (classic)*
      5) *Generate new token (classic)*
      6) Asigne un nombre
      7) Defina una duración

8) Seleccione “repo“



9) Click en *Generate token*
10) Copie el *token*

Si le solicita contraseña, ingrese el token.

Incluya en su reporte el link del repositorio y un *screenshot* de su repositorio con el archivo *.xml*.

Gestionando datos

Con DVC podemos contar con diversas fuentes de datos, llamados remotes. Para empezar configuremos y usemos una fuente local.

1. En el directorio /tmp cree una carpeta para almacenar los datos

   ```
   mkdir /tmp/dvcstore
   ```

2. Desde la carpeta del proyecto de DVC creada anteriormente (cd ~/dvc-proj) agregue esta fuente local con nombre fuentelocal

   ```
   dvc remote add -d fuentelocal /tmp/dvcstore
   ```

   Note que esta fuente queda definida como la fuente por defecto.
3. Haga un push de DVC para actualizar los datos en la fuente por defecto creada

   ```
   dvc push
   ```

   Note que esta fuente queda definida como la fuente por defecto.
4. Explore el contenido de la carpeta /tmp/dvcstore/ y observe que se ha agregado información sobre los datos registrados por DVC.
5. Hagamos una prueba eliminando los datos del caché y de la carpeta data con los comandos

   ```
   rm -rf .dvc/cache/
   ```

   ```
   rm -f data/data.xml
   ```

   Verifiquemos que los datos se han eliminado con

   ```
   ls data
   ```

   que no debe mostrar el archivo data.xml. Ahora realice un pull

   ```
   dvc pull
   ```

   y observe nuevamente el contenido de la carpeta data, verificando que el archivo data.xml se encuentra disponible.
6. Suponga ahora que se crean nuevos datos, lo cual simularemos duplicando los datos con los comandos

   ```
   cp data/data.xml /tmp/data.xml
   ```

   ```
   cat /tmp/data.xml >> data/data.xml
   ```

7. Ahora agregamos el archivo actualizado al registro de DVC

   ```
   dvc add data/data.xml
   ```

   y lo enviamos al repositorio de DVC con el comando

   ```
   dvc push
   ```

8. Ahora, agregamos los cambios al repositorio de Git haciendo un commit

```
git commit data/data.xml.dvc -m "Dataset actualizado"
```

Incluya en su reporte un *screenshot* del identificador md5 del archivo *.xml* e incluya el identificador en su reporte.

9. Clone el repositorio que creo en una nueva carpeta, para esto vuelva a la carpeta principal:

```
cd ~
```

Luego, clone el repositorio con el siguiente comando:

```
git clone https://github.com/usuario/nombre_repo.git
```

Incluya en su reporte un *screenshot* del repositorio clonado correctamente en otra carpeta.

10. Finalmente, en ocasiones es útil crear ramas en el repositorio de git. Lo anterior, permite la gestión y desarrollo colaborativo de proyectos de forma más eficiente y segura pues las ramas permiten:

a) **Desarrollo independiente**: Permite a los desarrolladores trabajar en diferentes características, correcciones de errores o mejoras en paralelo sin interferir entre sí. Cada rama es una línea de desarrollo independiente, lo que facilita la colaboración en proyectos con múltiples colaborales.

b) **Aislamiento de cambios**: Cuando se trabaja en una rama, los cambios que se realizan no afectan directamente al código en otras ramas. Esto ofrece un entorno seguro para probar nuevas ideas o implementaciones sin afectar la estabilidad del código en la rama principal o en otras ramas.

c) **Control de versiones**: Cada rama tiene su propio historial de cambios y versiones, lo que permite a los desarrolladores revisar, comparar y volver atrás en el tiempo para entender cómo evolucionó el código en cada línea de desarrollo.

d) **Pull Requests**: Las ramas son fundamentales para el flujo de trabajo de Pull Requests en GitHub. Los desarrolladores crean nuevas ramas para agregar nuevas características o solucionar problemas, y luego envían Pull Requests para fusionar sus cambios en la rama principal (por lo general, la rama "main.º "master"). Los Pull Requests permiten una revisión y discusión antes de que los cambios se incorporen al proyecto principal.

e) **Experimentación y pruebas**: Las ramas proporcionan un entorno aislado para experimentar con cambios importantes en el código sin comprometer la estabilidad del proyecto principal. Una vez que los cambios se prueban y validan en la rama, se pueden fusionar en la rama principal. **Versiones estables**: Al mantener una rama principal estable, es posible garantizar que siempre exista una versión funcional y utilizable del proyecto. Las nuevas características se desarrollan en ramas separadas, lo que ayuda a evitar la introducción de errores críticos en la rama principal.

Para crear una nueva rama y cambiar a ella, simplemente emita el siguiente comando en la terminal:

```
git checkout -b nombre_rama
```

Esto creará una nueva rama llamada **nombre_rama** y lo cambiará automáticamente a esa nueva rama.

Asimismo, el comando `dvc checkout` se utiliza generalmente después de realizar un `git checkout`, `git clone` o cualquier otra operación que cambie los archivos dvc.lock o .dvc actuales en el proyecto. Este comando restaura las versiones correspondientes de todos los archivos y directorios de datos rastreados por DVC desde el caché al espacio de trabajo.para descargar los datos y modelos que están versionados en DVC, pero que aún no están presentes en la copia local del repositorio. Esto es útil cuando usted está trabajando en un proyecto en equipo y otros colaboradores han versionado cambios en los datos o modelos. Al ejecutar `dvc checkout`, se asegura de tener las últimas versiones de los datos y modelos que aún no tiene en su copia local del repositorio.

En general, al cambiar a una nueva rama debe emitir los siguientes comandos:

```
git checkout nombre_rama
dvc checkout
```

Incluya en su reporte un *screenshot* realizando una exploración de los comandos anteriores.