

# La Bayeta de la Fortuna

Vamos a simular la creación de una aplicación web sencilla llamada “La Bayeta de la Fortuna”. Cada vez que accedamos a la web, nos dirá un texto auspicioso aleatorio. Para desarrollar y desplegar la aplicación tendremos que asegurarnos de lo siguiente:

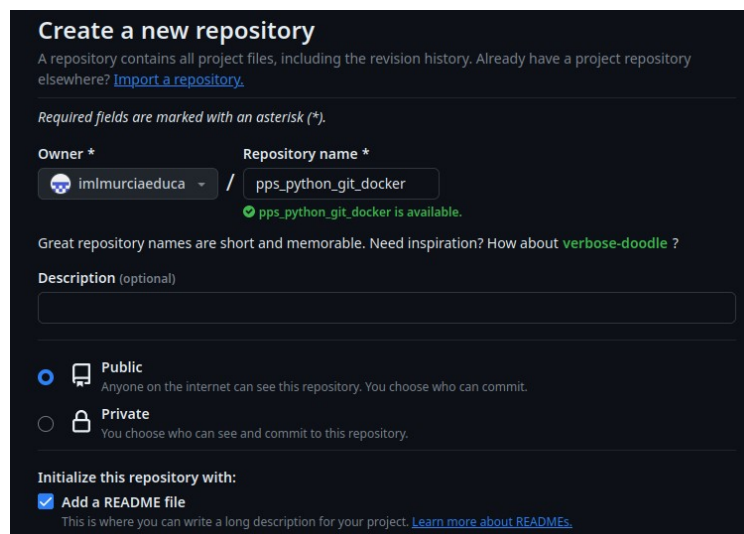


- 1) No perdemos código (ni tiempo repitiendo dicho código) y realizamos un mantenimiento del mismo. Además, trabajaremos colaborativamente y queremos publicar distintas versiones de nuestra aplicación conforme vayamos avanzando en su desarrollo
- 2) El proyecto debe poder clonarse y compilarse/ejecutarse sin que esto sea un problema de dependencias
- 3) Además, queremos asegurarnos de que el ecosistema en el que se ejecuta la aplicación se mantiene limpio y estable, de forma que no haya diferencias entre ejecutarlo en local o hacerlo en un servidor

Para ello vamos a utilizar (sí, lo has adivinado) Python (venv), Git y Docker

## 0. Introducción

1. Crea una cuenta en GitHub usando tu correo de Murciaeduca ([nre@alu.murciaeduca.es](mailto:nre@alu.murciaeduca.es))
2. Crea un proyecto público llamado “pps\_python\_git\_docker”. Recuerda añadir el fichero README.md



**Create a new repository**

A repository contains all project files, including the revision history. Already have a project repository elsewhere? [Import a repository.](#)

*Required fields are marked with an asterisk (\*).*

Owner \*  / Repository name \*

☒ pps\_python\_git\_docker is available.

Great repository names are short and memorable. Need inspiration? How about [verbose-doodle](#) ?

Description (optional)

☒ **Public**  
Anyone on the internet can see this repository. You choose who can commit.

☐ **Private**  
You choose who can see and commit to this repository.

Initialize this repository with:

☒ **Add a README file**  
This is where you can write a long description for your project. [Learn more about READMEs.](#)

3. Clona el proyecto en local, empezamos a trabajar. Para ello, tendrás que asociar previamente una clave SSH en tu cuenta de Github. Si no lo has hecho ya, investiga cómo hacerlo

## 1. El primer commit

1. Modifica (en local) el fichero README.md añadiendo más información sobre la aplicación. Recuerda que se trata de una aplicación al estilo de galleta de la fortuna/servilleta de bar
2. Crea un fichero app.py que imprima el típico (y cansino) “Hola, mundo”
3. Una vez te asegures de que funciona, añade los cambios al staging area (git add), haz un commit descriptivo y push
4. Comprueba que los cambios han llegado al proyecto de Github

## 2. Publicar versión

Una vez que nos hemos asegurado de que la versión más básica funciona, vamos a publicarla.

1. Desde Github, vamos al apartado de “releases” y le damos a “Create a new release”
2. Ponemos como tag “hola\_mundo” o algo parecido, “Target: main” y escribimos tanto el título de la Release como su descripción

Esto nos va a servir, entre otras cosas, para identificar el commit que hemos hecho sobre la rama en la que estamos (main) en el que garantizamos que se cumple la funcionalidad básica

## 3. Aislar el entorno

Queremos garantizar que nuestra aplicación se ejecuta bien sean cuales sean las librerías que tenemos instaladas en nuestro entorno de desarrollo. Esto también sirve para que nuestros compañeros de equipo, que también trabajan arduamente en la aplicación de la Bayeta de la Fortuna, puedan ejecutar la aplicación aunque tengan unas librerías distintas a las tuyas, o distintas versiones de las mismas

1. Creamos un entorno virtual de Python en el que ejecutaremos nuestra aplicación (seguimos usando nuestra máquina local). Para esta aplicación usaremos python venv. Activamos el entorno virtual, ¡consulta el cheatsheet!
2. Comprobamos la lista de librerías que tenemos instaladas con `pip list --local`
3. Deberíamos tener muy pocas librerías (1 o 2), pero la aplicación no debería necesitar nada en este punto
4. Ejecutamos la aplicación de nuevo
5. Una vez nos aseguremos de que funciona, exportamos las dependencias (requirements.txt)

## 4. Publicamos la versión aislada

1. Crea una rama nueva en Git, que parta de la rama actual (debería ser main) y muévete a dicha rama
2. Modifica el fichero README.md para indicar cómo resolver las dependencias adecuadamente. Indica también qué pasos debemos seguir para ejecutar la aplicación y comprobar que funciona

3. Si has creado el entorno virtual de Python en el mismo directorio en el que tienes configurado el proyecto de Git, crea un fichero `.gitignore` para que no considere el directorio del entorno virtual
4. Comprueba la lista de cambios con **git status**
5. Añade los cambios al staging area, haz commit con un mensaje descriptivo y push. Si te da error en el push es posible que sea porque esa rama aún no existe en el servidor. Lee bien el mensaje de error para resolverlo
6. Asegúrate de que funciona todo bien simplemente haciendo git clone de esa rama (usa el flag `-b`) en otro directorio y siguiendo los pasos del README.md
7. Una vez te hayas asegurado, haz un merge de esta rama en “main” y haz “push”
8. Sigue los pasos del apartado 2 para publicar una nueva versión

## 5. Trabajo colaborativo

Tras la reunión presencial que podía haber sido perfectamente online (o incluso un email y ancha es Castilla) se han repartido tareas para el desarrollo de la exitosa aplicación Bayeta de la Fortuna. Necesitamos disponer de una función que provea de frases auspiciosas para nuestra Bayeta de la Fortuna. Para ello, vamos a crear un nuevo fichero Python que se encargue de dicha responsabilidad. Para no frenar ninguna línea de desarrollo, se establece el siguiente acuerdo que todas las líneas deben cumplir:

- El script de Python que implementa la función de frases auspiciosas se llamará “**bayeta.py**”
- La función de Python que provee de frases auspiciosas tendrá la siguiente cabecera:

```
def frotar(n_frases: int = 1) -> list()
```

A partir de aquí, vas a colaborar con otro compañero. Va a haber 2 líneas de trabajo. En tu propio proyecto vas a seguir una línea “Experto en WEB”, pero seguirás la línea de “Experto en Python” en el proyecto de tu compañero, así como otro compañero completará la otra línea en tu proyecto

1. Crea una rama que parta de la “main” y muévete a dicha rama
2. Crea el fichero “bayeta.py” con la función “frotar” tal y como se describe arriba. El cuerpo de la función será la palabra “pass”, ya que está sin definir
3. Importa la función en “app.py”
4. Comprueba que la aplicación funciona
5. Git add, commit, push...
6. Mergea la rama en “main” y haz “push”

### 5.1 Línea “Experto en WEB”

1. Crea una rama nueva que parta de “main” (en tu proyecto)
2. Asegúrate de tener activado el entorno virtual de Python

3. Echa un vistazo a los siguientes enlaces para ver cómo crear una aplicación web sencilla con Flask:  
<https://flask.palletsprojects.com/en/2.3.x/quickstart/>  
<https://www.geeksforgeeks.org/flask-creating-first-simple-application/>
4. Modifica app.py para que el “Hola, mundo” se muestre en una web en localhost:5000
5. Modifica el método “frotar” para que devuelva una lista con N frases. Recuerda que no eres experto en Python, así que será siempre la misma frase, pero N veces. Esto se hace para salir del paso y seguir trabajando sin esperar el código de tu compañero
6. Crea un endpoint GET /frotar/<n\_frases>. Este debe devolver, en formato JSON, “n\_frases” frases. Las frases que devolverá este endpoint serán las de la llamada a “frotar”
7. Una vez te asegures de que funciona, exporta las nuevas dependencias (ahora debería haber bastantes más que antes)
8. Git add, commit, push...

## 5.2 Línea “Experto en Python”

1. Crea una rama nueva que parta de “main” (en el proyecto de tu compañero)
2. Asegúrate de tener activado el entorno virtual de Python
3. En el fichero “bayeta.py”, modifica la función “frotar” para que se comporte como se espera: debe de elegir N frases aleatorias de una lista de frases que tenga previamente almacenadas (idealmente en una BBDD pero, de momento, será un fichero de texto), añadirlas a una lista y devolver la lista
4. En el fichero “app.py”, modifica el print de “Hola mundo” para probar tu función
5. Una vez te asegures de que funciona, exporta las nuevas dependencias (si las hubiera)
6. Git add, commit, push...

## 6. Puesta en común

Una vez las 2 líneas están terminadas, toca fusionar código.

1. Desde la rama de la línea de “Experto Web”, haz merge de la rama de la línea “Experto en Python”. Probablemente no tengas esa rama en local. Para “descargarla”, haz “git fetch” primero
2. Te saldrán varios conflictos. Resuélvelos. Recuerda que tú eres el experto en web y tu compañero el experto en Python. Si tienes alguna duda de los cambios que ha hecho y cómo resolver los conflictos sin romper la aplicación, pregúntale
3. Asegúrate de que funciona la aplicación
4. Modifica el README.md
5. Git commit, git push

## 7. Nueva versión

1. Cámbiate a la rama “main”
2. Mergea la rama de la línea “Experto Web” en “main”. Si has hecho todo bien, no deberían detectarse conflictos y la aplicación debería funcionar
3. Una vez te asegures de que funciona, git push
4. Publica una nueva versión siguiendo los pasos del apartado 2

## 8. Despliegue seguro

A continuación, vamos a garantizar que el ecosistema en que se ejecuta la aplicación se mantiene consistente entre la máquina local y el servidor de producción. Para ello, vamos a hacer uso de contenedores

1. Crea una rama que parta de “main”
2. Crea un fichero Dockerfile multifase
  - Fase de resolución de dependencias (basada en python:slim)
  - Fase de ejecución (basada en la de resolución de dependencias)
3. Crea el fichero .dockerignore. Añade todo lo que no sea estrictamente necesario para construir la imagen
4. Construye la imagen, despliega un contenedor y prueba que funcione
5. Modifica el README.md
6. Git add, commit, push
7. Mergea en “main” la rama
8. Publica una nueva versión

## 9. Redes y MongoDB

Investiga un poco sobre MongoDB: <https://www.mongodb.com/docs/manual/introduction/>

Como puedes observar, la forma en la que se almacenan los datos es parecida a JSON (lo cual también recuerda mucho a los diccionarios de Python)

Vamos a desplegar en otro contenedor una BBDD de MongoDB para almacenar nuestras frases auspiciosas

En el Moodle tienes un fichero de ejemplo para inicializar y obtener datos de una BBDD de Mongo

1. Levanta un contenedor con la imagen de Mongo. Búscala en <https://hub.docker.com>
2. Asegúrate de tener activado el entorno virtual de Python
3. Crea una nueva rama que parta de “main”
4. Resuelve las dependencias que necesites

5. Ejecuta el script de Python de Moodle y comprueba que funciona adecuadamente
6. Exporta las dependencias a requirements.txt

A continuación, la idea es ejecutar Mongo en un contenedor y conectarnos desde la Bayeta de la Fortuna, realizando una inserción inicial de datos con un script de inicialización (si procede)

7. Dividiremos el fichero de Moodle en 3 funciones:
  - Instanciación (conexión con el motor, obtener la BBDD y la colección concreta)
  - Inicialización (insertar datos, si procede). Puedes modificar el ejemplo que te he dado para que inicialice usando el fichero de texto
  - Consulta
8. Haz los cambios que consideres que necesita para que funcione su ejecución en un contenedor de Docker (acuérdate de lo del nombre del contenedor de Mongo en lugar de localhost, deben estar en la misma red)
9. Modifica el fichero “bayeta.py” para importar la función de consulta de frases auspiciosas aleatorias del fichero de Mongo
10. Modifica el método “frotar” para que llame al del fichero de Mongo
11. Construye de nuevo la imagen, levanta el contenedor y comprueba que sigue funcionando
12. Modifica el README.md
13. Git add, commit, push...
14. Mergea la rama en “main”
15. Publica una nueva versión

## 10. Docker-Compose

Llegados a este punto ya se empieza a complicar bastante el despliegue de la aplicación, por lo que vamos a migrar a Docker Compose

1. Crea una rama que parta de “main”
2. Escribe el “compose.yml” para desplegar ambos contenedores (recuerda que para el de la aplicación usaremos build)
3. Modifica el .dockerignore para que no copie el compose.yml al construir la imagen de la Bayeta de la Fortuna
4. Comprueba que la aplicación funciona correctamente al desplegar con “docker compose up”
5. Modifica el README.md
6. Git add, commit, push
7. Mergea en “main”
8. Publica una nueva versión

## 11. Volúmenes

Investiga la ruta en la que MongoDB almacena los datos (el Hub de Docker te ayudará)

1. Crea una rama nueva que parta de “main”
2. Modifica “compose.yml” para que cree un volumen para MongoDB
3. Asegúrate de que la aplicación sigue funcionando
4. Git add, commit, push
5. Mergea en “main”
6. Publica una nueva versión

## 12. Añadir frases auspiciosas

Llegados a este punto, sabiendo que nuestro contenedor cuenta con la persistencia adecuada, podemos modificar nuestra aplicación para poder añadir nuevas frases auspiciosas

1. Crea una rama que parta de “main”
2. Modifica el fichero que interacciona con Mongo para añadirle una función de inserción de datos, si no la tiene ya
3. Crea una función de inserción en “bayeta.py” que llame a la de Mongo
4. Modifica “app.py” para añadir un endpoint POST /frotar/add. Dicho endpoint recibe un fichero JSON que contiene las frases a añadir y devuelve un código 200. Llama a la función de inserción de “bayeta.py”
5. Despliega la aplicación, añade algunas frases adicionales con tu cliente REST favorito
6. Para la ejecución de la aplicación
7. Despliega la aplicación de nuevo, pídele varias frases auspiciosas hasta que devuelva alguna de las nuevas que has insertado
8. Si funciona, git add, commit, push
9. Publica una nueva versión