Mi 1ª Elección EEUU

(Con Clara del Rey Analytica)

| **CURSO:** | Programa de Especialización Big Data |
| --- | --- |
| **FECHA:** | Enero de 2025 |
| **ASIGNATURA:** | Procesamiento de Datos Big Data |
| **ALUMNO:** | Alejandro Díez Redondo |

# Índice

[**Índice 2**](#_8osk1m4xxnf6)

[**1. Agradecimientos 3**](#_e6iu9x78qe8)

[**2. Introducción 4**](#_vxu91io6o594)

[2.1 Contexto y motivación del proyecto 4](#_kep1ryyv912c)

[2.2 Objetivos 4](#_lt9448ngpdy5)

[2.3 ¿Echamos una partida? 5](#_47sz2uw3qs8w)

[**3. Generador de Datos: Origen y Características de los Datos 7**](#_q4tj0394h69n)

[3.1 Propósito del generador 7](#_ix98x7krsgst)

[3.2 Diseño e implementación 7](#_v0vibq33j71)

[3.3 Almacenamiento, tipo y estructura de los datos 8](#_qg7yu49okoko)

[3.4 Configuración y personalización 9](#_cls8ijps9mb)

[**4. Entorno y Herramientas Utilizadas 11**](#_ej84nse2whi9)

[4.1 Docker 11](#_ya6tkp7ql5ba)

[4.2 Apache NiFi 11](#_xtn8hyieou8j)

[4.3 MongoDB y Mongo-Express 12](#_iczt9jp5waae)

[4.4 Spark/Jupyter 12](#_7qg6vwibxopf)

[**5. Desarrollo del Flujo de Procesamiento 13**](#_ufjxeheig2km)

[5.1 Diseño del flujo en NiFi 13](#_7ltahprf6ukh)

[5.2 Transformaciones aplicadas 14](#_him8euc7ol4a)

[**6. Pruebas y Validaciones 19**](#_vojfm8k0snz)

[6.1 Pruebas de flujo en NiFi 19](#_k4pcdo4e8c8j)

[6.2 Validación de datos exportados e ingesta en MongoDB 19](#_14sf0leyadcq)

[**7. Resultados y Análisis de Datos 20**](#_izucbq8nar09)

[7.1 Consultas en MongoDB 20](#_jprfdzja4ozu)

[7.2 Visualización de resultados en Spark/Jupyter 25](#_l1dpd59b3oui)

[**8. Conclusiones, Futuras Mejoras y Referencias 32**](#_s0am1o9erdh9)

[8.1 Resumen de logros 32](#_uhugob2dz6h5)

[8.2 Lecciones aprendidas 32](#_jd98bccu6rhp)

[8.3 Propuestas de mejora 32](#_z1m4oix0baqp)

[8.4 Referencias 32](#_v2zgz9qsj8b7)

[**ANEXOS 33**](#_tukmc1zi1su2)

[ANEXO I: Instalación de Docker en Ubuntu 22.04.2 LTS 33](#_ff6lvtsk30c2)

[ANEXO II: Instalación de NiFi con Docker 35](#_oqz34upwuch6)

[ANEXO III: Instalación de MongoDB con Docker 38](#_h6bcj0iyd2rm)

[ANEXO IV: Configuración de los procesadores en NiFi 42](#_e4fq5ekv7f8j)

[ANEXO V: Puesta en marcha de Spark/Jupyter 53](#_7n6ybophasyr)

# **1. Agradecimientos**

*A mi madre, por estar siempre disponible.*

*A los profesores, por transmitirme sus conocimientos.*

*A mis compañeros, por hacer el camino más soportable.*

*A ChatGPT, por su infinita ayuda.*

# **2. Introducción**

## **2.1 Contexto y motivación del proyecto**

Este proyecto se va a realizar en VirtualBox en la siguiente máquina virtual proporcionada por Jose María, el profesor de la asignatura de Procesamiento de Datos, para el curso 2024-25:



En esta máquina ya se encuentran instalados Spark y Jupyter. Además de estas 2 tecnologías, también se utilizarán: Python, Docker, Apache NiFi, MongoDB y Mongo-Express.

La idea de llevar a cabo este proyecto surgió debido a la cercanía que hubo con las elecciones estadounidenses del 5 de noviembre de 2024 y del revuelo que hace ya tiempo presenciamos cuando Cambridge Analytica intentó influir en las elecciones de 2016.

A este alumno se le ocurrió que podría ser divertido realizar un simulador de las elecciones estadounidenses de 2024 en el que 2 jugadores, uno representando al partido demócrata y otro al republicano; pudiesen escoger distintas acciones para ganar las elecciones e ir analizando el peso de sus estrategias con las herramientas de big data explicadas durante el curso lectivo. Todo esto a través de la implementación de procesos y flujos que permitieran la transformación, el almacenamiento y el análisis de los datos que se generan al tener lugar las elecciones del juego.

## **2.2 Objetivos**

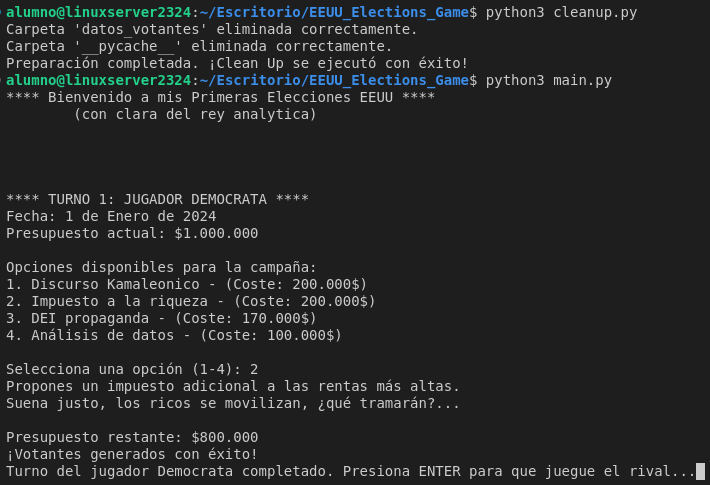
El objetivo principal de este proyecto es aprender y el secundario es divertirse, estamos haciendo un juego que podría clasificarse como edutainment: educación y entretenimiento; en el que 2 jugadores se embarcan en una aventura electoral con el objetivo de ganar las elecciones para su partido a través de la elección de acciones, estrategias y con la ayuda del análisis de datos. Este objetivo principal se nutre de objetivos específicos:

* **Generación de datos.** Desarrollar un generador de datos sintéticos que simule un escenario electoral realista.
* **Simulación interactiva.** Programar en Python las dinámicas de un juego por turnos para 2 jugadores de forma que tenga lugar la simulación de las elecciones.
* **Procesamiento de datos.** Implementar un flujo de procesamiento de datos con Apache NiFi para transformar los datos iniciales para que puedan ser ingestados por MongoDB y que estén listos para su uso en Jupyter y Spark.
* **Almacenamiento de datos.** Utilizar MongoDB como base de datos NoSQL para almacenar los datos procesados.
* **Análisis y visualización.** Realizar un análisis básico utilizando Spark y visualizar los datos mediante gráficas en Jupyter.
* **Evaluación del impacto.** Determinar el impacto de las diferentes estrategias electorales a través del análisis de los datos.

El objetivo principal de los jugadores, más allá de ganar las elecciones, es descubrir cómo funciona el juego, cómo afectan las acciones que escogen a su partido. Para ello tendrán que investigar con las herramientas de análisis de datos que se han mencionado anteriormente.

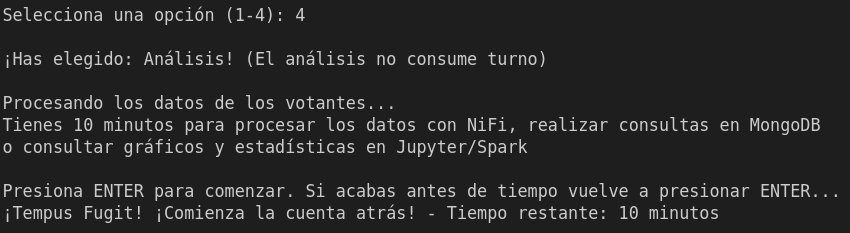
## **2.3 ¿Echamos una partida?**

A continuación se guiará al lector a lo largo de una partida de forma esquemática para que pueda ver la dinámica del juego en acción. Todo comienza de la siguiente manera, con el jugador demócrata, su cantidad inicial de dinero y la fecha de inicio, el 1 de Enero de 2024:



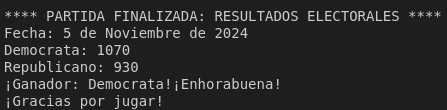
A cada jugador se le ofrecen 3 opciones aleatorias del pool de acciones.py y siempre se le ofrecerá la opción de "Análisis de datos" como 4ª opción. Estas 3 opciones aleatorias afectan a la generación de votantes parametrizada, pero para saber cómo afectan, el jugador debe de emplear la opción de análisis y poner en marcha NiFi para que los datos se ingesten en MongoDB y obtener el CSV con el que poder ver los datos en Jupyter o Spark. La opción de análisis no consume ningún turno, por lo que se puede volver a elegir otra opción. Todas las opciones tienen un coste que se descuenta del presupuesto del jugador.

Por ejemplo si eliges la opción de análisis, aparece lo siguiente:



El jugador tiene 10 minutos para activar NiFi, realizar el procesamiento de los datos; y poner en marcha MongoDB para lanzar algunas de las consultas que se mencionan más abajo en esta memoria o bien obtener el CSV para graficar datos en Jupyter desde el notebook que ya está diseñado.

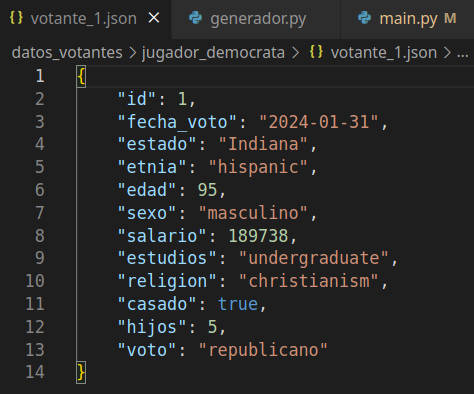
Otra forma de jugar es obviar la opción de análisis y llevar a cabo el programa de inicio a fin como si fuese un simulador de las elecciones EEUU con distintos eventos cánon, que será lo que haremos en esta memoria, generaremos todos los datos de ambos partidos y veremos en un análisis final cómo han afectado las decisiones en las elecciones y cuáles han sido los resultados. En este simulador la votación no va por estados, sino que gana quien haya recibido más votos en total.



# **3. Generador de Datos: Origen y Características de los Datos**

## **3.1 Propósito del generador**

El generador de datos tiene como propósito principal crear un conjunto de datos sintéticos que representen patrones reales de votantes en un entorno electoral. Este generador está diseñado para producir datos con diferentes atributos. El generador crea cada votante como un único archivo JSON con los siguientes campos:



Los posibles valores que pueden albergar estos campos se encuentran en: config.py y generador.py

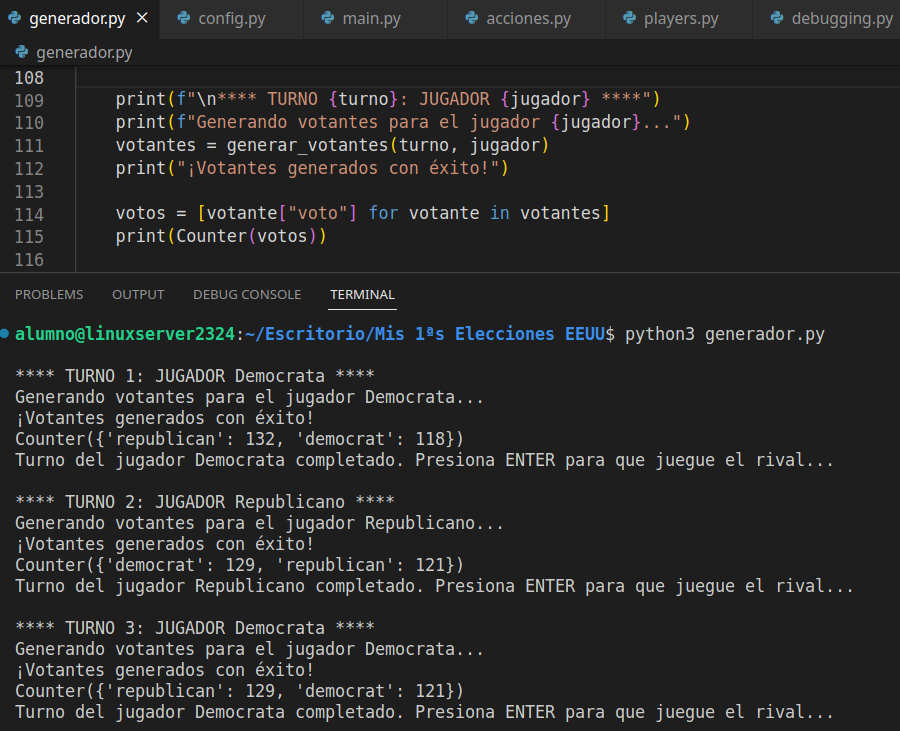
## **3.2 Diseño e implementación**

El generador se implementó en Python utilizando librerías como os, json, random, datetime… La generación de datos está parametrizada de tal forma que se definen pesos para distintos campos que influyen en el voto de los ciudadanos, tales como el estado, el sexo, el nivel educativo (estudios) o el salario. Estos pesos se combinan con los modelos de votante para cada partido: demócrata o republicano; definidos en el archivo de configuración, config.py, determinando por comparación a quién votará el ciudadano en cuestión.

Siendo más específicos, en generador.py, la función principal generar\_votantes se encarga de crear los datos de los votantes de manera iterativa, asignando valores aleatorios o "impactados" por las acciones específicas según los turnos y las decisiones durante el juego. Adicionalmente, el atributo voto de cada ciudadano es determinado mediante la función calcular\_voto, que utiliza una fórmula basada en los modelos predefinidos para calcular la probabilidad de que un votante apoye a un partido u otro a través del peso del resto de atributos.

El generador está diseñado para ser ejecutado de manera automática al inicio de cada turno, generando un conjunto de votantes únicos por cada jugador en cada etapa del juego.

Durante el desarrollo del generador se ajustaron cuidadosamente los pesos de las variables y los valores de los modelos, (config.py), para tener un generador equilibrado en el que el juego dependiera de las decisiones y estrategias de sus jugadores.



## **3.3 Almacenamiento, tipo y estructura de los datos**

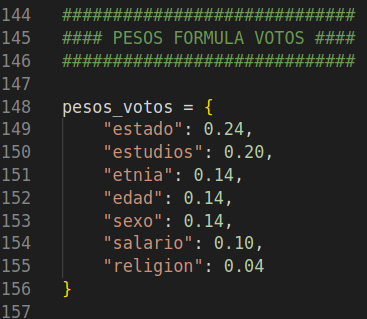
Como ha sido mencionado anteriormente, los datos generados por generador.py se almacenan en archivos JSON individuales para cada ciudadano. La estructura y la posibilidad de valores de cada registro es la siguiente:

* **ID.** Identificador único autoincremental del votante.
* **Fecha de voto.** Fecha en la que se registró el voto.
* **Estado.** Estado de residencia del votante: 50 posibles.
* **Etnia.** Grupo étnico del votante: white, black, hispanic, asian, indian, illegal.
* **Edad.** Rango de 18 - 99 años.
* **Sexo.** Masculino o femenino.
* **Salario.** Rango de 30.000 - 250.000 dólares.
* **Estudios.** Nivel máximo de estudios alcanzado: high school, undergraduate, graduate, masters y doctor.
* **Casado.** Indicador del estado civil: True o False.
* **Hijos.** Número de hijos, de 0 a 5.
* **Voto.** Partido político al que votó el ciudadano. Se calcula en un formato de probabilidad para cada partido con el resto de atributos anteriores y la función calcular\_voto, que compara los datos generados "aleatoriamente" o sesgados a través del impacto de las acciones con los modelos: modelo\_democrata y modelo\_republicano; y asigna un porcentaje de probabilidad de voto a cada partido. Se asigna como voto aquel que tenga mayor probabilidad. Hay 2 opciones: demócrata o republicano.

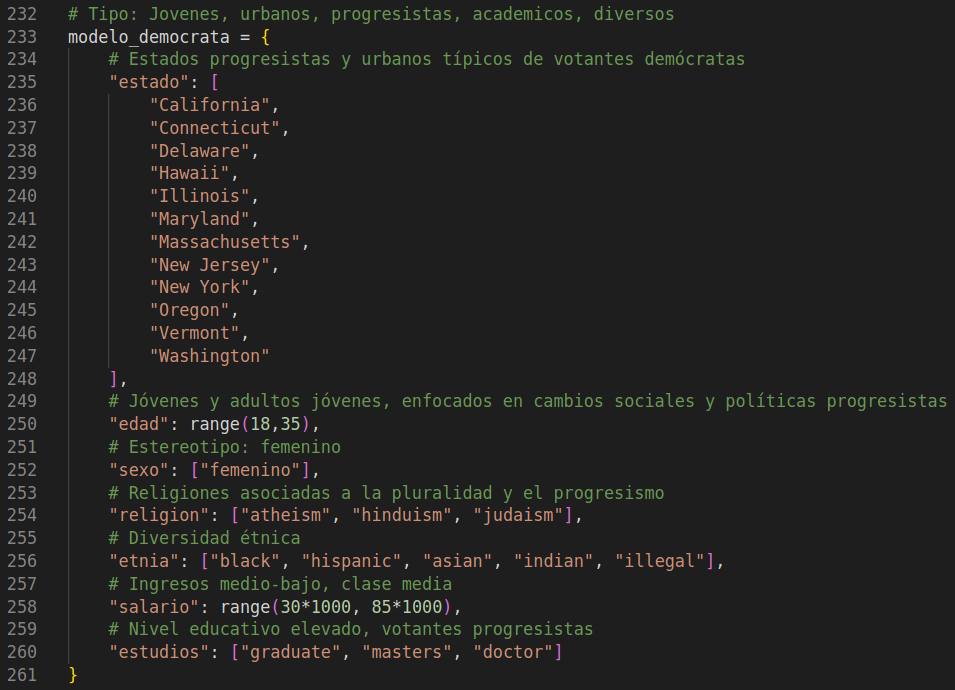
## **3.4 Configuración y personalización**

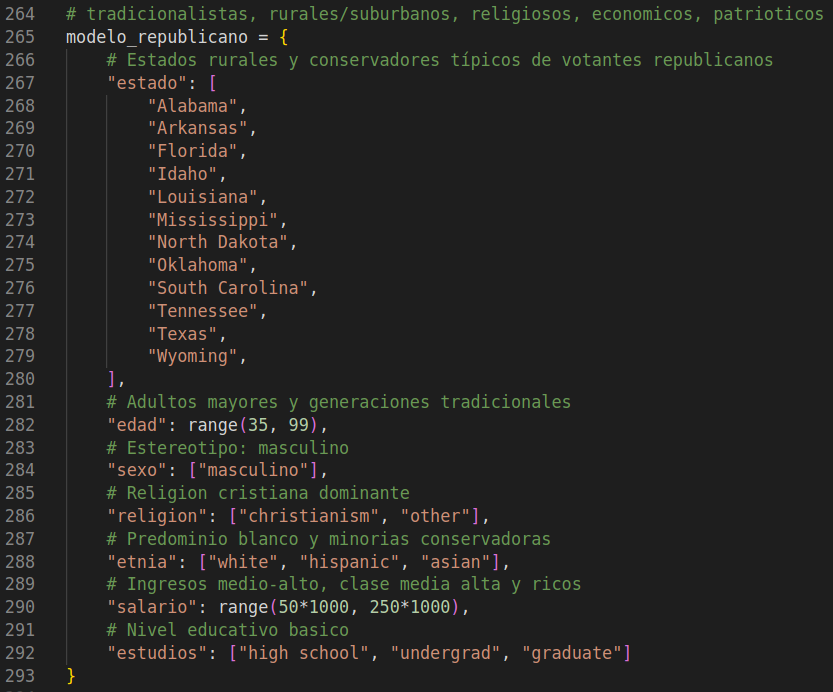
El generador permite ajustar parámetros clave a través de un archivo de configuración externo (config.py). Entre los parámetros configurables se incluyen:

* **Rango de fechas**: Fecha inicial y final de la simulación, comprendidas entre el 1 de enero de 2024 y el 5 de noviembre de 2024.
* **Número de turnos**: Se definieron 16 turnos en total, permitiendo que ambos jugadores exploren sus estrategias a lo largo del tiempo. Una partida más corta podría ser una con la mitad de turnos, 8.
* **Minutos de análisis**: la opción de análisis siempre está disponible para los jugadores a lo largo del juego, permite realizar consultas, análisis y gráficas en MongoDB, Jupyter y Spark. Por defecto son 10 minutos de análisis.
* **Los atributos de cada votante** mencionados anteriormente.
* **Pesos**: Valores que determinan la importancia relativa de cada atributo en la decisión de voto:

****

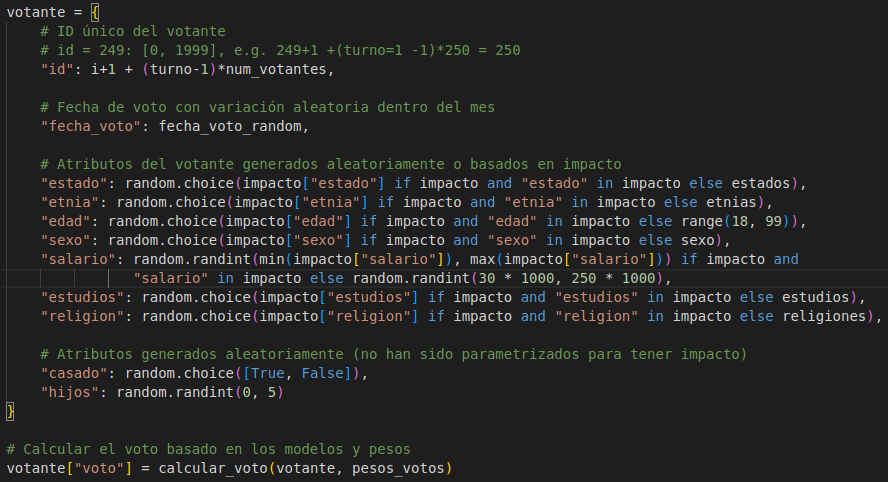
* **Modelos de votación**: Se establecieron 2 modelos diferentes:

**a) Modelo demócrata**: 

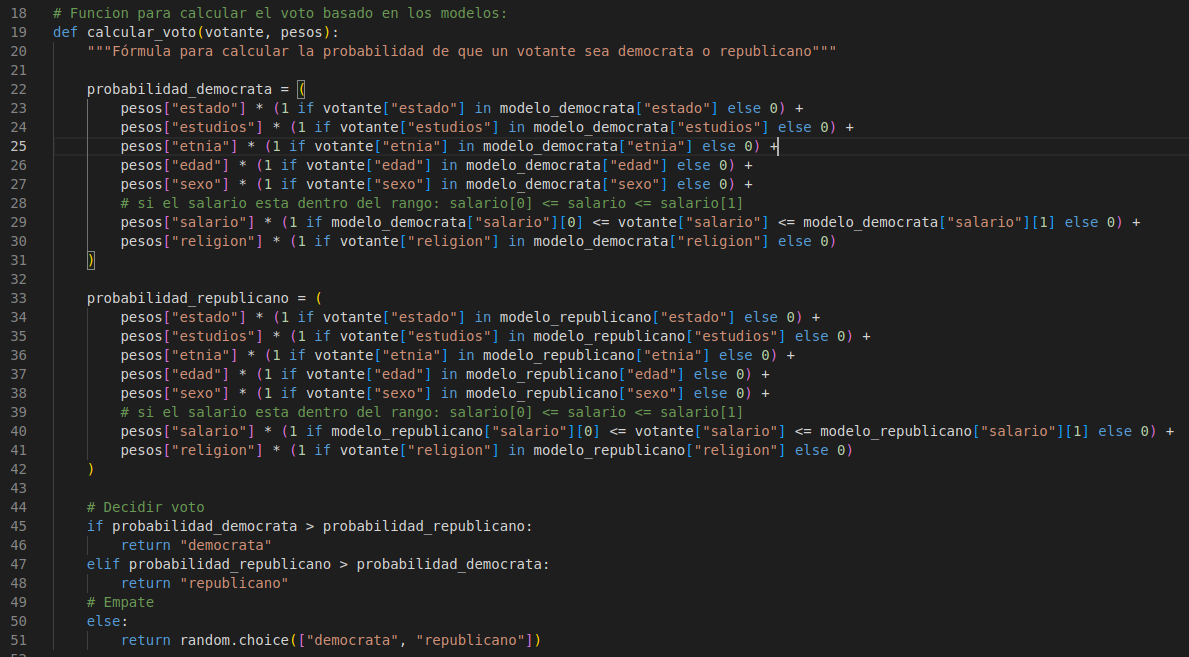
**b)Modelo republicano**:

Todo esto permite que el generador sea altamente personalizable, adaptándose a distintos escenarios de juego y ofreciendo una componente de experimentación.

Cabe mencionar de nuevo que a la hora de generar los atributos, estos se generan aleatoriamente o de manera impactada:

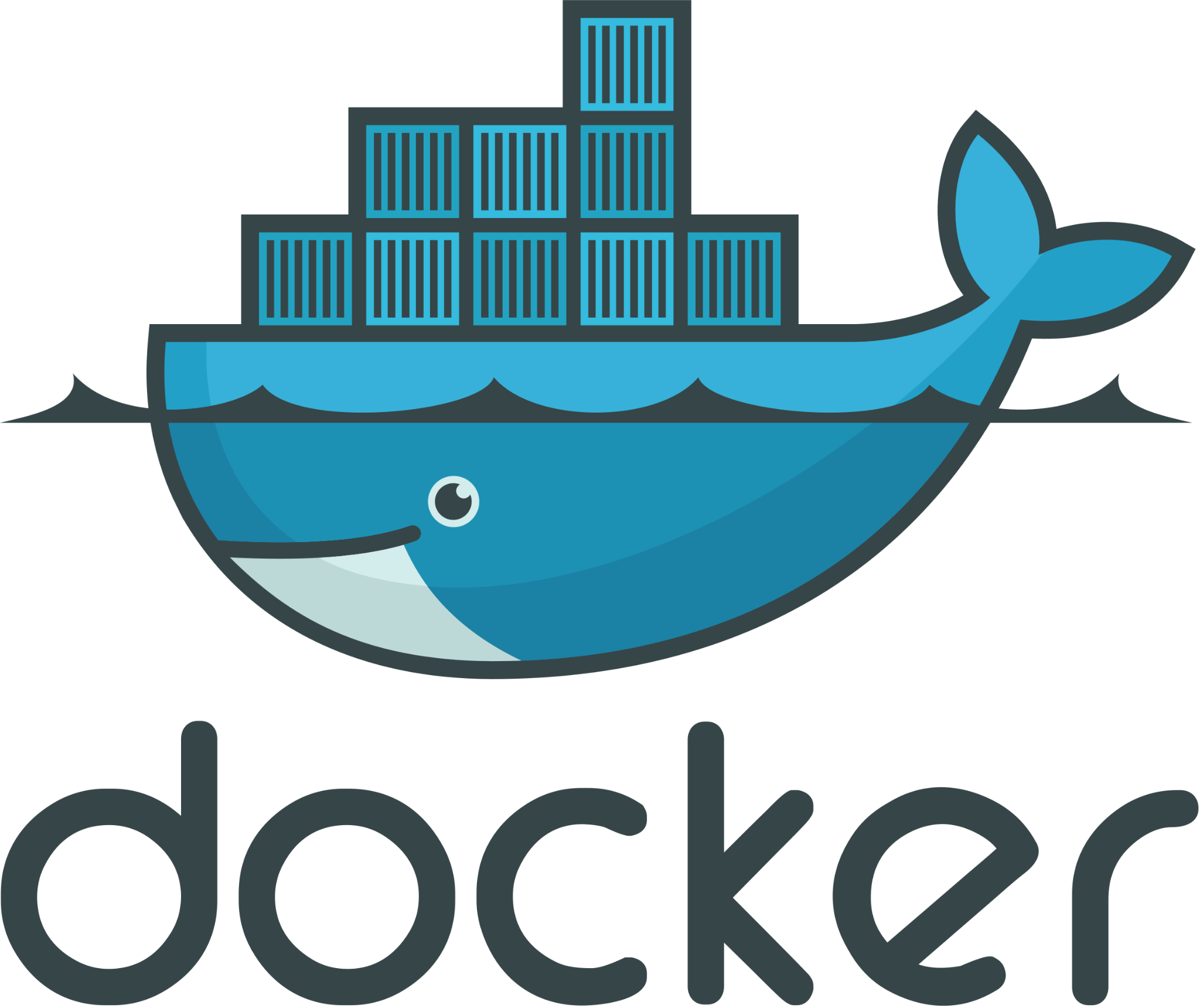


Y se calcula el voto teniendo en cuenta cuánto se asemeja el votante generado al modelo\_democrata y al modelo\_republicano, el modelo al que más se asemeje captará su voto:



# **4. Entorno y Herramientas Utilizadas**

## **4.1 Docker**

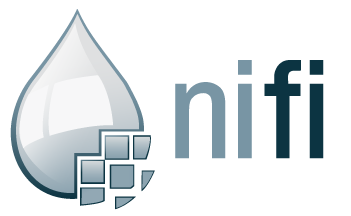


Docker es una plataforma que permite crear y gestionar contenedores, entornos virtualizados ligeros y portables que encapsulan aplicaciones y sus dependencias. En este proyecto, Docker se utilizó para contenedorización de 3 componentes clave:

* **Apache NiFi**: Para el flujo de procesamiento de datos.
* **MongoDB**: Como BBDD NoSQL para almacenar los datos procesados.
* **Mongo-Express**: Interfaz web para gestionar MongoDB.

El uso de Docker garantiza la portabilidad del proyecto, permitiendo replicar fácilmente el entorno en cualquier sistema que tenga Docker instalado.

## **4.2 Apache NiFi**



Apache NiFi es una herramienta de código abierto de procesamiento de datos que permite diseñar, gestionar y monitorear flujos de datos en tiempo real. Su interfaz gráfica facilita la creación de flujos mediante el uso de procesadores predefinidos que realizan tareas como filtrado, transformación y enrutamiento de datos.

En este proyecto, NiFi se encargó de recibir los datos generados de cada votante (los JSON individuales), procesarlos y prepararlos para su ingesta en MongoDB. Así como preparar los datos para compactarlos en un CSV que posteriormente utilizarían Jupyter y Spark.

## **4.3 MongoDB y Mongo-Express**



MongoDB es una base de datos NoSQL orientada a documentos que permite almacenar datos en formato JSON, lo que la hace ideal para manejar información semiestructurada y grandes volúmenes de datos. Mongo-Express es una interfaz web que facilita la administración de MongoDB, permitiendo a los usuarios consultar y gestionar los datos de manera visual.

En este proyecto, MongoDB se utilizó para almacenar los datos generados y procesados, mientras que Mongo-Express permitió comprobar visualmente la correcta ingesta de los datos.

## **4.4 Spark/Jupyter**

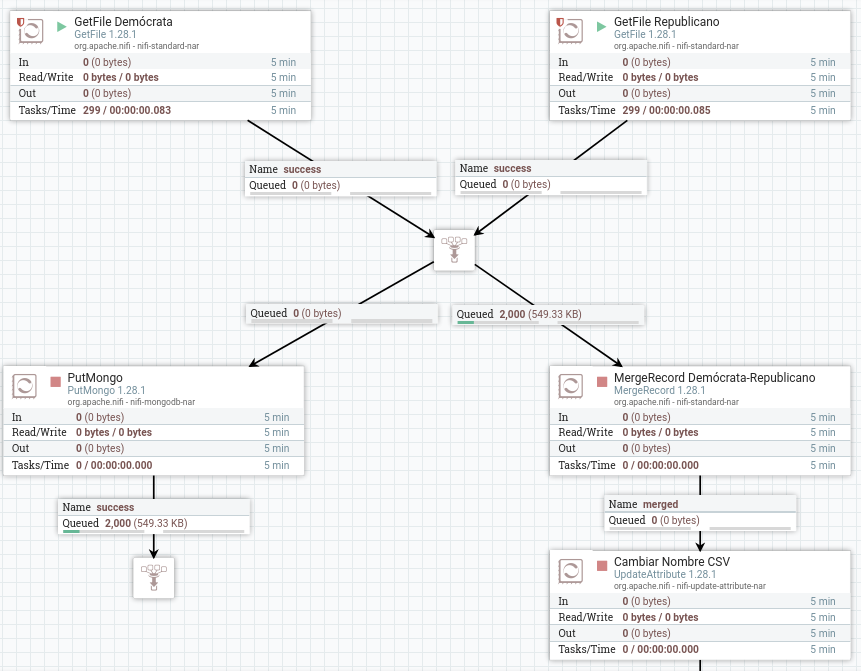


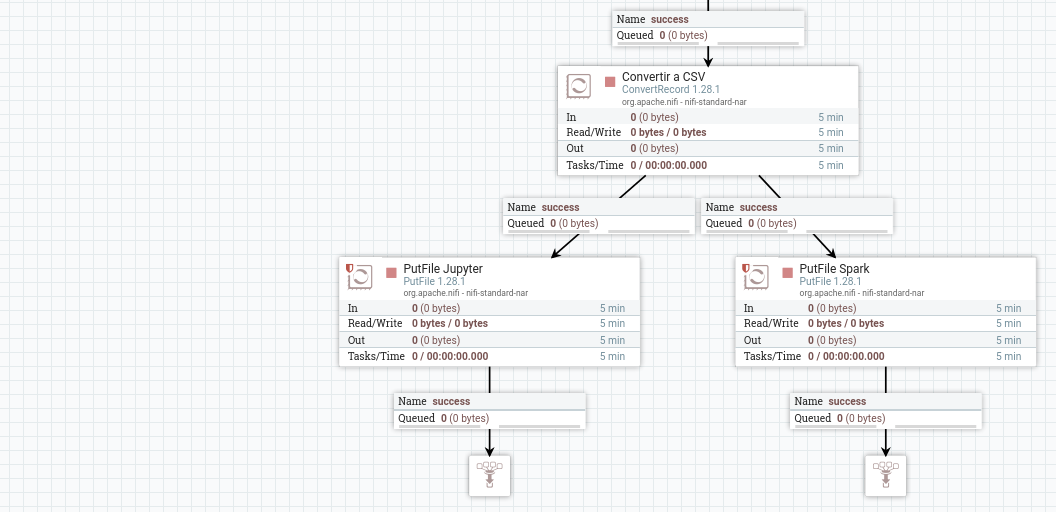
Apache Spark es un motor de procesamiento distribuido que permite el análisis de grandes volúmenes de datos en memoria, proporcionando una gran velocidad y escalabilidad. Jupyter Notebook, por su parte, es una aplicación web que permite crear y compartir documentos que contienen código en vivo, ecuaciones, visualizaciones y texto narrativo.

En este proyecto, Spark se utilizó breve y básicamente a partir del CSV generado por NiFi para llevar a cabo unos análisis básicos, mientras que Jupyter se empleó para generar gráficos y visualizaciones a partir del mismo CSV que utilizó Spark.

# **5. Desarrollo del Flujo de Procesamiento**

## **5.1 Diseño del flujo en NiFi**





La configuración de los procesadores y los controladores se puede encontrar en:

***ANEXO IV: Configuración de los procesadores en NiFi*.**

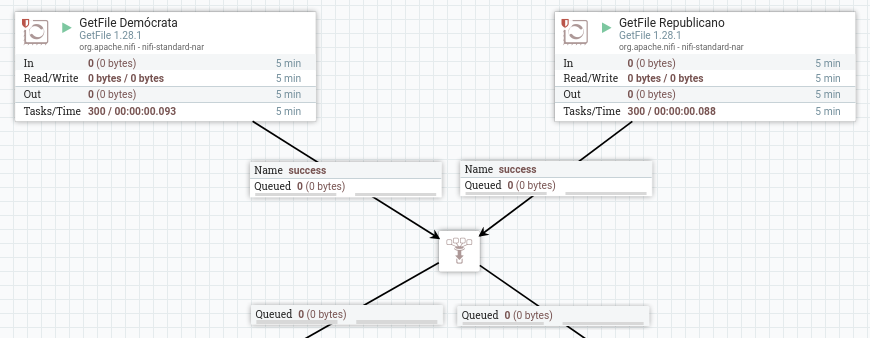
## **5.2 Transformaciones aplicadas**

Cuando jugamos una partida, en cada turno, el generador de votantes guarda cada votante generado como un archivo individual JSON en la carpeta correspondiente del jugador:

* /home/alumno/Escritorio/EEUU\_Elections\_Game/datos\_votantes/jugador\_democrata
* /home/alumno/Escritorio/EEUU\_Elections\_Game/datos\_votantes/jugador\_republicano

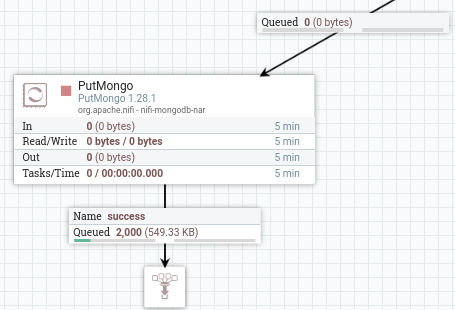


El contenedor de NiFi, tiene mapeadas estas carpetas de la máquina virtual a través de volúmenes para poder ingestar los datos con ayuda de los procesadores **GetFile Demócrata** y **GetFile Republicano:**



Una vez ingestados los datos, el funnel los duplica y divide por 2 senderos, uno para que Mongo los ingeste y otro para que se compacten y los JSON se transformen en un archivo CSV.

**Sendero 1: Ingesta en Mongo:**



Los datos se han almacenado en nuestro Mongo contenedorizado con docker gracias al **procesador PutMongo**, lo cual podemos comprobar:

$ sudo docker exec -it mongodb bash

# mongo mongodb://admin:1234@localhost:27017

> use votantesdb

> db.votantes.count()

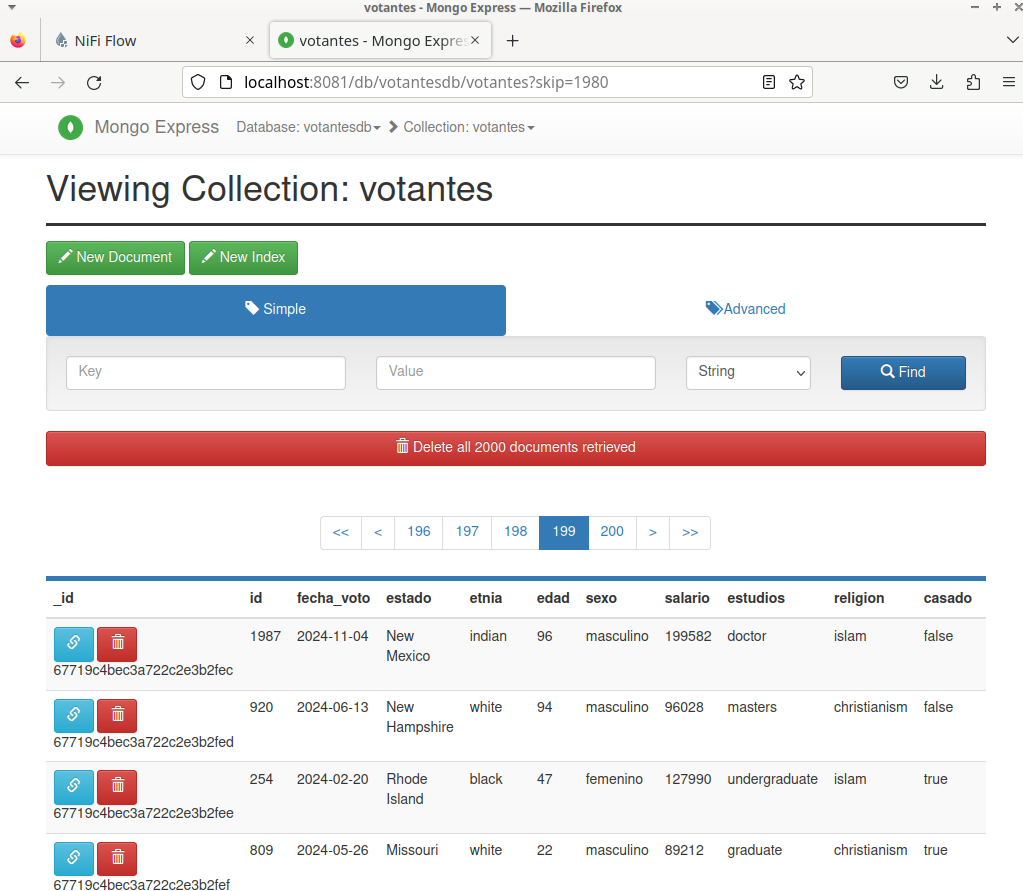
> db.votantes.findOne()



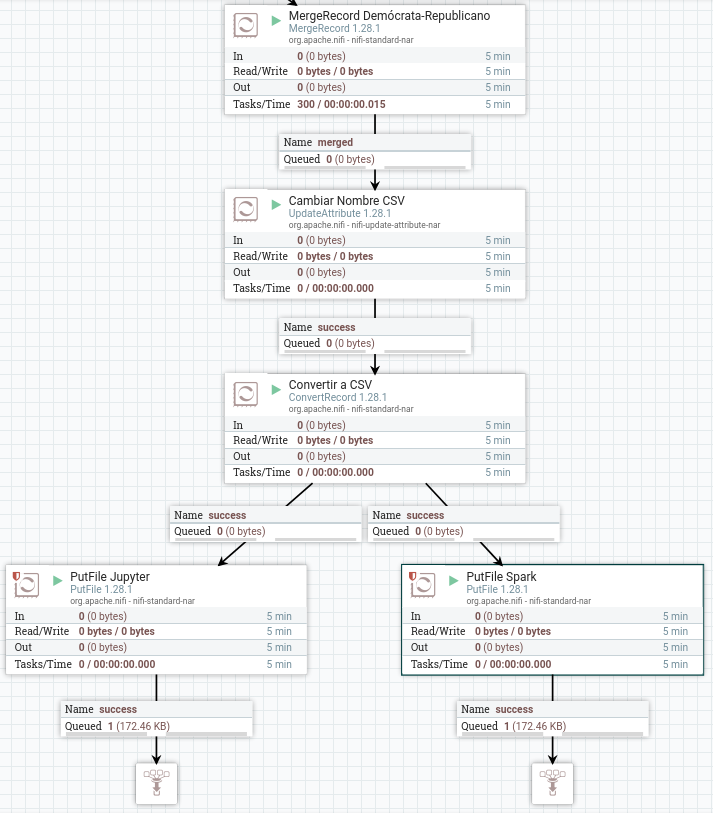
Borrar todos los registros para que no se contamine la siguiente partida:

> db.votantes.deleteMany({})

Si lo preferimos, también podemos visualizar los datos desde la interfaz web que nos proporciona Mongo-Express:



**Sendero 2: Compactación y transformación en CSV:**

****

**MergeRecord Demócrata-Republicano** compacta todos los ficheros individuales en un único archivo y a continuación el procesador **UpdateAttribute: Cambiar Nombre CSV**, cambia el nombre de dicho archivo siguiendo la siguiente expresión:

* votantes\_merged\_${now():format('dd\_MM\_yyyy\_HH\_mm')}**.csv**

Posteriormente, se convierte el archivo a CSV gracias al procesador **ConvertRecord: Convertir a CSV**; y se guarda este archivo CSV en la ruta dedicada para Jupyter, con el procesador **PutFile Jupyter**; y en la ruta dedicada para Spark, con el procesador **PutFile Spark.**

Quedando los datos de la siguiente manera:



# **6. Pruebas y Validaciones**

## **6.1 Pruebas de flujo en NiFi**

Tal y como nos aconsejó en clase el profesor José María, se realizaron pruebas individuales para cada procesador, **activándolos poco a poco y paso a paso**, en vez de todos de golpe, esto ha ayudado a la depuración de errores.

El **sendero 2** que se expuso anteriormente es más complejo de lo que podría ser, esto se debe a que al principio se optó por compactar los ficheros JSON de los votantes de entrada en un único fichero JSON de salida, sin embargo el **procesador PutMongo**, los ingesta más fácilmente cuando son ficheros individuales, por lo que se cambió a posteriori el flujo. Otra opción hubiese sido la inclusión de un **procesador SplitText** en el flujo original con tal de preservar así el archivo JSON compacto y subdividirlo antes de la ingesta en Mongo, esta opción se valoró y se descartó al contar ya con archivos CSV compactos.

## **6.2 Validación de datos exportados e ingesta en MongoDB**

Para verificar que los documentos se insertaron correctamente la validación del JSON ingestado en MongoDB se realizó ejecutando comandos como:

> db.votantes.count()

> db.votantes.findOne()

Una partida normal corta de 8 turnos sin acciones especiales son 2000 votantes, esto fue comprobado y constatado.

Además, se comprobó que los datos almacenados cumplían con el formato esperado, asegurando que los campos estuvieran correctamente mapeados y que las fechas se presentaran en el formato **YYYY-MM-DD**. Aunque las fechas estén almacenadas en MongoDB como strings, este formato nos permite realizar consultas sin realizar una conversión al formato fecha de MongoDB; y por lo tanto, consultas en MongoQL como la siguiente funcionan adecuadamente:

> db.votantes.find({fecha\_voto: { $gte: "2024-11-05" }})

Finalmente, se ejecutaron pruebas de borrado mediante comandos como:

> db.votantes.deleteMany({})

Lo que permitió confirmar que los datos podían ser eliminados completamente antes de iniciar una nueva partida, asegurando así un entorno limpio y listo para nuevos datos.

En cuanto a la validación del CSV generado, se revisaron los archivos compactados para confirmar que incluían todos los registros generados durante el proceso (2000). También se evaluó la estructura del archivo CSV resultante, verificando la presencia de encabezados adecuados y la consistencia en los campos.

# **7. Resultados y Análisis de Datos**

## **7.1 Consultas en MongoDB**

Tras la instalación de MongoDB con Docker, (*ANEXO III: Instalación de MongoDB con Docker*), Algunas consultas que pueden utilizar ambos jugadores son:

$ sudo docker exec -it mongodb bash

# mongo mongodb://admin:1234@localhost:27017

> use votantesdb

1. **Número total de votantes**

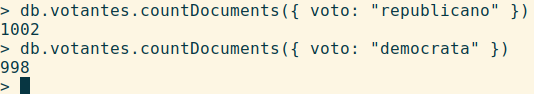
db.votantes.count()



1. **Número de votantes por partido**

db.votantes.countDocuments({ voto: "republicano" })

db.votantes.countDocuments({ voto: "democrata" })



1. **Número de votantes por estado y partido**

db.votantes.aggregate([

{ $group: {

\_id: { estado: "$estado", voto: "$voto" },

total: { $sum: 1 }

} },

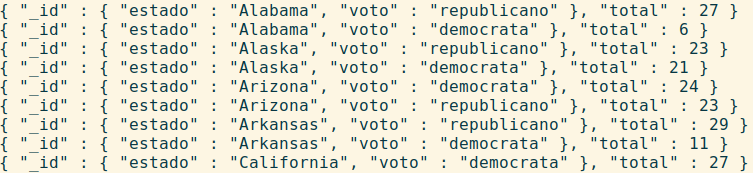
{ $sort: {

"\_id.estado": 1, // Orden alfabético por estado

total: -1 // Orden de mayor a menor en votos

} }

])



1. **Distribución salarial promedio por partido**

db.votantes.aggregate([

{ $group: {

\_id: "$voto",

salario\_promedio: { $avg: "$salario" },

total\_votantes: { $sum: 1 }

} },

{ $sort: { salario\_promedio: -1 } }

])



1. **Distribución por sexo y partido**

db.votantes.aggregate([

{ $group: {

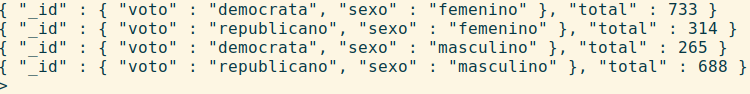
\_id: { voto: "$voto", sexo: "$sexo" },

total: { $sum: 1 }

} },

{ $sort: { "\_id.sexo": 1, "\_id.voto": 1 } }

])



1. **Distribución por etnia y partido**

db.votantes.aggregate([

{ $match: {

etnia: { $in: ["white", "black", "hispanic", "asian", "indian", "illegal"] }

} },

{ $group: {

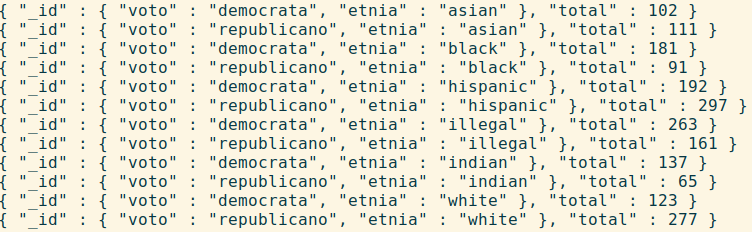
\_id: { voto: "$voto", etnia: "$etnia" },

total: { $sum: 1 }

} },

{ $sort: { "\_id.etnia": 1, "\_id.voto": 1 } }

])



1. **Votantes por rango de edad y partido**

db.votantes.aggregate([

{ $match: { edad: { $gte: 18, $lte: 99 } } },

{ $group: {

\_id: {

voto: "$voto",

grupo\_edad: {

$cond: [

// Rangos de edad

{ $lt: ["$edad", 30] }, "Jóvenes",

{ $cond: [

{ $and: [ { $gte: ["$edad", 31] }, { $lte: ["$edad", 50] } ] },

"Adultos",

"Mayores"

] }

]

}

},

total: { $sum: 1 }

} },

{ $addFields: {

orden\_grupo\_edad: {

$switch: {

branches: [

{ case: { $eq: ["$\_id.grupo\_edad", "Jóvenes"] }, then: 1 },

{ case: { $eq: ["$\_id.grupo\_edad", "Adultos"] }, then: 2 },

{ case: { $eq: ["$\_id.grupo\_edad", "Mayores"] }, then: 3 }

],

default: 4

}

}

} },

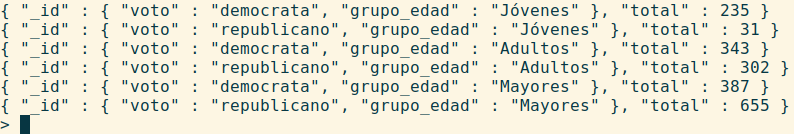
{ $sort: { "orden\_grupo\_edad": 1, "\_id.voto": 1 } }, // Ordena por grupo de edad y luego por voto

{ $project: {

"orden\_grupo\_edad": 0 // Excluye el campo orden\_grupo\_edad

} }

])



1. **Votantes por nivel de estudios y partido**

db.votantes.aggregate([

{ $match: { estudios: {

$in: ["high school", "undergraduate", "graduate", "masters", "doctor"]

} } },

{ $group: {

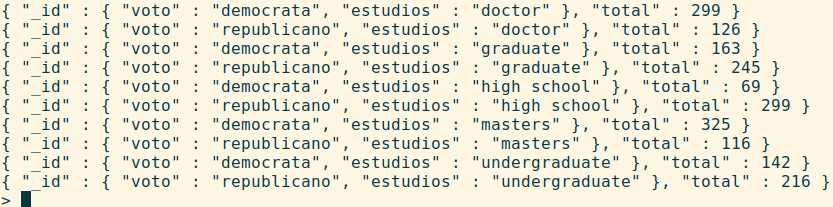
\_id: { voto: "$voto", estudios: "$estudios" },

total: { $sum: 1 }

} },

{ $sort: { "\_id.estudios": 1, "\_id.voto": 1 } }

])



1. **Distribución por religión y partido**

db.votantes.aggregate([

{ $match: { religion: { $in:

["judaism", "islam", "christianism", "hinduism", "atheism", "other"]

} } },

{ $group: {

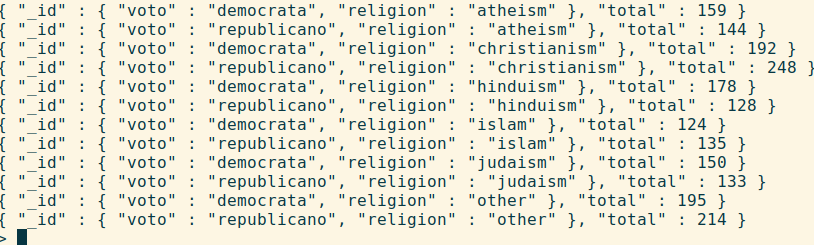
\_id: { voto: "$voto", religion: "$religion" },

total: { $sum: 1 }

} },

{ $sort: { "\_id.religion": 1, "\_id.voto": 1 } }

])



1. **Distribución por estado civil, hijos y partido**

db.votantes.aggregate([

{ $group: {

\_id: { voto: "$voto", casado: "$casado", hijos: "$hijos" },

total: { $sum: 1 }

} },

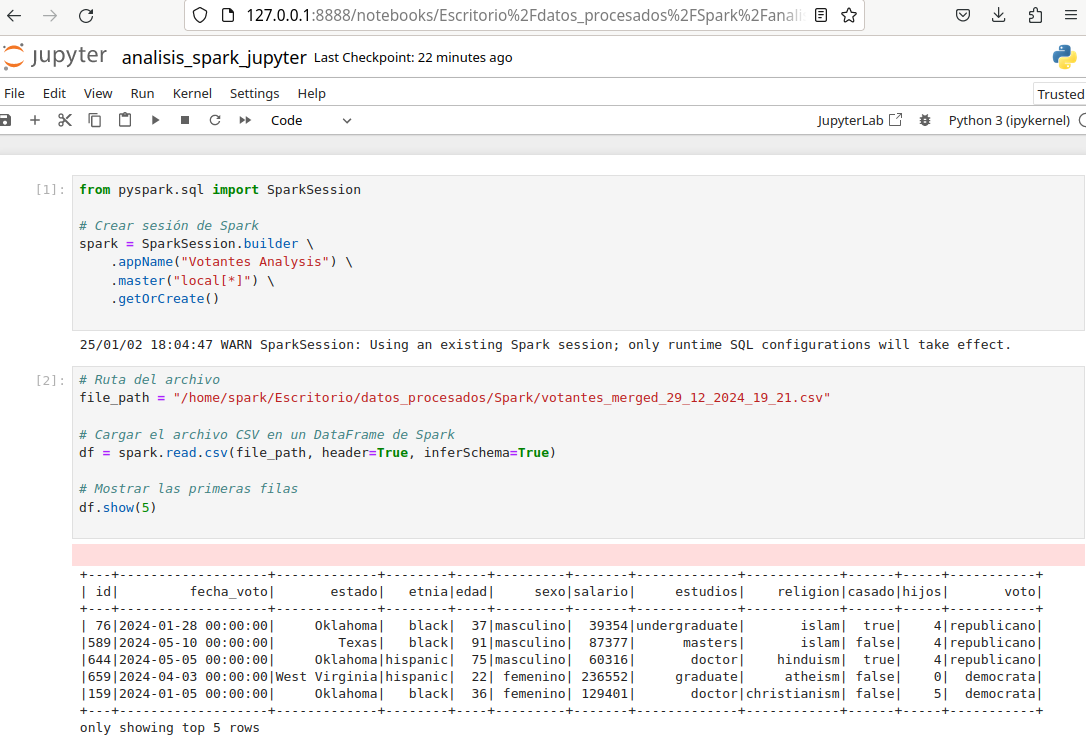
{ $sort: { "\_id.casado": 1, "\_id.hijos": 1, "\_id.voto": 1 } }

])

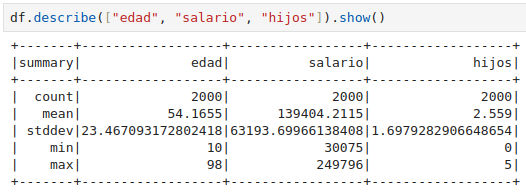


## **7.2 Visualización de resultados en Spark/Jupyter**

Después de seguir los pasos del: *ANEXO V: Puesta en marcha de Spark/Jupyter*, podremos utilizar los datos procesados para realizar algunas visualizaciones:

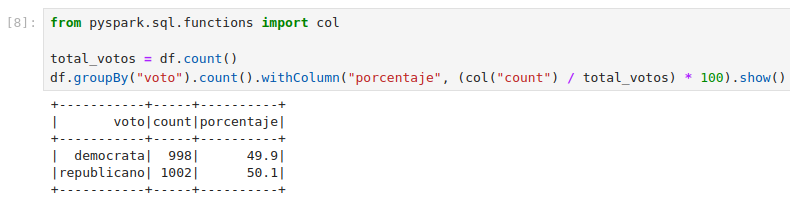


Por ejemplo, podemos ver estadísticas de los datos numéricos:

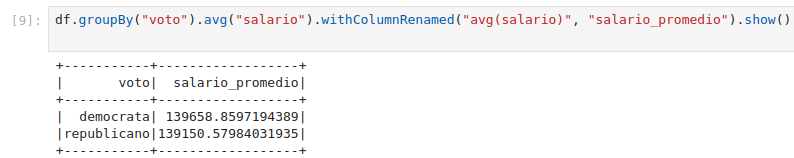


Algunos análisis básicos que podemos realizar con Spark son los siguientes:

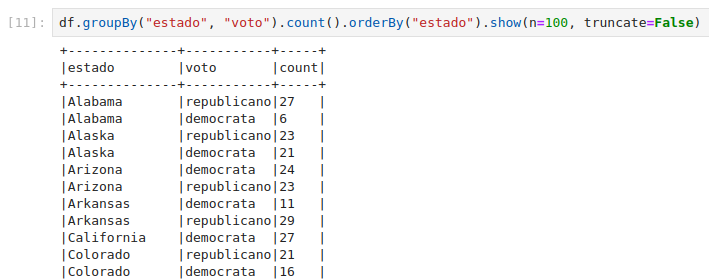
1. **Porcentaje de votos por partido**



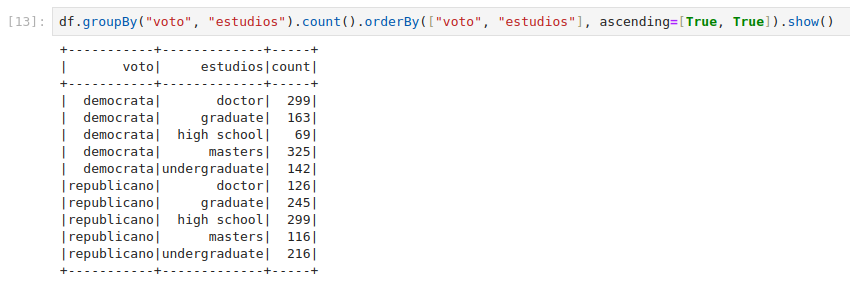
1. **Promedio salarial por partido**



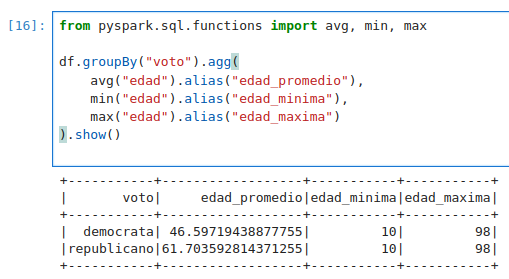
1. **Distribución de votos por estado**



1. **Nivel educativo de los votantes por partido**

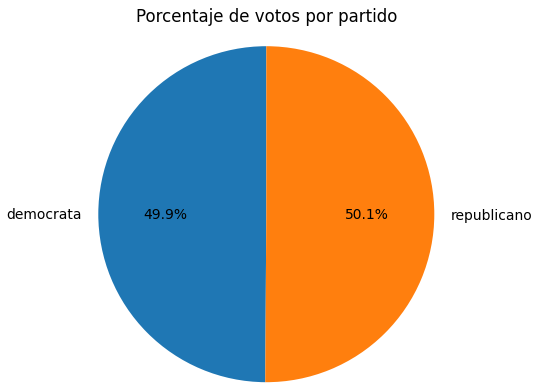


1. **Análisis de edades según el voto** (desconozco por qué la mínima es 10)

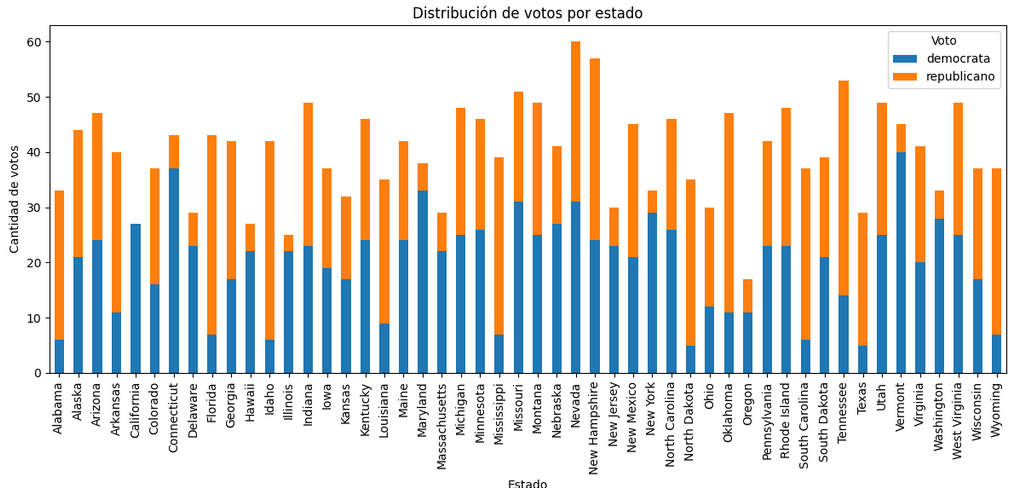


Tras instalar matplotlib, podemos ver los datos en gráficos, una representación visual siempre será más agradable a la vista que la interfaz de texto:

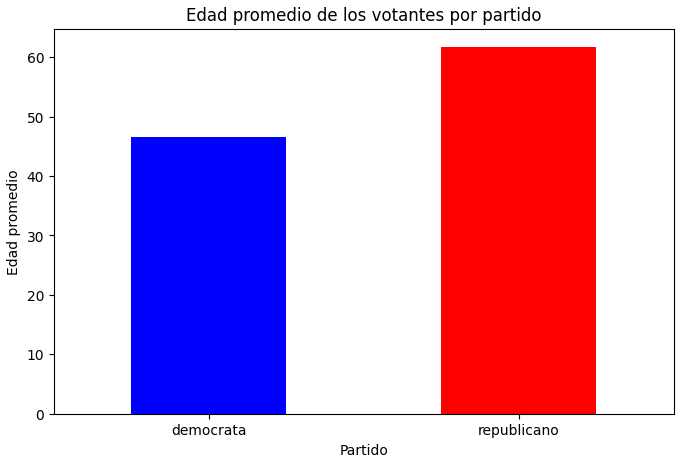
1. **Porcentaje de votos por partido**



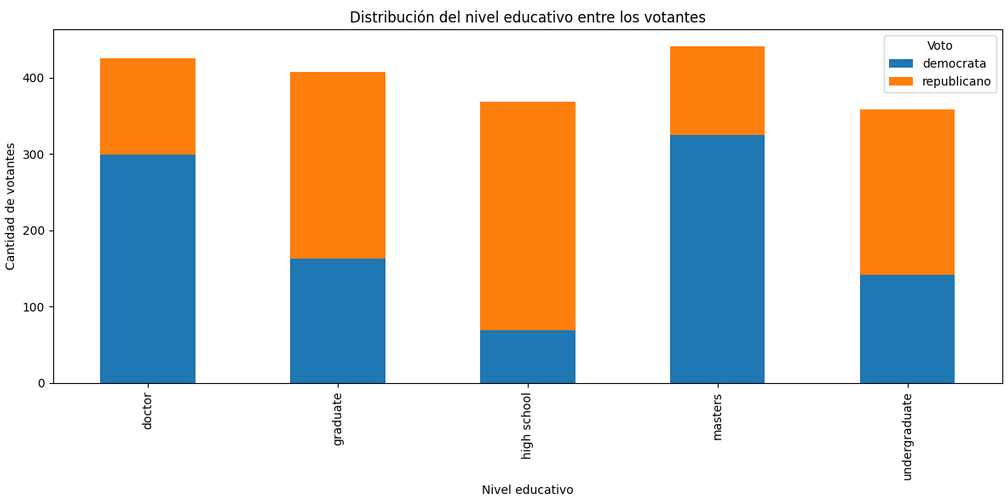
1. **Distribución de votos por estado**



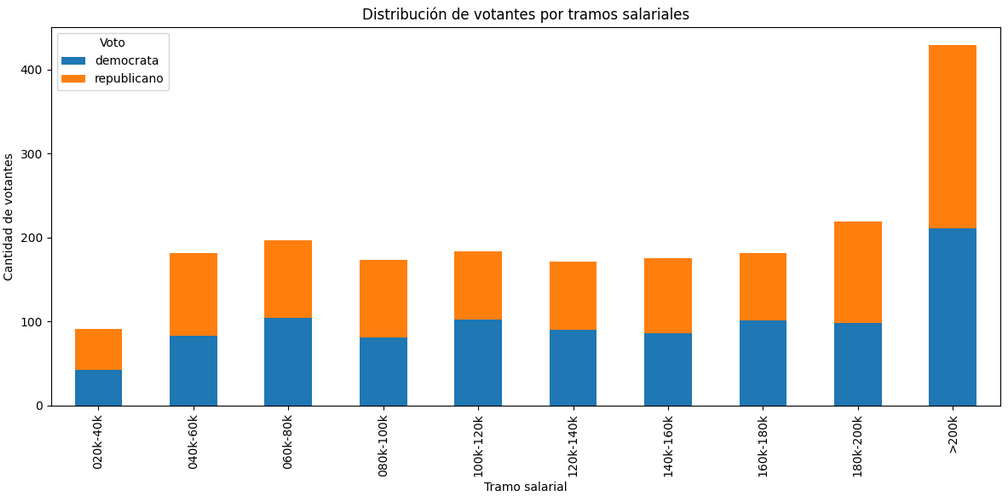
1. **Edad promedio de los votantes por partido**



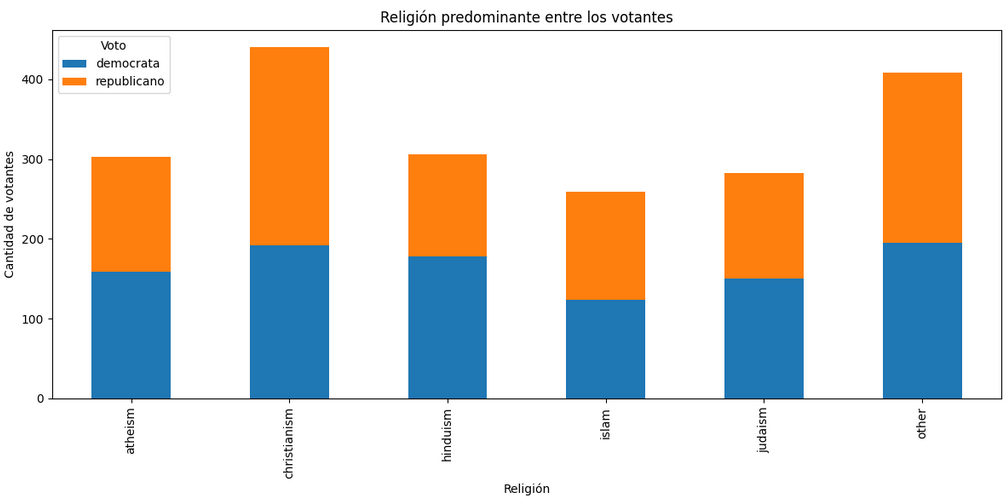
1. **Distribución del nivel educativo entre los votantes**



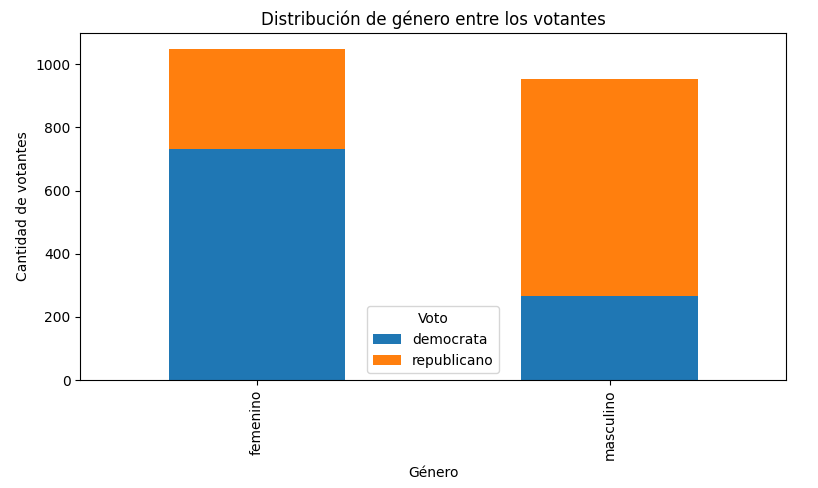
1. **Distribución de votantes por tramos salariales**



1. **Religión predominante entre los votantes**



1. **Distribución de género entre los votantes**



1. **Evolución de los votantes en función de la fecha del juego**



# **8. Conclusiones, Futuras Mejoras y Referencias**

## **8.1 Resumen de logros**

Redacto esta línea satisfecho por el trabajo realizado durante estas Navidades, que me ha permitido refrescar conceptos de Python a la hora de realizar el generador de votos y la dinámica de juego, contenedorizar NiFi, MongoDB y Mongo-Express; así como utilizarlos de forma útil en el proyecto para procesar, almacenar y consultar datos. Además se ha utilizado Spark y Jupyter, apenas vistos hasta la fecha en clase; ambos han permitido unas visualizaciones y consultas muy básicas pero útiles a la hora de jugar. Es por ello que estoy contento con el resultado final del proyecto: un simulador de elecciones realista y funcional que integra múltiples tecnologías.

## **8.2 Lecciones aprendidas**

A lo largo del desarrollo de este proyecto se han adquirido muchos conocimientos prácticos sobre diversas tecnologías de procesamiento y análisis de datos; y lo que es más, su contenedorización, coordinación e integración para que funcionasen en conjunto.

## **8.3 Propuestas de mejora**

Este proyecto se inició con el objetivo de ser un juego para 2 jugadores en el que compitiesen investigando con herramientas de Big Data vistas en clase para ganar las elecciones y parece haber acabado como una especie de simulador de elecciones en el que ambos jugadores crean un escenario en el que se desarrollan eventos canon que influyen en ellas y sobre el que después pueden investigar. Aunque esto en sí es una especie de juego, me habría gustado más que fuese todo más fluido, que existiese una interconexión entre los componentes que eliminase la fricción que existe actualmente a la hora de investigar.

Este juego ha intentado ser fidedigno a la realidad, una propuesta de mejora podría ser aleatorizar los datos de los modelos demócrata y republicano, a través de los cuales se forman los porcentajes de los votos de los ciudadanos, de forma que no estén fijos para cada partida, esto aseguraría una "diversión sin límites" ya que no sabríamos en cada partida qué está sucediendo hasta que no lo investiguemos con las herramientas propuestas anteriormente.

## **8.4 Referencias**

Apuntes, material de clase y chatGPT.

# **ANEXOS**

## **ANEXO I: Instalación de Docker en Ubuntu 22.04.2 LTS**

Seguiremos los pasos de la página oficial de Docker.

**URL**: <https://docs.docker.com/engine/install/ubuntu/>

**1. Set up del repositorio apt de Docker:**

**# Add Docker's official GPG key**

$ sudo apt-get update

$ sudo apt-get install ca-certificates curl

$ sudo install -m 0755 -d /etc/apt/keyrings

$ sudo curl -fsSL https://download.docker.com/linux/ubuntu/gpg -o /etc/apt/keyrings/docker.asc

$ sudo chmod a+r /etc/apt/keyrings/docker.asc

**# Add the repository to Apt sources**

$ echo \

"deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docker.asc] https://download.docker.com/linux/ubuntu \

$(. /etc/os-release && echo "$VERSION\_CODENAME") stable" | \

sudo tee /etc/apt/sources.list.d/docker.list > /dev/null

$ sudo apt-get update

**2. Instalar los paquetes de Docker:**

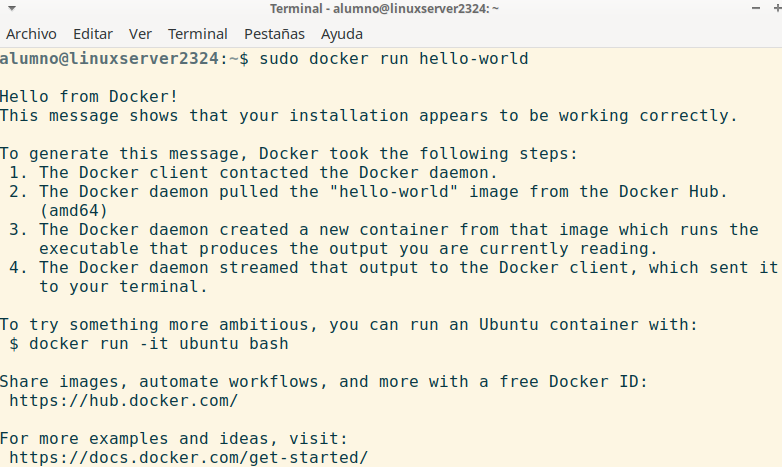
$ sudo apt-get install docker-ce docker-ce-cli containerd.io docker-buildx-plugin docker-compose-plugin

**3. Verificar la instalación con la imagen hello-world**

$ sudo docker run hello-world

**4. Habilitar docker al inicio**

$ sudo systemctl enable docker



## **ANEXO II: Instalación de NiFi con Docker**

Versiones disponibles de NiFi: <https://hub.docker.com/r/apache/nifi/tags>

Vamos a utilizar Docker Compose, **docker-compose.yml:**

services:

nifi:

image: apache/nifi:1.28.1

container\_name: nifi

network\_mode: host

ports:

- "8080:8080" # Puerto de NiFi

volumes:

- /home/alumno/Escritorio/EEUU\_Elections\_Game/datos\_votantes/jugador\_democrata:/opt/nifi/input/democrata

- /home/alumno/Escritorio/EEUU\_Elections\_Game/datos\_votantes/jugador\_republicano:/opt/nifi/input/republicano

- /home/alumno/Escritorio/EEUU\_Elections\_Game/datos\_procesados:/opt/nifi/output

environment:

- NIFI\_WEB\_HTTP\_PORT=8080

restart: always

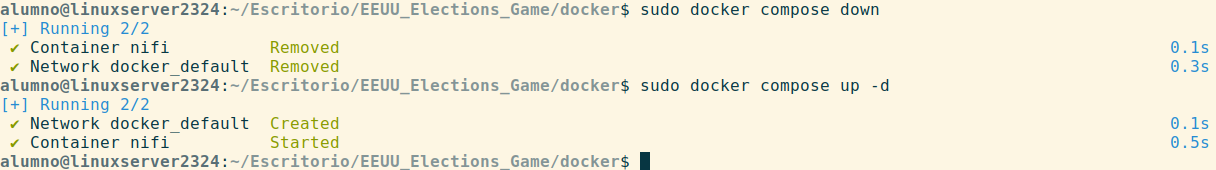
$ sudo docker compose down

$ sudo docker compose up

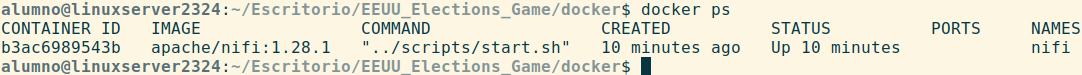


Una opción es correrlo con la opción detached, de forma que no se nos quede la consola bloqueada.

$ sudo docker compose up -d



$ docker ps



Verificamos los volúmenes presentes en el contenedor de NiFi, para ver que se encuentran los archivos:

$ sudo docker exec -it nifi bash

$ ls /opt/nifi/input/democrata | head -n 10

$ ls /opt/nifi/input/republicano | head -n 10

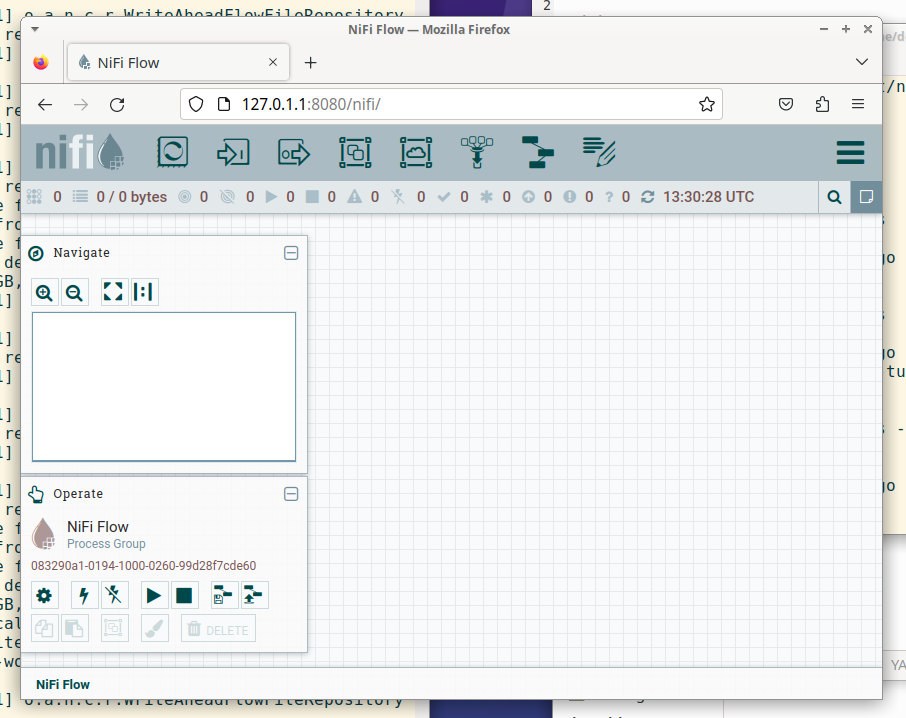
$ ls /opt/nifi/output

$ exit



Y nos conectamos a través del navegador por el puerto 8080:

http://127.0.1.1:8080/nifi/



Parar e iniciar el contenedor:

Opción 1, desde donde se encuentre nuestro docker-compose.yml:

$ docker compose up

Opción 2 con el nombre del contenedor:

$ docker stop nifi

$ docker start nifi

Ver los logs:

$ docker logs -f nifi

## **ANEXO III: Instalación de MongoDB con Docker**

Versiones disponibles de MongoDB: <https://hub.docker.com/_/mongo/tags>

Versiones disponibles de Mongo-Express: <https://hub.docker.com/_/mongo-express/tags>

Vamos a utilizar Docker Compose, **docker-compose.yml:**

services:

mongo:

image: mongo:4.4

container\_name: mongodb

network\_mode: host # Compartir la red del host

ports: # Puerto por defecto de MongoDB

- "27017:27017"

volumes: # Persistencia de datos

- /home/alumno/Escritorio/EEUU\_Elections\_Game/datos\_procesados/MongoDB:/data/db

environment:

MONGO\_INITDB\_ROOT\_USERNAME: admin

MONGO\_INITDB\_ROOT\_PASSWORD: 1234

restart: always

mongo-express:

image: mongo-express

container\_name: mongo-express

network\_mode: host

ports:

- "8081:8081" # Mapea el puerto 8081 del contenedor al host

environment:

ME\_CONFIG\_MONGODB\_ADMINUSERNAME: admin

ME\_CONFIG\_MONGODB\_ADMINPASSWORD: 1234

ME\_CONFIG\_MONGODB\_SERVER: localhost

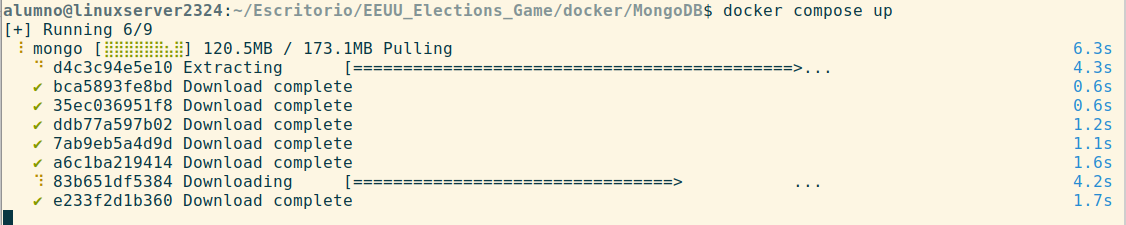
depends\_on:

- mongo

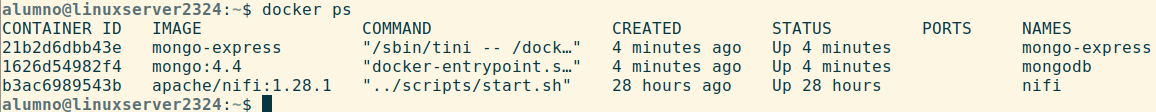
restart: always

$ sudo docker compose down

$ sudo docker compose up

****

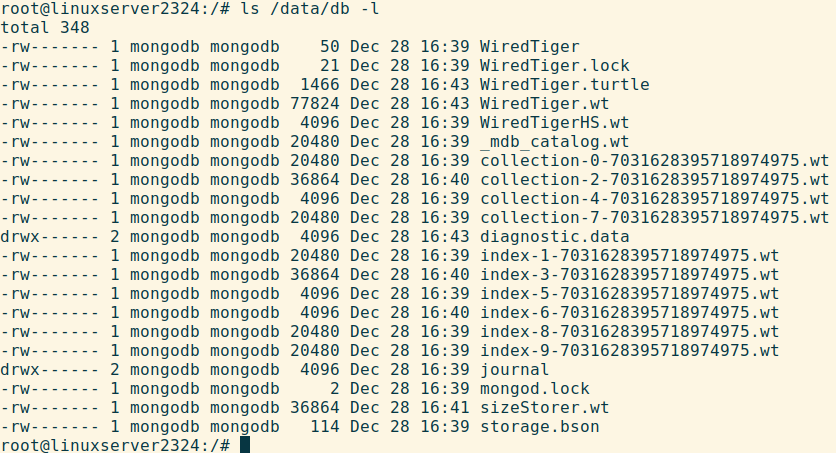
$ docker ps

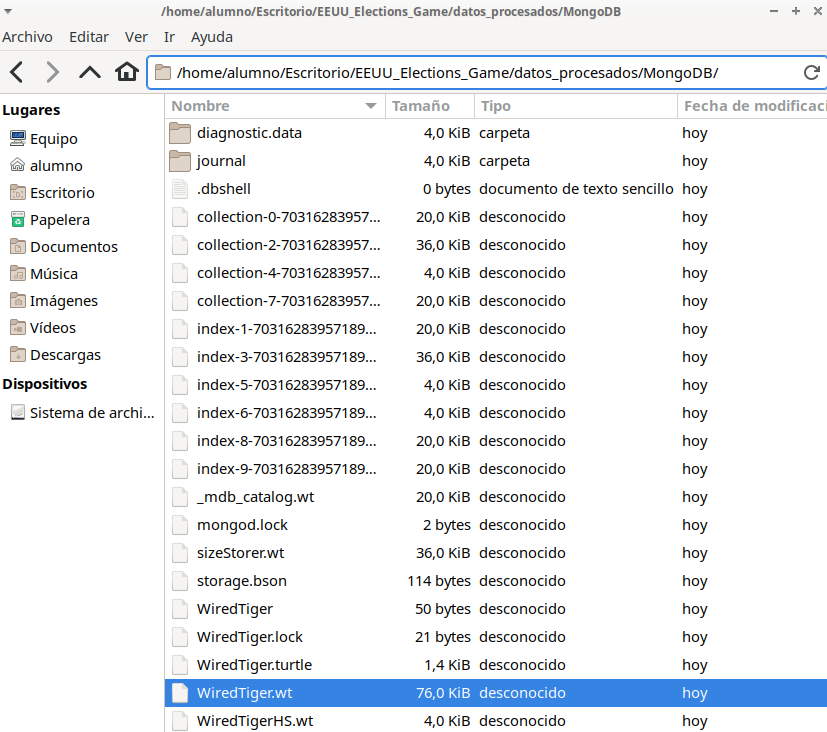


Verificamos los volúmenes presentes en el contenedor de Mongo, para ver que se encuentran los archivos:

$ sudo docker exec -it mongodb bash

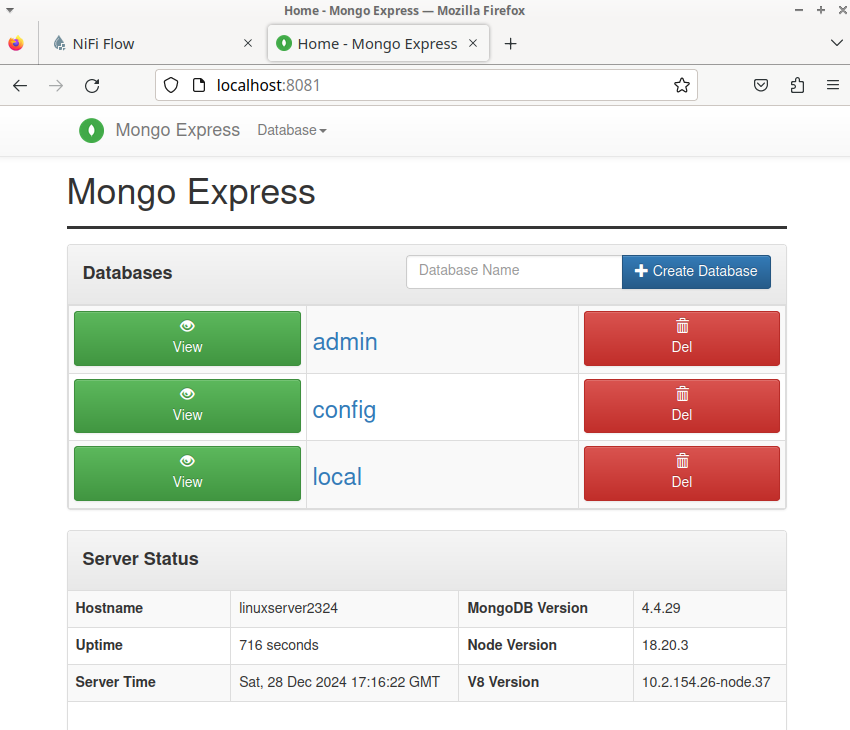
# ls /data/db -l



Vemos que coincide con el directorio que ha sido mapeado en la MV:  
  


Vamos a probar a conectarnos a mongo-express a través del navegador, aunque hemos configurado antes el usuario y la contraseña, se nos muestra en la terminal unos valores distintos, admin:pass





Parar e iniciar los contenedores:

Opción 1, desde donde se encuentre nuestro docker-compose.yml:

$ docker compose up

Opción 2 con el nombre de los contenedores:

$ docker stop mongodb mongo-express

$ docker start mongodb mongo-express

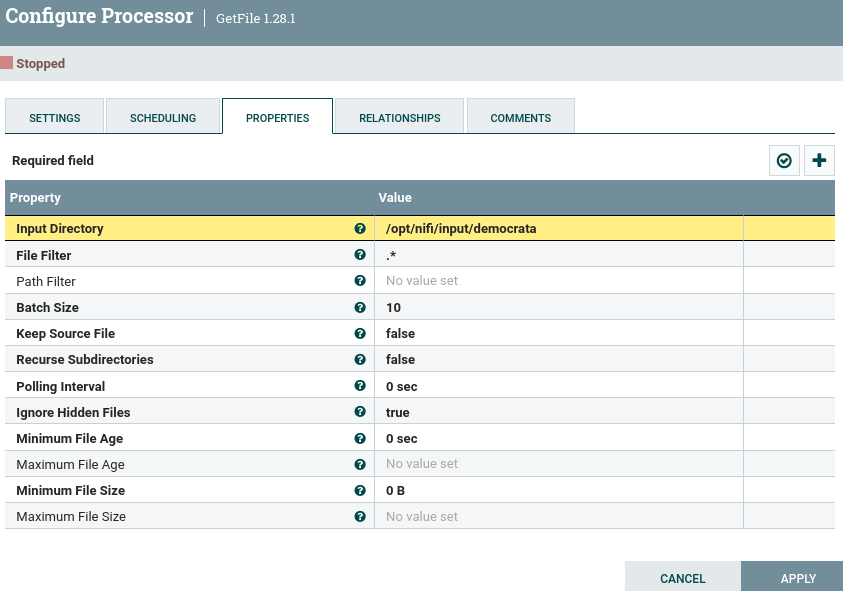
Ver los logs:

$ docker logs -f mongodb

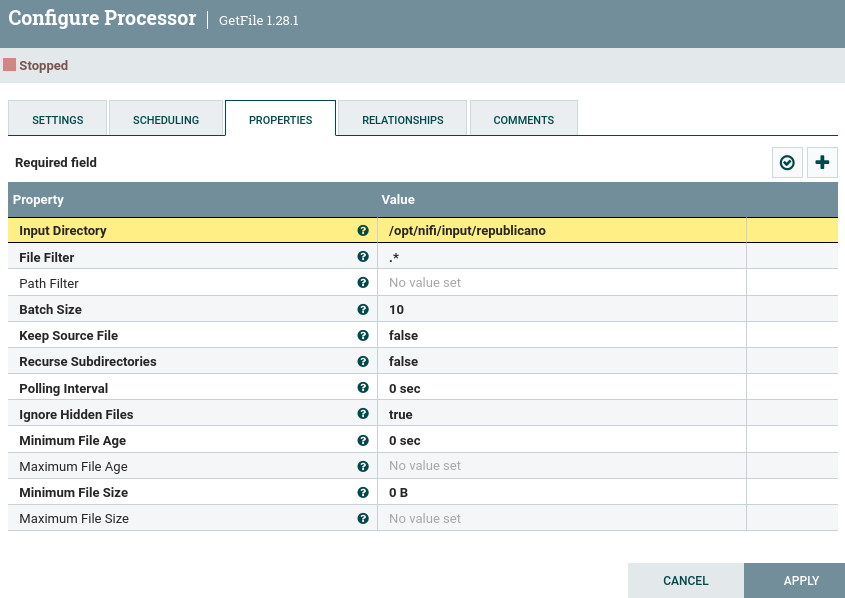
$ docker logs -f mongo-express

## **ANEXO IV: Configuración de los procesadores en NiFi**

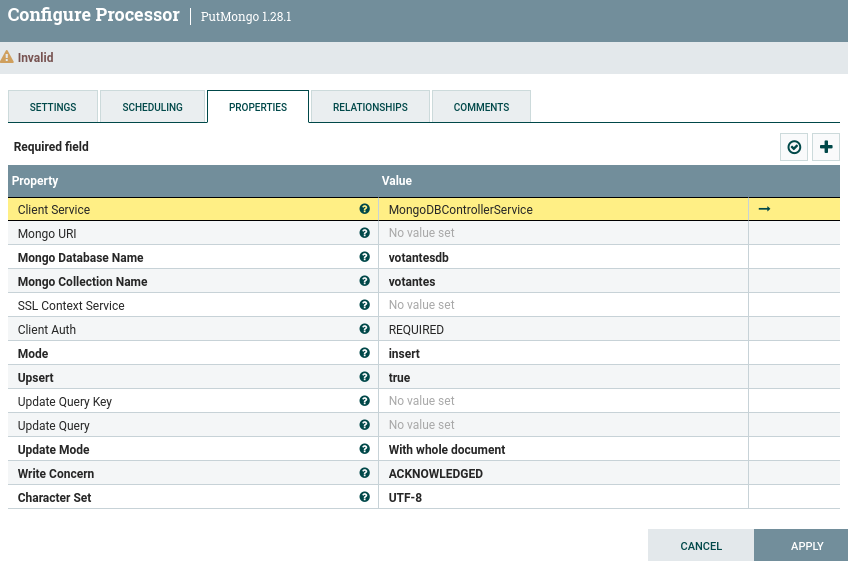
**GetFile 1.28.1: Demócrata**

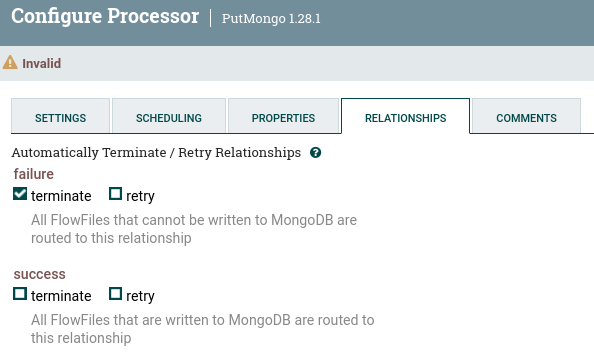


**GetFile 1.28.1: Republicano**



**PutMongo 1.28.1: PutMongo**

****

****

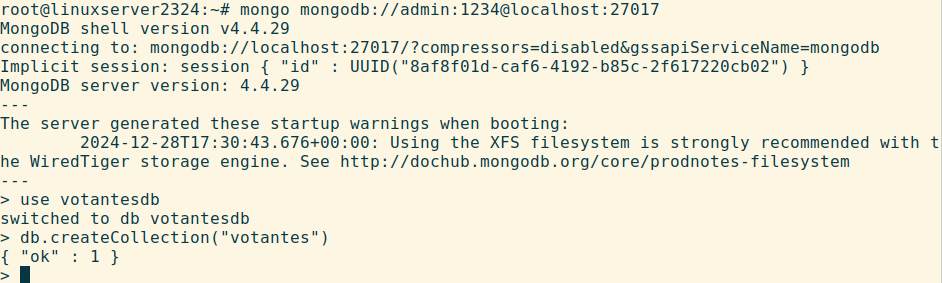
Es necesario que la base de datos votantesdb y la colección existan de antes:

$ sudo docker exec -it mongodb bash

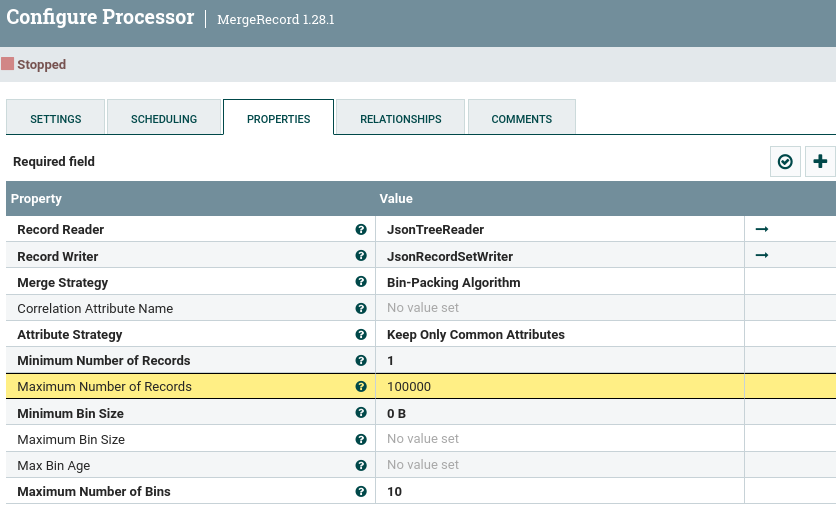
# mongo mongodb://admin:1234@localhost:27017

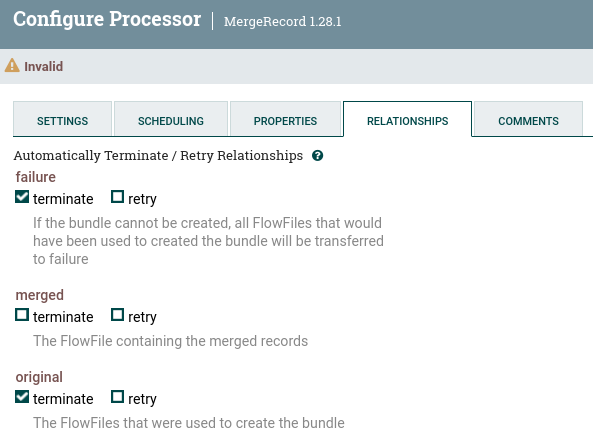
> use votantesdb

> db.createCollection("votantes")

****

**MergeRecord 1.28.1: Demócrata-Republicano**

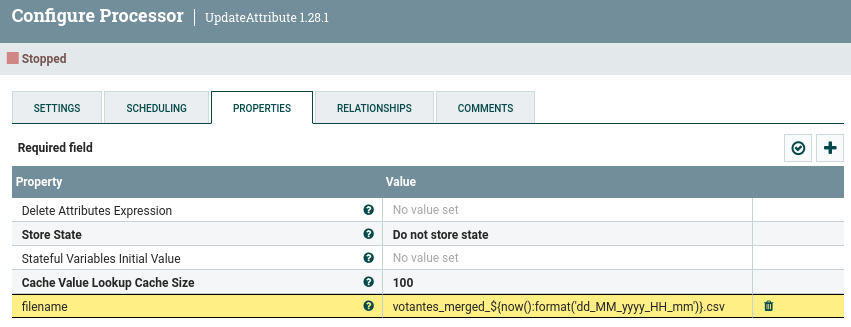




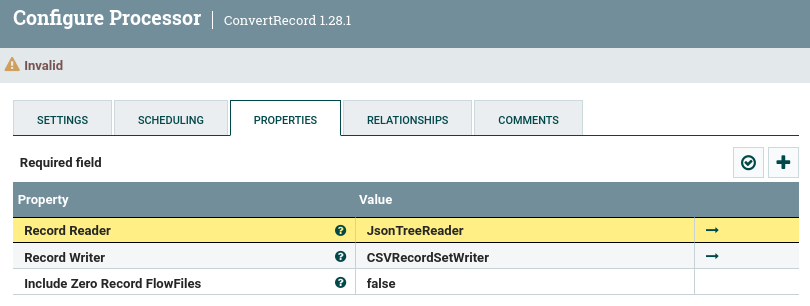
**UpdateAttribute 1.28.1: Cambiar Nombre CSV**

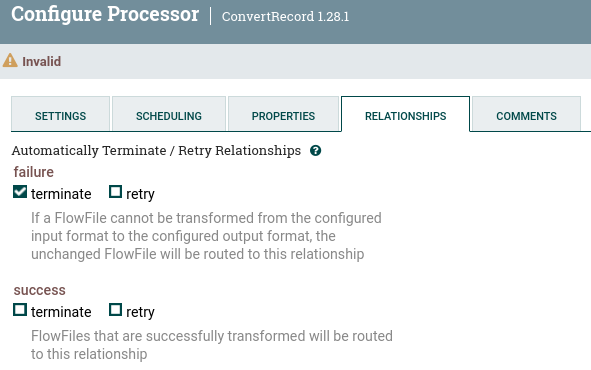
Hay que añadir un atributo:

**filename**: votantes\_merged\_${now():format('dd\_MM\_yyyy\_HH\_mm')}.csv

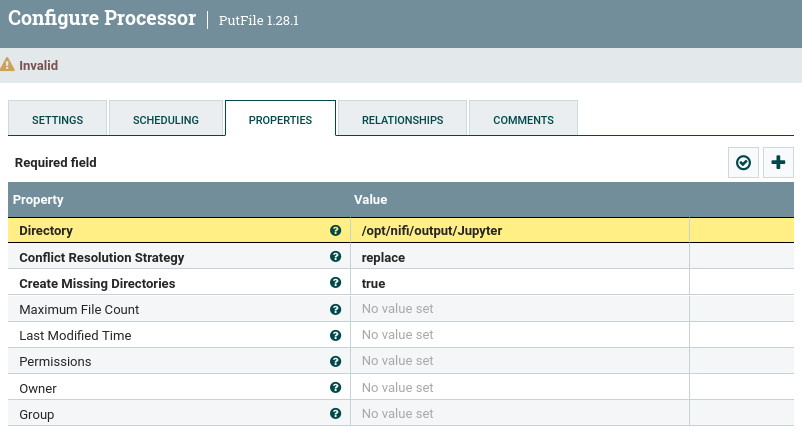


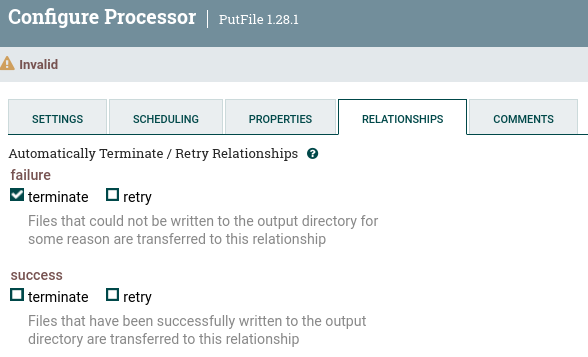
**ConvertRecord 1.28.1: Convertir a CSV**

****

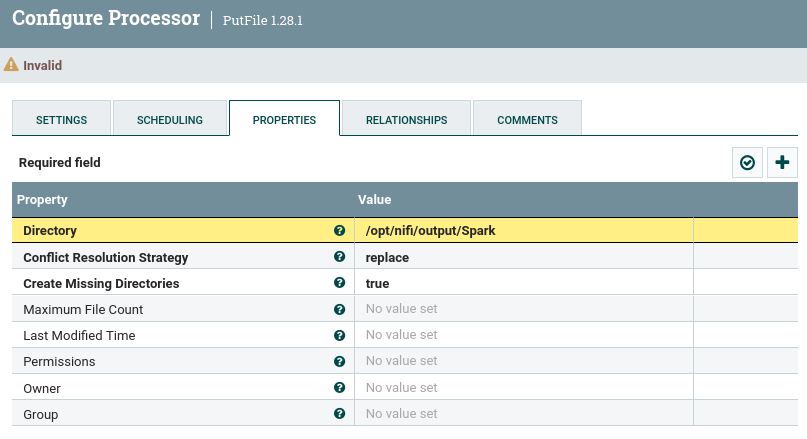
****

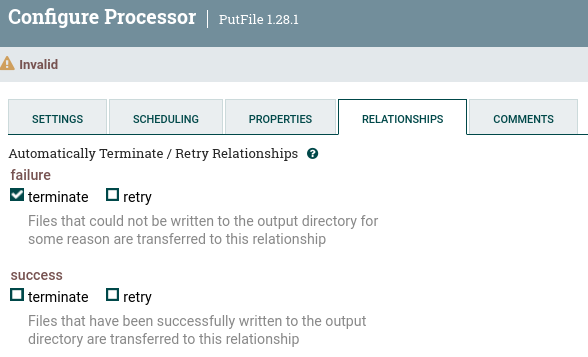
**PutFile 1.28.1: PutFile Jupyter**

****

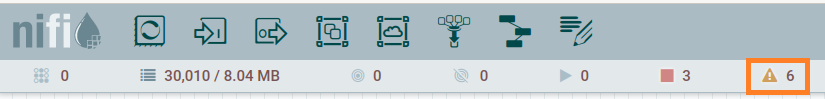
****

**PutFile 1.28.1: PutFile Spark**

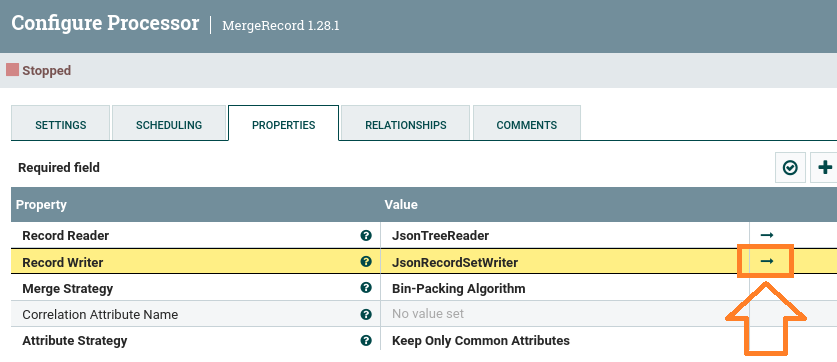
****



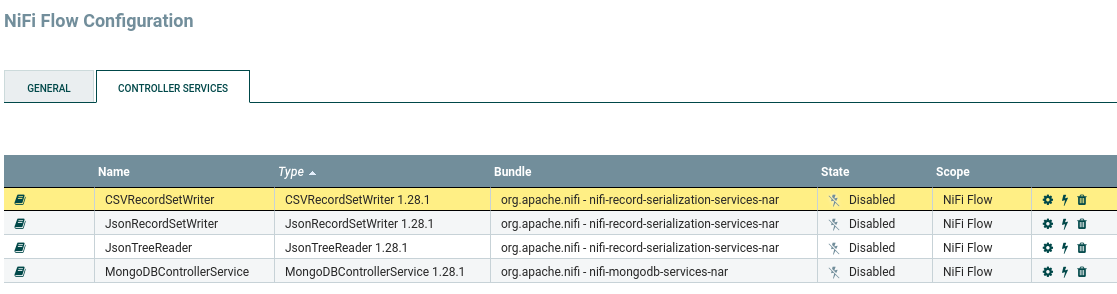
Vemos que nos aparecen varios warnings de "Invalid Components" que podemos solucionar desde **NiFi Flow Configuration**, activando los controladores.



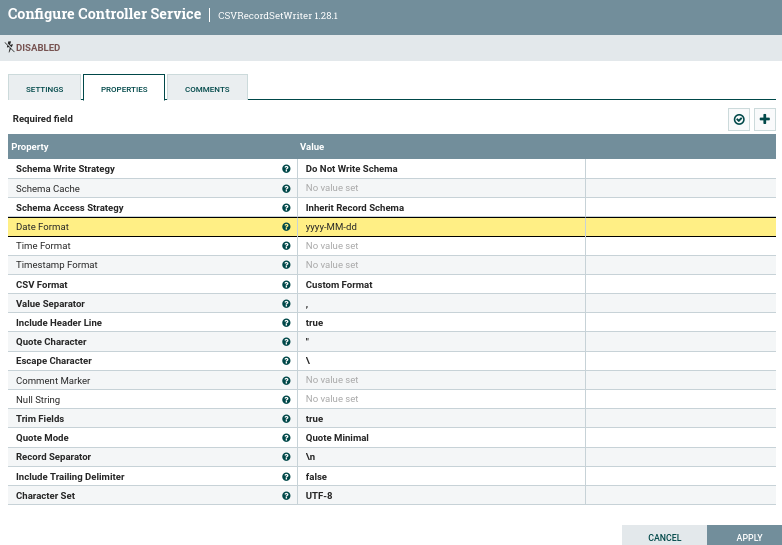
Para acceder a **NiFi Flow Configuration**, nos vamos por ejemplo al procesador MergeRecord y hacemos click en la flechita:



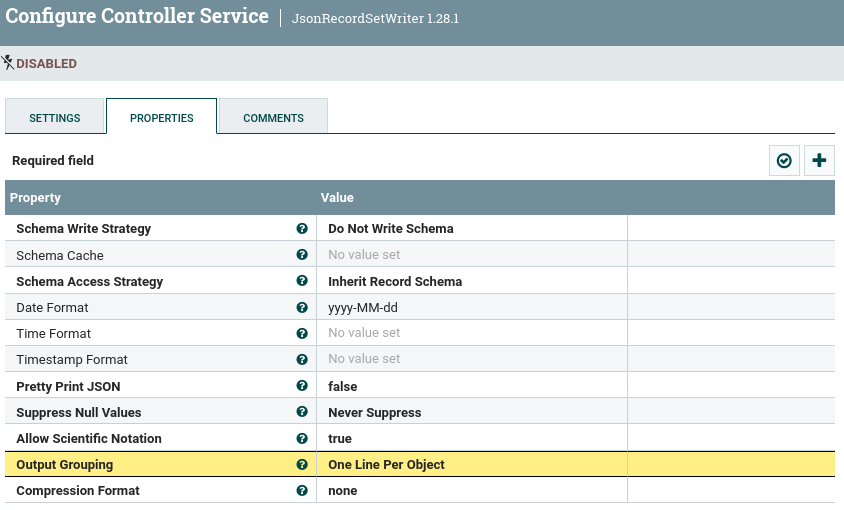
Antes de activarlos, vamos a configurar algunos de ellos con la ruedecita de configuración:



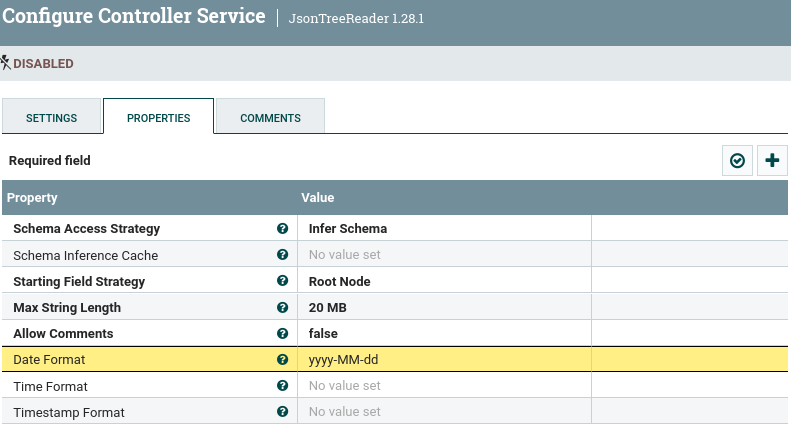
**CSVRecordSetWriter:**



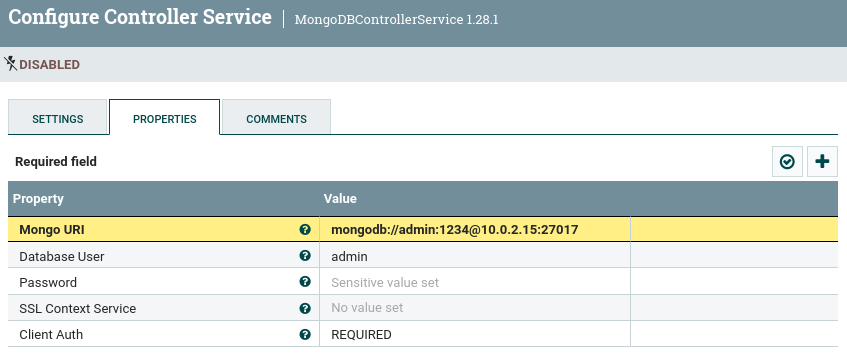
**JsonRecordSetWriter:**



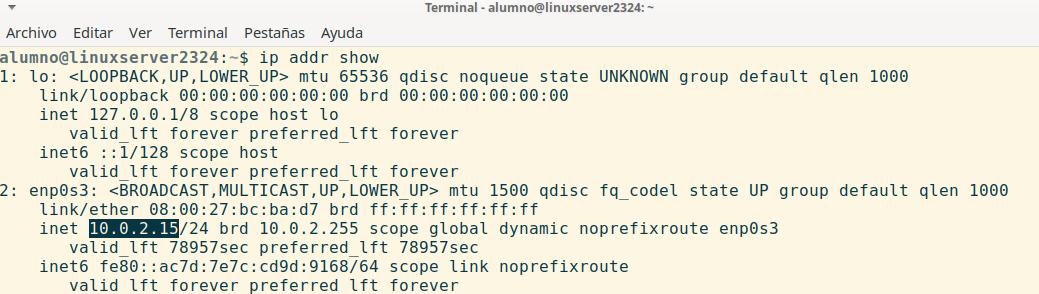
**JsonTreeReader:**



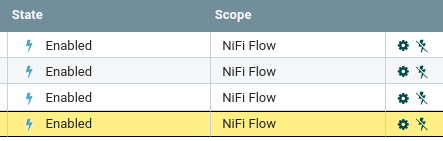
Y el **MongoDBControllerService**, lo dejamos de la siguiente manera (admin:1234):



Cómo estamos contenedorizando, nos interesa la siguiente ip:



Para activarlos es necesario darle al **rayito** a la derecha de la rueda de configuración en **NiFi Flow Configuration**, quedando de la siguiente manera:

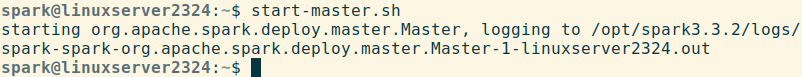


## **ANEXO V: Puesta en marcha de Spark/Jupyter**

Para visualizar los datos en Spark y Jupyter, primero tenemos que loguearnos como el usuario spark (spark/spark) en nuestra MV, recordamos que tanto Jupyter como Spark se encuentran instalados previamente en esta máquina:

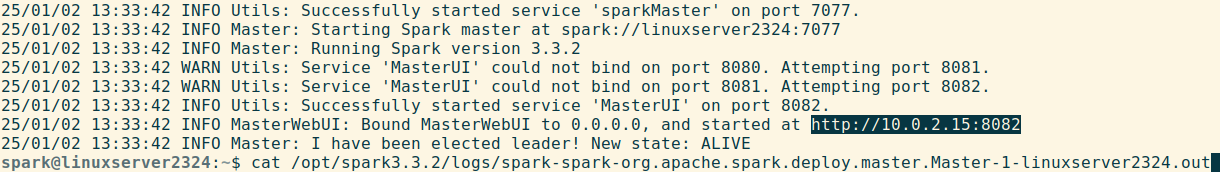
alumno@linuxserver2324:~$ sudo su - spark

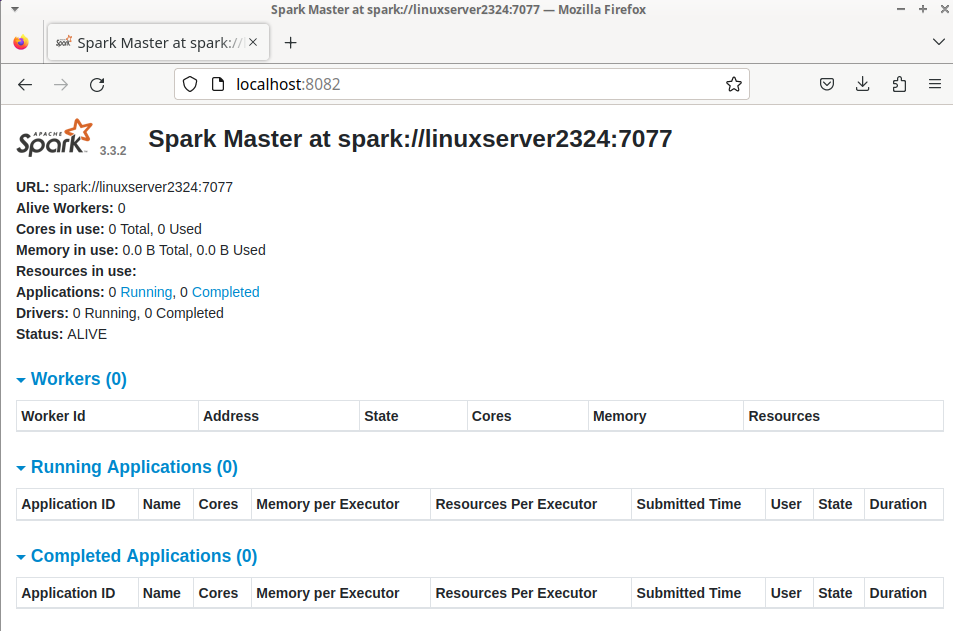
Iniciamos el nodo master de Spark:  
  
spark@linuxserver2324:~$ start-master.sh



Spark no se podrá iniciar ni en el puerto 8080 (NiFi), ni en el 8081 (Mongo-Express), por lo que se ejecutará en el 8082:

spark@linuxserver2324:~$ cat /opt/spark3.3.2/logs/spark-spark-org.apache.spark.deploy.master.Master-1-linuxserver2324.out





Procedemos ahora a iniciar un Worker con el siguiente comando:

spark@linuxserver2324:~$ /opt/spark3.3.2/sbin/start-worker.sh spark://linuxserver2324:7077



Agregar las siguientes líneas al archivo spark-env.sh

$ nano /opt/spark3.3.2/conf/spark-env.sh

SPARK\_LOCAL\_IP=10.0.2.15

SPARK\_MASTER\_HOST=10.0.2.15

SPARK\_WORKER\_CORES=2

SPARK\_WORKER\_MEMORY=4g

Parar el master y el worker:

$ /opt/spark3.3.2/sbin/stop-master.sh

$ /opt/spark3.3.2/sbin/stop-worker.sh

Cambiar permisos, borrar archivos temporales e iniciar de nuevo los servicios:

$ sudo chown -R spark:spark /opt/spark3.3.2/work

$ sudo chown -R spark:spark /tmp

$ sudo chmod -R 755 /tmp

$ rm -rf /tmp/spark-\*

$ sudo chown -R spark:spark /opt/spark3.3.2/work

$ /opt/spark3.3.2/sbin/start-master.sh

$ /opt/spark3.3.2/sbin/start-worker.sh spark://10.0.2.15:7077

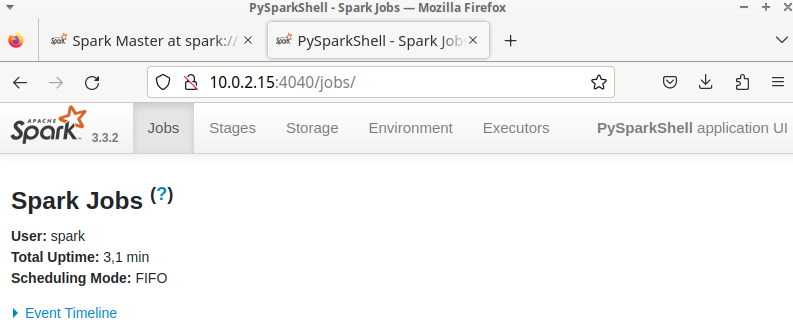
Después de esto, la nueva URL pasa a ser: http://10.0.2.15:8080/

Ahora podemos probar a iniciar PySpark:

$ /opt/spark3.3.2/bin/pyspark --master spark://10.0.2.15:7077



La URL de PySpark es: http://10.0.2.15:4040/



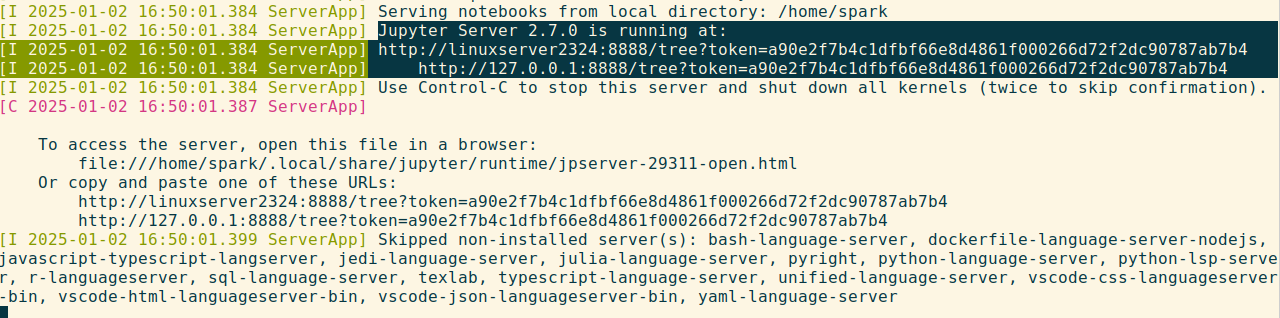
Ahora necesitamos configurar Jupyter para trabajar con Spark, editamos el archivo .bashrc, agregando las 3 últimas líneas como variables de entorno:

$ nano ~/.bashrc



Iniciamos ahora Jupyter Notebook con PySpark:

spark@linuxserver2324:~$ pyspark --master spark://10.0.2.15:7077



La URL del Jupyter Notebook es la siguiente:

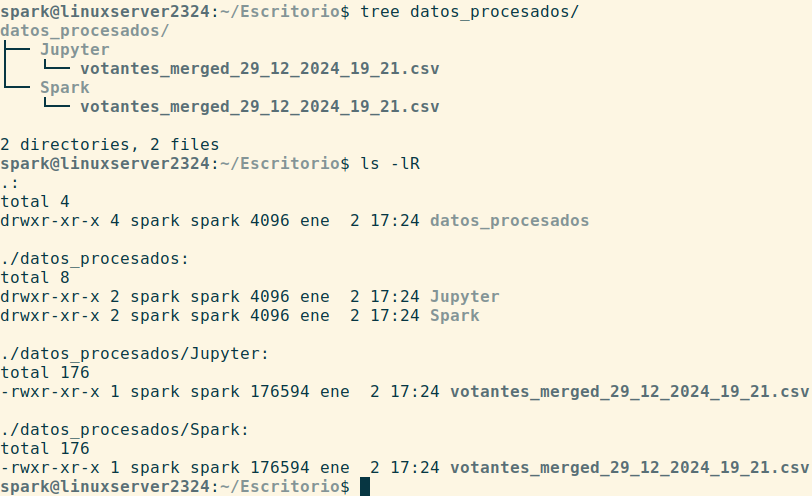
<http://127.0.0.1:8888/tree>

Copiamos los archivos CSV con los que queremos trabajar al escritorio del usuario spark:

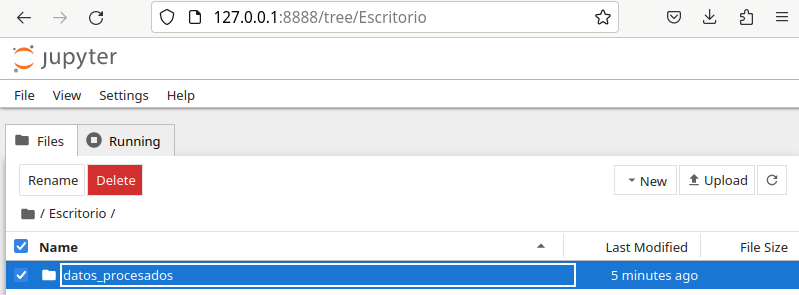
$ sudo chmod -R 755 /home/alumno/Escritorio/datos\_procesados/

$ sudo cp -R /home/alumno/Escritorio/datos\_procesados/ /home/spark/Escritorio/

$ sudo chown -R spark:spark /home/spark/Escritorio/datos\_procesados



Después de esto ya podemos acceder a nuestro Jupyter Notebook y encontrar allí los archivos necesarios para trabajar con ellos:



¡Y no nos olvidemos de instalar matplotlib!

spark@linuxserver2324:~$ pip install matplotlib