

SmartParking Flow: Monitorización inteligente de plazas con Kafka, NiFi, MongoDB, Flask y Dremio

Ángel Manuel Pereira Rodríguez

Noviembre 2025

Índice general

1. Introducción	1
1.1. Contextualización	1
1.2. Justificación	1
1.3. Objetivos	2
1.4. Alcance del proyecto	2
1.5. Planificación	3
2. Marco Teórico	4
2.1. Apache Kafka	4
2.2. Apache NiFi	4
2.3. MongoDB Atlas	5
2.4. Flask	5
2.5. Dremio	5
3. Fase de Análisis	6
3.1. Requisitos Funcionales (RF)	6
3.2. Requisitos No Funcionales (RNF)	7
3.3. Modelo de Datos	7
4. Fase de Diseño	8
4.1. Arquitectura del Sistema	8
4.2. Diseño de Base de Datos (MongoDB)	9
4.2.1. Colección <code>bays</code>	9
4.2.2. Colección <code>events</code>	9
4.3. Diseño del Flujo de NiFi	10
4.4. Diseño de la API REST (Flask)	10
A. Anexo A: Configuración de la Máquina Virtual (Lubuntu)	12
B. Anexo B: Configuración de MongoDB Atlas	29
C. Anexo C: Configuración de Apache Kafka	55
D. Anexo D: Configuración del Flujo de NiFi	71

E. Anexo E: Aplicación Flask (API y Visualización)	118
E.1. Código de la Aplicación (app.py)	118
E.2. Despliegue y Funcionamiento	126
F. Anexo F: Consultas de Análisis en Dremio	128

Índice de figuras

4.1. Diagrama de la arquitectura del sistema (Flujo de NiFi).	8
A.1. Configuración inicial de la MV "Lubuntu Proyecto UD1. ^{en} VirtualBox.	12
A.2. Menú de arranque (GRUB) de la ISO de Lubuntu.	13
A.3. Inicio del instalador de Lubuntu.	13
A.4. Selección de "Install Lubuntu".	14
A.5. Selección de idioma (Español).	14
A.6. Selección de ubicación (Madrid).	15
A.7. Selección de teclado (Spanish).	15
A.8. Selección de tipo de instalación (Normal).	16
A.9. Configuración de particiones (Borrar disco).	16
A.10. Creación del usuario y nombre de equipo.	17
A.11. Resumen de la instalación.	17
A.12. Confirmación de inicio de instalación.	18
A.13. Proceso de instalación.	18
A.14. Instalación finalizada.	19
A.15. Extracción del disco de instalación virtual al reiniciar.	19
A.16. Escritorio de Lubuntu y apertura de QTerminal.	20
A.17. Actualización de paquetes ('sudo apt update').	20
A.18. Insertando imagen de CD de las "Guest Additions" de VirtualBox.	21
A.19. Aviso de medio extraíble (Guest Additions).	21
A.20. Navegación al directorio de las Guest Additions.	22
A.21. Ejecución del script de instalación VBoxLinuxAdditions.run.	22
A.22. Extracción del disco virtual de las Guest Additions.	23
A.23. Aviso de forzar desmontaje del disco óptico.	23
A.24. Habilitando portapapeles bidireccional en VirtualBox.	24
A.25. Terminal de Lubuntu post-reinicio.	24
A.26. Comprobación de la dirección IP de la MV ('ip addr') - 192.168.1.73.	25
A.27. Búsqueda de Símbolo del sistema (CMD) en el anfitrión (Windows).	26
A.28. Prueba de conectividad (ping) desde el anfitrión (Windows) a la MV.	27
A.29. Instalación de paquetes base en Lubuntu ('curl', 'wget', 'git', 'python3-pip').	27
A.30. Creación de directorios del proyecto y comprobación de versiones ('python3' y 'pip').	28
B.1. Búsqueda inicial de MongoDB Atlas.	29
B.2. Formulario de registro en MongoDB Atlas.	30
B.3. Paso de verificación de correo electrónico.	30

B.4.	Recepción del correo de verificación.	31
B.5.	Contenido del correo "Verify Your MongoDB Email Address".	31
B.6.	Confirmación de email verificado.	32
B.7.	Configuración opcional de Multi-Factor Authentication (MFA).	32
B.8.	Recepción del código de verificación de MongoDB.	33
B.9.	Pantalla de configuración de seguridad de la cuenta.	33
B.10.	Pantalla de bienvenida a la nueva navegación de Atlas.	34
B.11.	Panel de ^A All Projects inicial.	34
B.12.	Renombrando el proyecto a "SmartParking Flow".	35
B.13.	Panel principal del proyecto "SmartParking Flow".	35
B.14.	Configuración del cluster gratuito (M0).	36
B.15.	Verificación Captcha para la creación del cluster.	36
B.16.	Creación del usuario de la base de datos.	37
B.17.	Confirmación de creación de usuario.	37
B.18.	Selección del método de conexión al cluster.	38
B.19.	Cargando datos de ejemplo (sample data).	38
B.20.	Vista del cluster tras finalizar la carga de datos de ejemplo.	39
B.21.	Configuración de ÍP Access List"(IP actual).	39
B.22.	Añadiendo una nueva entrada a ÍP Access List".	40
B.23.	Añadiendo '0.0.0.0/0' (Allow Access From Anywhere).	40
B.24.	ÍP Access List. ^a ctualizada con acceso global.	41
B.25.	Vista del cluster con datos de ejemplo cargados (74.12 MB).	41
B.26.	Explorador de datos (Data Explorer) viendo 'sample_mflix.comments'.	42
B.27.	Creación de la base de datos 'smartparking' y la colección 'events'.	42
B.28.	Vista de la colección 'events' vacía.	43
B.29.	Creación de la colección 'bays'.	43
B.30.	Vista de las colecciones 'bays' y 'events' creadas.	44
B.31.	Creación del índice para la colección 'bays'.	44
B.32.	Confirmación de creación de índice en 'bays' sobre 'bay_id'.	45
B.33.	Índice 'bay_id_1' creado y listado en la colección 'bays'.	45
B.34.	Vista de la pestaña Índexes"de la colección 'events'.	46
B.35.	Definición del índice compuesto para la colección 'events'.	46
B.36.	Confirmación de creación de índice en 'events'.	47
B.37.	Índice 'bay_event_ts_desc_idx' creado en la colección 'events'.	47
B.38.	Vista del cluster (Data Size: 116.16 MB).	48
B.39.	Opciones de conexión al cluster.	48
B.40.	Obteniendo la cadena de conexión (Connection String) para Python.	49
B.41.	Instalando 'pymongo[srv]' en el anfitrión (Windows).	49
B.42.	Script de prueba <code>pruebas.py</code> en VS Code (Windows).	50
B.43.	Resultado de la ejecución del script de prueba en Windows.	50
B.44.	Documento de prueba "TEST-001insertado en 'smartparking.bays'	51
B.45.	Vista del documento "TEST-001.en Data Explorer.	51
B.46.	Documento "TEST-001"marcado para eliminación.	52
B.47.	Colección 'smartparking.bays' vacía tras borrado.	52
B.48.	Instalando 'python3-venv' en la MV de Lubuntu.	53
B.49.	Instalando 'pymongo[srv]' en el entorno virtual de Lubuntu.	53

B.50.Editando el script de prueba <code>pruebas.py</code> en Lubuntu (nano).	54
B.51.Resultado de la ejecución del script de prueba en Lubuntu.	54
C.1. Descarga de Apache Kafka ('wget').	55
C.2. Descompresión del archivo ('tar -xzf') y renombrado de la carpeta.	56
C.3. Creación del directorio de datos ('/opt/kafka-data') y edición de 'server.properties'.	56
C.4. Configuración de 'server.properties': 'node.id' y 'controller.quorum.bootstrap.servers'.	57
C.5. Configuración de 'server.properties': 'listeners' y 'advertised.listeners'.	57
C.6. Configuración de 'server.properties': 'log.dirs'.	58
C.7. Configuración de 'server.properties': Políticas de retención de logs.	58
C.8. Instalación de Java (OpenJDK 11 y 21) en la MV.	59
C.9. Selección de la versión de Java (JDK 11) usando 'update-alternatives'.	59
C.10.Selección de la versión de 'javac' (JDK 11).	60
C.11.Editando el script 'kafka-run-class.sh'.	60
C.12.Añadiendo 'JAVA_HOME' al script 'kafka-run-class.sh'.	61
C.13.Formateo del directorio de almacenamiento de Kafka ('kafka-storage.sh for- mat').	61
C.14.Inicio del servidor Kafka ('kafka-server-start.sh').	62
C.15.Editando el archivo '/.bashrc' para añadir variables de entorno.	62
C.16.Añadiendo 'KAFKA_HOME' y actualizando el 'PATH' en '/.bashrc'.	63
C.17.Recargando la configuración de la terminal ('source /.bashrc').	63
C.18.Comando para crear un nuevo tópico de Kafka.	64
C.19.Creación del tópico <code>parking-events</code> y listado de tópicos.	64
C.20.Instalación de la librería 'kafka-python' con 'pip3'.	65
C.21.Código del simulador de sensores: 'ParkingSensorSimulator.__init__'.	65
C.22.Código del simulador de sensores: 'ParkingSensorSimulator.generate_event'.	66
C.23.Código del publicador de Kafka: 'KafkaPublisher'.	66
C.24.Código de la función 'run_simulation' (lógica principal del productor).	67
C.25.Código del 'argparse' para configurar el simulador desde la terminal.	68
C.26.Dando permisos de ejecución ('chmod +x') al script del simulador.	68
C.27.Ejecución del script simulador de sensores ('parking_simulator.py').	69
C.28.Comando para iniciar un consumidor de consola en formato JSON.	69
C.29.Consumidor de consola de Kafka mostrando los mensajes JSON entrantes.	70
D.1. Descarga del binario de Apache NiFi ('wget').	71
D.2. Descompresión del archivo ('unzip').	72
D.3. Accediendo al directorio de configuración ('conf').	72
D.4. Editando 'nifi.properties': 'nifi.web.http.host=0.0.0.0'.	73
D.5. Editando 'nifi.properties': 'nifi.sensitive.props.key'.	73
D.6. Editando 'nifi.properties': 'nifi.remote.input.host'.	74
D.7. Editando 'nifi.properties': 'nifi.security.user.authorizer'.	74
D.8. Accediendo al archivo 'bootstrap.conf'.	75
D.9. Editando 'bootstrap.conf': Configuración de memoria JVM (Xms2g, Xmx2g). .	75
D.10.Editando '/.bashrc' para añadir 'NIFI_HOME'.	76
D.11.Añadiendo 'NIFI_HOME' y actualizando el 'PATH' en '/.bashrc'.	76
D.12.Accediendo al script de entorno 'nifi-env.sh'.	77
D.13.Estableciendo 'JAVA_HOME' en 'nifi-env.sh'.	77

D.14.Iniciando el servicio de NiFi ('nifi.sh start').	78
D.15.Comprobando los logs de inicio de NiFi ('tail -f nifi-app.log')	78
D.16.Accediendo a la interfaz web de NiFi (localhost:8080/nifi).	79
D.17.Arrastrando un nuevo procesador al canvas.	79
D.18.Buscando el procesador 'ConsumeKafka'.	80
D.19.Configuración (Settings) del procesador 'ConsumeKafka'.	80
D.20.Configuración (Properties) del procesador 'ConsumeKafka'.	81
D.21.Creación de un nuevo Controller Service: 'Kafka3ConnectionService'.	81
D.22.Asignando el 'Kafka3ConnectionService' al procesador.	82
D.23.Guardando cambios en el procesador.	82
D.24.Accediendo a la configuración del Controller Service ('Kafka3ConnectionService').	83
D.25.Configurando el 'Kafka3ConnectionService' (Bootstrap Servers: localhost:9092).	83
D.26.Controller Service 'Kafka3ConnectionService' en estado "Disabled".	84
D.27.Habilitando el 'Kafka3ConnectionService'.	84
D.28.Controller Service 'Kafka3ConnectionService' siendo habilitado.	85
D.29.Controller Service 'Kafka3ConnectionService' en estado ".Enabled".	85
D.30.Configuración final del procesador 'ConsumeKafka' (Tópico: parking-events).	86
D.31.Procesador 'Consume Parking Sensors' en el canvas.	86
D.32.Añadiendo el procesador 'EvaluateJsonPath'.	87
D.33.Procesadores 'Consume Parking Sensors' y 'EvaluateJsonPath' en el canvas.	87
D.34.Creando conexión "success."entre 'ConsumeKafka' y 'EvaluateJsonPath'. . .	88
D.35.Configuración (Properties) de 'EvaluateJsonPath': 'Destination: flowfile-attribute'.	88
D.36.Configuración (Properties) de 'EvaluateJsonPath': Extracción de atributos (battery, bay_id, ...).	89
D.37.Configuración (Settings) de 'EvaluateJsonPath': 'Name: Extract JSON Attributes'.	89
D.38.Configuración (Relationships) de 'EvaluateJsonPath': Terminar ünmatched".	90
D.39.Añadiendo el procesador 'RouteOnAttribute'.	90
D.40.Configuración (Settings) de 'RouteOnAttribute': 'Name: Validate Required Fields'.	91
D.41.Configuración (Relationships) de 'RouteOnAttribute': Terminar ünmatched".	91
D.42.Configuración (Properties) de 'RouteOnAttribute': 'Routing Strategy'. . .	92
D.43.Configuración (Properties) de 'RouteOnAttribute': Añadiendo propiedades de validación.	92
D.44.Flujo con los tres procesadores iniciales.	93
D.45.Creando conexión "matched."entre 'EvaluateJsonPath' y 'RouteOnAttribute'. .	93
D.46.Accediendo a la configuración de Controller Services del Process Group. . .	94
D.47.Vista de 'Kafka3ConnectionService' habilitado.	94
D.48.Añadiendo un nuevo Controller Service: 'JsonTreeReader'.	95
D.49.Controller Service 'JsonTreeReader' en estado "Disabled".	95
D.50.Habilitando el 'JsonTreeReader'.	96
D.51.Confirmación de habilitación del 'JsonTreeReader'.	96
D.52.Ambos Controller Services ('JsonTreeReader' y 'Kafka3ConnectionService') habilitados.	97
D.53.Añadiendo el procesador 'PutMongoRecord'.	97

D.54.Flujo con el procesador 'PutMongoRecord' añadido.	98
D.55.Creando conexión "matched."entre 'RouteOnAttribute' y 'PutMongoRecord'. .	98
D.56.Configuración (Settings) de 'PutMongoRecord': 'Name: Insert to Events Collection'.	99
D.57.Configuración (Properties) de 'PutMongoRecord' (vacía).	99
D.58.Creando un nuevo Controller Service: 'MongoDBCPService' (obsoleto, se usará otro).	100
D.59.Configuración de 'PutMongoRecord' (Properties) - se usará 'MongoDBControllerService'.	100
D.60.Aviso de guardado de Controller Service.	101
D.61.Accediendo a la configuración del 'MongoDBControllerService'.	101
D.62.Editando 'MongoDBControllerService': Pegando la URI de conexión de Atlas.	102
D.63.Configuración (Properties) de 'MongoDBControllerService' con la URI.	102
D.64.Habilitando el 'MongoDBControllerService'.	103
D.65.Confirmación de habilitación del 'MongoDBControllerService'.	103
D.66.Habilitando el 'MongoDBControllerService'.	104
D.67.Todos los Controller Services habilitados.	104
D.68.Configuración de 'PutMongoRecord': 'Mongo Database Name: smartparking', 'Collection: events'.	105
D.69.Añadiendo el procesador 'ReplaceText'.	105
D.70.Flujo con el procesador 'ReplaceText' añadido.	106
D.71.Creando conexión "success."entre 'Insert to Events Collection' y 'ReplaceText'. .	106
D.72.Configuración (Settings) de 'ReplaceText': 'Name: Prepare Bay Document'. .	107
D.73.Configuración (Relationships) de 'ReplaceText': Terminar 'failure'.	107
D.74.Configuración (Properties) de ReplaceText: Replacement Value (JSON del documento bays).	108
D.75.Configuración (Properties) de ReplaceText: Search Value: (?s)(*) . . .	108
D.76.Añadiendo el procesador 'PutMongo' (para el upsert).	109
D.77.Flujo con el procesador 'PutMongo' añadido.	109
D.78.Creando conexión "success."entre 'ReplaceText' y 'PutMongo'.	110
D.79.Configuración (Settings) de 'PutMongo': 'Name: Upsert to Bays Collection'. .	110
D.80.Configuración (Relationships) de 'PutMongo': Terminar "failurez "success". .	111
D.81.Configuración (Properties) de 'PutMongo': 'Collection: bays', 'Mode: upsert', 'Key: bay_id'.	111
D.82.Añadiendo el procesador 'LogAttribute' (para errores).	112
D.83.Flujo con el procesador 'LogAttribute' añadido.	112
D.84.Creando conexión "failure."entre 'Upsert to Bays Collection' y 'Log Errors'. .	113
D.85.Configuración (Settings) de 'LogAttribute': 'Name: Log Errors'.	113
D.86.Configuración (Relationships) de 'LogAttribute': Terminar "success".	114
D.87.Configuración (Properties) de 'LogAttribute': 'Log Level: error'.	114
D.88.Configuración (Relationships) de 'Insert to Events Collection': Terminar "failurez "success".	115
D.89.Vista general del flujo de NiFi (incompleta).	115
D.90.Vista general del flujo de datos completo en Apache NiFi.	116
D.91.Datos de la colección 'events' vistos desde MongoDB Atlas.	116
D.92.Datos de la colección 'bays' vistos desde MongoDB Atlas.	117

E.1. Instalación de dependencias de Python (<code>requirements.txt</code>)	126
E.2. Contenido del archivo de variables de entorno (' <code>.env.template</code> ')	126
E.3. Ejecución del servidor Flask (' <code>python app.py</code> ')	126
E.4. Interfaz web de SmartParking mostrando el estado en tiempo real.	127
F.1. Descarga de Dremio Community (' <code>wget</code> ')	128
F.2. Descompresión, inicio del servicio (' <code>dremio start</code> ') y comprobación de logs.	129
F.3. Creación de la cuenta de administrador en la interfaz web de Dremio.	129
F.4. Panel principal de Dremio (Datasets)	130
F.5. Añadiendo una nueva fuente de datos (Add Data Source): MongoDB.	130
F.6. Configuración de la fuente "MongoDB Source": Host y Puerto.	131
F.7. Configuración de la autenticación de la fuente "MongoDB Source".	131
F.8. Fuente de datos "MongoDBAtlas" conectada, mostrando la base de datos 'smartparking'.	132
F.9. Navegando en las colecciones 'bays' y 'events' dentro de Dremio.	132
F.10. Consulta SQL 1: Ocupación actual por nivel.	133
F.11. Guardando la consulta (View) como ".cupacion Actual por Nivel".	133
F.12. Consulta SQL 2: Porcentaje de ocupación por nivel.	134
F.13. Consulta SQL 3: Plazas con nivel bajo de batería (mantenimiento).	134
F.14. Consulta SQL 4: Horas del día con mayor número de eventos de ocupación.	135
F.15. Consulta SQL 5: Plazas más ocupadas (Top 10).	135
F.16. Resultado de la consulta SQL 5, mostrando las plazas con más eventos.	136

Capítulo 1

Introducción

1.1 Contextualización

Este proyecto se desarrolla en el marco de la iniciativa *SmartCity Cádiz*, que busca aplicar tecnologías avanzadas para mejorar la eficiencia de los servicios urbanos y la calidad de vida de los ciudadanos.

Uno de los desafíos más significativos en las ciudades modernas es la gestión del tráfico y el aparcamiento. El proyecto **SmartParking Flow** aborda este reto implementando un sistema de monitorización inteligente para aparcamientos. El objetivo es desarrollar e implementar un sistema avanzado para la recolección, almacenamiento y visualización en tiempo real de los datos generados por los sensores de un aparcamiento inteligente.

1.2 Justificación

La gestión eficiente del aparcamiento reduce la congestión del tráfico, disminuye la contaminación y mejora la experiencia del conductor. Los sistemas tradicionales de gestión de parking carecen de la capacidad de procesar y reaccionar a los datos en tiempo real.

Este proyecto se justifica por la necesidad de implementar un flujo de datos (un *pipeline*) robusto que gestione la información desde el sensor hasta el usuario final. Se emplearán tecnologías de Big Data para optimizar este flujo:

- **Apache Kafka** se utilizará para la recepción continua y fiable de millones de mensajes de los sensores.
- **Apache NiFi** orquestará el flujo, automatizando la ingestión, validación y almacenamiento de los datos.
- **MongoDB Atlas** servirá como base de datos NoSQL flexible y escalable, capaz de gestionar tanto el estado actual de las plazas como un histórico de eventos.
- **Flask** y **Dremio** proporcionarán las capas de visualización y análisis, permitiendo a los usuarios ver la disponibilidad en tiempo real y a los gestores analizar patrones de uso.

1.3 Objetivos

Los objetivos principales y específicos del proyecto *SmartParking Flow* son los siguientes:

- **Configurar Apache Kafka:** Desplegar y configurar Kafka para gestionar la transmisión en tiempo real de los datos generados por los sensores, garantizando una comunicación continua y fiable.
- **Orquestar con Apache NiFi:** Automatizar el flujo de ingestión y procesamiento desde los tópicos de Kafka hasta MongoDB, asegurando que cada mensaje se valide, transforme e inserte correctamente.
- **Configurar MongoDB:** Estructurar la base de datos en MongoDB Atlas con dos colecciones diferenciadas: una para el histórico de eventos y otra para el estado actual de cada plaza (optimizada para operaciones *upsert*).
- **Desarrollar aplicación Flask:** Implementar una aplicación web que consuma los datos de MongoDB y muestre un mapa visual del aparcamiento (plazas en verde/-rojo) que se actualice automáticamente.
- **Integrar Dremio:** Utilizar Dremio como plataforma de análisis para ejecutar consultas descriptivas (SQL) sobre los datos de MongoDB, extrayendo indicadores sobre ocupación, uso por franjas horarias, etc..
- **Garantizar un flujo robusto:** Asegurar que todo el *pipeline* de datos, desde el sensor hasta la visualización, sea robusto, escalable y automatizado.

1.4 Alcance del proyecto

El alcance de este proyecto cubre el ciclo de vida completo del dato, desde su generación simulada hasta su análisis final.

- **Entorno de desarrollo:** Configuración de una máquina virtual (VM) con Lubuntu 24.04 en VirtualBox (Ver Anexo A).
- **Generación de datos:** Creación de un script en Python (`sensores.py`) que simula la actividad de los sensores (cambios de estado, métricas de batería y temperatura) y publica los datos en formato JSON en un tópico de Kafka (Ver Anexo C).
- **Ingesta y ETL:** Configuración de Apache Kafka (tópico `parking-events`) y un flujo en Apache NiFi que consume de dicho tópico, procesa los mensajes y los enruta a MongoDB (Ver Anexo D).
- **Almacenamiento:** Uso de la plataforma cloud MongoDB Atlas para alojar la base de datos `smartparking` con sus dos colecciones `bays` y `events` (Ver Anexo B).
- **Visualización (Web App):** Desarrollo de una aplicación web con Flask (`app.py`) que expone una API REST y presenta una interfaz gráfica en `index.html` que muestra el estado en tiempo real (Ver Anexo E).

- **Análisis (BI):** Instalación y conexión de Dremio a la fuente de MongoDB Atlas para la ejecución de, al menos, 5 consultas SQL analíticas significativas (Ver Anexo F).

1.5 Planificación

La ejecución del proyecto se distribuyó en varias jornadas de trabajo, cubriendo las diferentes fases de instalación, configuración y desarrollo, tal como se detalla en el diario de trabajo.

Capítulo 2

Marco Teórico

Para la implementación del proyecto *SmartParking Flow*, se ha seleccionado un conjunto de tecnologías (un *stack*) que permiten gestionar de forma eficiente el ciclo de vida de los datos en tiempo real.

2.1 Apache Kafka

Apache Kafka actúa como el sistema nervioso central de la arquitectura. Es una plataforma de *streaming* de eventos distribuida, diseñada para manejar grandes volúmenes de datos con alta velocidad y baja latencia.

En este proyecto, se utiliza para:

- **Desacoplar** a los productores de datos (sensores) de los consumidores (NiFi).
- **Actuar como buffer**, permitiendo que el sistema de ingesta procese los datos a su propio ritmo sin riesgo de pérdida de información, incluso si hay picos de eventos.
- **Garantizar la fiabilidad** y la tolerancia a fallos en la transmisión de mensajes.

2.2 Apache NiFi

Apache NiFi (NiagaraFiles) es una herramienta de orquestación y automatización de flujos de datos. Su principal fortaleza radica en su interfaz gráfica de usuario (GUI) basada en flujos, que permite diseñar, controlar y monitorizar la ruta de los datos de forma visual.

Sus funciones clave en el proyecto son:

- **Consumir** datos del tópico de Kafka en tiempo real.
- **Validar y transformar** los mensajes JSON recibidos (por ejemplo, extrayendo atributos).
- **Enrutar** los datos, implementando la lógica de negocio para la doble inserción en MongoDB (histórico y estado actual).

2.3 MongoDB Atlas

MongoDB es la plataforma de almacenamiento principal. Es una base de datos NoSQL orientada a documentos, lo que significa que almacena los datos en estructuras flexibles similares a JSON (llamadas BSON).

Se ha elegido MongoDB por:

- **Flexibilidad de esquema:** Ideal para los datos de sensores, que pueden evolucionar.
- **Escalabilidad:** Permite crecer horizontalmente para manejar grandes volúmenes de datos.
- **Rendimiento:** Optimizado para consultas rápidas y operaciones de actualización eficientes, como el *upsert*.

Se utiliza **MongoDB Atlas**, la versión gestionada en la nube, para simplificar el despliegue y la administración de la base de datos.

2.4 Flask

Flask es un micro-framework web ligero para Python. Se utiliza para construir la capa de presentación y la API REST del proyecto. Su simplicidad permite un desarrollo rápido.

En el proyecto, Flask es responsable de:

- **Exponer una API REST** que consulta la colección 'bays' y 'events' de MongoDB.
- **Renderizar la interfaz web (`index.html`)** que los usuarios ven en su navegador.
- **Servir como backend** para las peticiones AJAX (JavaScript) que actualizan el mapa de plazas automáticamente.

2.5 Dremio

Dremio es una plataforma de análisis de datos que permite ejecutar consultas SQL federadas sobre múltiples fuentes, incluyendo bases de datos NoSQL como MongoDB.

Su papel es el de **capa de análisis (BI)**. Permite a los gestores del parking ejecutar consultas SQL estándar sobre los datos JSON almacenados en MongoDB, facilitando la creación de informes y la extracción de *insights* (como horas punta, plazas más usadas, etc.) sin necesidad de mover o transformar los datos (ETL) a un almacén de datos tradicional.

Capítulo 3

Fase de Análisis

En esta fase se definen los requisitos del sistema, se identifica el formato de los datos y se establecen las bases para el diseño de la arquitectura.

3.1 Requisitos Funcionales (RF)

Los requisitos funcionales describen *qué* debe hacer el sistema:

- **RF-01: Ingesta de eventos.** El sistema debe ser capaz de consumir mensajes en formato JSON desde un tópico de Apache Kafka llamado `parking-events`.
- **RF-02: Almacenamiento histórico.** Cada evento de sensor recibido debe ser almacenado de forma inmutable en una colección de MongoDB ('events') para su posterior auditoría o análisis histórico.
- **RF-03: Almacenamiento de estado actual.** El sistema debe mantener una colección en MongoDB ('bays') que refleje el último estado conocido de cada plaza de aparcamiento. Esta colección debe actualizarse mediante operaciones *upsert*.
- **RF-04: Visualización de estado.** El sistema debe proveer una interfaz web que muestre un mapa visual del parking, donde cada plaza se represente gráficamente.
- **RF-05: Codificación por color.** Las plazas en la interfaz web deben cambiar de color (verde para "libre", rojo para ".cupada") basándose en su estado actual en la base de datos.
- **RF-06: Actualización automática.** La interfaz web debe refrescar el estado de las plazas automáticamente cada pocos segundos sin necesidad de recargar la página.
- **RF-07: Panel de estadísticas.** La web debe mostrar un resumen estadístico (Total de plazas, Libres, Ocupadas, Porcentaje de Ocupación).
- **RF-08: Análisis de datos.** El sistema debe permitir a un usuario analista ejecutar consultas SQL descriptivas sobre los datos almacenados en MongoDB.

3.2 Requisitos No Funcionales (RNF)

Los requisitos no funcionales describen *cómo* debe operar el sistema:

- **RNF-01: Rendimiento (Latencia).** El tiempo desde que un sensor emite un evento hasta que se refleja en la interfaz web debe ser de pocos segundos.
- **RNF-02: Fiabilidad.** El sistema no debe perder mensajes de los sensores. El flujo de datos debe ser tolerante a fallos.
- **RNF-03: Escalabilidad.** La arquitectura debe ser capaz de escalar horizontalmente para soportar un incremento futuro en el número de sensores, parkings o usuarios.
- **RNF-04: Mantenibilidad.** El flujo de datos (ETL) debe ser fácilmente modificable y monitorizable, preferiblemente a través de una interfaz visual (cumplido por NiFi).
- **RNF-05: Flexibilidad.** El esquema de la base de datos debe ser flexible para admitir nuevos campos en los mensajes de los sensores (ej. nuevos tipos de métricas) sin interrumpir el servicio.

3.3 Modelo de Datos

El formato de datos (contrato) que se utiliza en todo el flujo, desde el productor de Kafka hasta el almacenamiento, es un documento JSON. Este formato es ligero, legible por humanos y fácilmente procesable por todas las herramientas del *stack*.

Un ejemplo del documento enviado por los sensores se define de la siguiente manera:

```
1 {
2     "bay\_id": "L1-A-023",
3     "parking\_id": "PK-CADIZ-01",
4     "level": "L1",
5     "occupied": true,
6     "last\_event\_ts": "2025-10-07T10:15:30Z",
7     "metrics": {
8         "temperature\_c": 23.4,
9         "battery\_pct": 78
10    },
11    "updated\_at": "2025-10-07T10:15:31Z"
12 }
```

Listing 3.1: Ejemplo de documento JSON de un evento de sensor.

Capítulo 4

Fase de Diseño

Basado en los requisitos analizados, se diseña una arquitectura de sistema que cumple con los objetivos funcionales y no funcionales.

4.1 Arquitectura del Sistema

La arquitectura diseñada sigue un patrón de *pipeline* de datos en tiempo real, separando las responsabilidades en capas claras. El flujo de datos es unidireccional, desde los sensores hasta las aplicaciones de usuario (Figura 4.1).

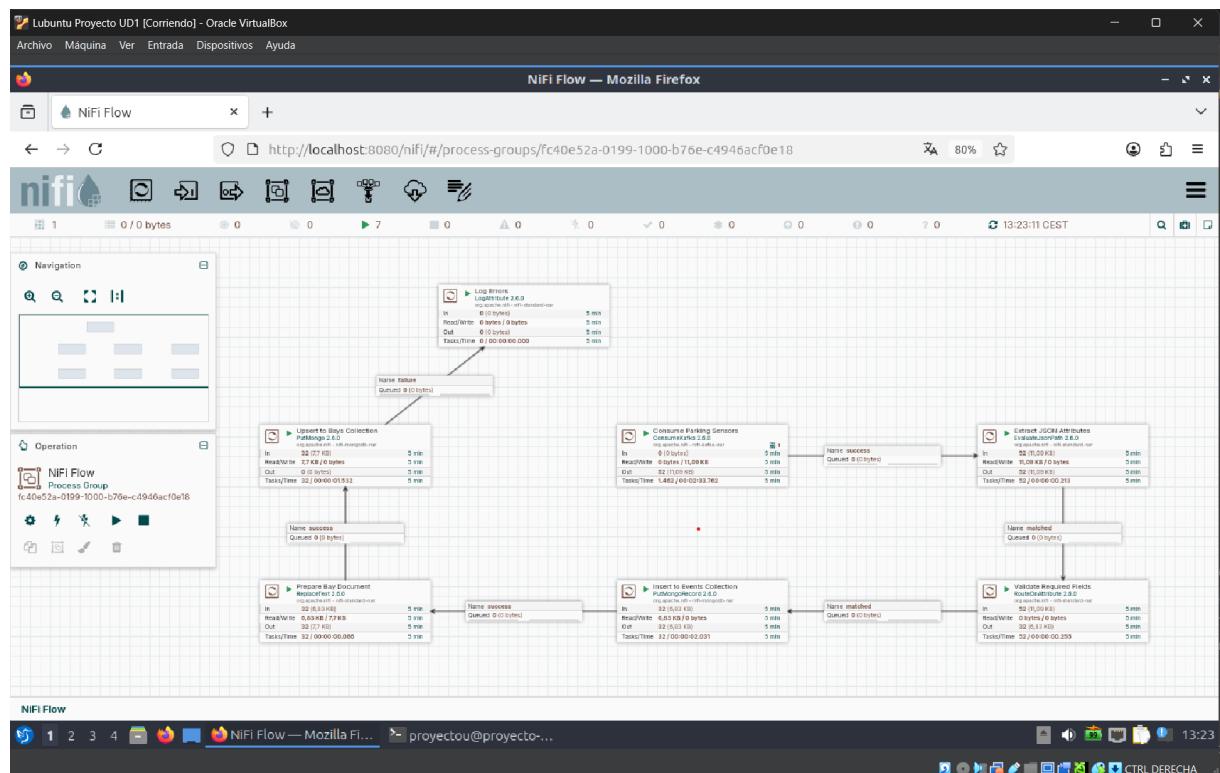


Figura 4.1: Diagrama de la arquitectura del sistema (Flujo de NiFi).

El flujo de datos es el siguiente:

1. **Sensores (Simulados):** El script `sensores.py` genera los datos JSON y los publica en el tópico `parking-events` de Kafka.
2. **Broker (Kafka):** Recibe y almacena temporalmente los mensajes en el tópico.
3. **ETL (NiFi):** Un flujo de NiFi consume los mensajes de Kafka.
4. **Bifurcación (NiFi):** El flujo se divide en dos ramas paralelas:
 - **Rama Histórica:** Los datos se insertan directamente en la colección `events` de MongoDB.
 - **Rama de Estado:** Los datos se utilizan para realizar un *upsert* en la colección `bays`.
5. **Almacenamiento (MongoDB Atlas):** La base de datos en la nube persiste los datos.
6. **Visualización (Flask):** La aplicación Flask consulta la colección `bays` para obtener el estado actual y lo sirve a través de su API REST.
7. **Análisis (Dremio):** Dremio se conecta directamente a MongoDB Atlas para ejecutar consultas SQL sobre las colecciones `bays` y `events`.

4.2 Diseño de Base de Datos (MongoDB)

Se ha diseñado una base de datos en MongoDB Atlas llamada `smartparking`, que contiene dos colecciones principales para satisfacer los requisitos RF-02 y RF-03.

4.2.1. Colección `bays`

Esta colección cumple con el requisito RF-03 de mantener el estado actual.

- **Propósito:** Almacenar el documento más reciente y completo de cada `bay_id` único.
- **Operación de NiFi:** *upsert*.
- **Clave de Upsert:** `bay_id`.
- **Índices:** Se crea un índice único sobre el campo `bay_id` para garantizar que no haya duplicados y acelerar las operaciones de *upsert* y las consultas de la API (Ver Anexo B, Figura B.33).

4.2.2. Colección `events`

Esta colección cumple con el requisito RF-02 de mantener un histórico.

- **Propósito:** Almacenar un registro inmutable de cada evento de sensor recibido.
- **Operación de NiFi:** *insert*.

- **Índices:** Se crea un índice compuesto descendente sobre `last_event_ts` y `bay_id` para optimizar las consultas de series temporales o la auditoría de una plaza específica (Ver Anexo B, Figura B.37).

4.3 Diseño del Flujo de NiFi

El flujo de NiFi es el motor de ETL que implementa la lógica de negocio (Ver Figura D.90). El diseño utiliza los siguientes procesadores principales:

- **ConsumeKafkaRecord_2_6:** Configurado para conectarse al *bootstrap server* de Kafka (ej. `localhost:9092`) y suscribirse al tópico `parking-events`. Utiliza un `JsonTreeReader` para interpretar los datos entrantes.
- **EvaluateJsonPath:** Extrae los campos clave del cuerpo del JSON (como `$.bay_id`, `$.occupied`, `$.metrics.battery_pct`, etc.) y los almacena como atributos del *FlowFile*.
- **RouteOnAttribute:** Utilizado para validar que los campos requeridos (ej. `bay_id`) no sean nulos o vacíos, enrutando los *FlowFiles* válidos a la rama "matched".
- **Insert to Events Collection (PutMongoRecord):**
 - **Conexión:** Usa un `MongoDBControllerService` con la URI de conexión de Atlas (Ver Anexo D, Figura D.63).
 - **Configuración:** Apunta a Mongo Database Name: `smartparking` y Mongo Collection Name: `events` (Ver Anexo D, Figura D.68).
 - **Operación:** Modo `insert`.
- **Upsert to Bays Collection (PutMongo):**
 - **Conexión:** Utiliza el mismo `MongoDBControllerService`.
 - **Configuración:** Apunta a Mongo Database Name: `smartparking` y Mongo Collection Name: `bays`.
 - **Operación:** Modo `upsert` con Update Query Key: `bay_id` (Ver Anexo D, Figura D.81).
- **LogAttribute:** Se utiliza para registrar errores (en las relaciones *failure* o *unmatched*) y facilitar la depuración.

4.4 Diseño de la API REST (Flask)

Para cumplir con los requisitos de visualización (RF-04, RF-07), la aplicación Flask (`app.py`) expone los siguientes *endpoints* REST:

- **GET /:**
 - **Descripción:** Sirve la página web principal `index.html`.

- **Respuesta:** text/html
- **GET /api/bays:**
 - **Descripción:** Obtiene el estado actual de todas las plazas. Es consumido por el JavaScript de la web para refrescar el mapa.
 - **Lógica:** Ejecuta un `db.bays.find({}, {'_id': 0})`
 - **Respuesta:** application/json con una lista de documentos de plazas.
- **GET /api/stats:**
 - **Descripción:** Obtiene las estadísticas agregadas para el panel principal.
 - **Lógica:** Utiliza un *pipeline* de agregación de MongoDB (`db.bays.aggregate(...)`) para calcular los totales y agrupar por nivel.
 - **Respuesta:** application/json con un objeto que contiene `total`, `occupied`, `free`, `occupancy_rate` y `levels`.
- **GET /api/health:**
 - **Descripción:** Endpoint de monitorización para verificar el estado de la aplicación y su conexión a MongoDB.
 - **Respuesta:** application/json con estado `healthy` o `unhealthy`.

Apéndice A

Anexo A: Configuración de la Máquina Virtual (Lubuntu)

Esta sección muestra el proceso de instalación y configuración de la máquina virtual Lubuntu 24.04 sobre la cual se despliega el entorno de desarrollo.

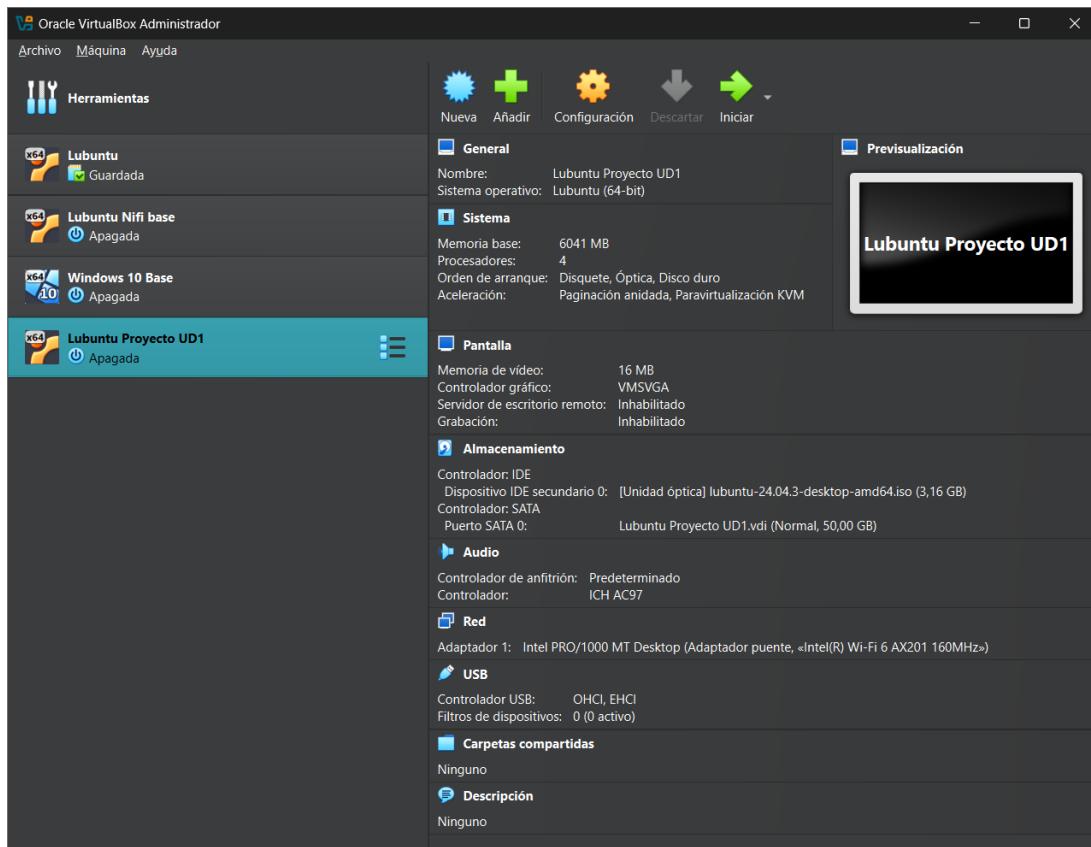


Figura A.1: Configuración inicial de la MV "Lubuntu Proyecto UD1.^{en}" VirtualBox.

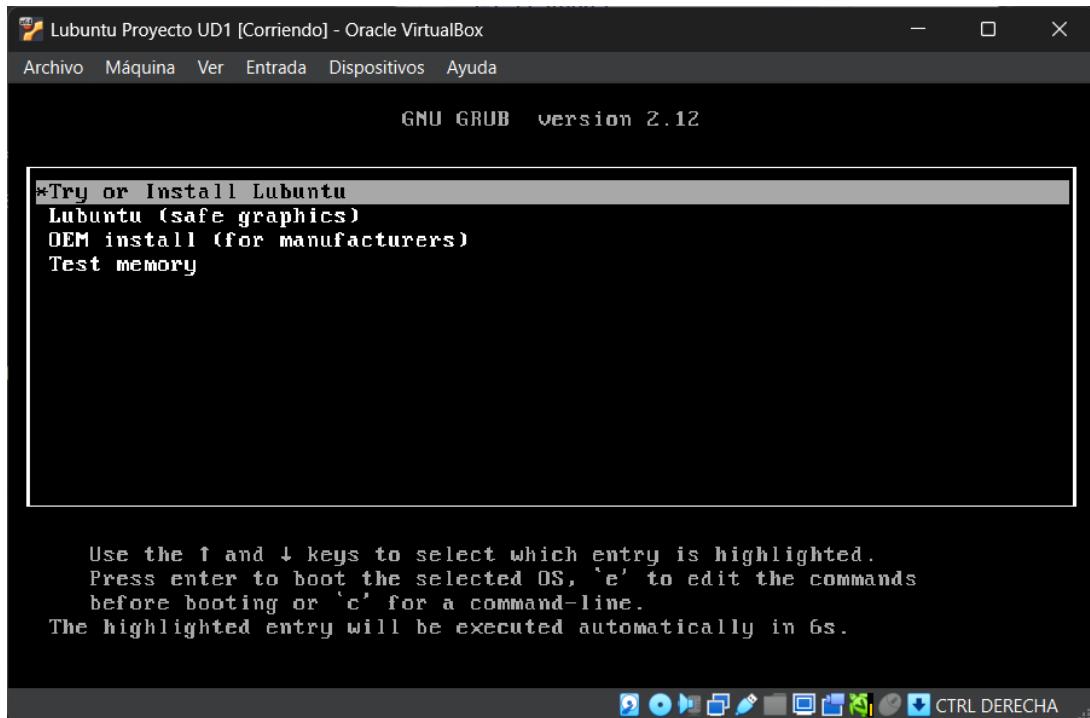


Figura A.2: Menú de arranque (GRUB) de la ISO de Lubuntu.



Figura A.3: Inicio del instalador de Lubuntu.

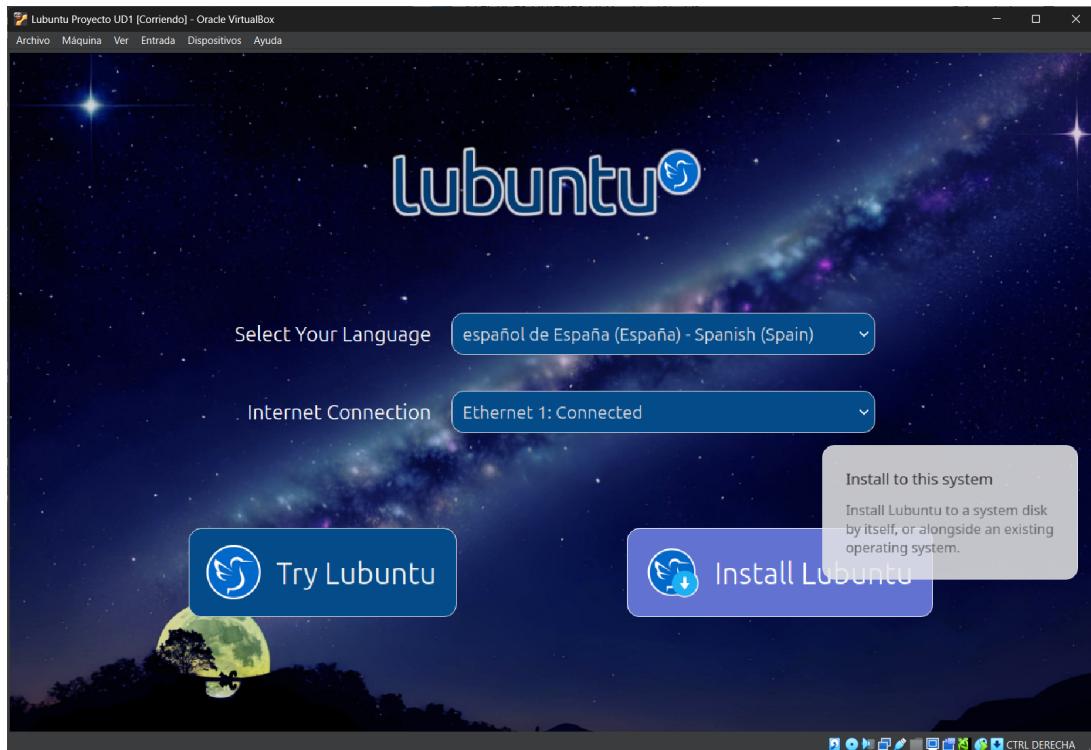


Figura A.4: Selección de "Install Lubuntu".

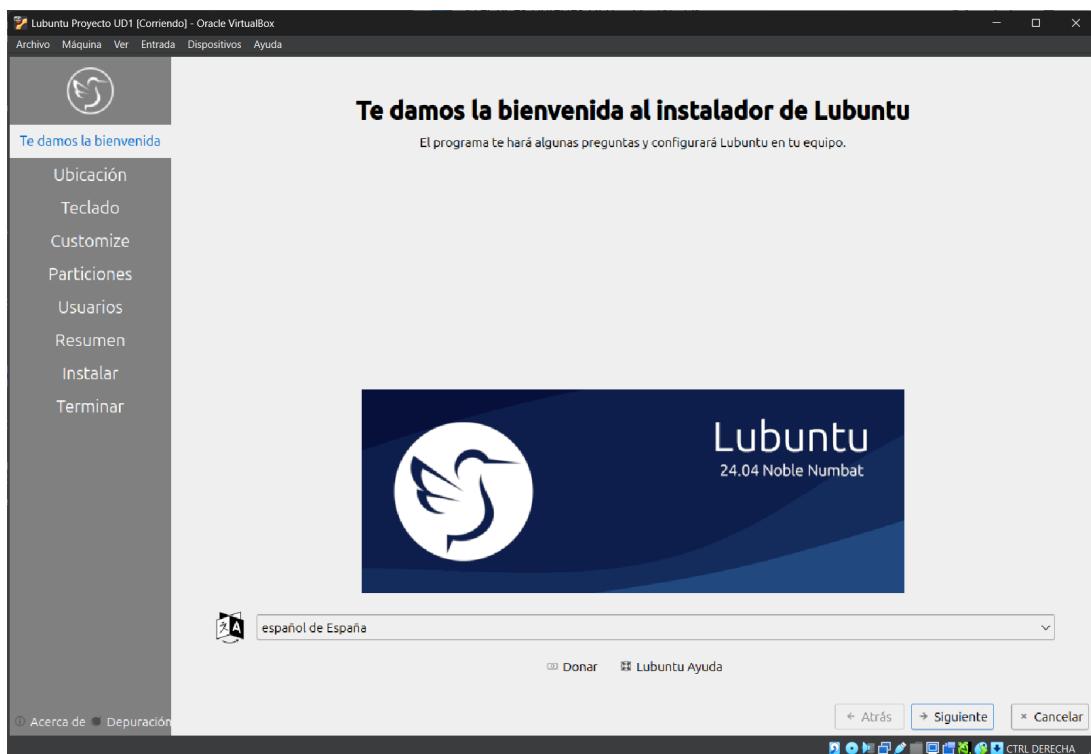


Figura A.5: Selección de idioma (Español).

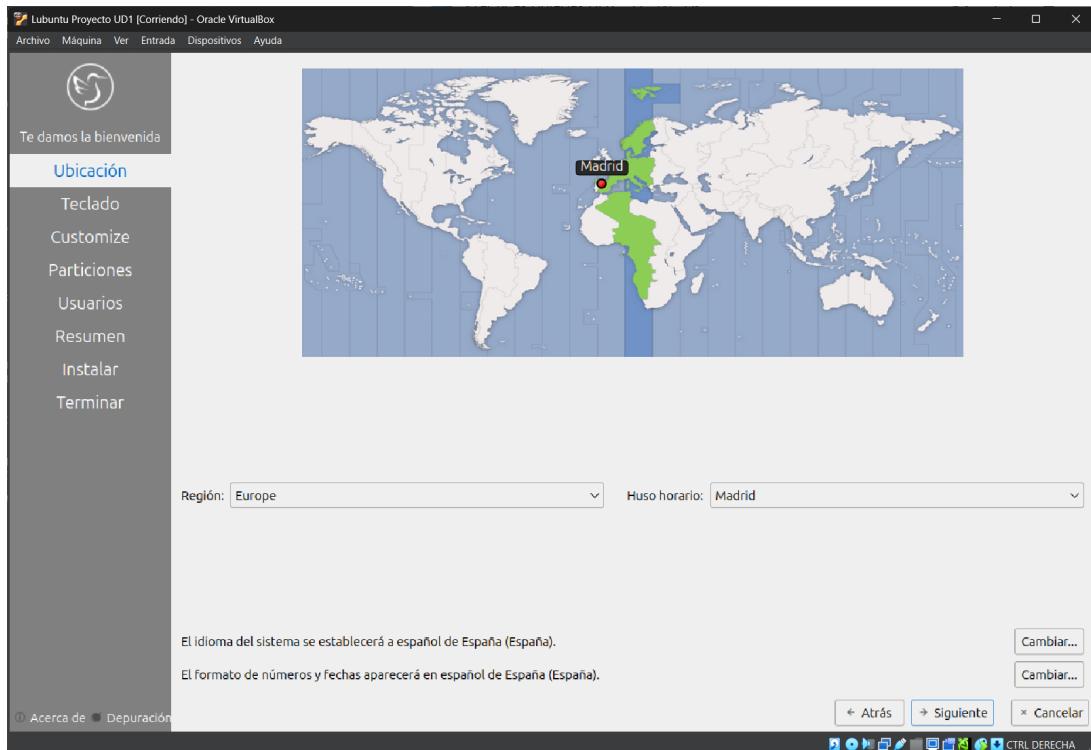


Figura A.6: Selección de ubicación (Madrid).

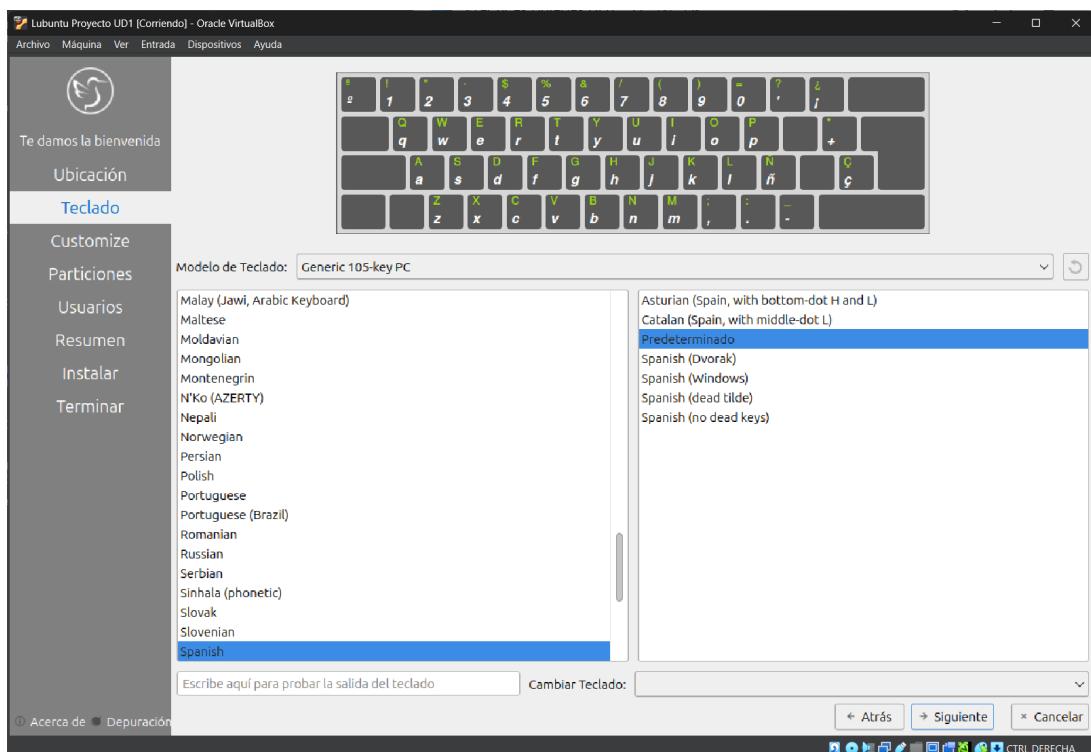


Figura A.7: Selección de teclado (Spanish).

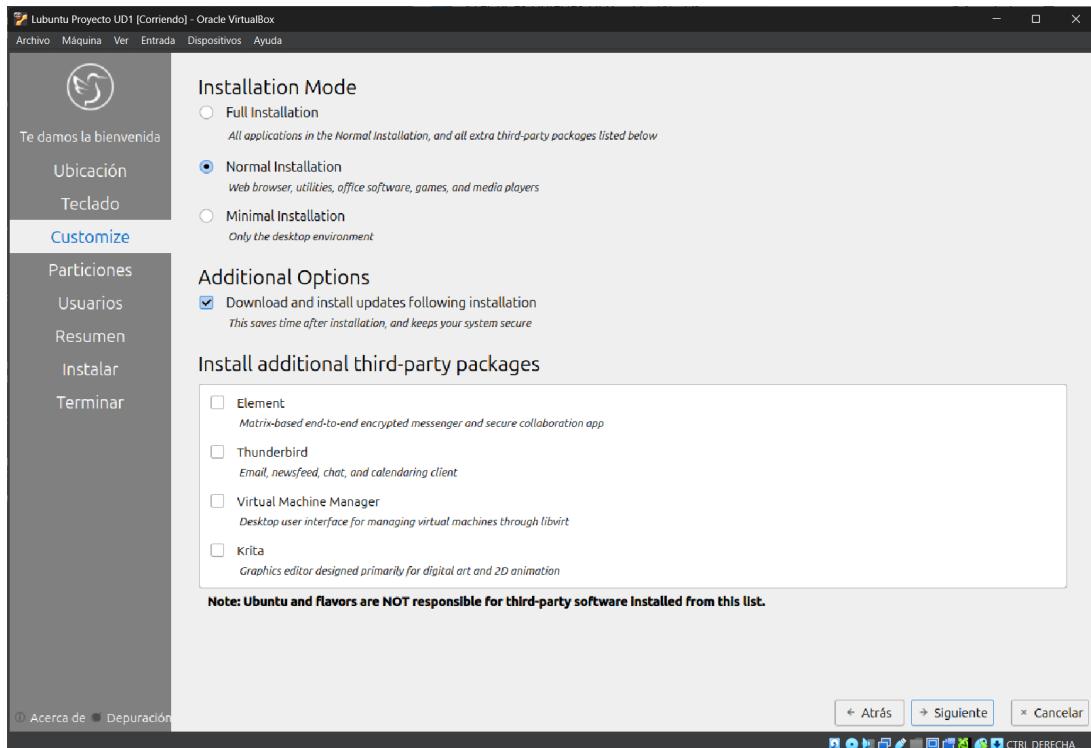


Figura A.8: Selección de tipo de instalación (Normal).

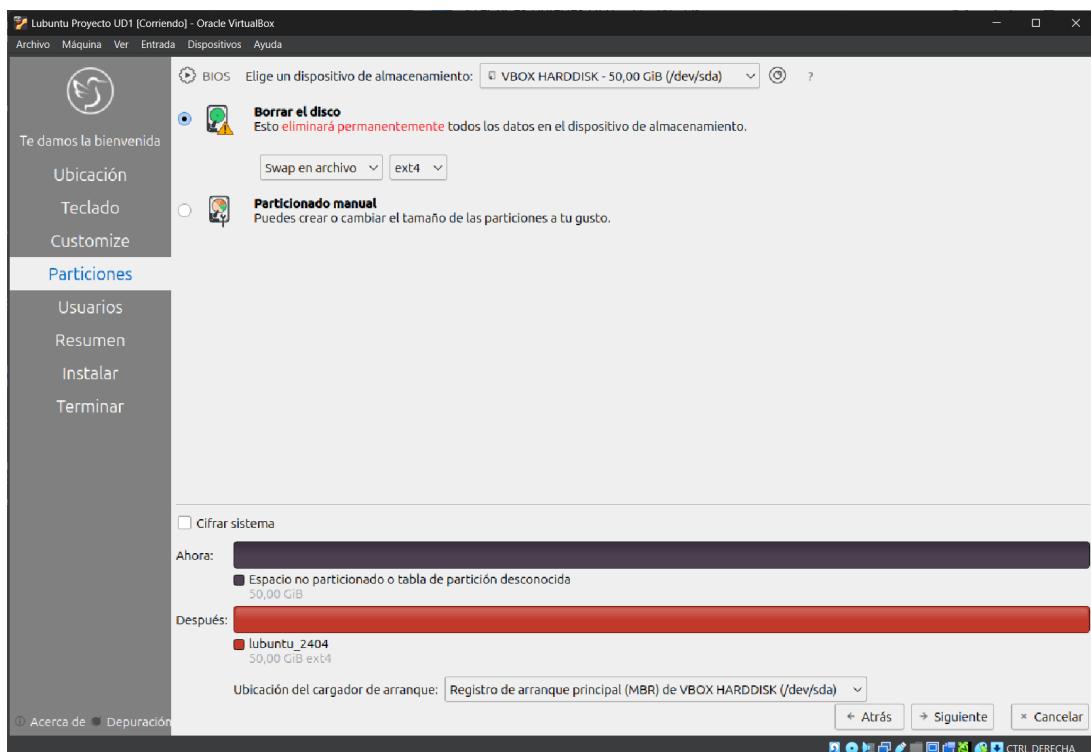


Figura A.9: Configuración de particiones (Borrar disco).

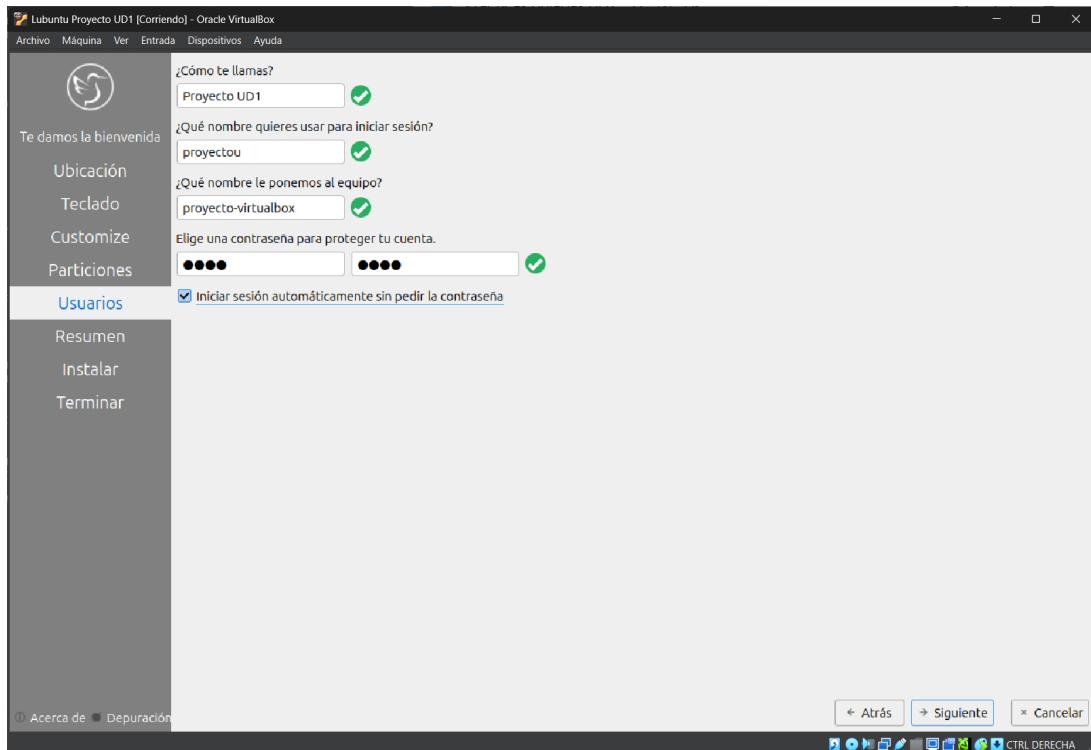


Figura A.10: Creación del usuario y nombre de equipo.

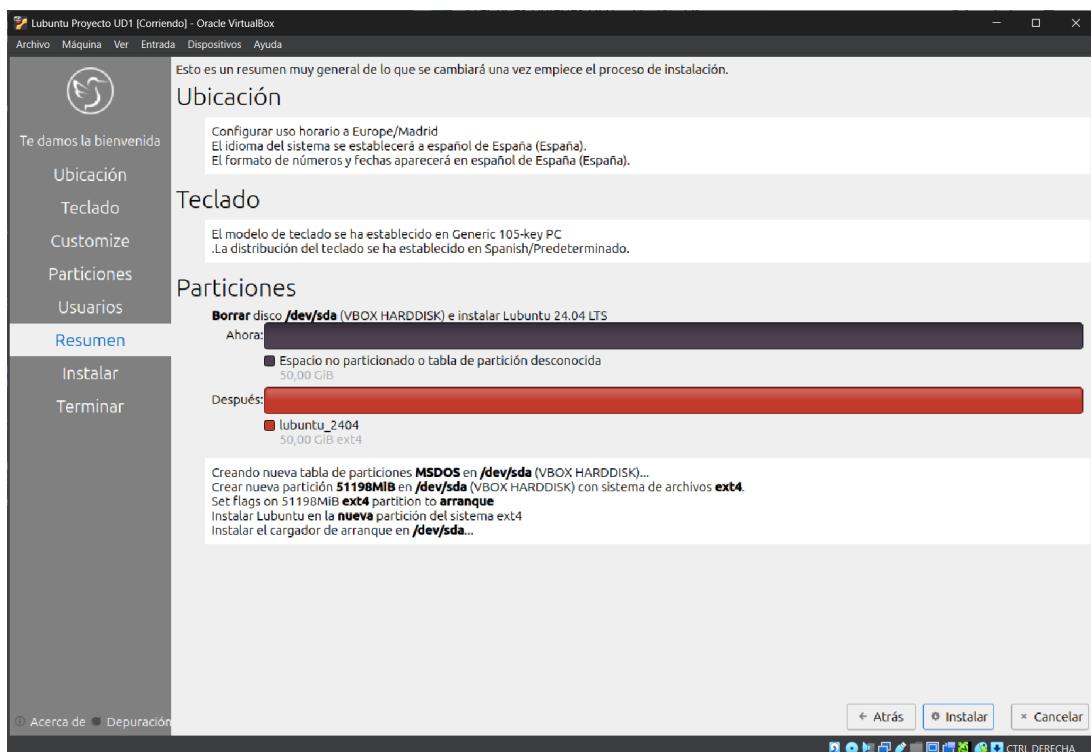


Figura A.11: Resumen de la instalación.

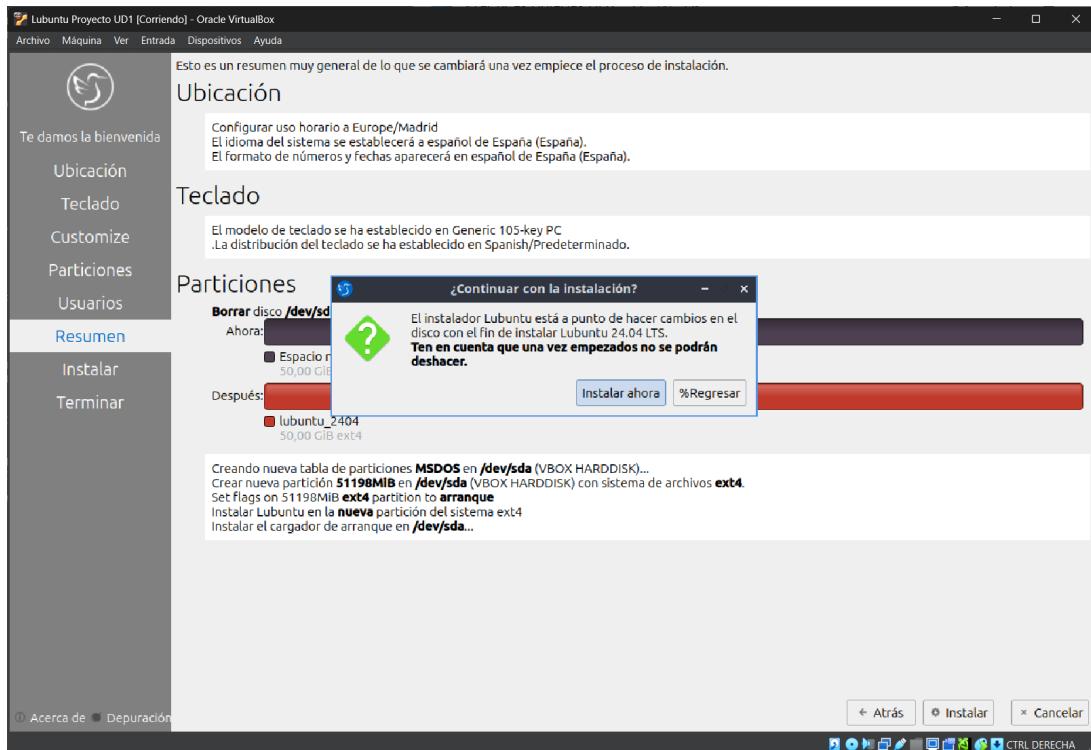


Figura A.12: Confirmación de inicio de instalación.

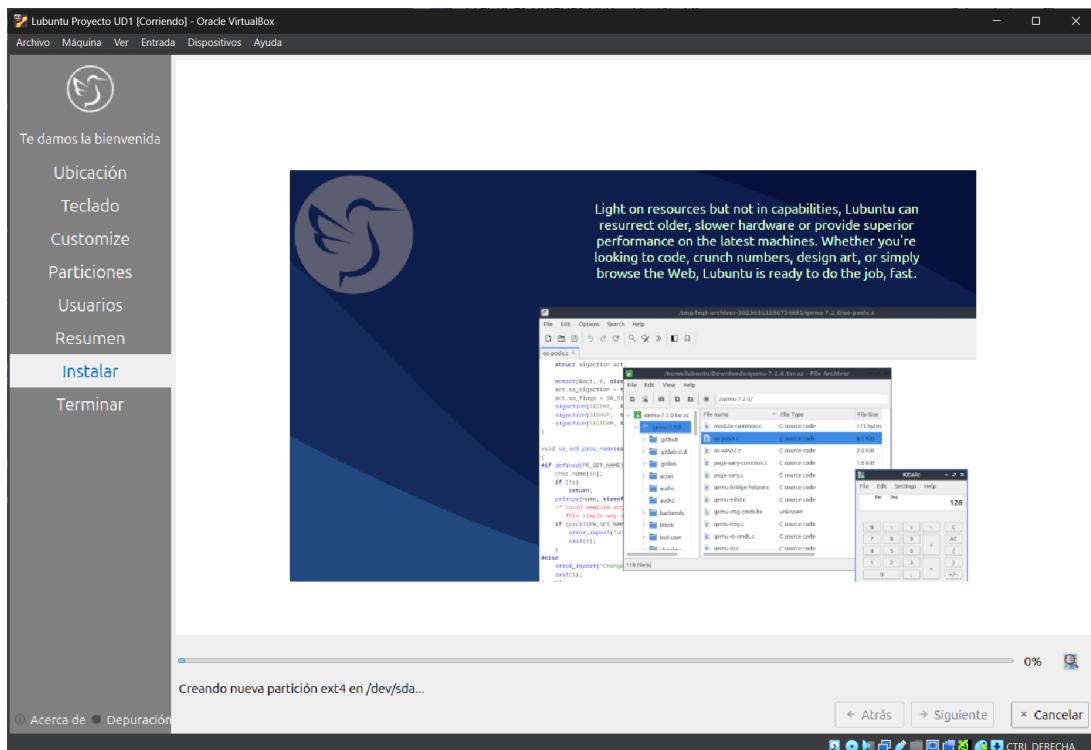


Figura A.13: Proceso de instalación.

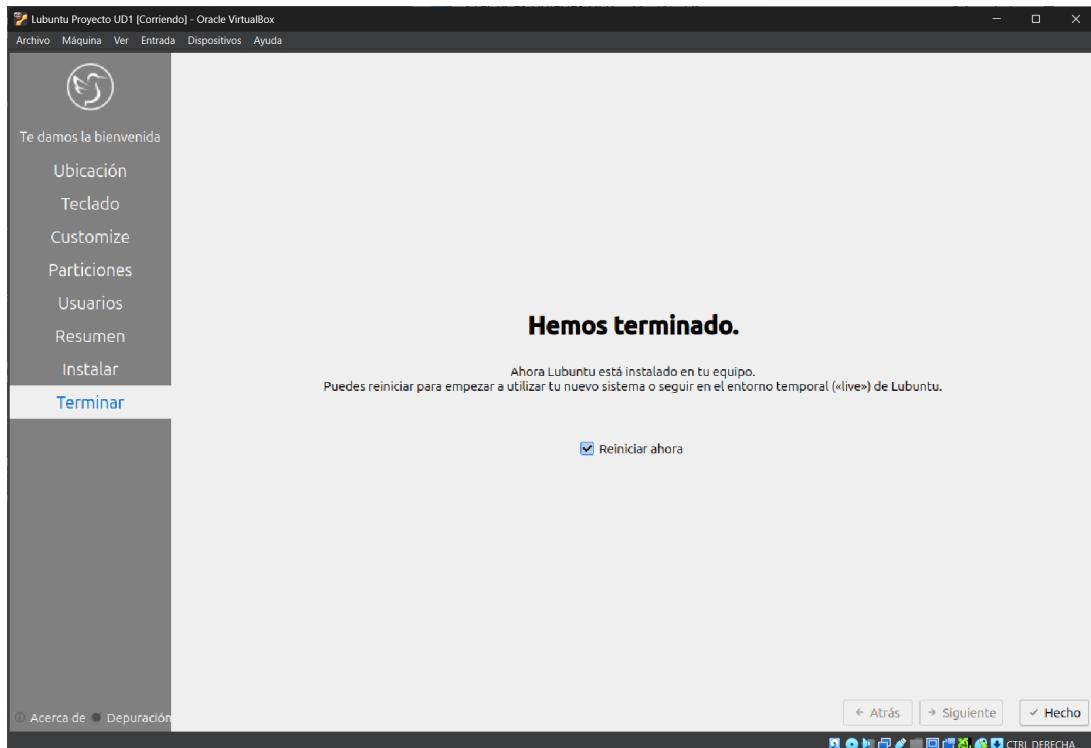


Figura A.14: Instalación finalizada.

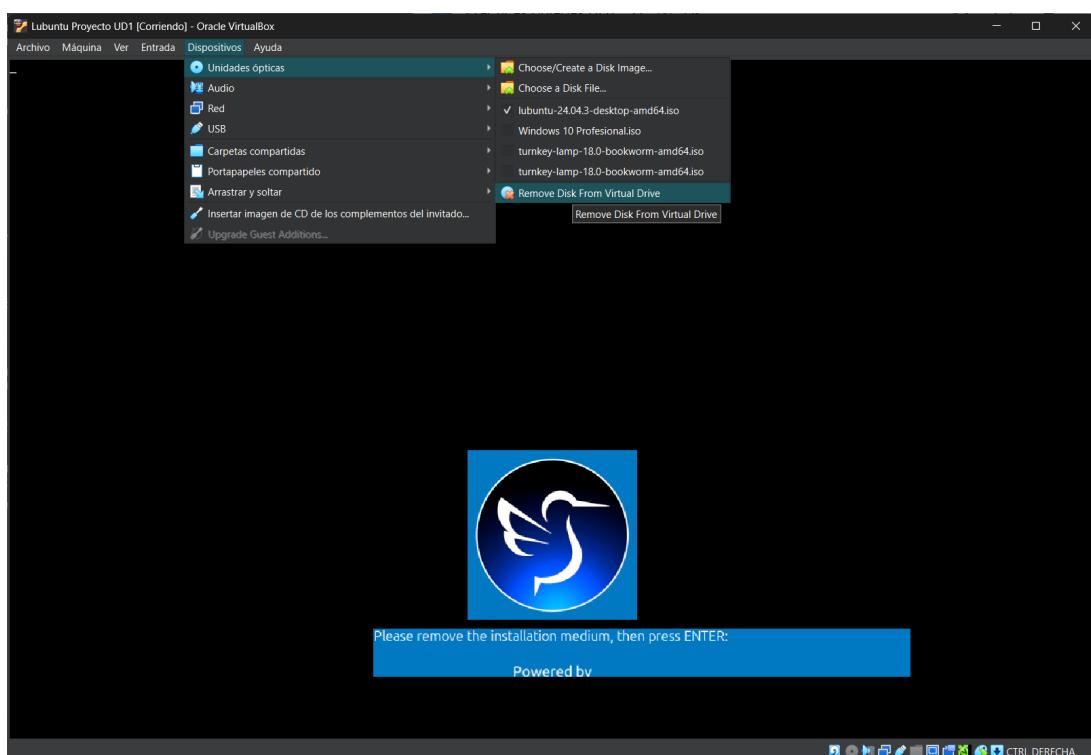


Figura A.15: Extracción del disco de instalación virtual al reiniciar.

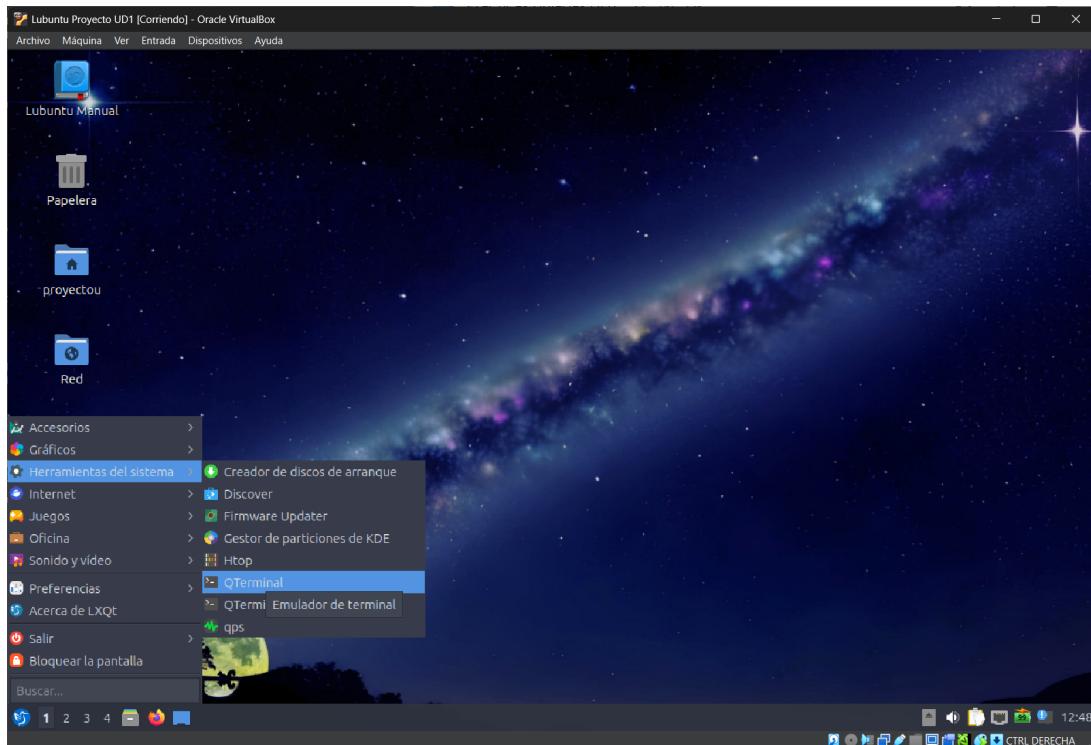


Figura A.16: Escritorio de Lubuntu y apertura de QTerminal.

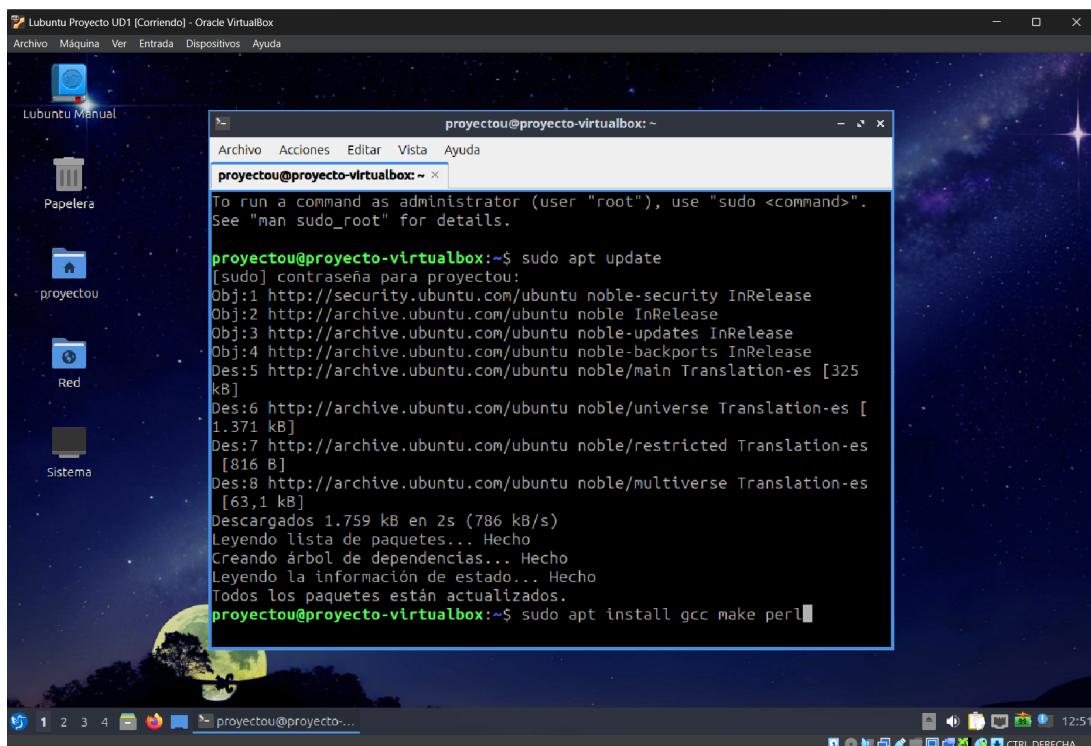


Figura A.17: Actualización de paquetes ('sudo apt update').

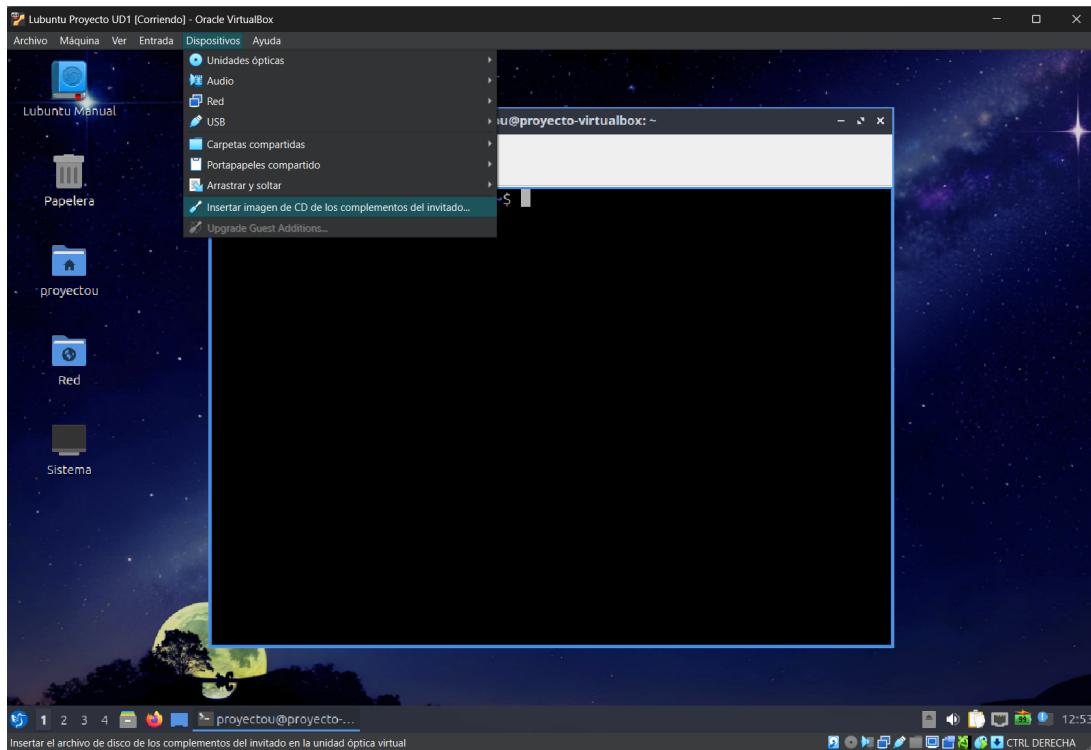


Figura A.18: Insertando imagen de CD de las "Guest Additions"de VirtualBox.

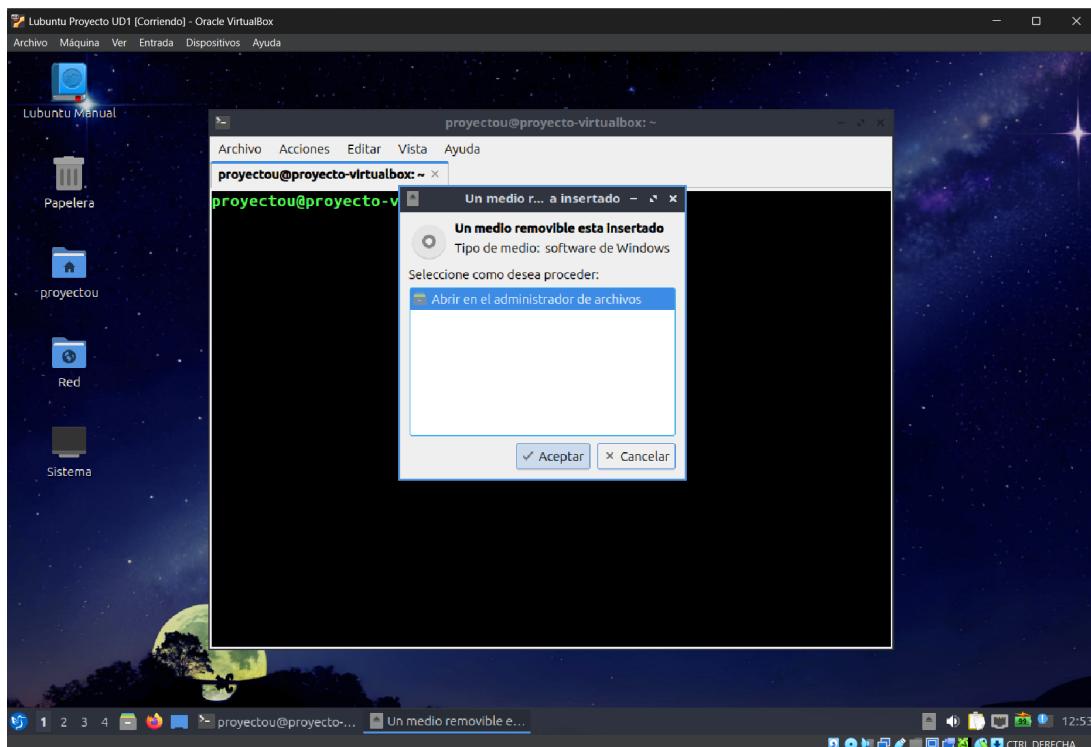


Figura A.19: Aviso de medio extraíble (Guest Additions).

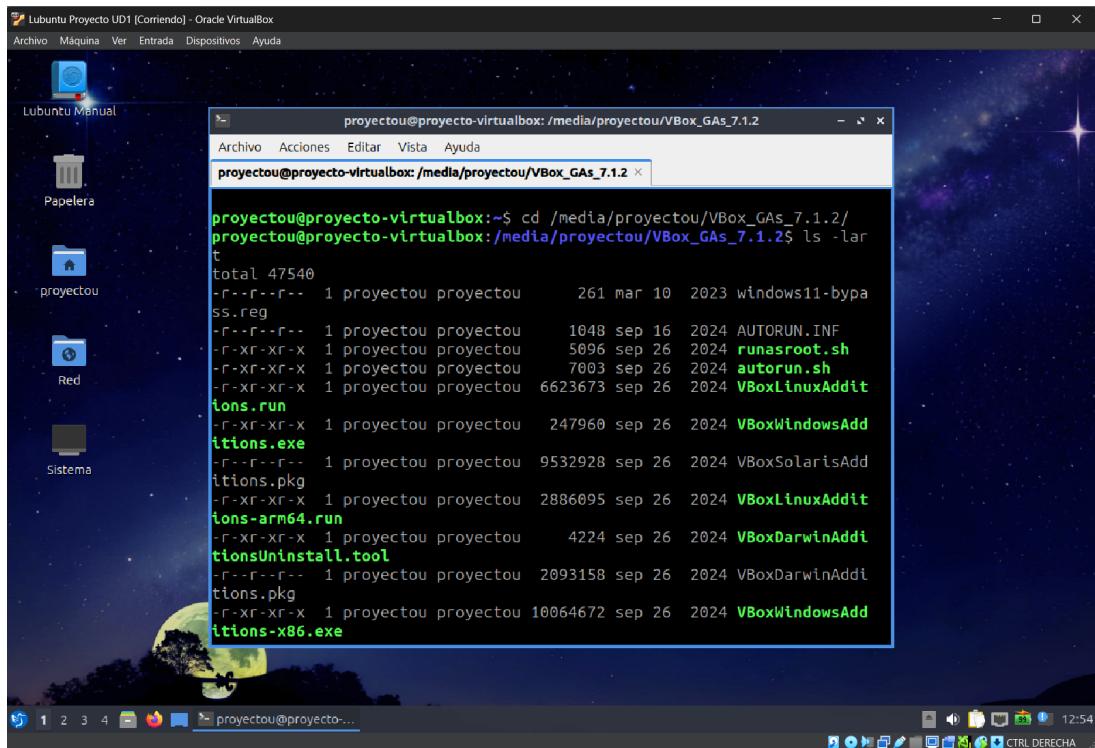


Figura A.20: Navegación al directorio de las Guest Additions.

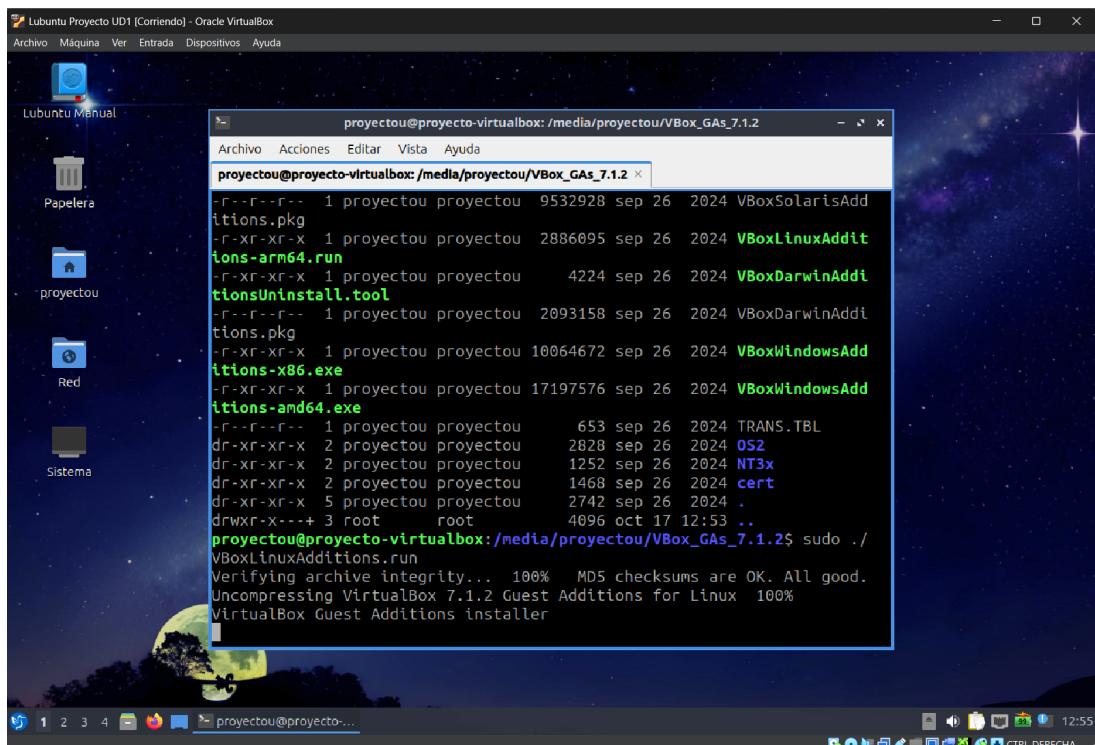


Figura A.21: Ejecución del script de instalación VBoxLinuxAdditions.run.

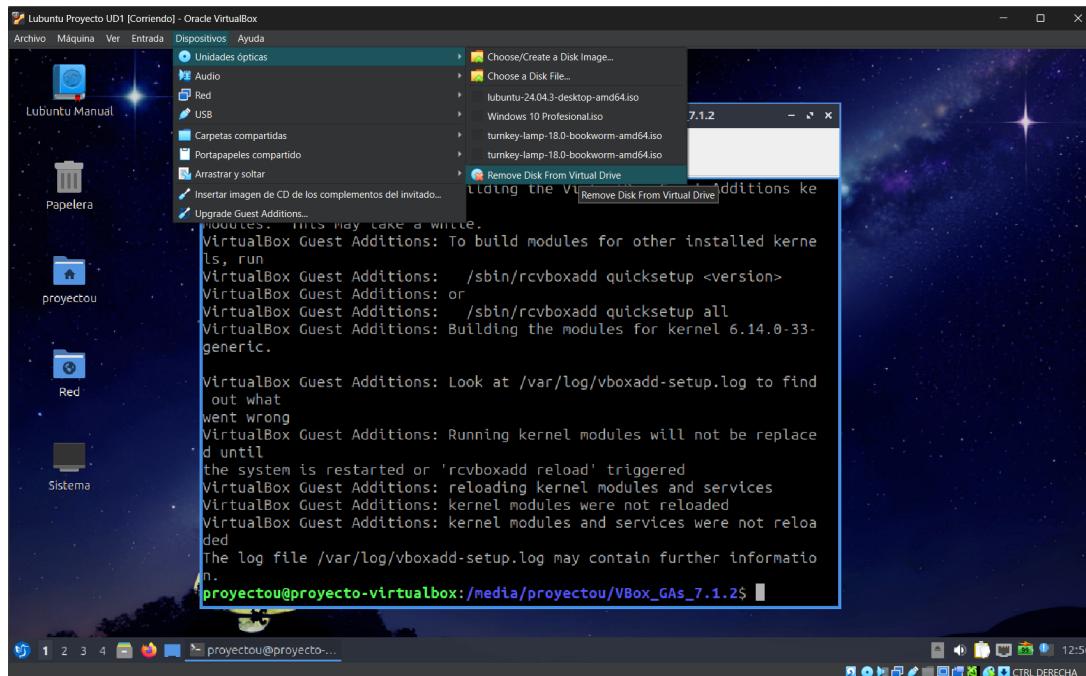


Figura A.22: Extracción del disco virtual de las Guest Additions.

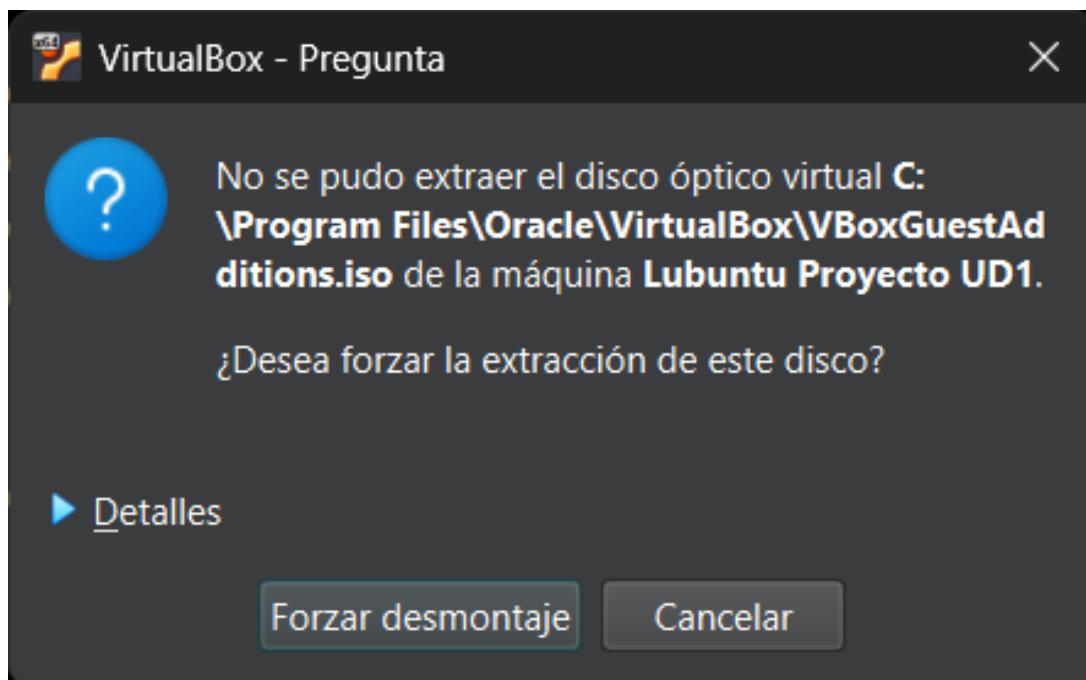


Figura A.23: Aviso de forzar desmontaje del disco óptico.

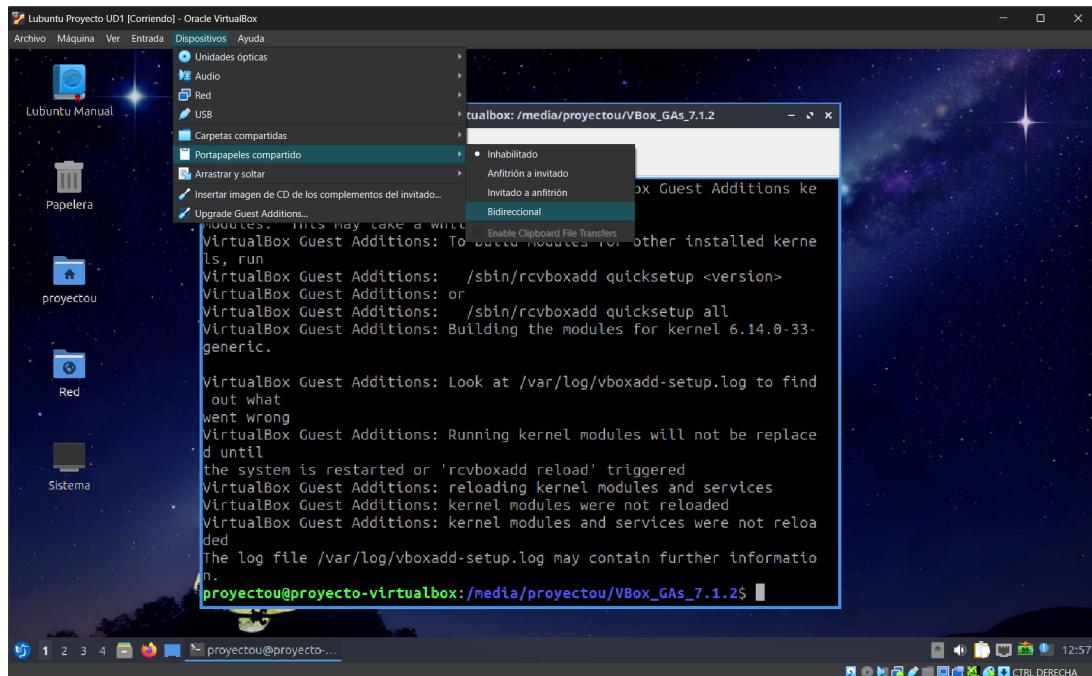


Figura A.24: Habilitando portapapeles bidireccional en VirtualBox.

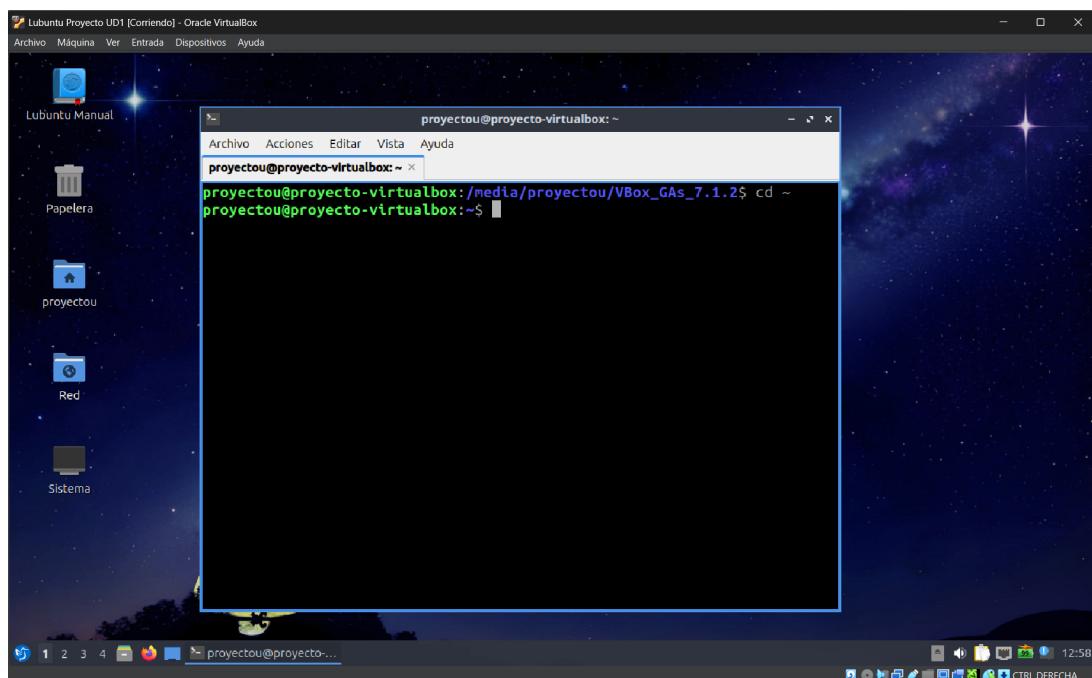


Figura A.25: Terminal de Lubuntu post-reinicio.

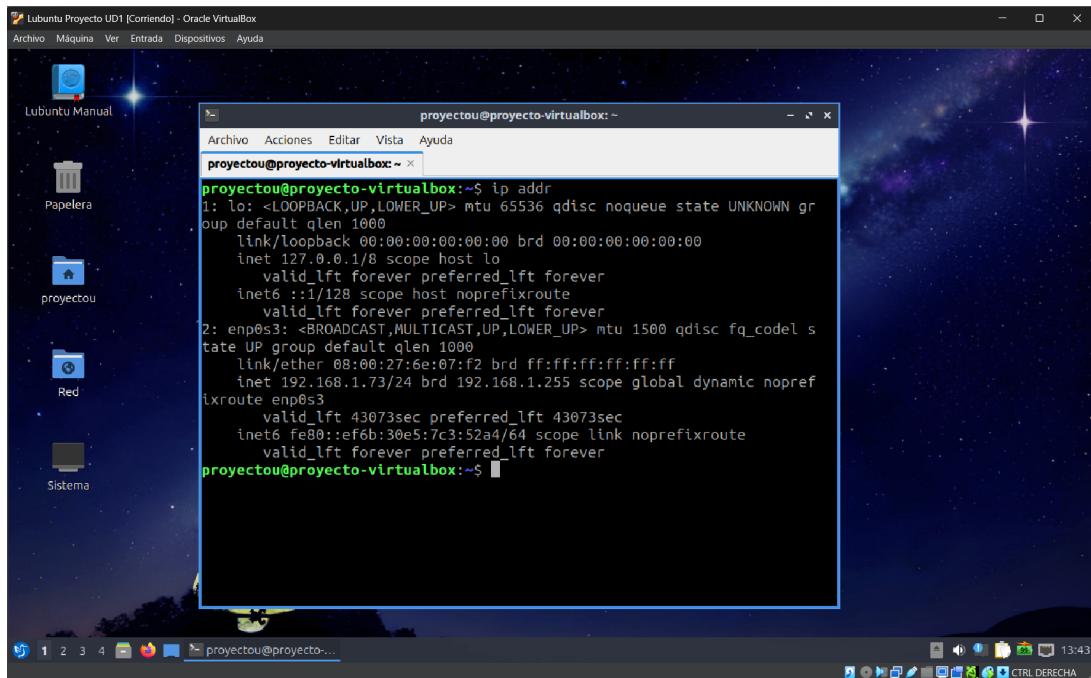


Figura A.26: Comprobación de la dirección IP de la MV ('ip addr') - 192.168.1.73.

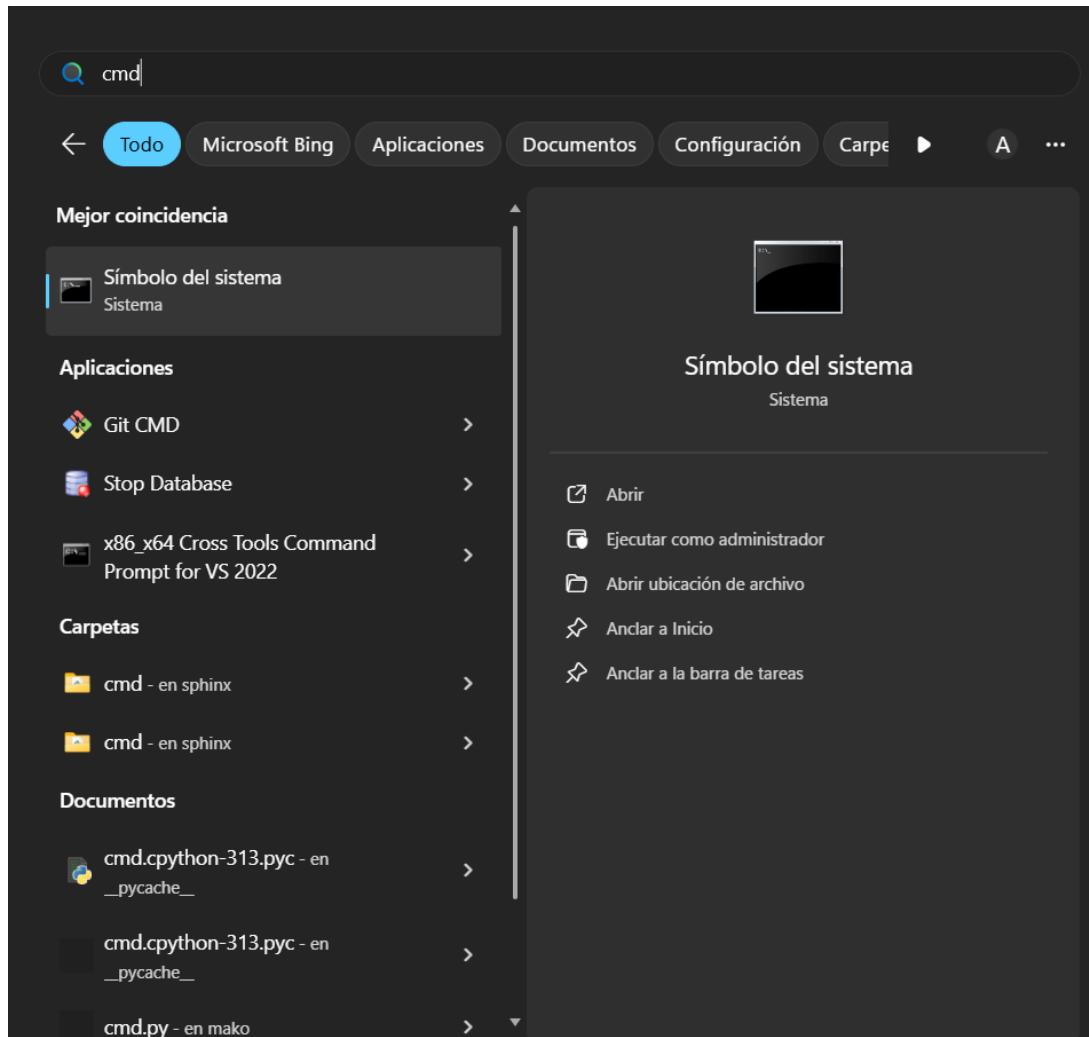
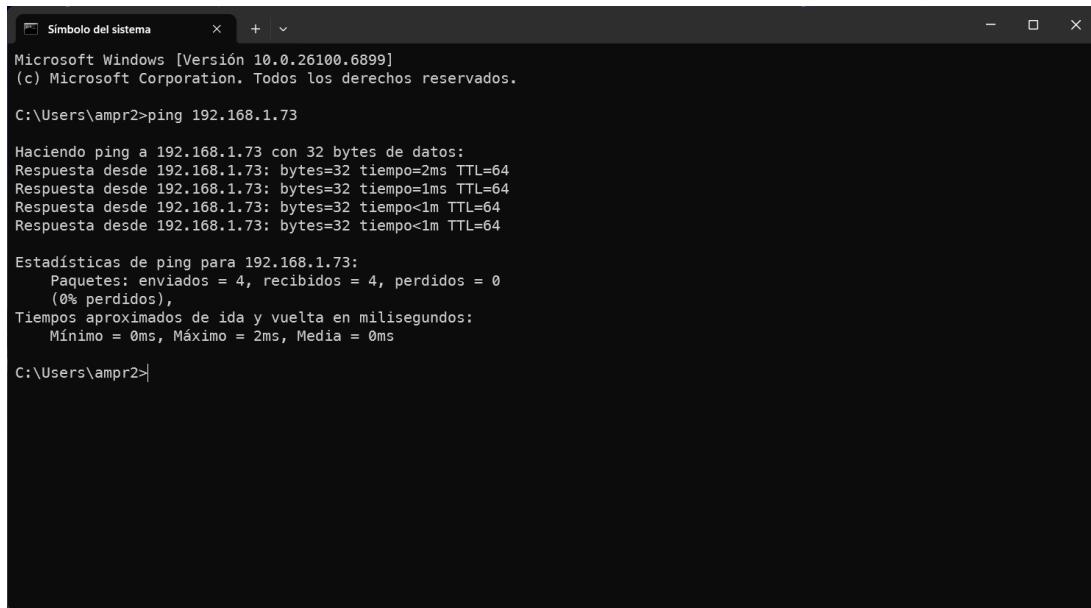


Figura A.27: Búsqueda de Símbolo del sistema (CMD) en el anfitrión (Windows).



```
Símbolo del sistema
Microsoft Windows [Versión 10.0.26100.6899]
(c) Microsoft Corporation. Todos los derechos reservados.

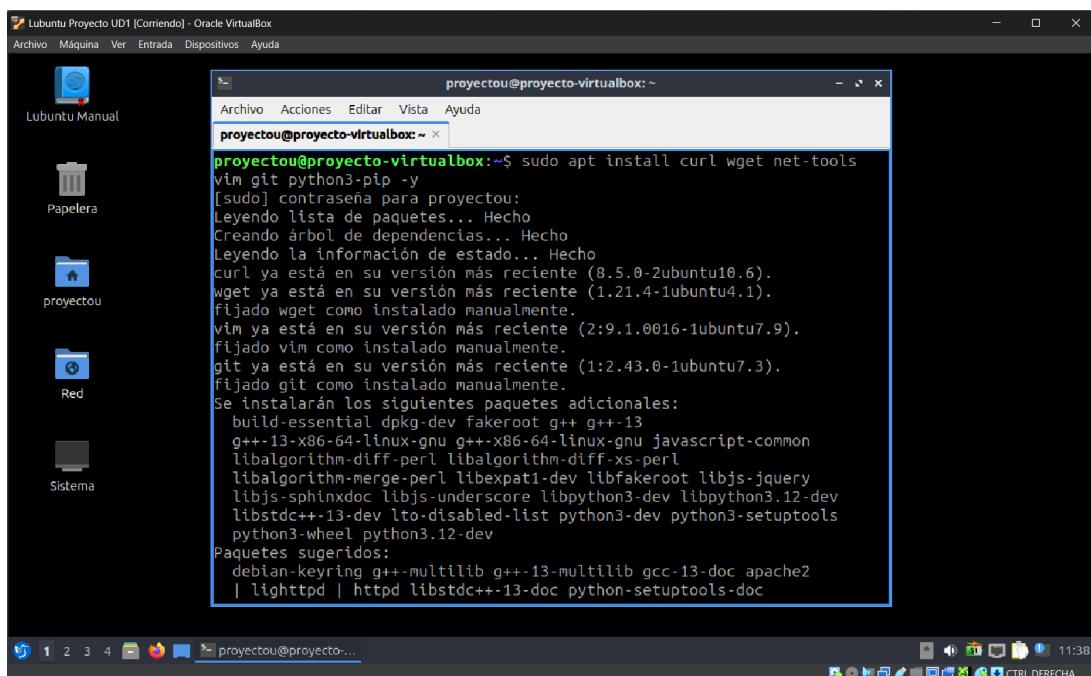
C:\Users\ampr2>ping 192.168.1.73

Haciendo ping a 192.168.1.73 con 32 bytes de datos:
Respuesta desde 192.168.1.73: bytes=32 tiempo=2ms TTL=64
Respuesta desde 192.168.1.73: bytes=32 tiempo=1ms TTL=64
Respuesta desde 192.168.1.73: bytes=32 tiempo<1ms TTL=64
Respuesta desde 192.168.1.73: bytes=32 tiempo<1ms TTL=64

Estadísticas de ping para 192.168.1.73:
    Paquetes: enviados = 4, recibidos = 4, perdidos = 0
                (0% perdidos),
    Tiempos aproximados de ida y vuelta en milisegundos:
        Mínimo = 0ms, Máximo = 2ms, Media = 0ms

C:\Users\ampr2>
```

Figura A.28: Prueba de conectividad (ping) desde el anfitrión (Windows) a la MV.



```
Lubuntu Proyecto UD1 [Corriendo] - Oracle VirtualBox
Archivo Máquina Ver Entrada Dispositivos Ayuda

proyecto@proyecto-virtualbox: ~
Archivo Acciones Editar Vista Ayuda
proyecto@proyecto-virtualbox: ~ | x
proyecto@proyecto-virtualbox:~$ sudo apt install curl wget net-tools
vim git python3-pip -y
[sudo] contraseña para proyecto:
Leyendo lista de paquetes... Hecho
Creando árbol de dependencias... Hecho
Leyendo la información de estado... Hecho
curl ya está en su versión más reciente (8.5.0-2ubuntu10.6).
wget ya está en su versión más reciente (1.21.4-1ubuntu4.1).
Fijado wget como instalado manualmente.
vim ya está en su versión más reciente (2:9.1.0016-1ubuntu7.9).
Fijado vim como instalado manualmente.
git ya está en su versión más reciente (1:2.43.0-1ubuntu7.3).
Fijado git como instalado manualmente.
Se instalarán los siguientes paquetes adicionales:
  build-essential dpkg-dev fakeroot g++ g++-13
  g++-13-x86-64-linux-gnu g++-x86-64-linux-gnu javascript-common
  libalgorithm-diff-perl libalgorithm-diff-xs-perl
  libalgorithm-merge-perl libexpat1-dev libfakeroot libjs-jquery
  libjs-sphinxdoc libjsunderscore libpython3 dev libpython3.12-dev
  libstdc++-13-dev lto-disabled-list python3-dev python3-setuptools
  python3-wheel python3.12-dev
Paquetes sugeridos:
  debian-keyring g++-multilib g++-13-multilib gcc-13-doc apache2
  | lighttpd | httpd libstdc++-13-doc python-setuptools-doc
```

Figura A.29: Instalación de paquetes base en Lubuntu ('curl', 'wget', 'git', 'python3-pip').

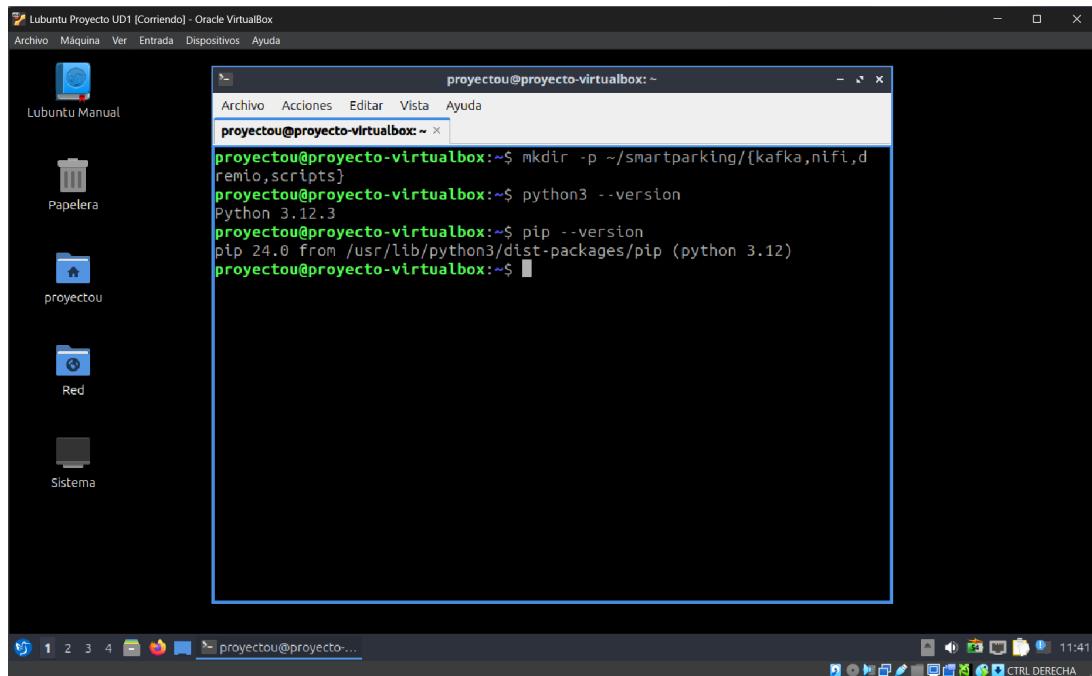


Figura A.30: Creación de directorios del proyecto y comprobación de versiones ('python3' y 'pip').

Apéndice B

Anexo B: Configuración de MongoDB Atlas

Proceso de creación de la cuenta, despliegue del cluster, configuración de la seguridad y creación de la base de datos y colecciones en MongoDB Atlas.

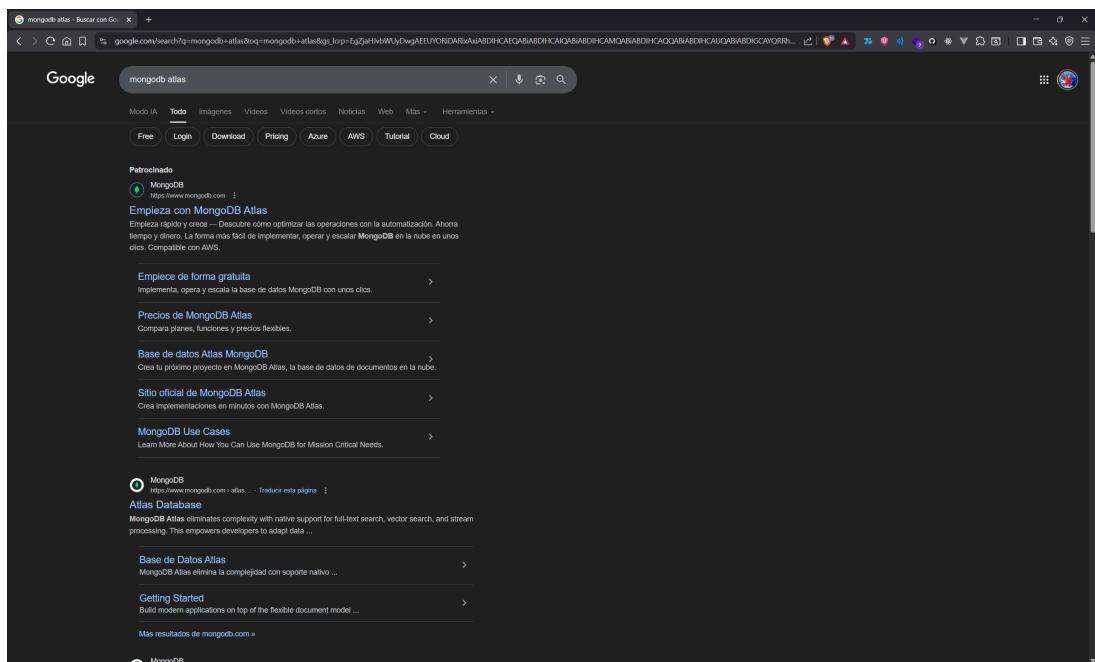


Figura B.1: Búsqueda inicial de MongoDB Atlas.

SmartParking Flow — Monitorización Inteligente

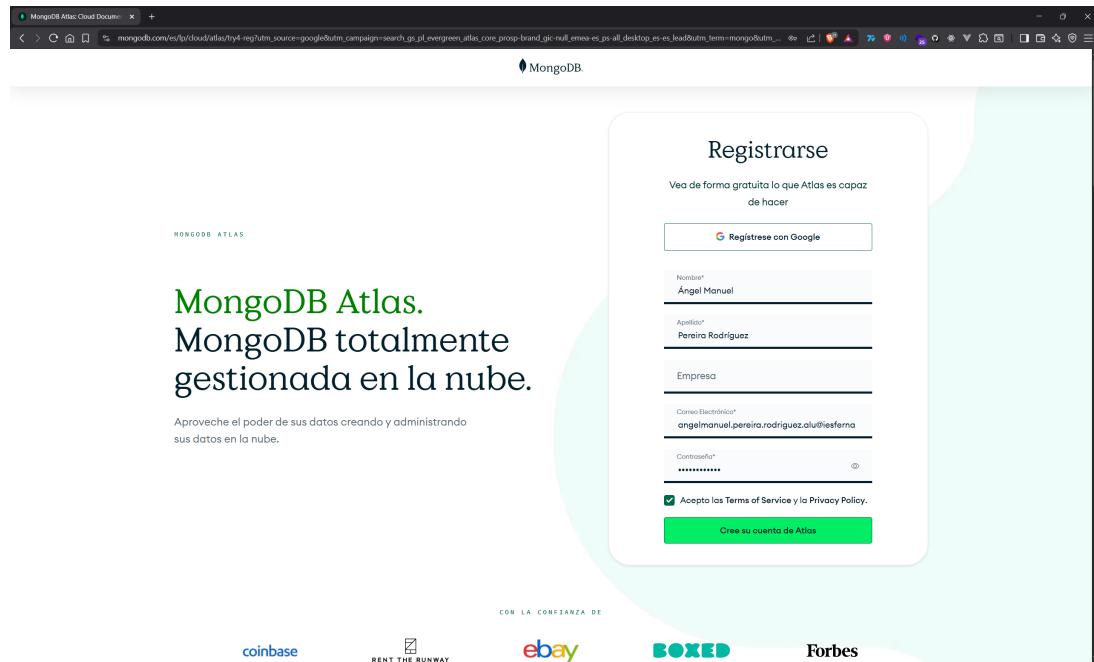


Figura B.2: Formulario de registro en MongoDB Atlas.

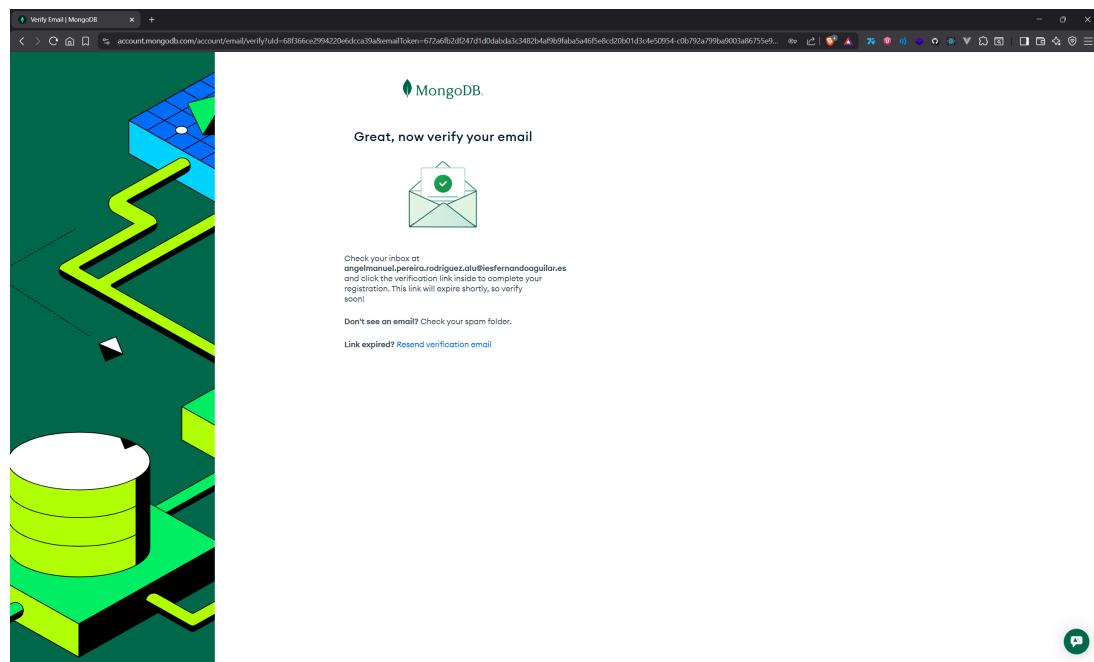


Figura B.3: Paso de verificación de correo electrónico.

SmartParking Flow — Monitorización Inteligente

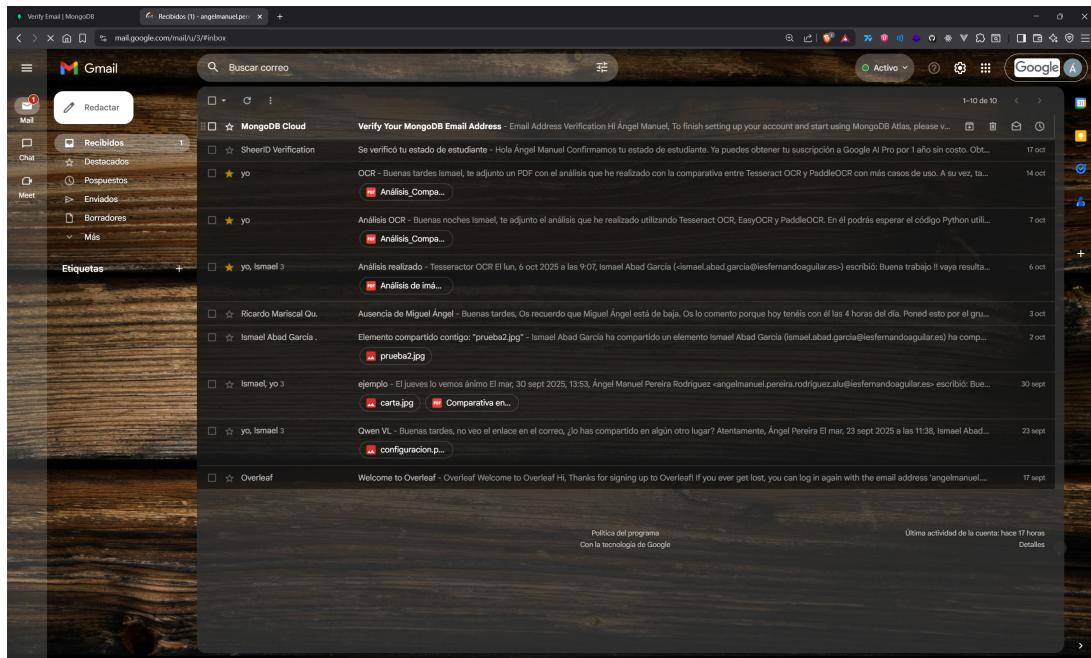


Figura B.4: Recepción del correo de verificación.

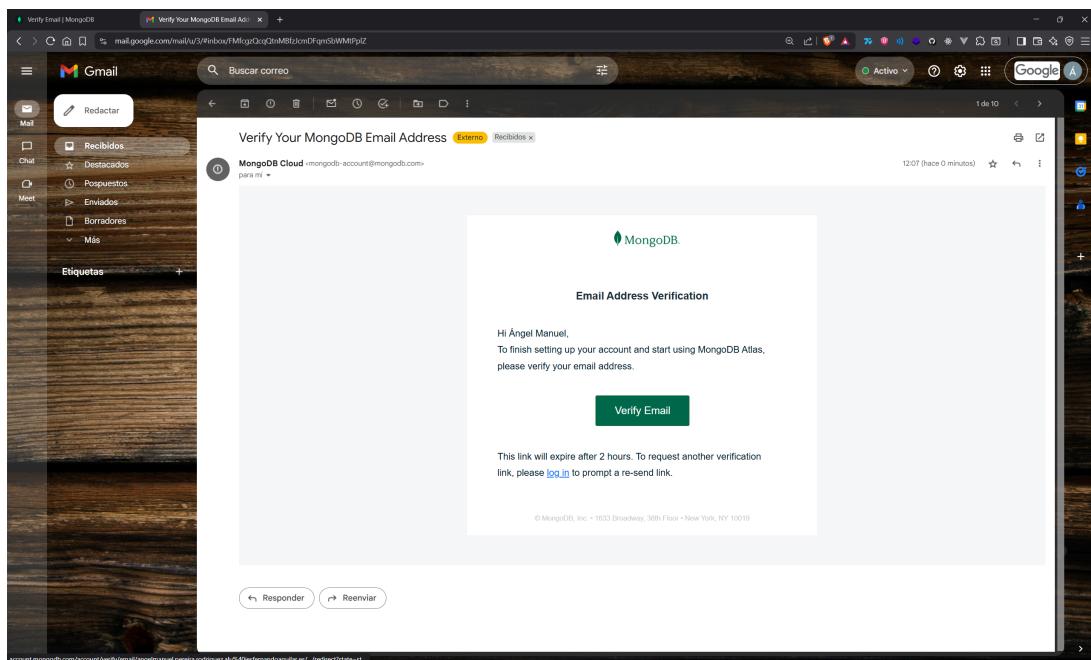


Figura B.5: Contenido del correo "Verify Your MongoDB Email Address".

SmartParking Flow — Monitorización Inteligente

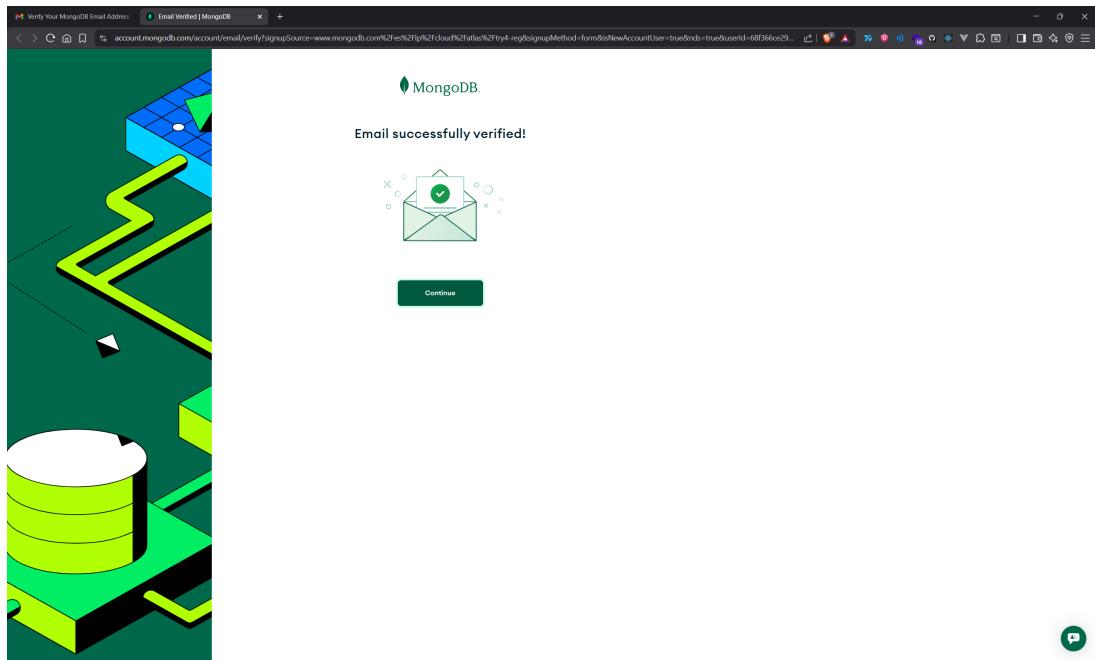


Figura B.6: Confirmación de email verificado.

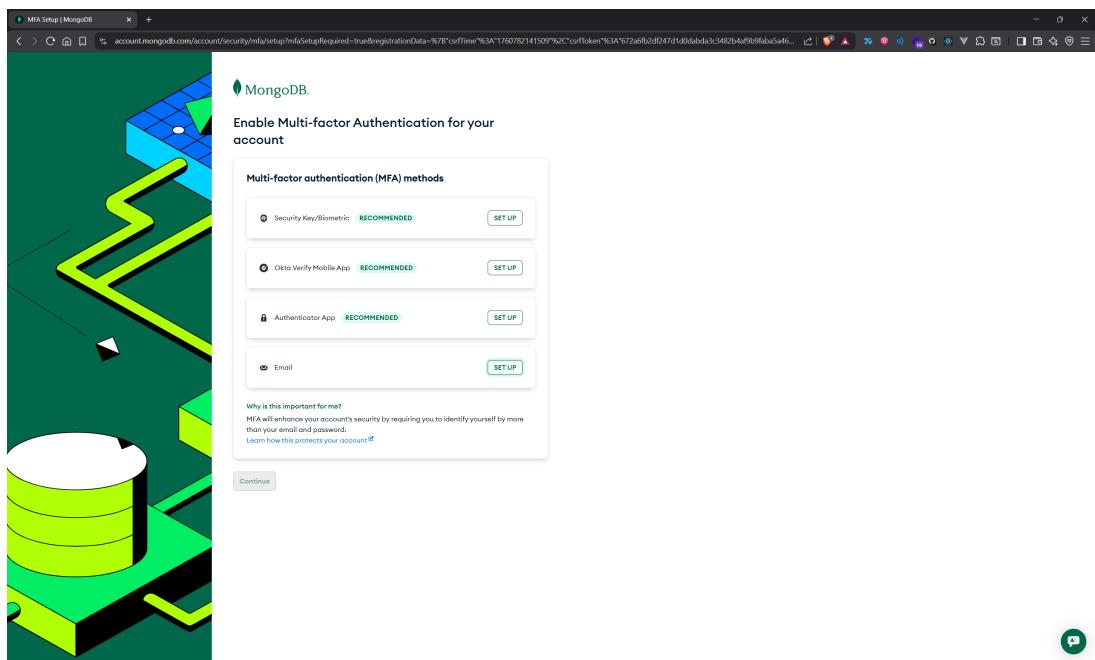


Figura B.7: Configuración opcional de Multi-Factor Authentication (MFA).

SmartParking Flow — Monitorización Inteligente

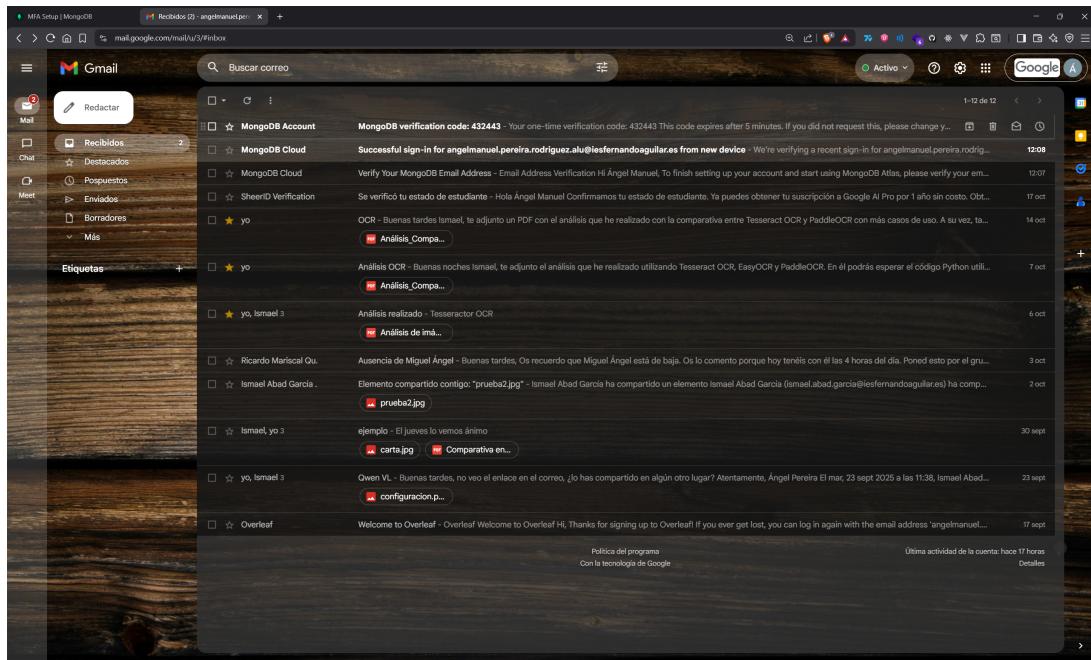


Figura B.8: Recepción del código de verificación de MongoDB.

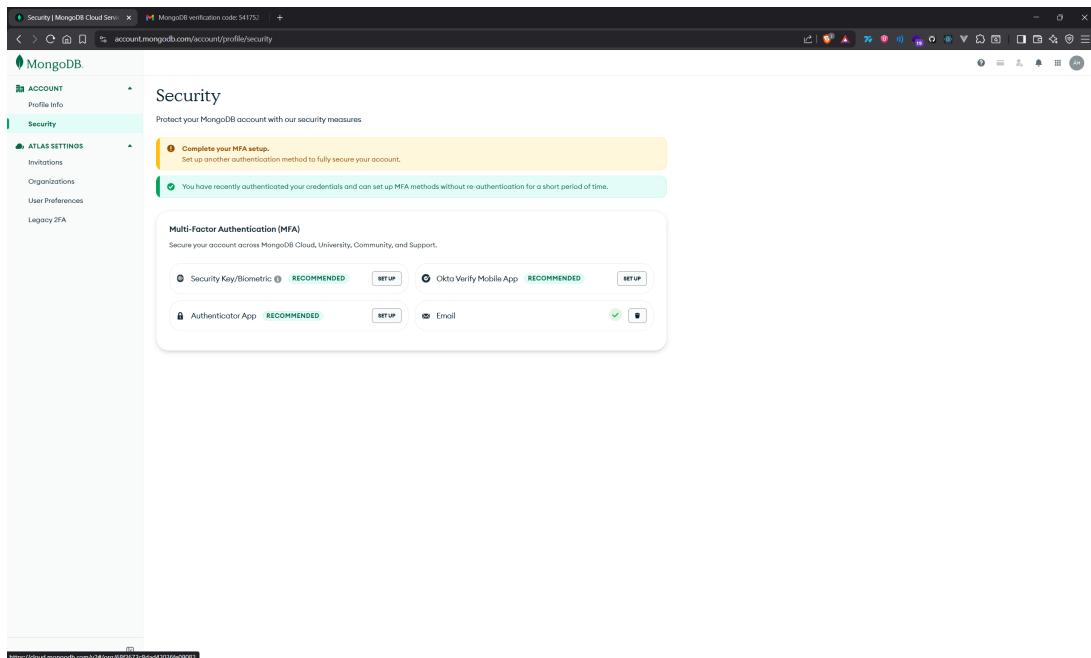


Figura B.9: Pantalla de configuración de seguridad de la cuenta.

SmartParking Flow — Monitorización Inteligente

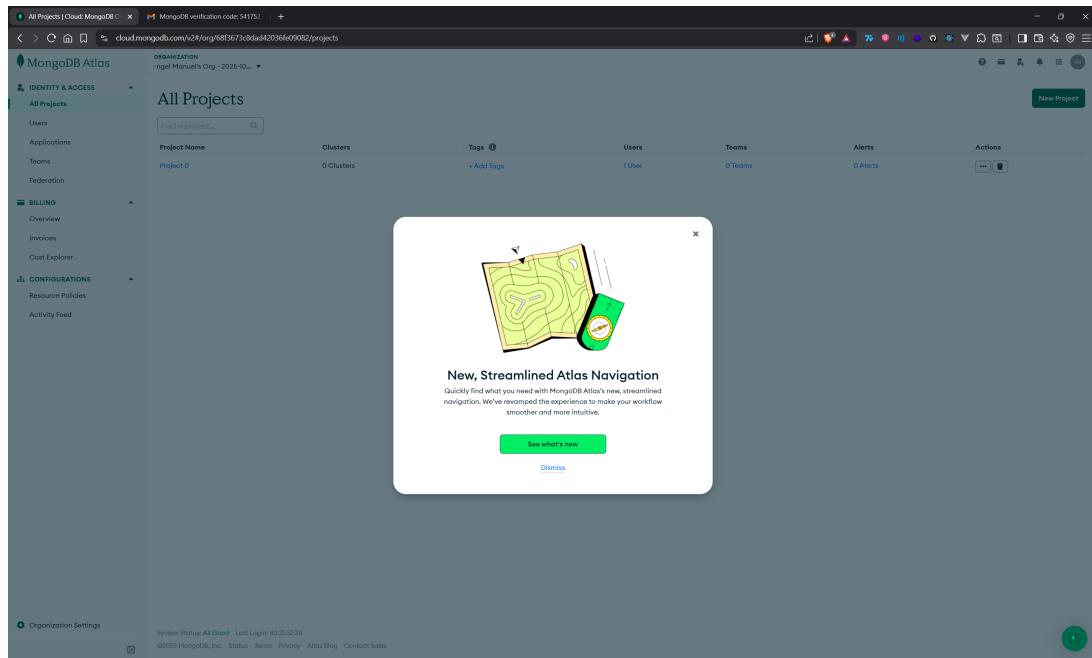


Figura B.10: Pantalla de bienvenida a la nueva navegación de Atlas.

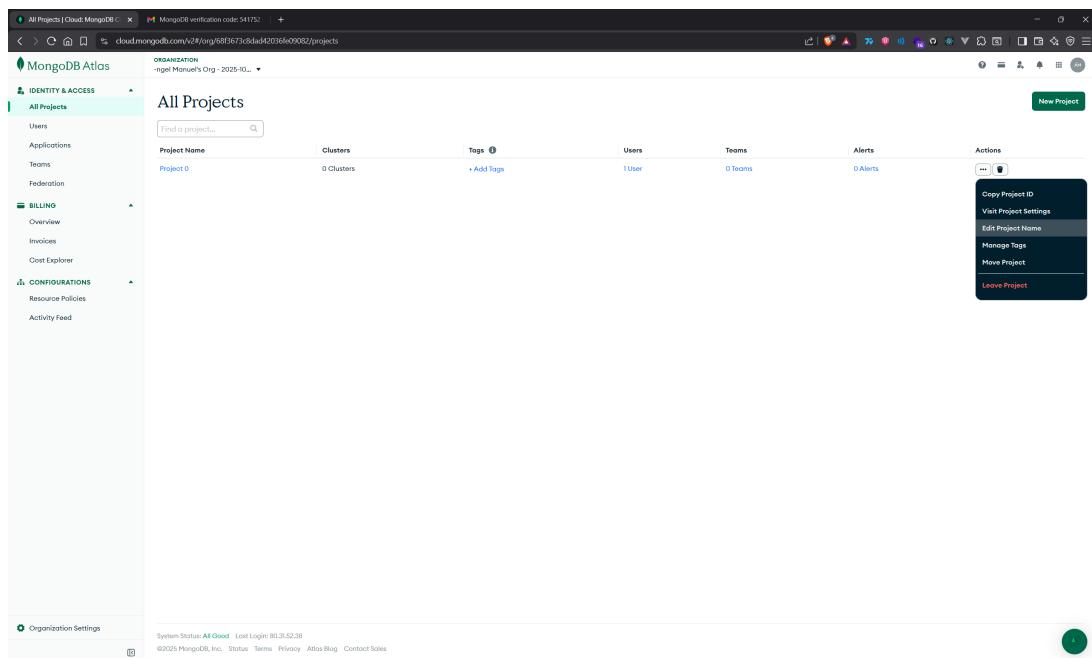


Figura B.11: Panel de .^All Projects^ inicial.

SmartParking Flow — Monitorización Inteligente

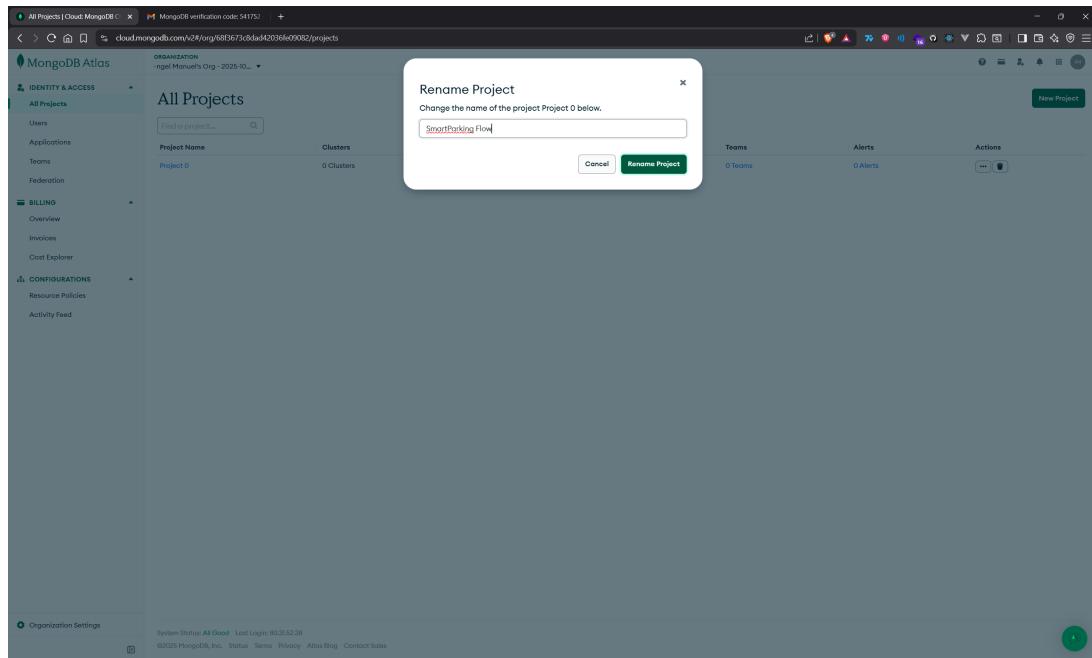


Figura B.12: Renombrando el proyecto a "SmartParking Flow".

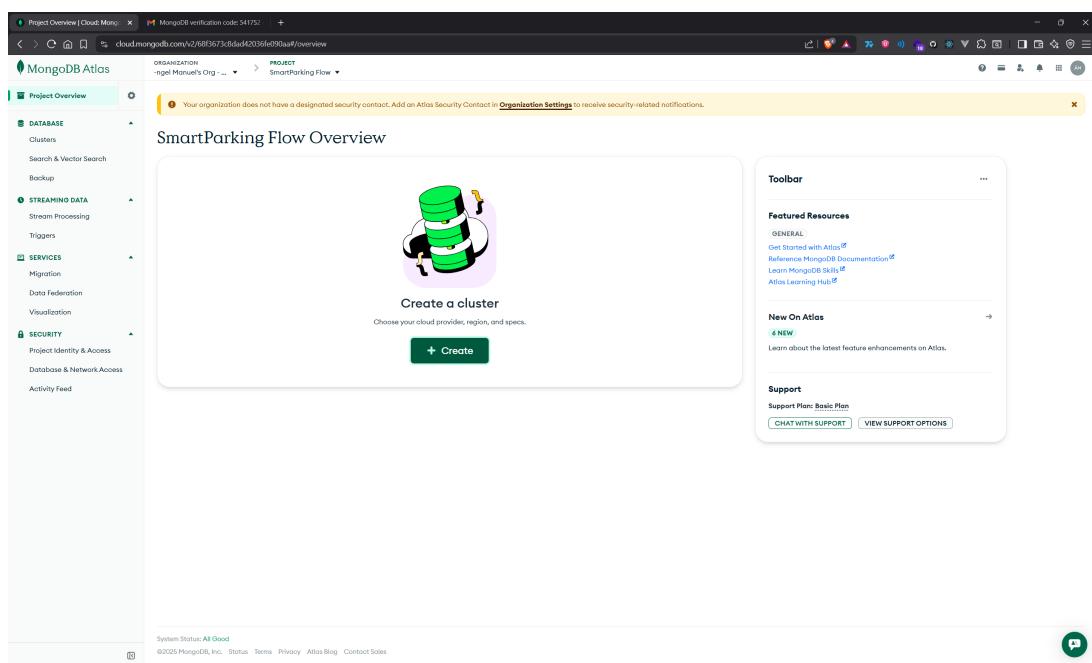


Figura B.13: Panel principal del proyecto "SmartParking Flow".

SmartParking Flow — Monitorización Inteligente

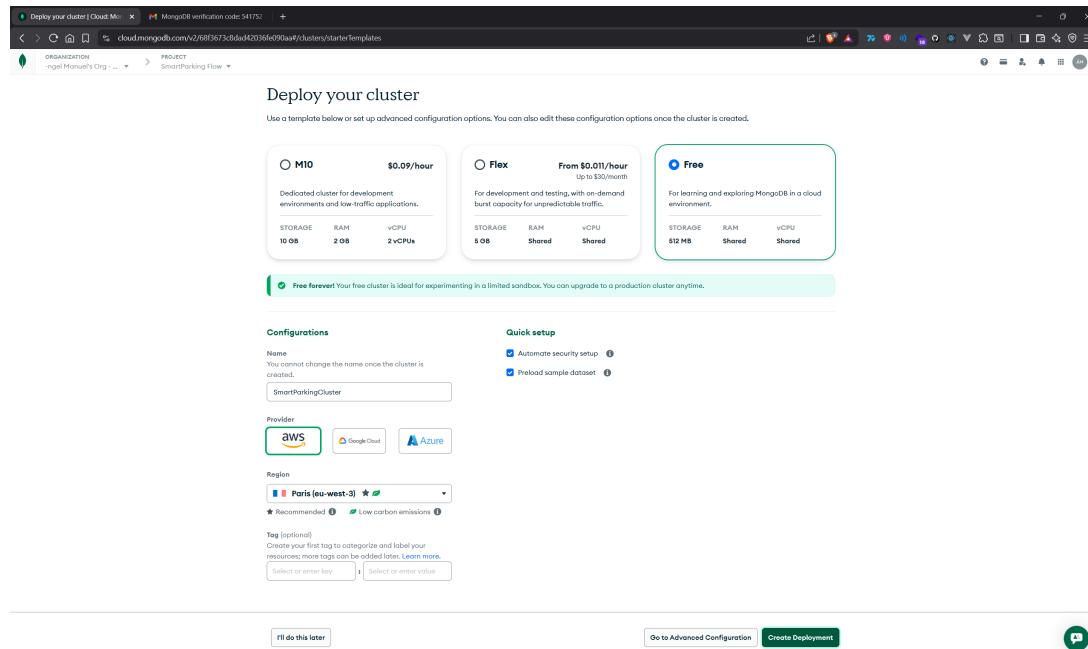


Figura B.14: Configuración del cluster gratuito (M0).

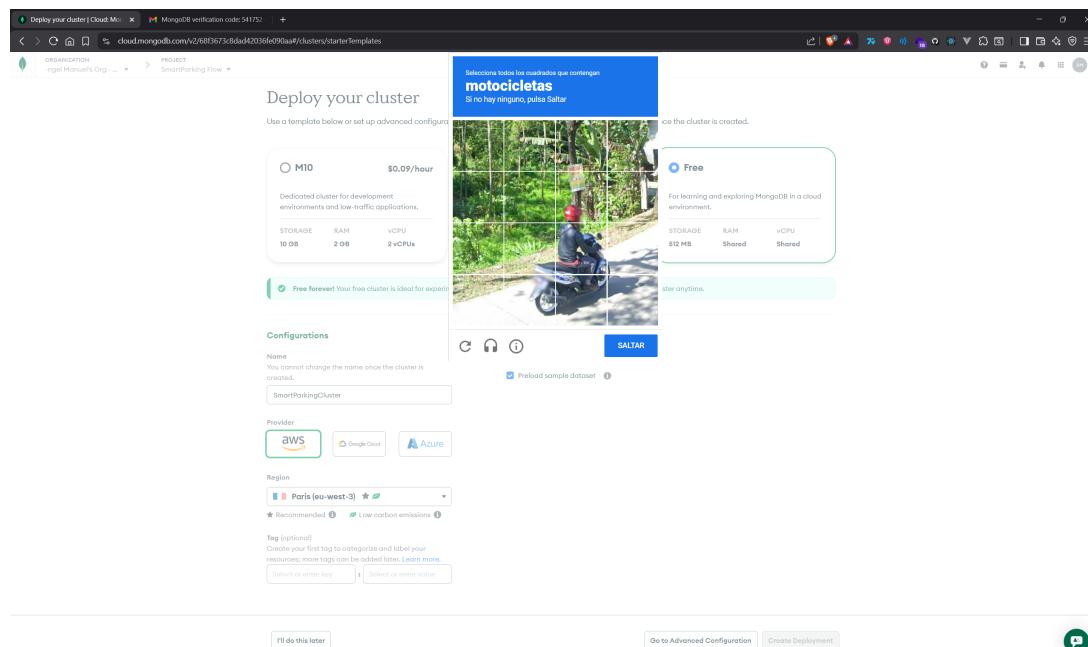


Figura B.15: Verificación Captcha para la creación del cluster.

SmartParking Flow — Monitorización Inteligente

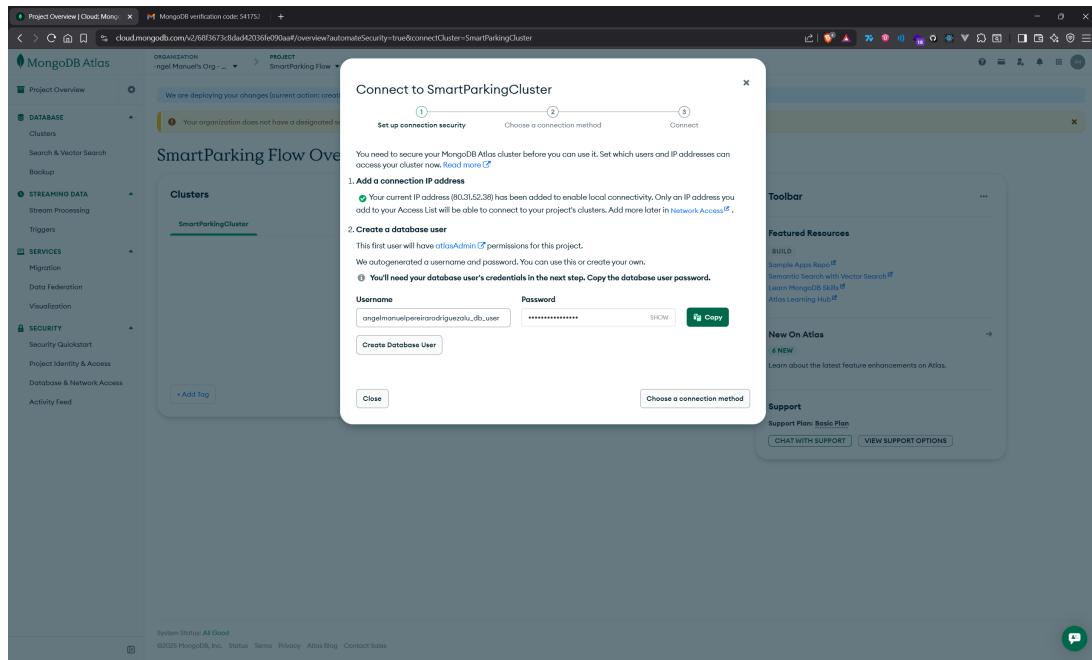


Figura B.16: Creación del usuario de la base de datos.

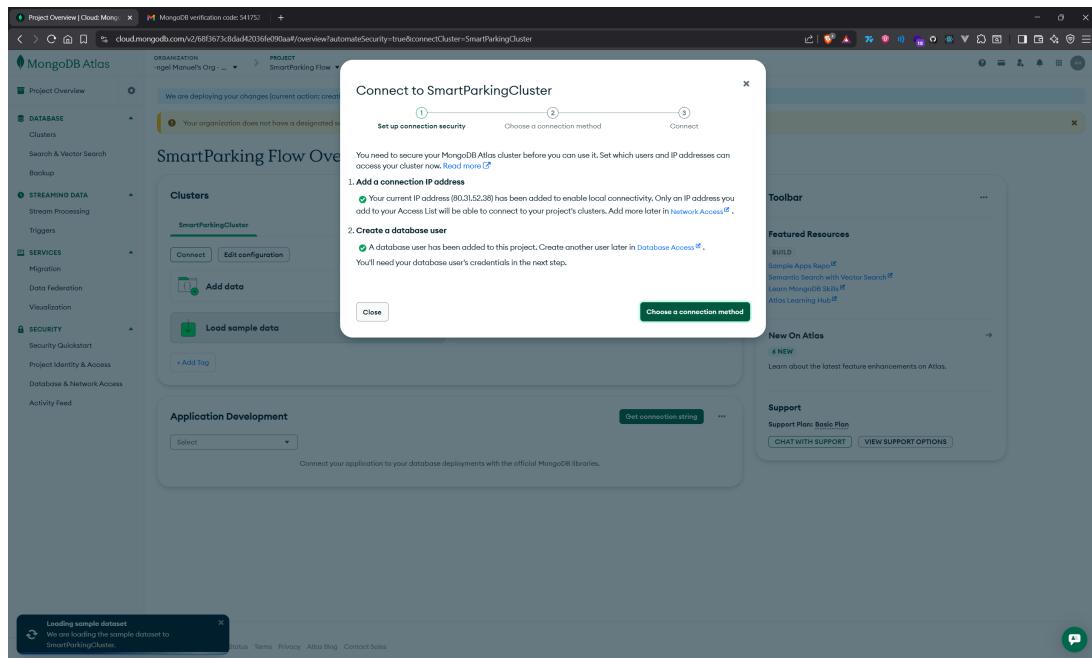


Figura B.17: Confirmación de creación de usuario.

SmartParking Flow — Monitorización Inteligente

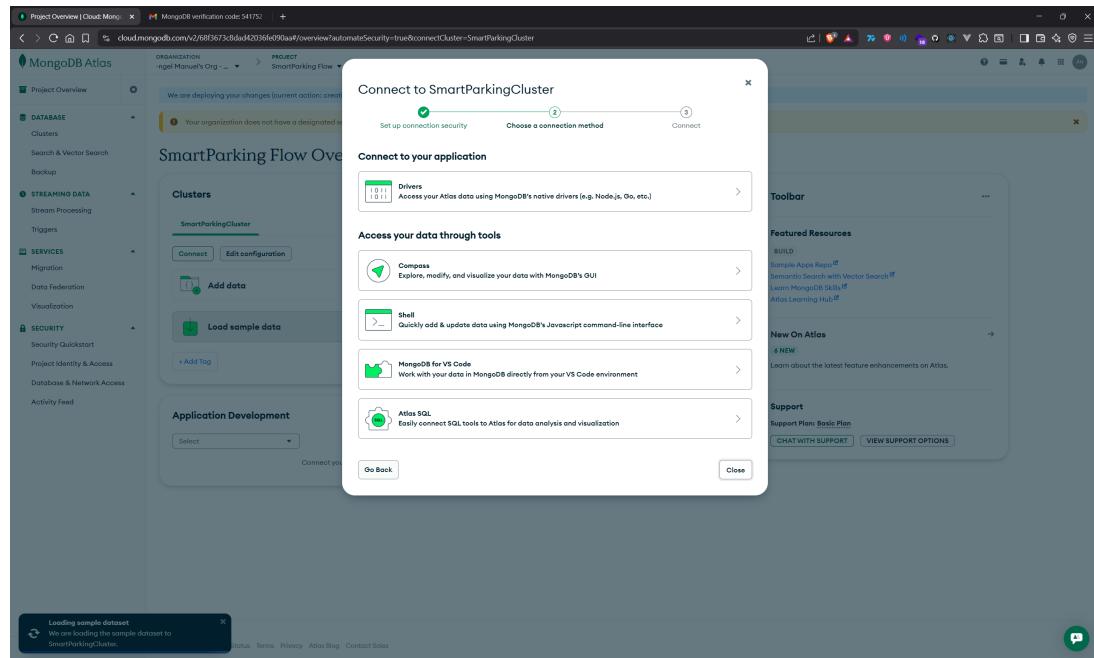


Figura B.18: Selección del método de conexión al cluster.

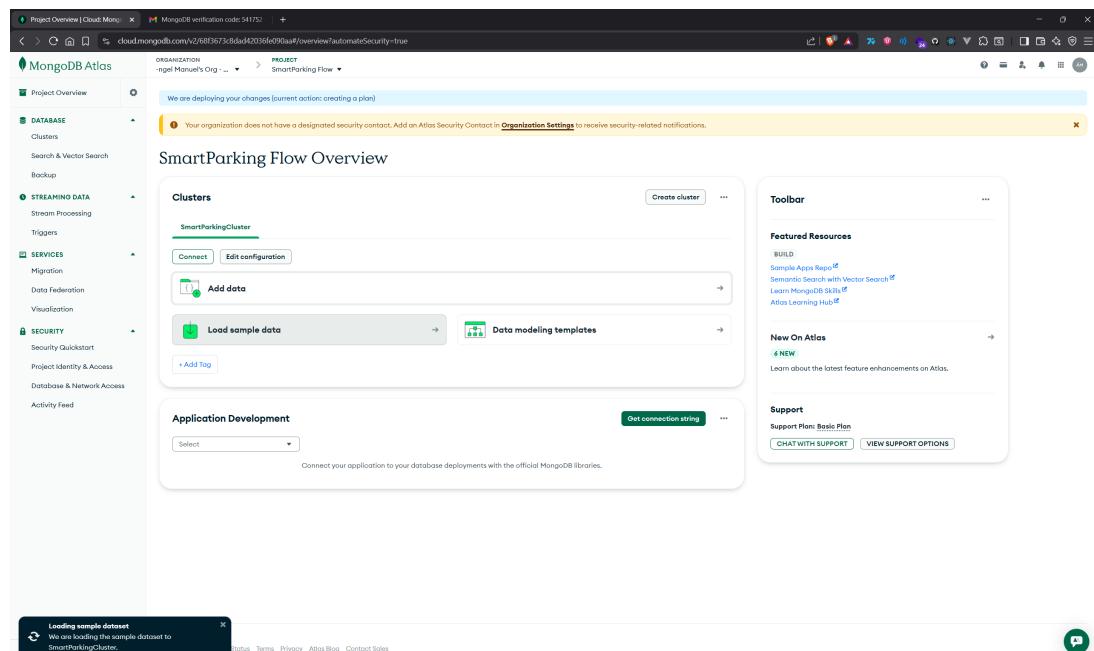


Figura B.19: Cargando datos de ejemplo (sample data).

SmartParking Flow — Monitorización Inteligente

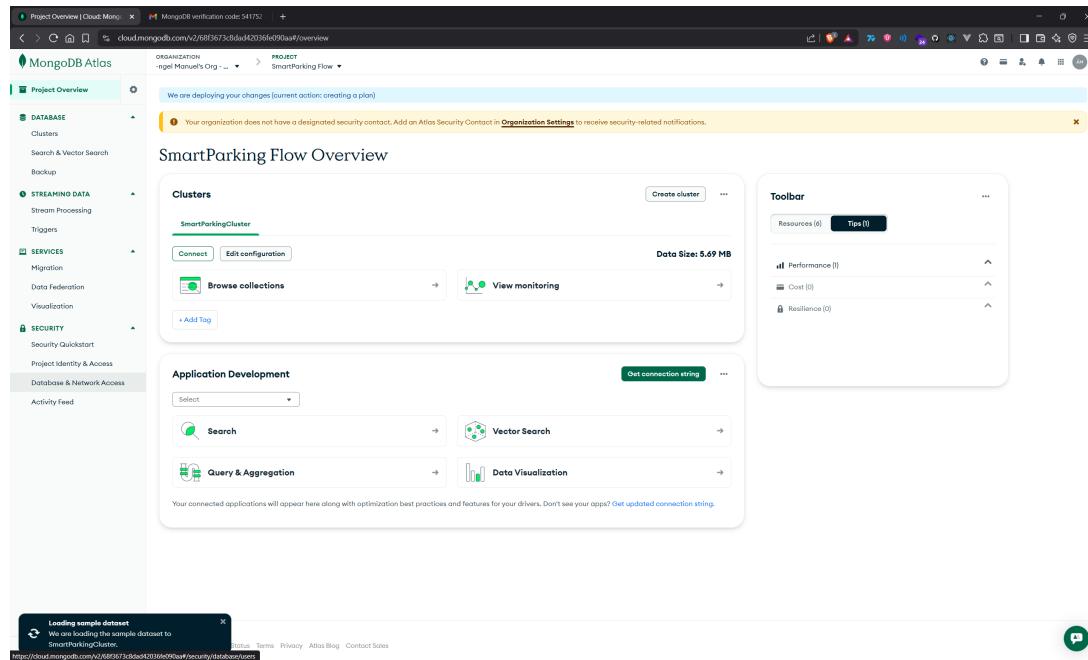


Figura B.20: Vista del cluster tras finalizar la carga de datos de ejemplo.

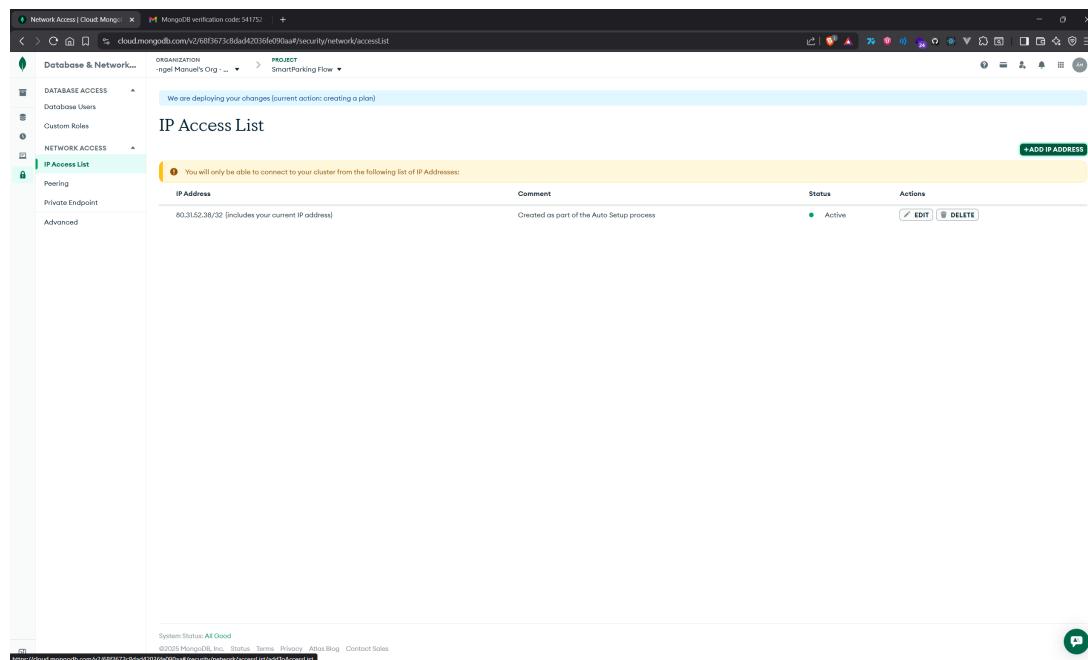


Figura B.21: Configuración de "IP Access List"(IP actual).

SmartParking Flow — Monitorización Inteligente

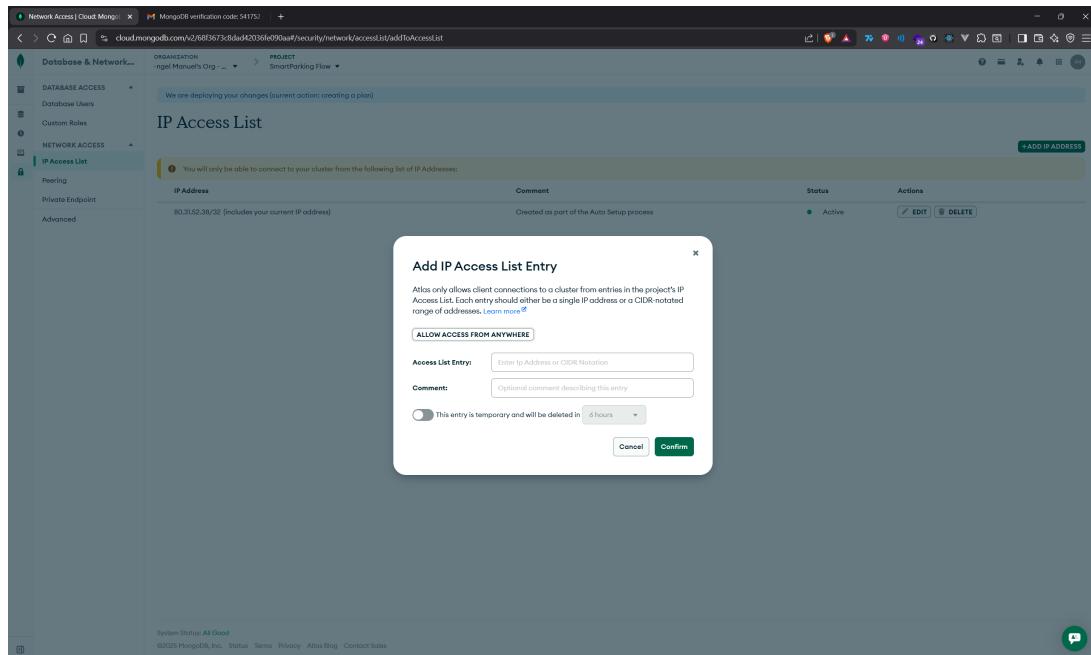


Figura B.22: Añadiendo una nueva entrada a "IP Access List".

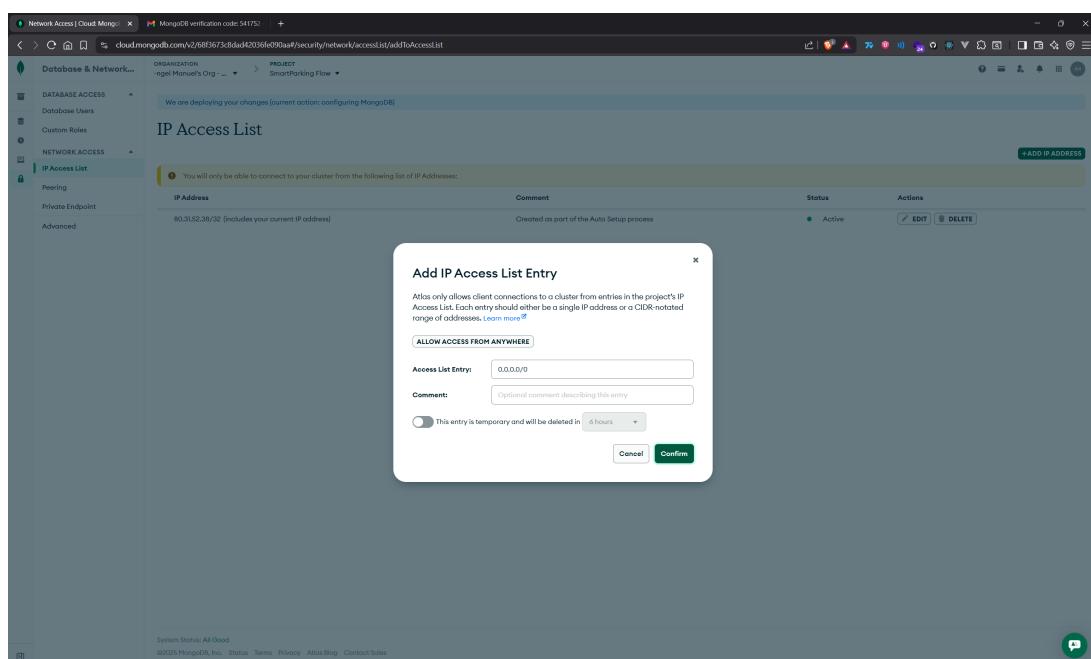


Figura B.23: Añadiendo '0.0.0.0/0' (Allow Access From Anywhere).

SmartParking Flow — Monitorización Inteligente

The screenshot shows the MongoDB Cloud interface for the project "SmartParking Flow". The left sidebar has sections for Database & Network, Network Access, and IP Access List, which is currently selected. The main area displays the "IP Access List" with two entries: "0.0.0.0/0 [includes your current IP address]" and "60.31.52.36/32 [includes your current IP address]". Both entries are marked as "Active". A message at the top says "We are deploying your changes [current action: creating a plan]". At the bottom, it says "System Status: All Good".

Figura B.24: IP Access List actualizada con acceso global.

The screenshot shows the MongoDB Atlas Project Overview page for the "SmartParking Flow" project. The left sidebar includes sections for Project Overview, Database, Streamlined Data, Services, and Security. The main area shows a cluster named "SmartParkingCluster" with a status message: "Sample data loaded successfully. Get started with a sample query in Data Explorer or by connecting with MongoDB Shell." It also shows "Data Size: 74.12 MB". Below this, there's a "Clusters" section with a "Create cluster" button and a "Toolbar" section with "Resources", "Tips", "Performance", "Cost", and "Resilience".

Figura B.25: Vista del cluster con datos de ejemplo cargados (74.12 MB).

SmartParking Flow — Monitorización Inteligente

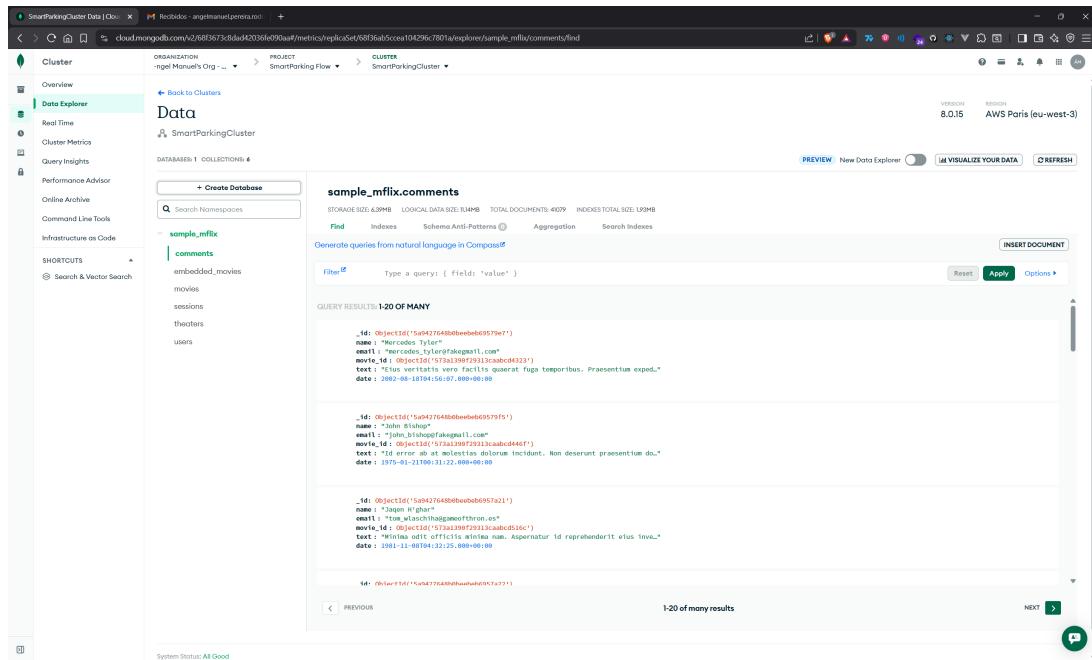


Figura B.26: Explorador de datos (Data Explorer) viendo 'sample_mflix.comments'.

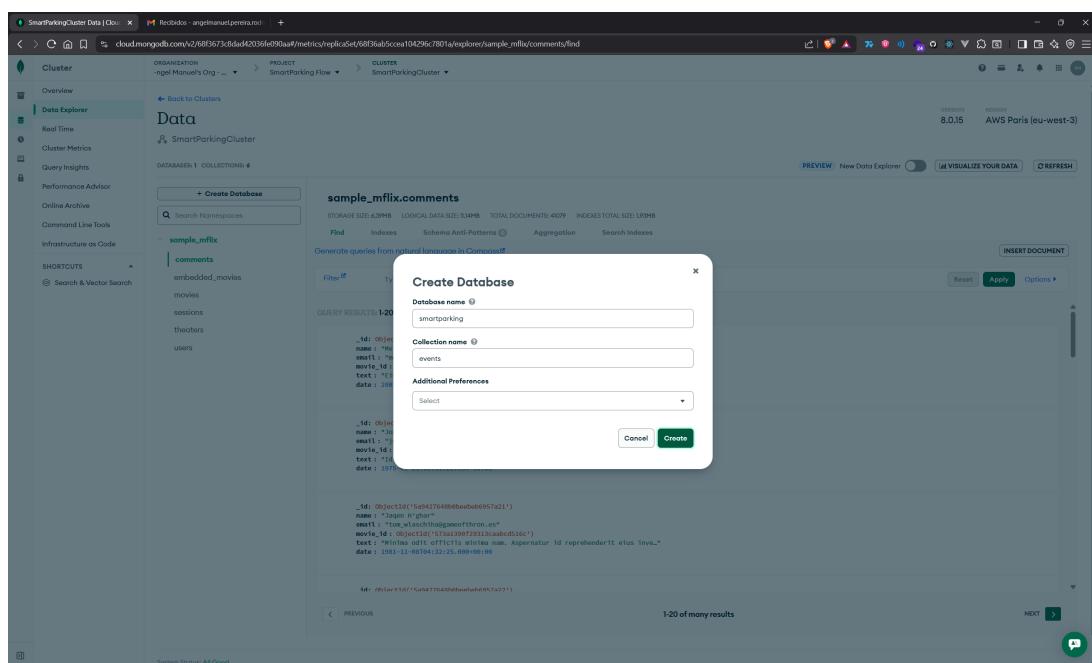


Figura B.27: Creación de la base de datos 'smartparking' y la colección 'events'.

SmartParking Flow — Monitorización Inteligente

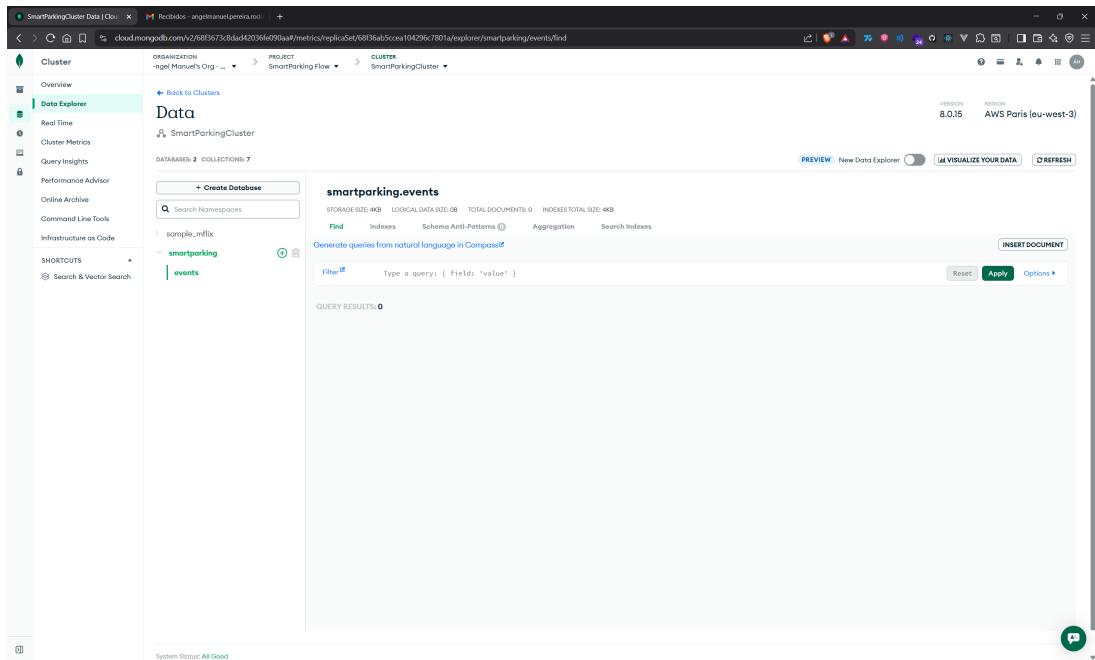


Figura B.28: Vista de la colección 'events' vacía.

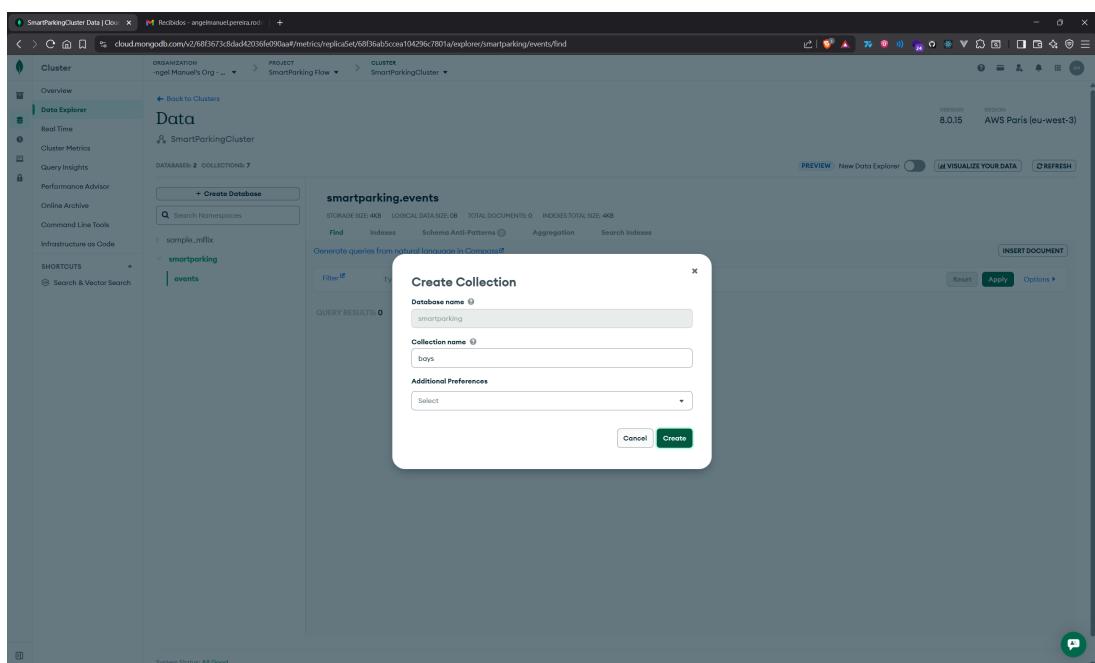
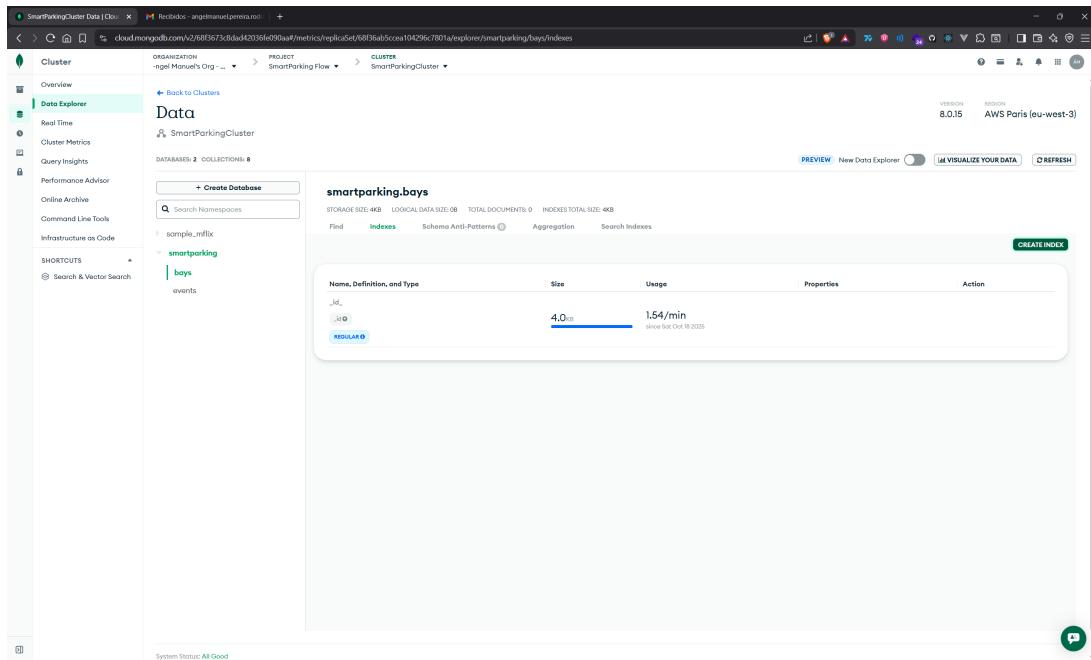


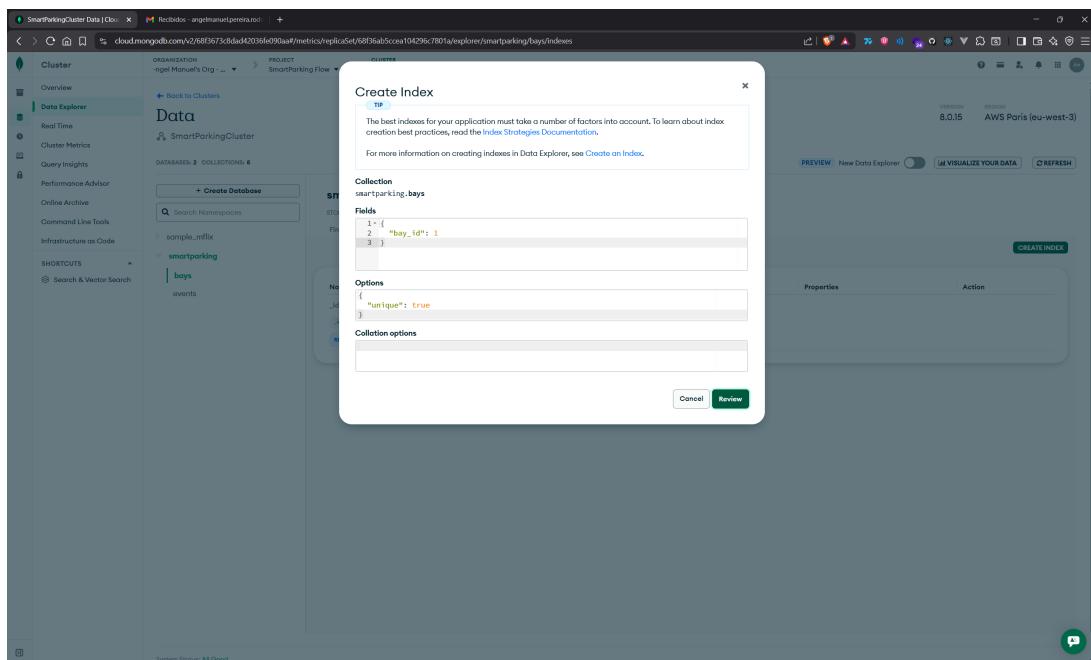
Figura B.29: Creación de la colección 'bays'.

SmartParking Flow — Monitorización Inteligente



The screenshot shows the MongoDB Data Explorer interface. On the left, the sidebar includes 'Cluster', 'Overview', 'Data Explorer' (which is selected), 'Real Time', 'Cluster Metrics', 'Query Insights', 'Performance Advisor', 'Online Archive', 'Command Line Tools', and 'Infrastructure as Code'. Under 'Data Explorer', there are 'SHORTCUTS' (Search & Vector Search) and a 'Data' section with 'SmartParkingCluster' and 'Databases: 2 Collections: 8'. The 'bays' collection is highlighted. The main panel displays the 'smartparking.bays' collection with 4 documents. It shows fields '_id' and '_id' with sizes of 4.0 bytes and usage of 1.54/min respectively. A 'CREATE INDEX' button is visible at the top right of the collection view.

Figura B.30: Vista de las colecciones 'bays' y 'events' creadas.



This screenshot shows the 'Create Index' dialog box overlaid on the MongoDB Data Explorer. The dialog has a 'Fields' section containing the code: '1: { 2: { "bay_id": 1 3: } }'. Below it is an 'Options' section with the code: '["unique": true]'. At the bottom are 'Cancel' and 'Review' buttons. The background shows the same MongoDB interface as Figure B.30, with the 'bays' collection listed in the collections list.

Figura B.31: Creación del índice para la colección 'bays'.

SmartParking Flow — Monitorización Inteligente

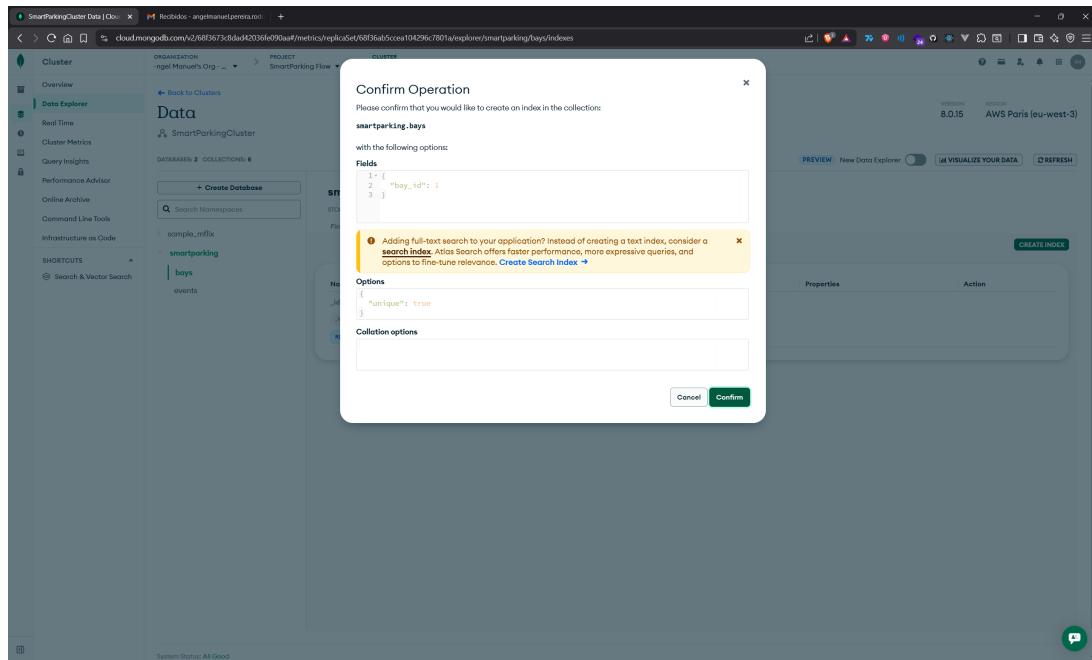


Figura B.32: Confirmación de creación de índice en 'bays' sobre 'bay_id'.

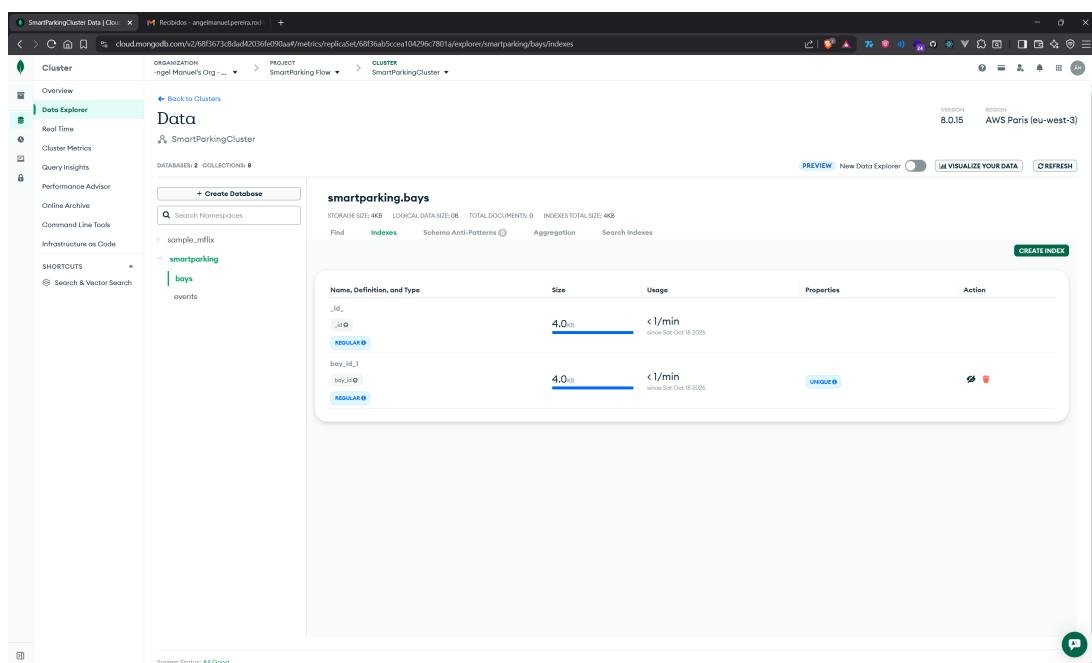


Figura B.33: Índice 'bay_id_1' creado y listado en la colección 'bays'.

SmartParking Flow — Monitorización Inteligente

The screenshot shows the MongoDB Data Explorer interface. On the left, the sidebar includes 'Cluster', 'Overview', 'Data Explorer' (selected), 'Real Time', 'Cluster Metrics', 'Query Insights', 'Performance Advisor', 'Online Archive', 'Command Line Tools', and 'Infrastructure as Code'. Under 'Data Explorer', there are 'SHORTCUTS' (Search & Vector Search) and a 'Data' section with 'Real Time' and 'Cluster Metrics' tabs. The 'Data' section also contains a 'Create Database' button, a search bar for namespaces, and a list of databases and collections: 'sample_mflix', 'smartparking' (selected), 'boys', and 'events'. The main panel shows the 'smartparking.events' collection. It displays storage details: STORAGE SIZE: 4KB, LOGICAL DATA SIZE: 0B, TOTAL DOCUMENTS: 0, INDEXES TOTAL SIZE: 4KB. Below this are tabs for 'Find', 'Indexes' (selected), 'Schema Anti-Patterns', 'Aggregation', and 'Search Indexes'. A table lists the index definition: '_id_'. The table columns are 'Name, Definition, and Type', 'Size', 'Usage', 'Properties', and 'Action'. The 'CREATE INDEX' button is located at the bottom right of the table area.

Figura B.34: Vista de la pestaña Índices"de la colección 'events'.

The screenshot shows the 'Create Index' dialog box overlaid on the MongoDB Data Explorer interface. The dialog has a 'TIP' section with a note about creating indexes. It includes sections for 'Collection' (smartparking.events), 'Fields' (containing a JSON object with fields: '_id': 1, 'bay_id': 1, 'last_event_ts': -1), 'Options' (containing a JSON object with a field: 'name': 'bay_event_ts_desc_idx'), and 'Collection options' (empty). At the bottom are 'Cancel' and 'Review' buttons. The background shows the same interface as Figure B.34, with the 'Indexes' tab selected for the 'events' collection.

Figura B.35: Definición del índice compuesto para la colección 'events'.

SmartParking Flow — Monitorización Inteligente

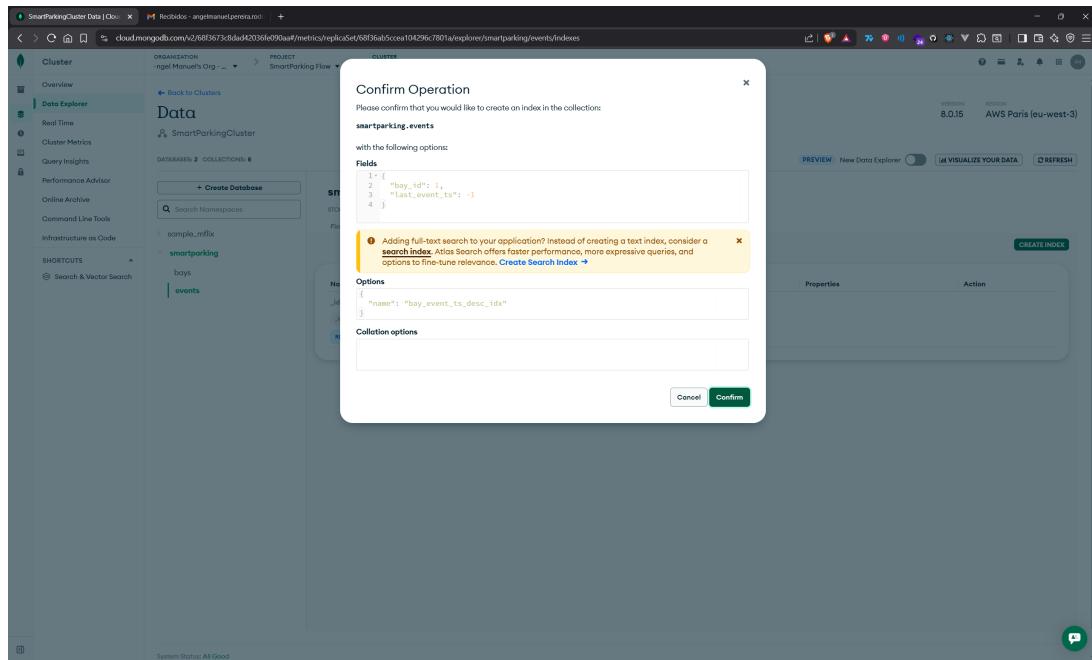


Figura B.36: Confirmación de creación de índice en 'events'.

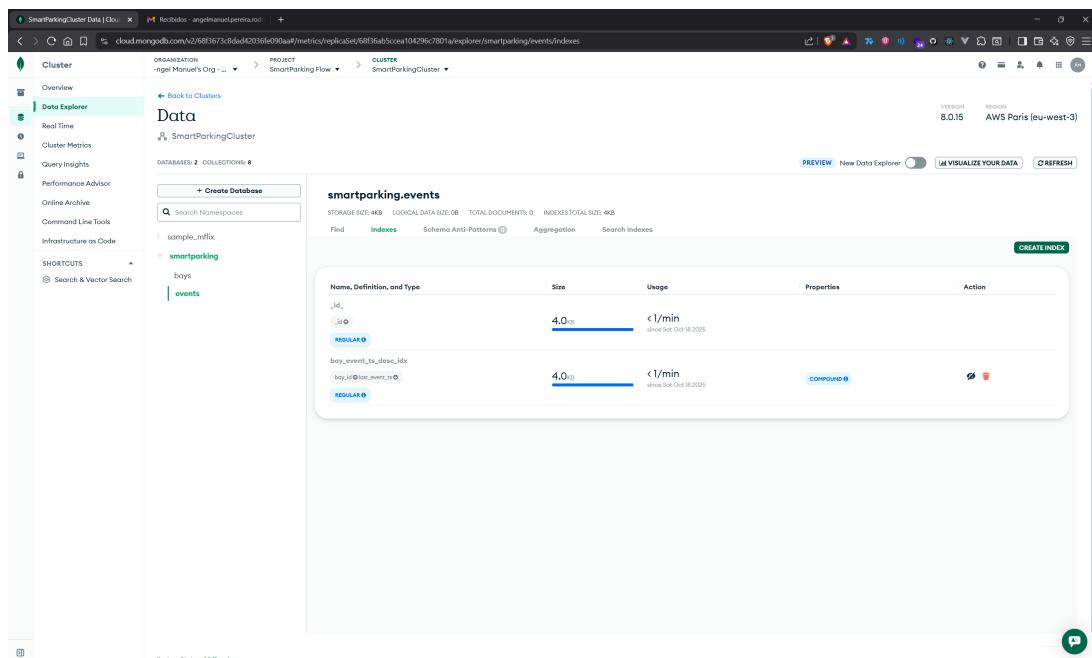


Figura B.37: Índice 'bay_event_ts_desc_idx' creado en la colección 'events'.

SmartParking Flow — Monitorización Inteligente

The screenshot shows the MongoDB Atlas Project Overview page for the 'SmartParking Flow' project. On the left, a sidebar navigation includes sections for Project Overview, Database, Stream Data, Services, and Security. The main area displays the 'SmartParking Cluster' with a message indicating sample data has been loaded successfully. It shows a data size of 116.16 MB and provides options to Connect, Browse collections, and View monitoring. A 'Toolbar' on the right offers links to Performance, Cost, and Resilience metrics. At the bottom, there's a footer with system status and a copyright notice.

Figura B.38: Vista del cluster (Data Size: 116.16 MB).

The screenshot shows a modal dialog titled 'Connect to SmartParkingCluster' overlaid on the MongoDB Atlas interface. The dialog is divided into three tabs: 'Set up connection security', 'Choose a connection method', and 'Connect'. The 'Choose a connection method' tab is active, showing options for connecting to an application via Drivers (MongoDB native drivers like Node.js, Go, etc.) or using various tools like Compose, Shell, MongoDB for VS Code, and Atlas SQL. Navigation buttons 'Go Back' and 'Close' are at the bottom of the dialog.

Figura B.39: Opciones de conexión al cluster.

SmartParking Flow — Monitorización Inteligente

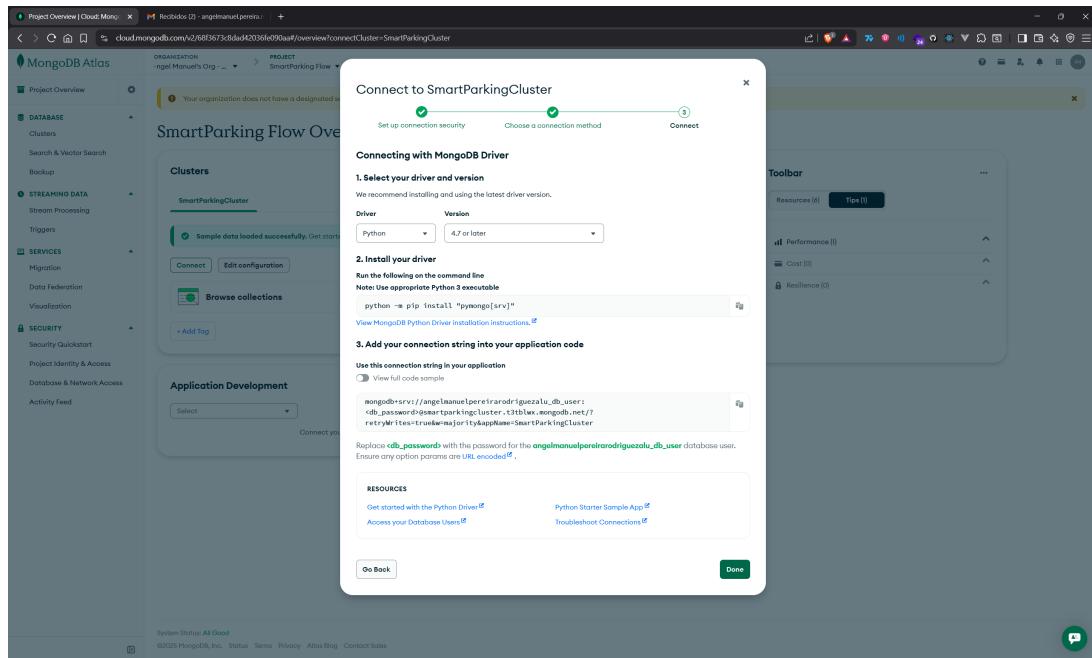


Figura B.40: Obteniendo la cadena de conexión (Connection String) para Python.

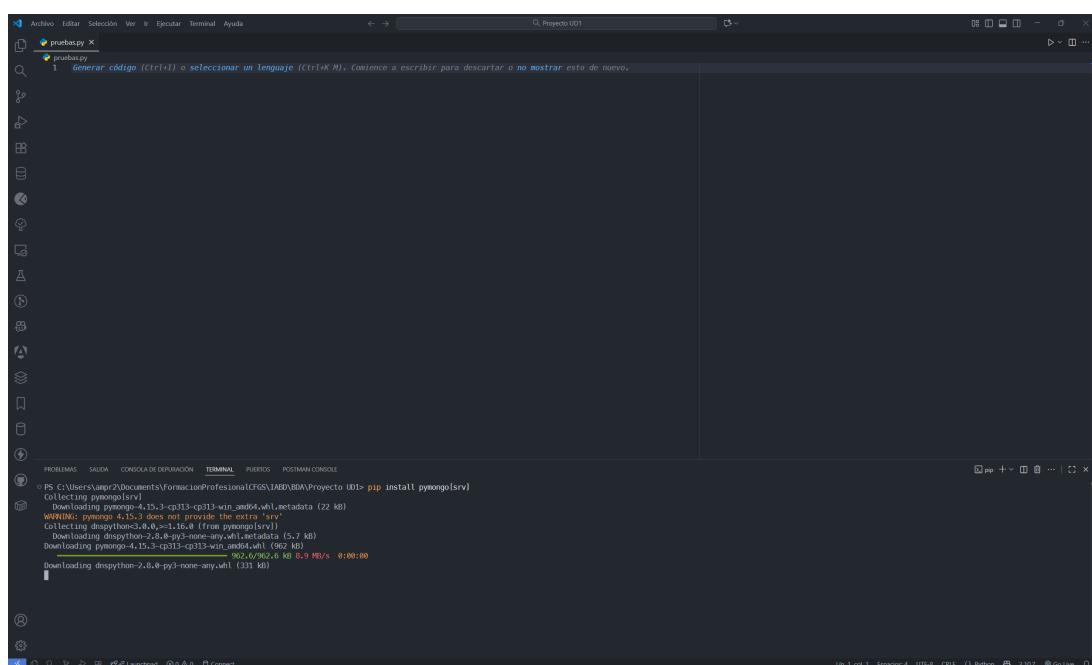


Figura B.41: Instalando 'pymongo[srv]' en el anfitrión (Windows).

SmartParking Flow — Monitorización Inteligente

The screenshot shows the VS Code interface with the 'pruebas.py' file open in the editor. The code connects to a MongoDB database and inserts a test document. Below the editor is a terminal window showing the execution of the script and its output.

```
from pymongo import MongoClient
uri = "mongodb+srv://angelmanuelperierarodriguezalv_db_user:<db_password>@smartparkingcluster.t3tblwx.mongodb.net/?retryWrites=true&w=majority&appName=SmartParkingCluster"
client = MongoClient(uri)

# Probar conexión
db = client.smartparking
print(db.list_collection_names())

# Insertar documento de prueba
test_buy = {
    "_id": "TEST-001",
    "parking_id": "PK-C001Z-01",
    "level": "L1",
    "occupied": False,
    "updated_at": "2025-10-18T10:00:00Z"
}
db.bays.insert_one(test_buy)
print("Conexión exitosa a MongoDB Atlas")
```

```
PS C:\Users\apr2\Documents\FormacionProfesional\CFGS\IABD\BDA\Proyecto UD1> pip install pymongo[srv]
Collecting pymongo[srv]
  Downloading pymongo-4.15.3-py3-none-any.whl (22 kB)
    WARNING: pymongo-4.15.3 does not provide the extra 'srv'
Collecting dnspython<3.0.0,>=1.16.4 (from pymongo[srv])
  Downloading dnspython-2.8.0-py3-none-any.whl.metadata (5.7 kB)
  Downloading dnspython-2.8.0-py3-none-any.whl (331 kB)
  Downloading dnspython-2.8.0-py3-none-any.whl (331 kB)
Successfully installed dnspython-2.8.0 pymongo-4.15.3
PS C:\Users\apr2\Documents\FormacionProfesional\CFGS\IABD\BDA\Proyecto UD1> []

PS C:\Users\apr2\Documents\FormacionProfesional\CFGS\IABD\BDA\Proyecto UD1> python pruebas.py
[{'_id': 'TEST-001', 'events': []}
Conexión exitosa a MongoDB Atlas]
```

Figura B.42: Script de prueba `pruebas.py` en VS Code (Windows).

The screenshot shows the VS Code interface with the 'pruebas.py' file open in the editor. The terminal window below shows the command to run the script and its output, which displays the inserted document and a success message.

```
PS C:\Users\apr2\Documents\FormacionProfesional\CFGS\IABD\BDA\Proyecto UD1> & C:/Users/apr2/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/apr2/Documents/FormacionProfesional\CFGS\IABD\BDA\Proyecto UD1/pruebas.py"
[{'_id': 'TEST-001', 'events': []}
Conexión exitosa a MongoDB Atlas]
```

Figura B.43: Resultado de la ejecución del script de prueba en Windows.

SmartParking Flow — Monitorización Inteligente

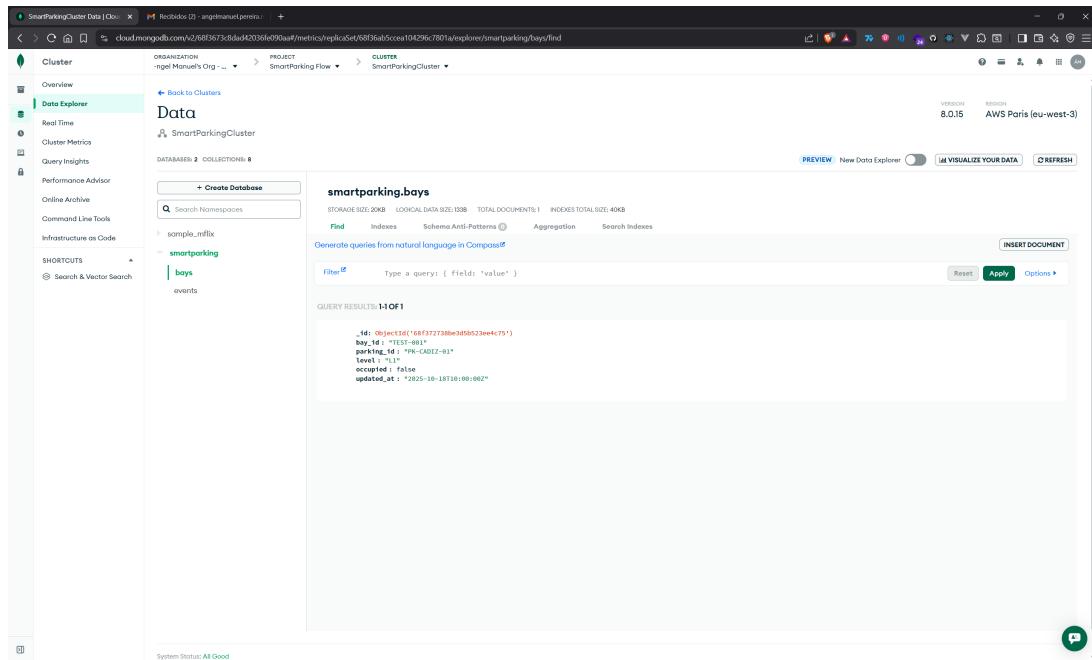


Figura B.44: Documento de prueba "TEST-001" insertado en 'smartparking.bays'.

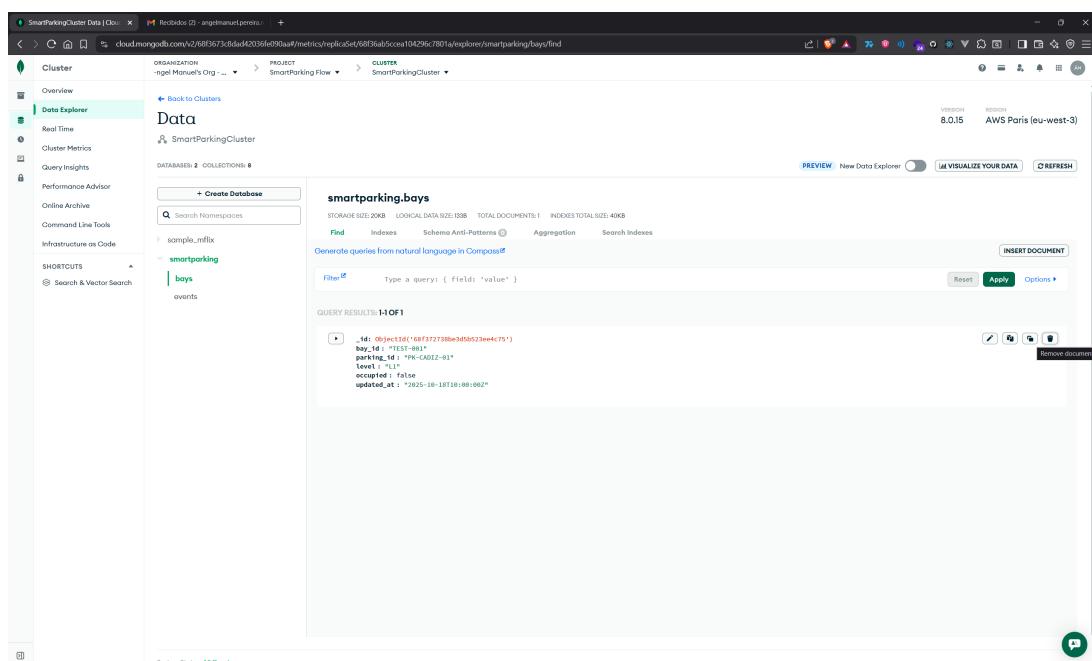


Figura B.45: Vista del documento "TEST-001.en Data Explorer.

SmartParking Flow — Monitorización Inteligente

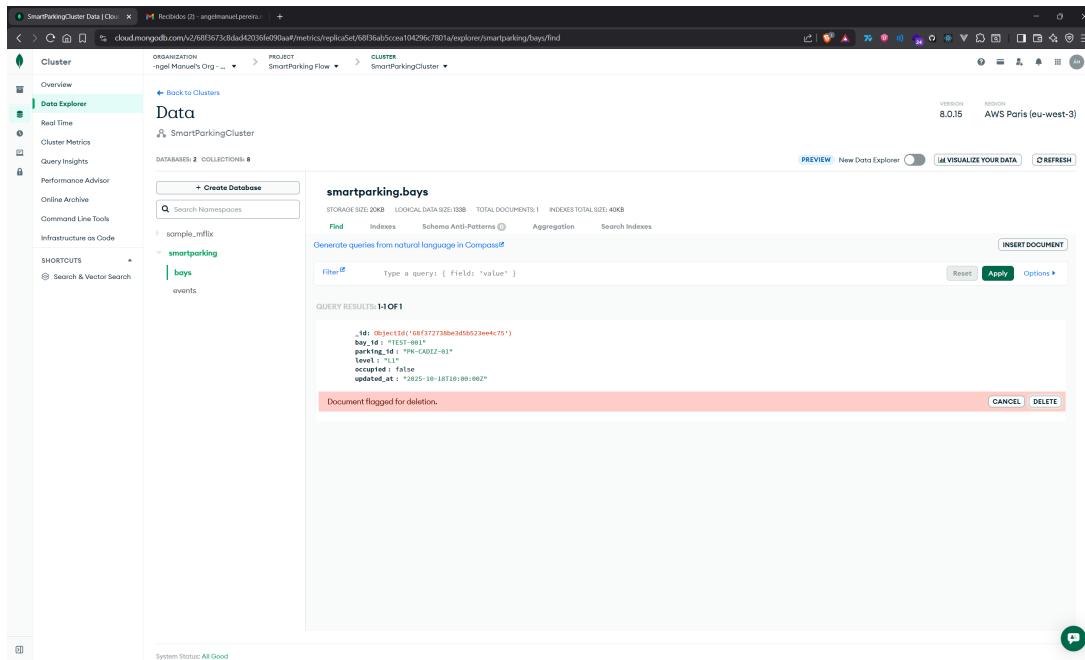


Figura B.46: Documento "TEST-001" marcado para eliminación.

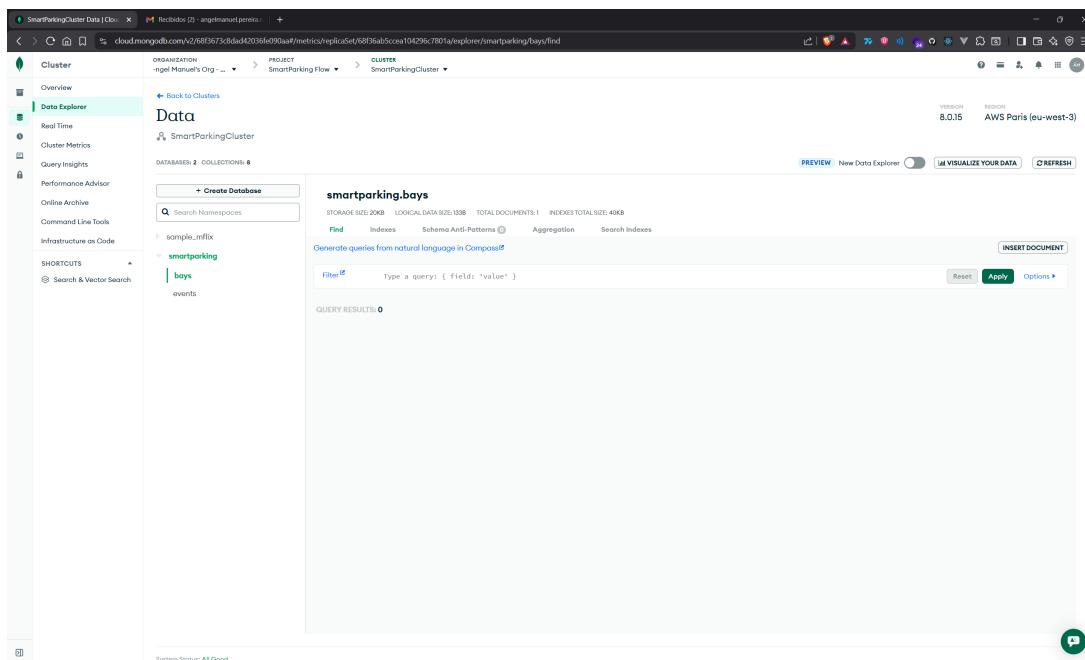
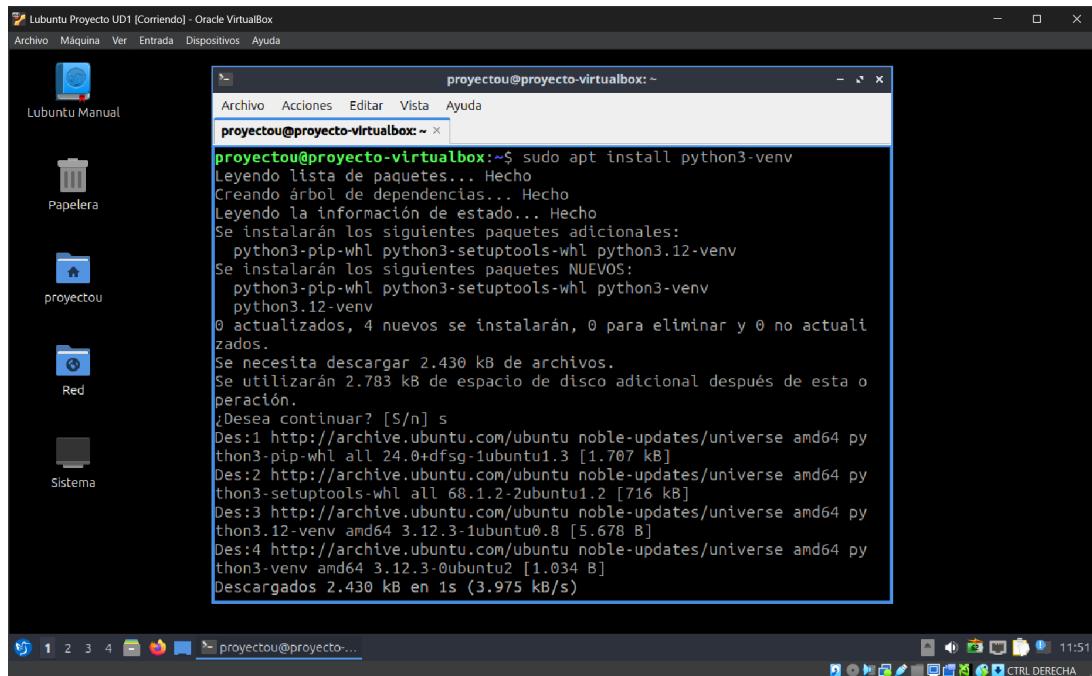


Figura B.47: Colección 'smartparking.bays' vacía tras borrado.

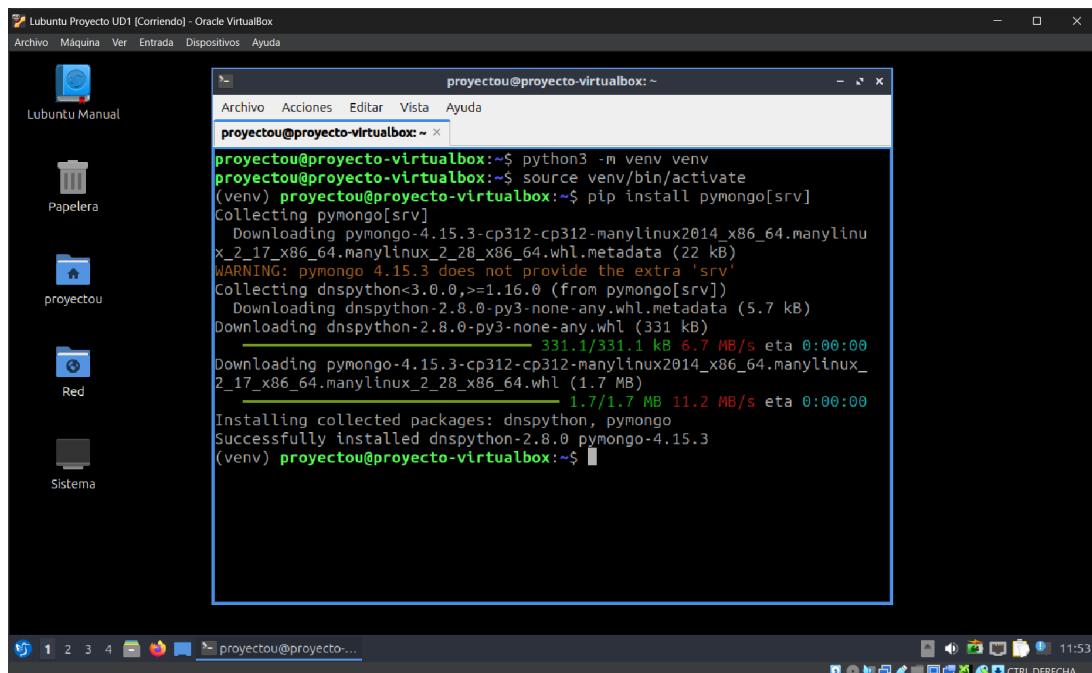


```

Lubuntu Proyecto UD1 [Corriendo] - Oracle VirtualBox
Archivo Máquina Ver Entrada Dispositivos Ayuda
proyecto@proyecto-virtualbox:~ %
proyecto@proyecto-virtualbox:~ $ sudo apt install python3-venv
Leyendo lista de paquetes... Hecho
Creando árbol de dependencias... Hecho
Leyendo la información de estado... Hecho
Se instalarán los siguientes paquetes adicionales:
  python3-pip-whl python3-setuptools-whl python3.12-venv
Se instalarán los siguientes paquetes NUEVOS:
  python3-pip-whl python3-setuptools-whl python3-venv
  python3.12-venv
0 actualizados, 4 nuevos se instalarán, 0 para eliminar y 0 no actualizados.
Se necesita descargar 2.430 kB de archivos.
Se utilizarán 2.783 kB de espacio de disco adicional después de esta operación.
¿Desea continuar? [S/n] s
Des:1 http://archive.ubuntu.com/ubuntu noble-updates/universe amd64 python3-pip-whl all 24.0+dfsg-1ubuntu1.3 [1.707 kB]
Des:2 http://archive.ubuntu.com/ubuntu noble-updates/universe amd64 python3-setuptools-whl all 68.1.2-2ubuntu1.2 [716 kB]
Des:3 http://archive.ubuntu.com/ubuntu noble-updates/universe amd64 python3.12-venv amd64 3.12.3-1ubuntu0.8 [5.678 kB]
Des:4 http://archive.ubuntu.com/ubuntu noble-updates/universe amd64 python3-venv amd64 3.12.3-0ubuntu2 [1.034 kB]
Descargados 2.430 kB en 1s (3.975 kB/s)

```

Figura B.48: Instalando 'python3-venv' en la MV de Lubuntu.



```

Lubuntu Proyecto UD1 [Corriendo] - Oracle VirtualBox
Archivo Máquina Ver Entrada Dispositivos Ayuda
proyecto@proyecto-virtualbox:~ %
proyecto@proyecto-virtualbox:~ $ python3 -m venv venv
proyecto@proyecto-virtualbox:~ $ source venv/bin/activate
(venv) proyecto@proyecto-virtualbox:~ $ pip install pymongo[srv]
Collecting pymongo[srv]
  Downloading pymongo-4.15.3-cp312-cp312-manylinux2014_x86_64.manylinux_2_17_x86_64.manylinux_2_28_x86_64.whl.metadata (22 kB)
WARNING: pymongo 4.15.3 does not provide the extra 'srv'
Collecting dnspython<3.0.0,>=1.16.0 (from pymongo[srv])
  Downloading dnspython-2.8.0-py3-none-any.whl.metadata (5.7 kB)
  Downloading dnspython-2.8.0-py3-none-any.whl (331 kB)
  _____ 331.1/331.1 kB 6.7 MB/s eta 0:00:00
  Downloading pymongo-4.15.3-cp312-cp312-manylinux2014_x86_64.manylinux_2_17_x86_64.manylinux_2_28_x86_64.whl (1.7 MB)
  _____ 1.7/1.7 kB 11.2 MB/s eta 0:00:00
Installing collected packages: dnspython, pymongo
Successfully installed dnspython-2.8.0 pymongo-4.15.3
(venv) proyecto@proyecto-virtualbox:~ $

```

Figura B.49: Instalando 'pymongo[srv]' en el entorno virtual de Lubuntu.

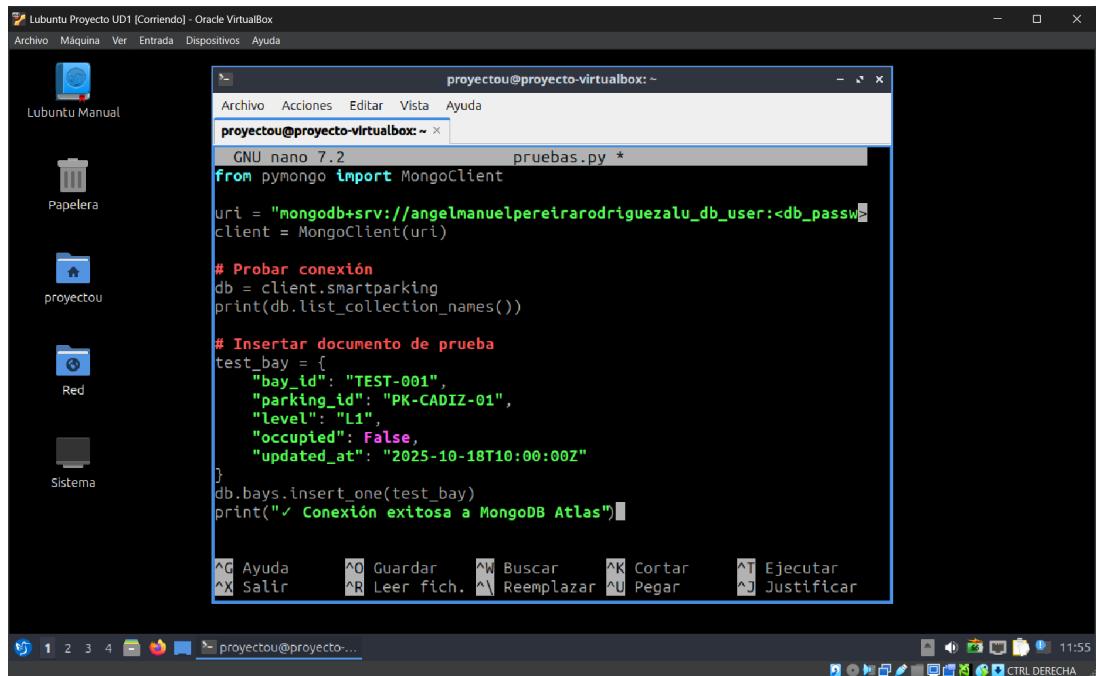


Figura B.50: Editando el script de prueba `pruebas.py` en Lubuntu (nano).

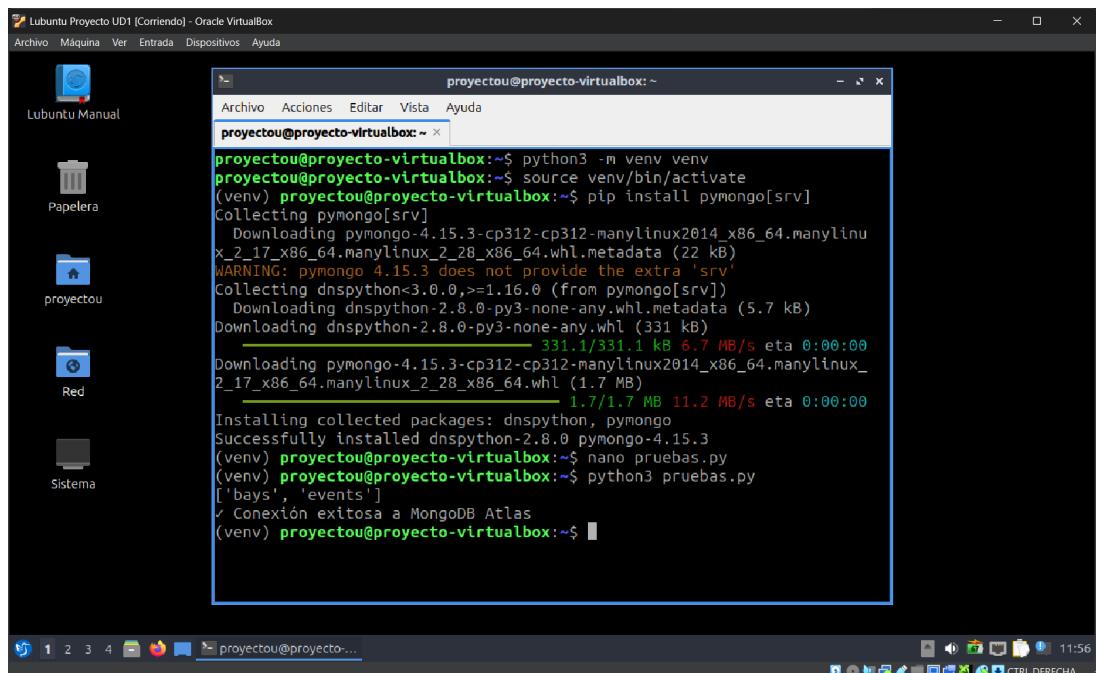


Figura B.51: Resultado de la ejecución del script de prueba en Lubuntu.

Apéndice C

Anexo C: Configuración de Apache Kafka

Instalación y configuración de Apache Kafka en la máquina virtual, incluyendo la descarga, configuración de servicios, creación de tópicos y el desarrollo del script de simulación de sensores.

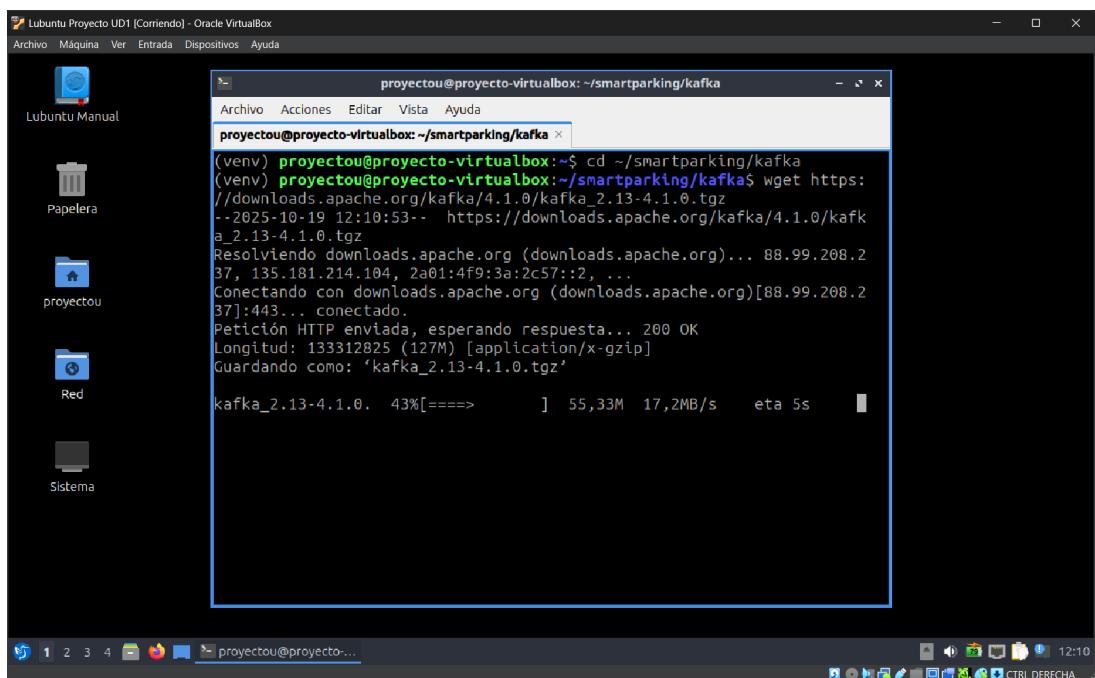


Figura C.1: Descarga de Apache Kafka ('wget').

SmartParking Flow — Monitorización Inteligente

The screenshot shows a Linux desktop environment with a terminal window open. The terminal window title is "proyecto@projeto-virtualbox: ~/smartparking/kafka/kafka". The terminal content shows the download and extraction of the Kafka 2.13-4.1.0 tarball from Apache's website. The file "kafka_2.13-4.1.0.tgz" is downloaded at 20.4 MB/s and extracted into the current directory. The Kafka directory is then cd'd into, and the "export KAFKA_HOME=\$PWD" command is run to set the environment variable.

```
//downloads.apache.org/kafka/4.1.0/kafka_2.13-4.1.0.tgz  
--2025-10-19 12:10:53-- https://downloads.apache.org/kafka/4.1.0/kafka_2.13-4.1.0.tgz  
Resolviendo downloads.apache.org (downloads.apache.org)... 88.99.208.2  
37, 135.181.214.104, 2a01:af9:3a:2c57::12, ...  
Conectando con downloads.apache.org (downloads.apache.org)[88.99.208.2  
37]:443... conectado.  
Petición HTTP enviada, esperando respuesta... 200 OK  
Longitud: 133312825 (127M) [application/x-gzip]  
Guardando como: 'kafka_2.13-4.1.0.tgz'  
  
kafka_2.13-4.1.0. 100%[=====] 127,14M 23,9MB/s en 6,2s  
  
2025-10-19 12:11:00 (20,4 MB/s) - 'kafka_2.13-4.1.0.tgz' guardado [133  
312825/133312825]  
  
(venv) proyecto@projeto-virtualbox:~/smartparking/kafka$ tar -xzf ka  
fka_2.13-4.1.0.tgz  
(venv) proyecto@projeto-virtualbox:~/smartparking/kafka$ mv kafka_2.  
13-4.1.0 kafka  
(venv) proyecto@projeto-virtualbox:~/smartparking/kafka$ cd kafka  
(venv) proyecto@projeto-virtualbox:~/smartparking/kafka$ export  
KAFKA_HOME=$PWD  
(venv) proyecto@projeto-virtualbox:~/smartparking/kafka/kafka$
```

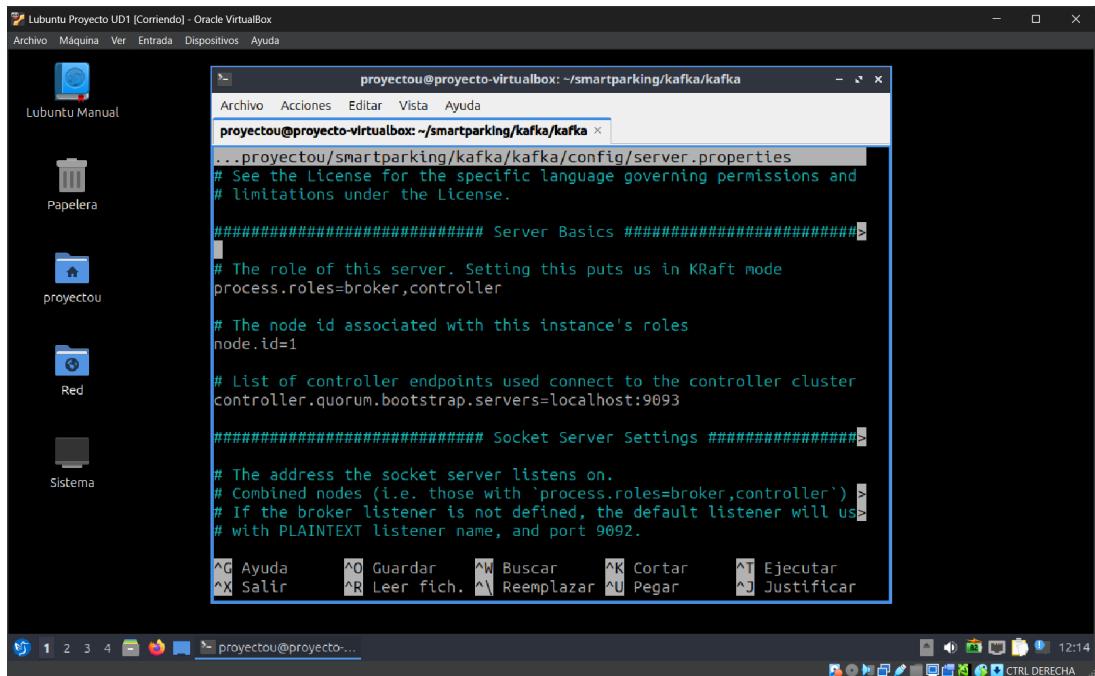
Figura C.2: Descompresión del archivo ('tar -xzf') y renombrado de la carpeta.

The screenshot shows a Linux desktop environment with a dark theme. On the left is a file explorer window titled "Lubuntu Manual" containing icons for "Papelera", "proyectou", "Red", and "Sistema". The main area features a terminal window titled "proyectou@projeto-virtualbox: ~/smartparking/kafka/kafka" with the following command history:

```
(venv) proyectou@projeto-virtualbox:~/smartparking/kafka/kafka$ sudo
mkdir -p /opt/kafka-data
[sudo] contraseña para proyectou:
(venv) proyectou@projeto-virtualbox:~/smartparking/kafka/kafka$ sudo
chown -R proyectou:proyectou /opt/kafka-data
(venv) proyectou@projeto-virtualbox:~/smartparking/kafka/kafka$ nano
$KAFKA_HOME/config/server.properties
```

The bottom taskbar has icons for various applications like a browser, file manager, and system tools. The status bar at the bottom right shows the date and time as "12:14".

Figura C.3: Creación del directorio de datos ('/opt/kafka-data') y edición de 'server.properties'.



```
proyectou@projeto-virtualbox:~/smartparking/kafka/kafka$ cat config/server.properties
...proyectou@projeto-virtualbox:~/smartparking/kafka/kafka$ cat config/server.properties
# See the License for the specific language governing permissions and
# limitations under the License.

#####
# Server Basics #####
#
# The role of this server. Setting this puts us in KRaft mode
process.roles=broker,controller

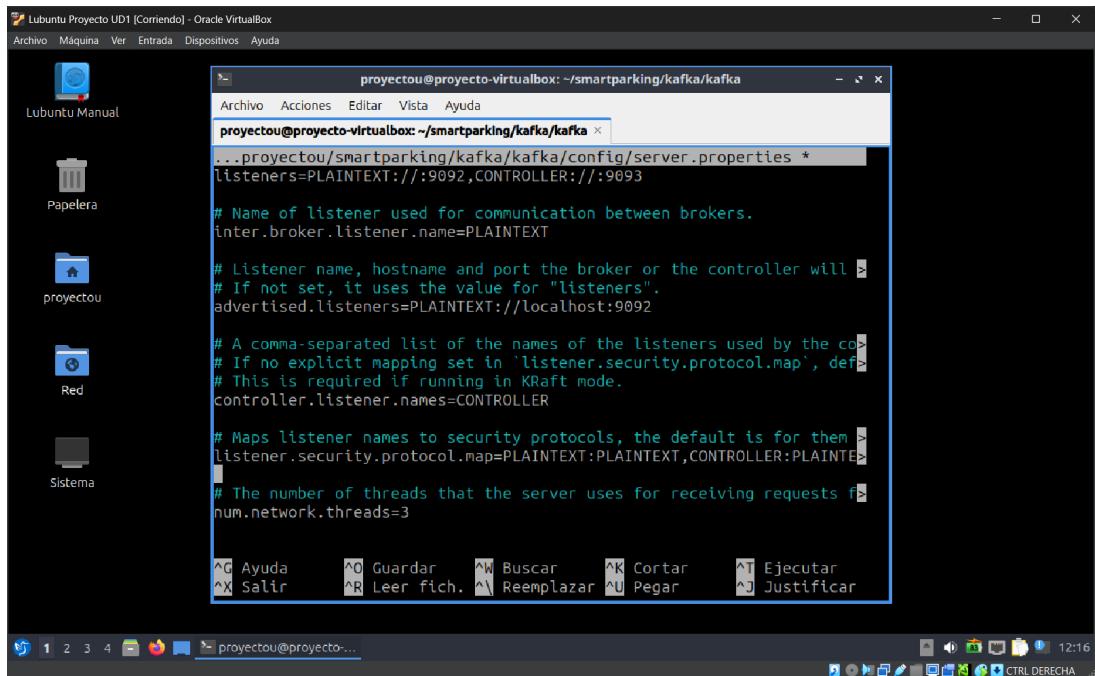
# The node id associated with this instance's roles
node.id=1

# List of controller endpoints used connect to the controller cluster
controller.quorum.bootstrap.servers=localhost:9093

#####
# Socket Server Settings #####
#
# The address the socket server listens on.
# Combined nodes (i.e. those with 'process.roles=broker,controller') >
# If the broker listener is not defined, the default listener will us>
# with PLAINTEXT listener name, and port 9092.

^G Ayuda      ^O Guardar     ^W Buscar     ^K Cortar     ^T Ejecutar
^X Salir      ^R Leer fich.  ^\ Reemplazar  ^U Pegar      ^J Justificar
```

Figura C.4: Configuración de 'server.properties': 'node.id' y 'controller.quorum.bootstrap.servers'.



```
proyectou@projeto-virtualbox:~/smartparking/kafka/kafka$ cat config/server.properties
...proyectou@projeto-virtualbox:~/smartparking/kafka/kafka$ cat config/server.properties *
listeners=PLAINTEXT://:9092,CONTROLLER://:9093

# Name of listener used for communication between brokers.
inter.broker.listener.name=PLAINTEXT

# Listener name, hostname and port the broker or the controller will >
# If not set, it uses the value for "listeners".
advertised.listeners=PLAINTEXT://localhost:9092

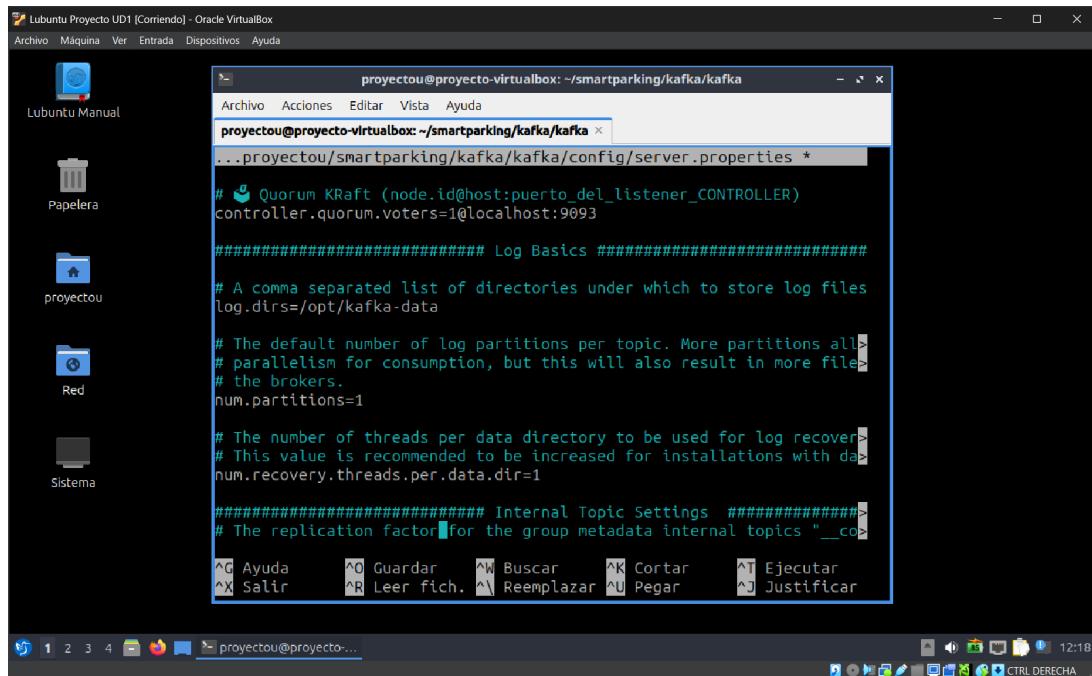
# A comma-separated list of the names of the listeners used by the co>
# If no explicit mapping set in 'listener.security.protocol.map', def>
# This is required if running in KRaft mode.
controller.listener.names=CONTROLLER

# Maps listener names to security protocols, the default is for them >
listener.security.protocol.map=PLAINTEXT:PLAINTEXT,CONTROLLER:PLAINTE>

# The number of threads that the server uses for receiving requests f>
num.network.threads=3

^G Ayuda      ^O Guardar     ^W Buscar     ^K Cortar     ^T Ejecutar
^X Salir      ^R Leer fich.  ^\ Reemplazar  ^U Pegar      ^J Justificar
```

Figura C.5: Configuración de 'server.properties': 'listeners' y 'advertised.listeners'.



```
proyecto@proyecto-virtualbox:~/smartparking/kafka/kafka$ cat config/server.properties
# Quorum KRaft (node.id@host:puerto_del_listener_CONTROLLER)
controller.quorum.voters=1@localhost:9093

#####
# A comma separated list of directories under which to store log files
log.dirs=/opt/kafka-data

# The default number of log partitions per topic. More partitions allow
# parallelism for consumption, but this will also result in more file
# the brokers.
num.partitions=1

# The number of threads per data directory to be used for log recovery
# This value is recommended to be increased for installations with da
num.recovery.threads.per.data.dir=1

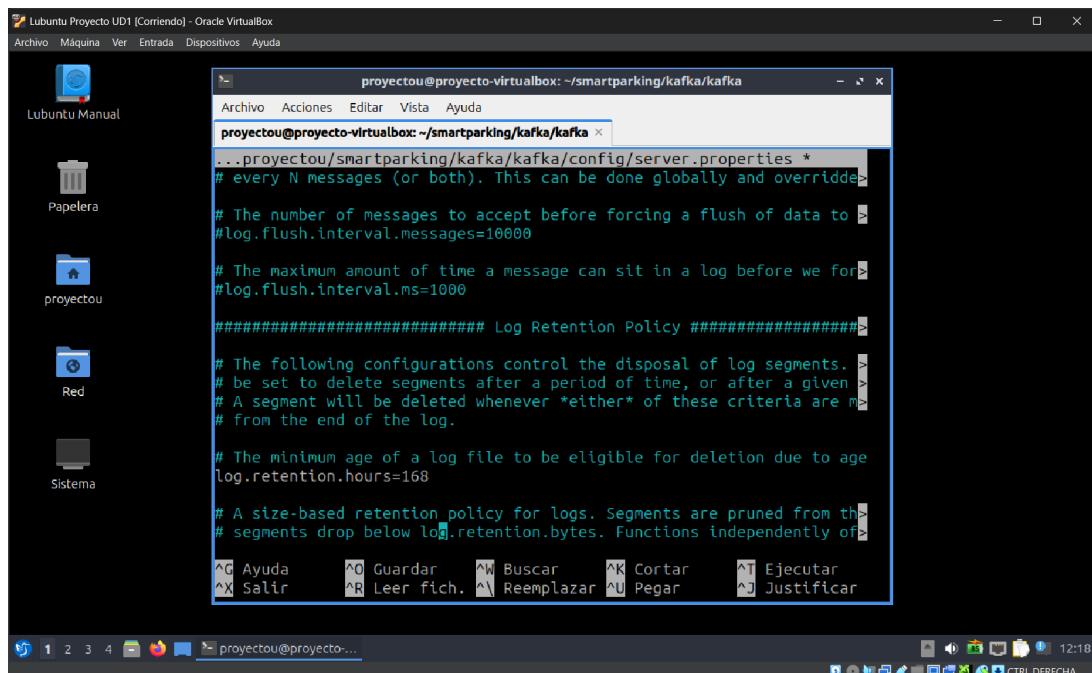
#####
# Internal Topic Settings
# The replication factor for the group metadata internal topics "_co"
# and _schemas
replication.factor=1

# Log Basics
log.flush.interval.messages=10000
log.flush.interval.ms=1000

#####
# Log Retention Policy
# The following configurations control the disposal of log segments.
# be set to delete segments after a period of time, or after a given
# A segment will be deleted whenever *either* of these criteria are m
# from the end of the log.
# The minimum age of a log file to be eligible for deletion due to age
log.retention.hours=168

# A size-based retention policy for logs. Segments are pruned from th
# segments drop below log.retention.bytes. Functions independently of
# the log retention policy
log.retention.bytes=104857600
```

Figura C.6: Configuración de 'server.properties': 'log.dirs'.



```
proyecto@proyecto-virtualbox:~/smartparking/kafka/kafka$ cat config/server.properties
# every N messages (or both). This can be done globally and overridden by
# log.flush.interval.messages=10000

# The number of messages to accept before forcing a flush of data to disk
#log.flush.interval.messages=10000

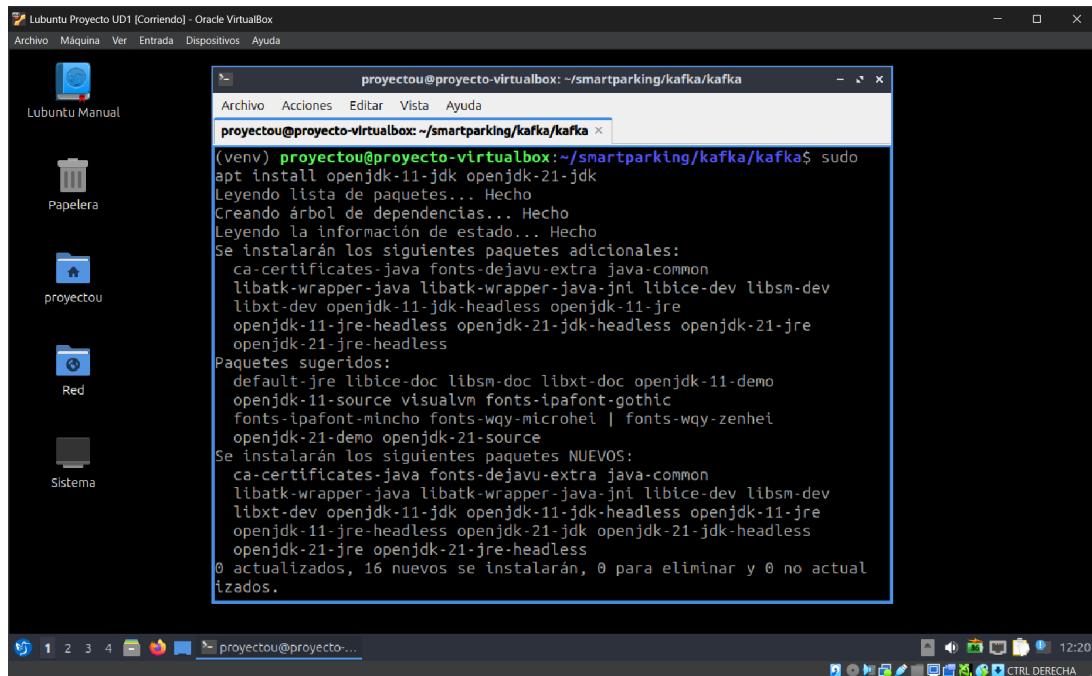
# The maximum amount of time a message can sit in a log before we force a
#log.flush.interval.ms=1000

#####
# Log Retention Policy
# The following configurations control the disposal of log segments.
# be set to delete segments after a period of time, or after a given
# A segment will be deleted whenever *either* of these criteria are met
# from the end of the log.

# The minimum age of a log file to be eligible for deletion due to age
log.retention.hours=168

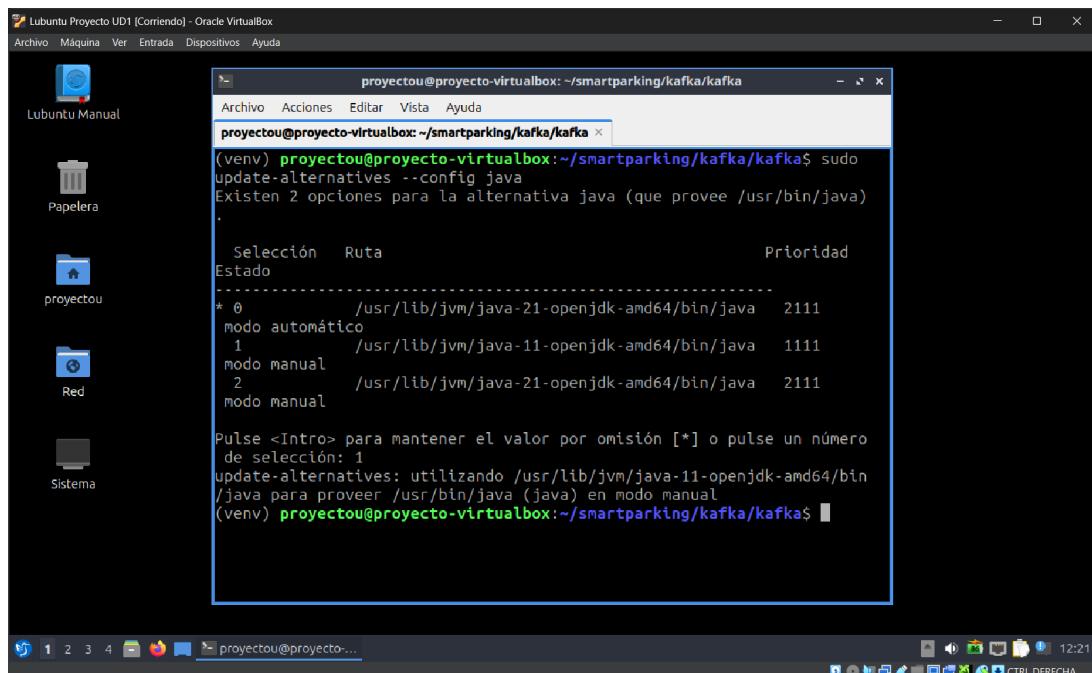
# A size-based retention policy for logs. Segments are pruned from the log
# segments drop below log.retention.bytes. Functions independently of the
# log retention policy
log.retention.bytes=104857600
```

Figura C.7: Configuración de 'server.properties': Políticas de retención de logs.



```
(venv) proyecto@proyecto-virtualbox:~/smartparking/kafka/kafka$ sudo
apt install openjdk-11-jdk openjdk-21-jdk
Leyendo lista de paquetes... Hecho
Creando árbol de dependencias... Hecho
Leyendo la información de estado... Hecho
Se instalarán los siguientes paquetes adicionales:
  ca-certificates-java fonts-dejavu-extra java-common
  libatk-wrapper-java libatk-wrapper-java-jni libice-dev libsm-dev
  libxt-dev openjdk-11-jdk-headless openjdk-11-jre
  openjdk-11-jre-headless openjdk-21-jdk-headless openjdk-21-jre
  openjdk-21-jre-headless
Paquetes sugeridos:
  default-jre libice-doc libsm-doc libxt-doc openjdk-11-demo
  openjdk-11-source visualvm fonts-ipafont-gothic
  fonts-ipafont-mincho fonts-wqy-microhei | fonts-wqy-zenhei
  openjdk-21-demo openjdk-21-source
Se instalarán los siguientes paquetes NUEVOS:
  ca-certificates-java fonts-dejavu-extra java-common
  libatk-wrapper-java libatk-wrapper-java-jni libice-dev libsm-dev
  libxt-dev openjdk-11-jdk openjdk-11-jdk-headless openjdk-11-jre
  openjdk-11-jre-headless openjdk-21-jdk openjdk-21-jdk-headless
  openjdk-21-jre openjdk-21-jre-headless
0 actualizados, 16 nuevos se instalarán, 0 para eliminar y 0 no actualizados.
```

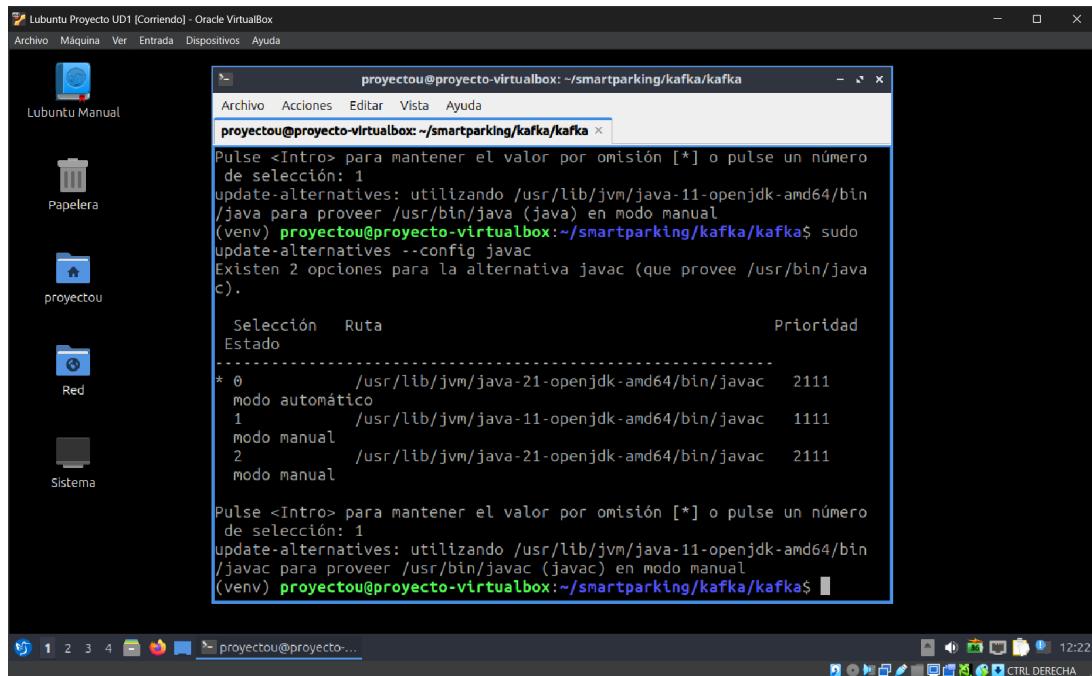
Figura C.8: Instalación de Java (OpenJDK 11 y 21) en la MV.



```
(venv) proyecto@proyecto-virtualbox:~/smartparking/kafka/kafka$ sudo
update-alternatives --config java
Existen 2 opciones para la alternativa java (que provee /usr/bin/java)
  *
    Selección     Ruta                               Prioridad
Estado
-----
* 0           /usr/lib/jvm/java-21-openjdk-amd64/bin/java  2111
  modo automático
  1           /usr/lib/jvm/java-11-openjdk-amd64/bin/java  1111
  modo manual
  2           /usr/lib/jvm/java-21-openjdk-amd64/bin/java  2111
  modo manual

Pulse <Intro> para mantener el valor por omisión [*] o pulse un número
de selección: 1
update-alternatives: utilizando /usr/lib/jvm/java-11-openjdk-amd64/bin/
/java para proveer /usr/bin/java (java) en modo manual
(venv) proyecto@proyecto-virtualbox:~/smartparking/kafka/kafka$
```

Figura C.9: Selección de la versión de Java (JDK 11) usando 'update-alternatives'.



Lubuntu Proyecto UD1 [Corriendo] - Oracle VirtualBox

Archivo Máquina Ver Entrada Dispositivos Ayuda

proyectou@projeto-virtualbox:~/smartparking/kafka/kafka

```
Pulse <Intro> para mantener el valor por omisión [*] o pulse un número de selección: 1
update-alternatives: utilizando /usr/lib/jvm/java-11-openjdk-amd64/bin/java para proveer /usr/bin/java (java) en modo manual
(venv) proyectou@projeto-virtualbox:~/smartparking/kafka/kafka$ sudo
update-alternatives --config javac
Existen 2 opciones para la alternativa javac (que provee /usr/bin/java c).

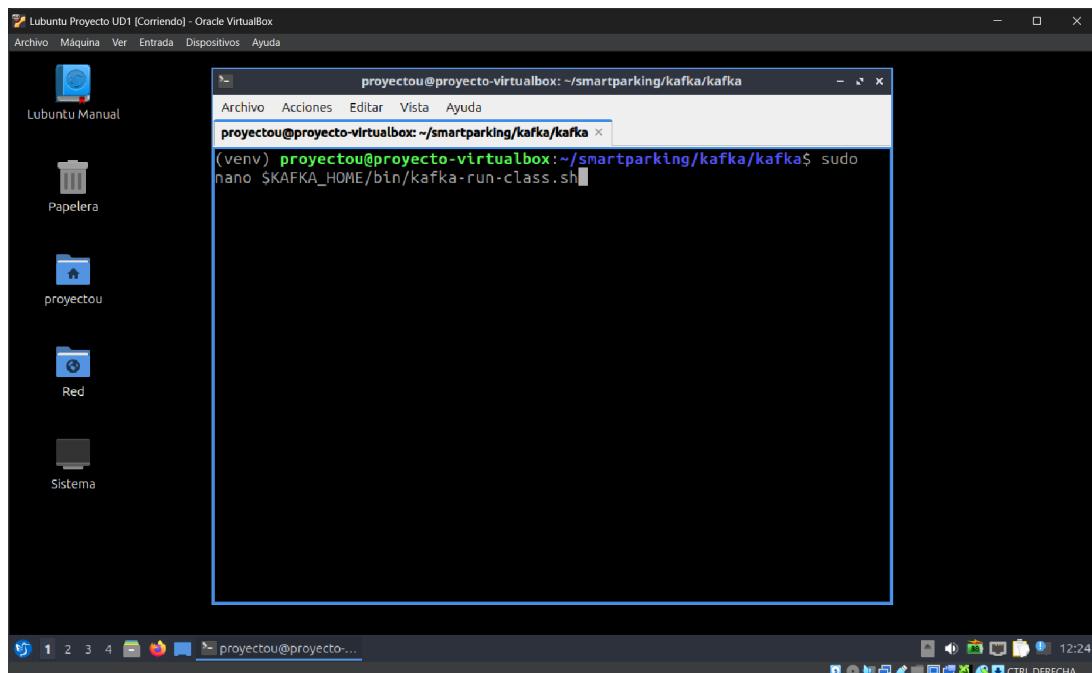
  Selección     Ruta                               Prioridad
  Estado

-----+
* 0           /usr/lib/jvm/java-21-openjdk-amd64/bin/javac   2111
  modo automático
  1           /usr/lib/jvm/java-11-openjdk-amd64/bin/javac   1111
  modo manual
  2           /usr/lib/jvm/java-21-openjdk-amd64/bin/javac   2111
  modo manual

Pulse <Intro> para mantener el valor por omisión [*] o pulse un número de selección: 1
update-alternatives: utilizando /usr/lib/jvm/java-11-openjdk-amd64/bin/javac para proveer /usr/bin/javac (javac) en modo manual
(venv) proyectou@projeto-virtualbox:~/smartparking/kafka/kafka$
```

1 2 3 4 🖥 12:22

Figura C.10: Selección de la versión de 'javac' (JDK 11).



Lubuntu Proyecto UD1 [Corriendo] - Oracle VirtualBox

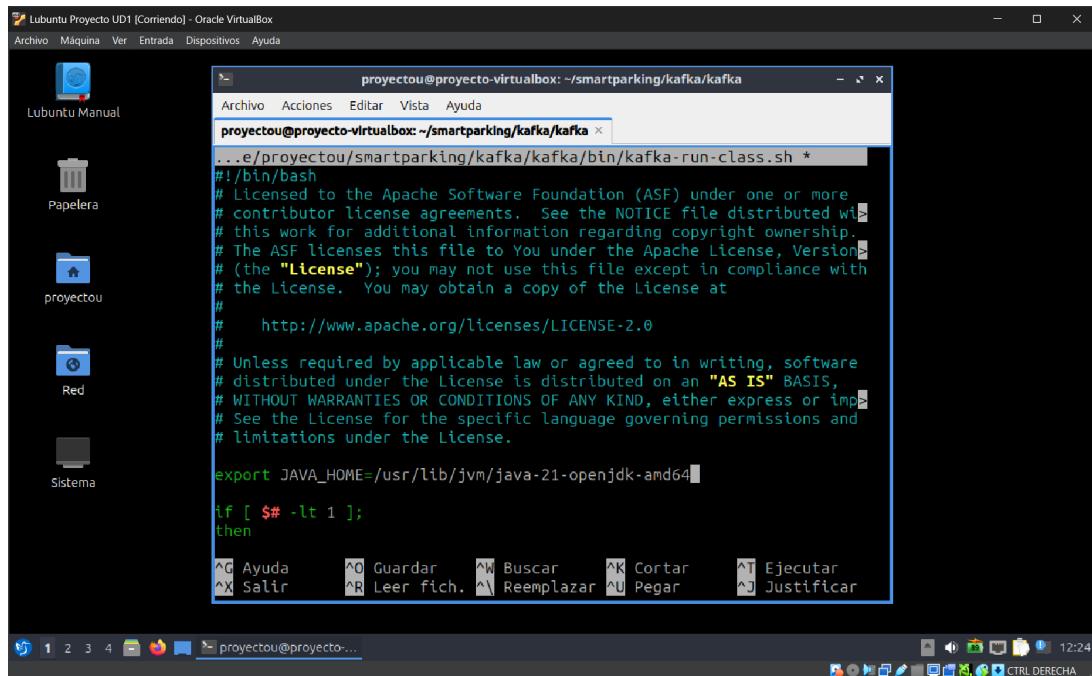
Archivo Máquina Ver Entrada Dispositivos Ayuda

proyectou@projeto-virtualbox:~/smartparking/kafka/kafka

```
(venv) proyectou@projeto-virtualbox:~/smartparking/kafka/kafka$ sudo
nano $KAFKA_HOME/bin/kafka-run-class.sh
```

1 2 3 4 🖥 12:24

Figura C.11: Editando el script 'kafka-run-class.sh'.



Lubuntu Proyecto UD1 [Corriendo] - Oracle VirtualBox

Archivo Máquina Ver Entrada Dispositivos Ayuda

proyectou@projeto-virtualbox: ~/smartparking/kafka/kafka

```
#!/bin/bash
# Licensed to the Apache Software Foundation (ASF) under one or more
# contributor license agreements. See the NOTICE file distributed with
# this work for additional information regarding copyright ownership.
# The ASF licenses this file to You under the Apache License, Version
# (the "License"); you may not use this file except in compliance with
# the License. You may obtain a copy of the License at
#
#     http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.

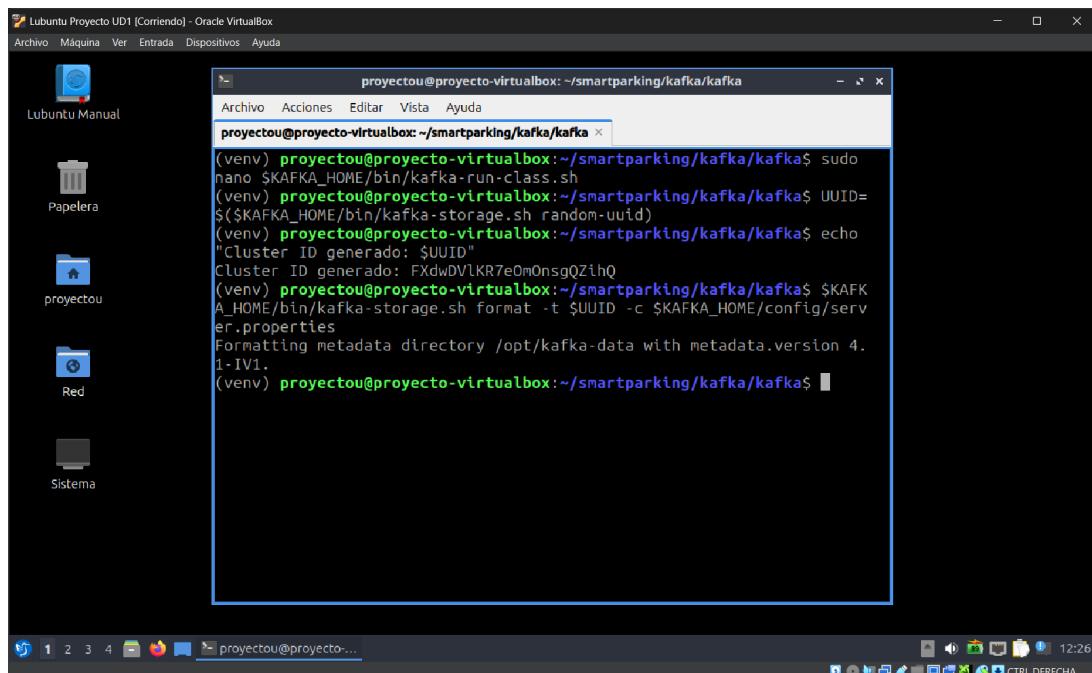
export JAVA_HOME=/usr/lib/jvm/java-21-openjdk-amd64

if [ $# -lt 1 ];
then
```

Ayuda Guardar Buscar Cortar Ejecutar Salir Leer fich. Reemplazar Pegar Justificar

proyectou@projeto-virtualbox: ~

Figura C.12: Añadiendo 'JAVA_HOME' al script 'kafka-run-class.sh'.



Lubuntu Proyecto UD1 [Corriendo] - Oracle VirtualBox

Archivo Máquina Ver Entrada Dispositivos Ayuda

proyectou@projeto-virtualbox: ~/smartparking/kafka/kafka

```
(venv) proyectou@projeto-virtualbox:~/smartparking/kafka/kafka$ sudo
nano $KAFKA_HOME/bin/kafka-run-class.sh
(venv) proyectou@projeto-virtualbox:~/smartparking/kafka/kafka$ UUID=
${$KAFKA_HOME/bin/kafka-storage.sh random-uuid}
(venv) proyectou@projeto-virtualbox:~/smartparking/kafka/kafka$ echo
"Cluster ID generado: $UUID"
Cluster ID generado: FXdwDVlKR7e0mOnsgQZihQ
(venv) proyectou@projeto-virtualbox:~/smartparking/kafka/kafka$ $KAF
KAFKA_HOME/bin/kafka-storage.sh format -t $UUID -c $KAFKA_HOME/config/server.properties
Formatting metadata directory /opt/kafka-data with metadata.version 4.
1-IV1.
(venv) proyectou@projeto-virtualbox:~/smartparking/kafka/kafka$
```

proyectou@projeto-virtualbox: ~

Figura C.13: Formateo del directorio de almacenamiento de Kafka ('kafka-storage.sh format').

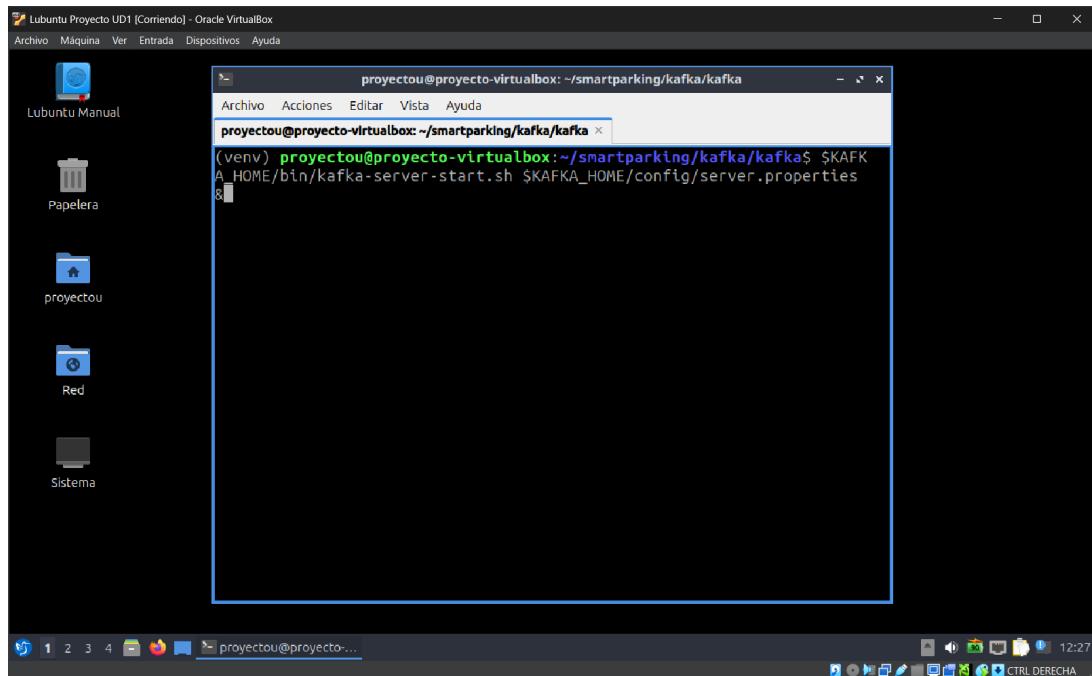


Figura C.14: Inicio del servidor Kafka ('kafka-server-start.sh').

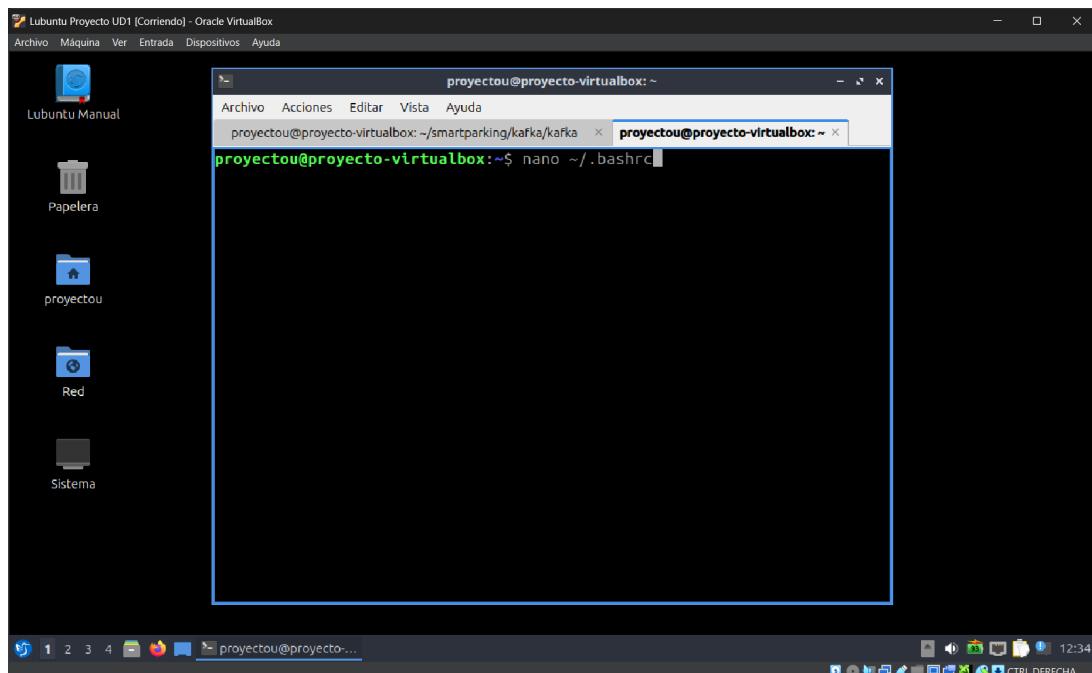


Figura C.15: Editando el archivo ' /.bashrc' para añadir variables de entorno.

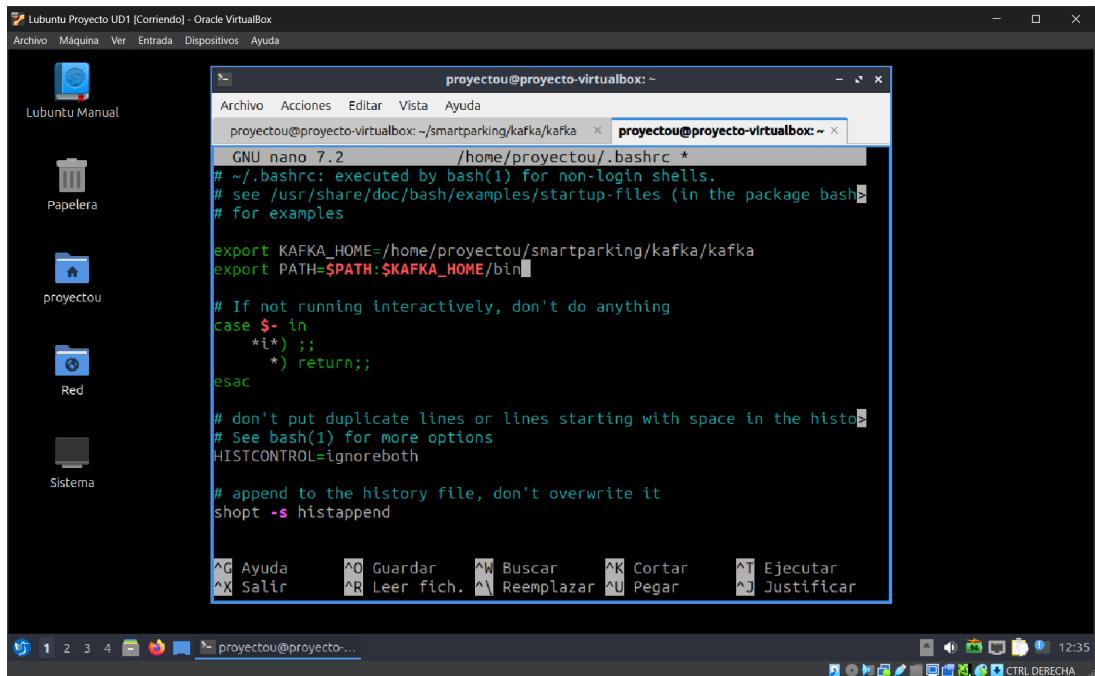


Figura C.16: Añadiendo 'KAFKA_HOME' y actualizando el 'PATH' en ' /.bashrc'.

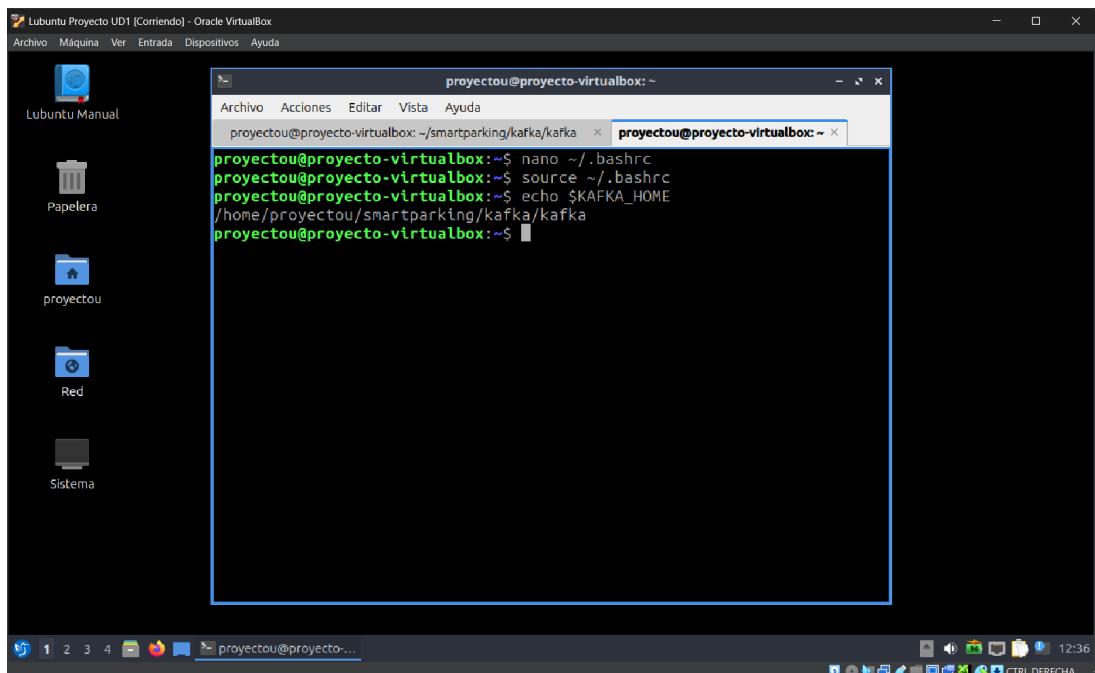


Figura C.17: Recargando la configuración de la terminal ('source ~.bashrc').

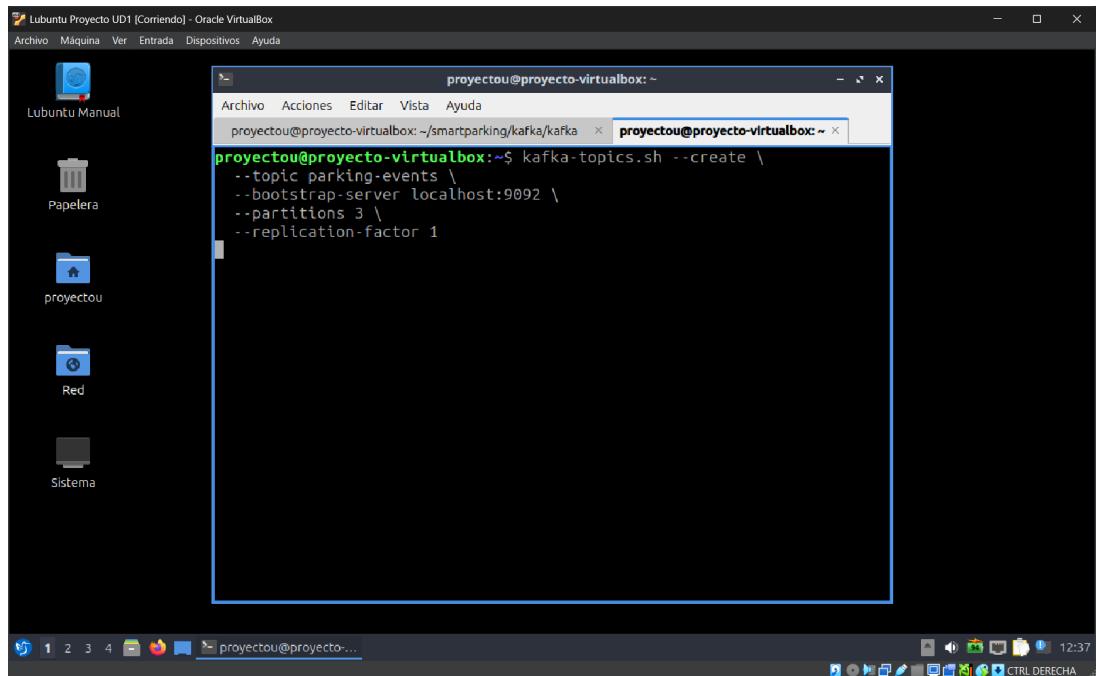


Figura C.18: Comando para crear un nuevo tópico de Kafka.

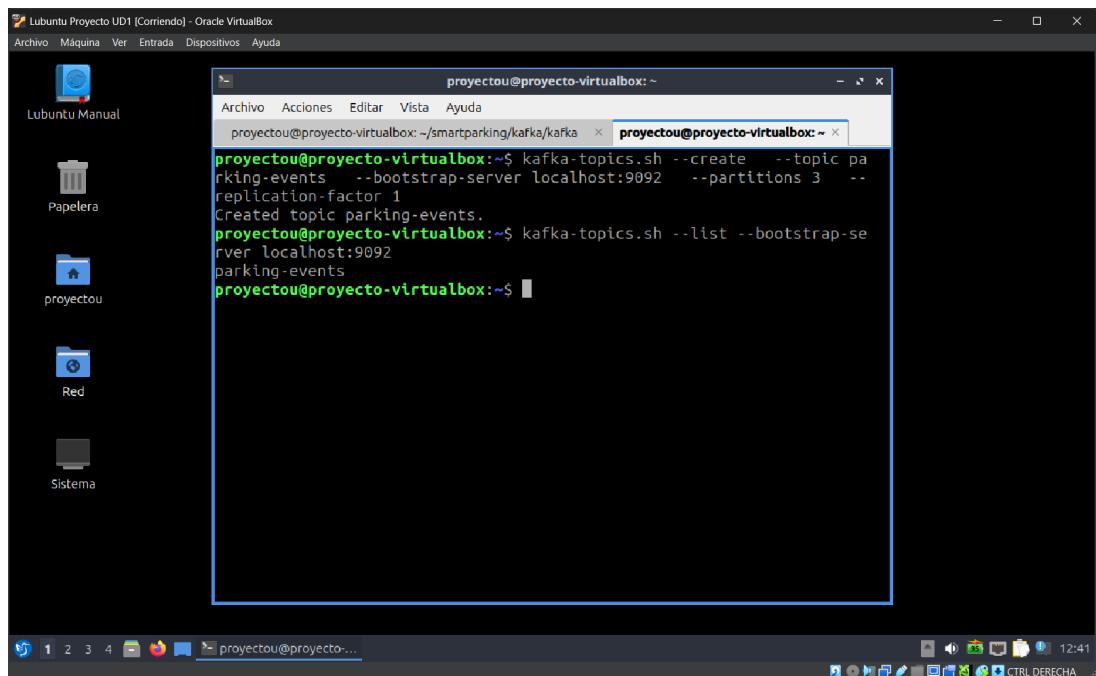


Figura C.19: Creación del tópico `parking-events` y listado de tópicos.

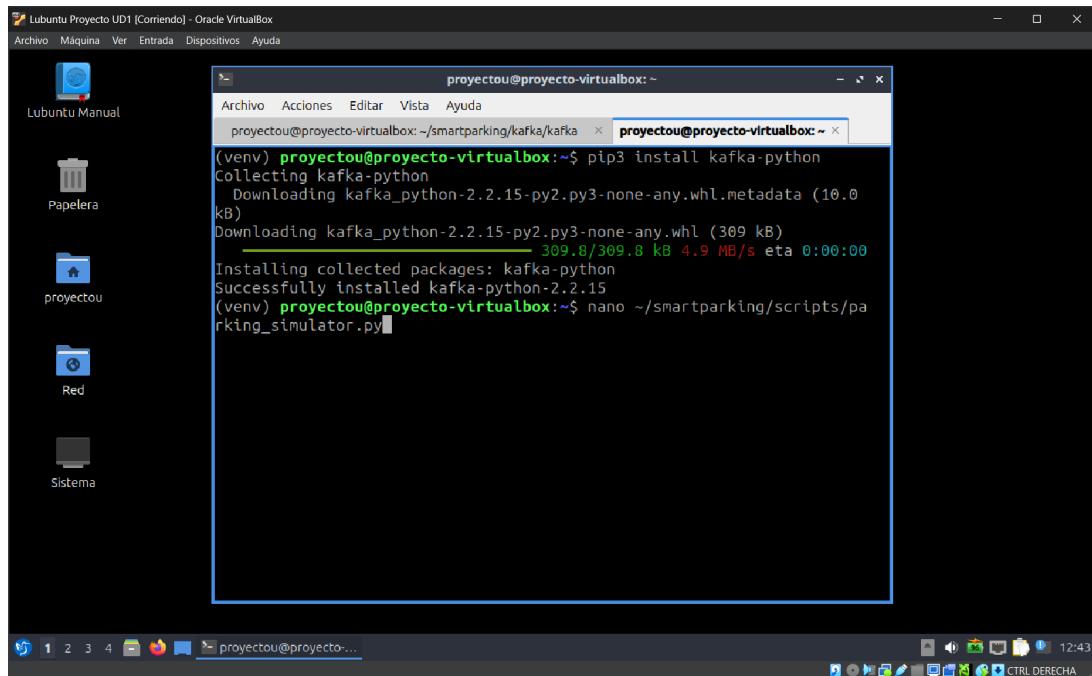


Figura C.20: Instalación de la librería 'kafka-python' con 'pip3'.

```

class ParkingSensorSimulator:
    """Simula sensores de un parking inteligente de 2 plantas con 20 plazas"""
    def __init__(self, parking_id="PK-C012-01", levels=2, bays_per_level=10):
        self.parking_id = parking_id
        self.levels = levels
        self.bays_per_level = bays_per_level
        self.bays = self._initialize_bays()

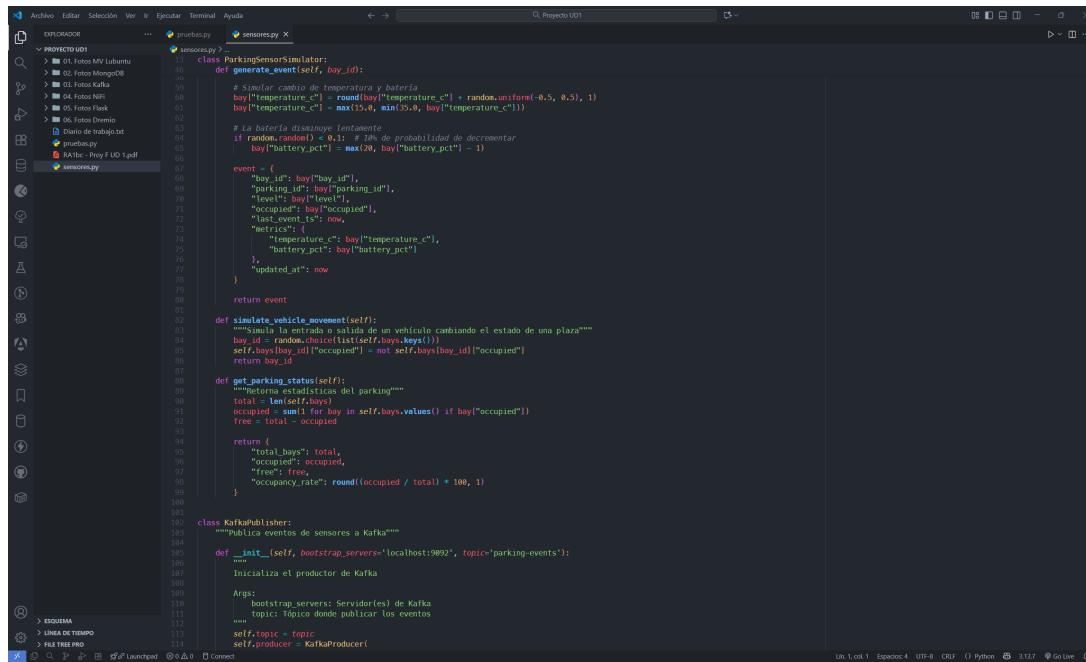
    def _initialize_bays(self):
        """Inicializa el estado de todas las plazas"""
        bays = []
        for level in range(1, self.levels + 1):
            for bay in range(1, self.bays_per_level + 1):
                bay_id = f"({level})-{bay}:{self.bay_id}"
                bays.append({
                    "bay_id": bay_id,
                    "parking_id": self.parking_id,
                    "level": f"({level})",
                    "occupied": random.choice([True, False]),
                    "temperature": round(random.uniform(10.0, 28.0), 1),
                    "battery_pct": random.randint(60, 100)
                })
        return bays

    def generate_event(self, bay_id):
        """Genera un evento de sensor para una plaza específica"""
        bay = self.bays[bay_id]
        now = datetime.utcnow().strftime("%Y-%m-%dT%H:%M:%S%z")
        return {
            "bay_id": bay_id,
            "event_time": now,
            "occupancy": bay["occupied"],
            "temperature": bay["temperature"],
            "battery_pct": bay["battery_pct"]
        }

```

Figura C.21: Código del simulador de sensores: 'ParkingSensorSimulator.__init__'.

SmartParking Flow — Monitorización Inteligente



```

    Archivo Editar Selección Ver Ir Ejecutar Terminal Ayuda
    PROYECTO U01 ...
    pruebas.py sensors.py X
    sensors.py @ run_simulation
    class ParkingSensorSimulator:
        def generate_event(self, bay_id):
            # Similar cambio de temperatura y batería
            bay["temperature_c"] = round(bay["temperature_c"] + random.uniform(-0.5, 0.5), 1)
            bay["temperature_c"] = max(15.0, min(35.0, bay["temperature_c"]))
            # La batería disminuye lentamente
            if random.random() < 0.1: # 10% de probabilidad de decrementar
                bay["battery_pct"] = max(20, bay["battery_pct"] - 1)

            event = {
                "bay_id": bay["bay_id"],
                "parking_id": bay["parking_id"],
                "level": bay["level"],
                "occupied": bay["occupied"],
                "last_event_ts": now,
                "metas": [
                    {"temperature_c": bay["temperature_c"],
                     "battery_pct": bay["battery_pct"]}
                ],
                "updated_at": now
            }
            return event

        def simulate_vehicle_movement(self):
            """Simula el movimiento de un vehículo cambiando el estado de una plaza"""
            bay_id = random.choice(list(self.bays.keys()))
            self.bays[bay_id]["occupied"] = not self.bays[bay_id]["occupied"]
            return bay_id

        def get_parking_status(self):
            """Retorna estadísticas del parking"""
            total = len(self.bays)
            occupied = sum(1 for bay in self.bays.values() if bay["occupied"])
            free = total - occupied
            return {
                "total_bays": total,
                "occupied": occupied,
                "free": free,
                "occupancy_rate": round(occupied / total) * 100
            }

        class KafkaPublisher:
            """Publica eventos de sensores a Kafka"""
            def __init__(self, bootstrap_servers='localhost:9092', topic='parking-events'):
                """Inicializa el productor de Kafka
                Args:
                    bootstrap_servers: Servidor(es) de Kafka
                    topic: Tópico donde publicar los eventos
                """
                self.topic = topic
                self.producer = KafkaProducer(
                    bootstrap_servers=bootstrap_servers,
                    value_serializer=lambda v: json.dumps(v).encode('utf-8'),
                    acks='all',
                    retries=3
                )

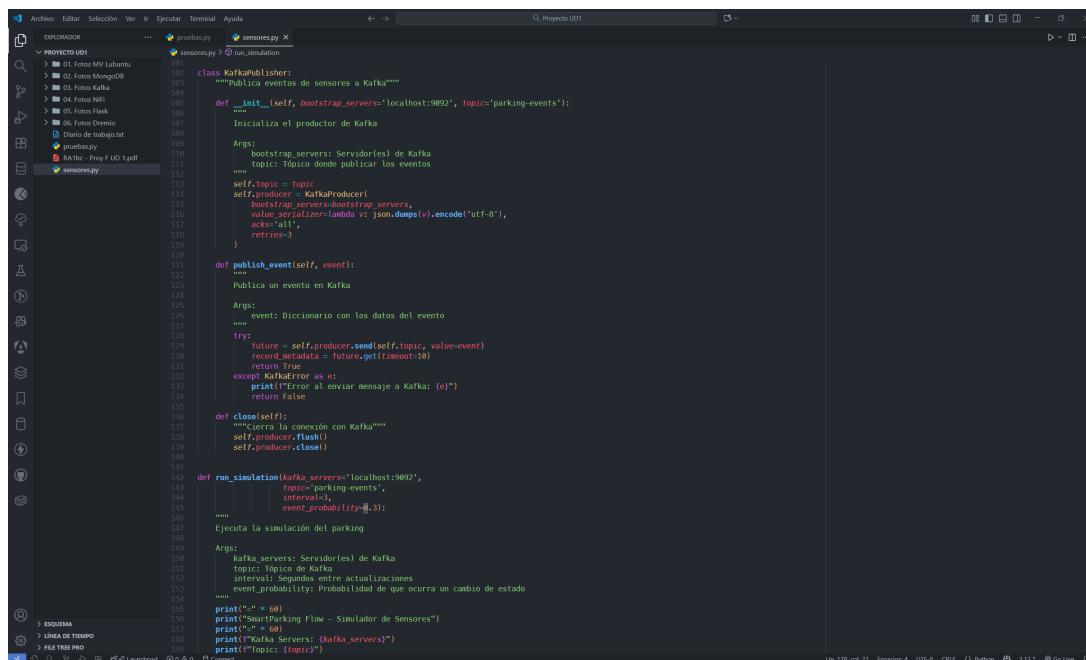
            def publish_event(self, event):
                """Publica un evento en Kafka
                Args:
                    event: Diccionario con los datos del evento
                """
                try:
                    future = self.producer.send(self.topic, value=event)
                    record_metadata = future.get(timeout=10)
                    return True
                except KafkaError as e:
                    print(f"Error al enviar mensaje a Kafka: {e}")
                    return False

            def close(self):
                """Cierra la conexión con Kafka"""
                self.producer.flush()
                self.producer.close()

            def run_simulation(self, kafka_servers='localhost:9092', topic='parking-events', interval=1, event_probability=0.3):
                """Ejecuta la simulación del parking
                Args:
                    kafka_servers: Servidor(es) de Kafka
                    topic: Tópico de Kafka
                    interval: Segundos entre actualizaciones
                    event_probability: Probabilidad de que ocurra un cambio de estado
                """
                print("-" * 60)
                print("SmartParking Flow - Simulador de Sensores")
                print(f"Kafka Servers: {kafka_servers}")
                print(f"Topic: {topic}")

```

Figura C.22: Código del simulador de sensores: 'ParkingSensorSimulator.generate_event'.



```

    Archivo Editar Selección Ver Ir Ejecutar Terminal Ayuda
    PROYECTO U01 ...
    pruebas.py sensors.py X
    sensors.py @ run_simulation
    class KafkaPublisher:
        """Publica eventos de sensores a Kafka"""
        def __init__(self, bootstrap_servers='localhost:9092', topic='parking-events'):
            """Inicializa el productor de Kafka
            Args:
                bootstrap_servers: Servidor(es) de Kafka
                topic: Tópico donde publicar los eventos
            """
            self.topic = topic
            self.producer = KafkaProducer(
                bootstrap_servers=bootstrap_servers,
                value_serializer=lambda v: json.dumps(v).encode('utf-8'),
                acks='all',
                retries=3
            )

        def publish_event(self, event):
            """Publica un evento en Kafka
            Args:
                event: Diccionario con los datos del evento
            """
            try:
                future = self.producer.send(self.topic, value=event)
                record_metadata = future.get(timeout=10)
                return True
            except KafkaError as e:
                print(f"Error al enviar mensaje a Kafka: {e}")
                return False

        def close(self):
            """Cierra la conexión con Kafka"""
            self.producer.flush()
            self.producer.close()

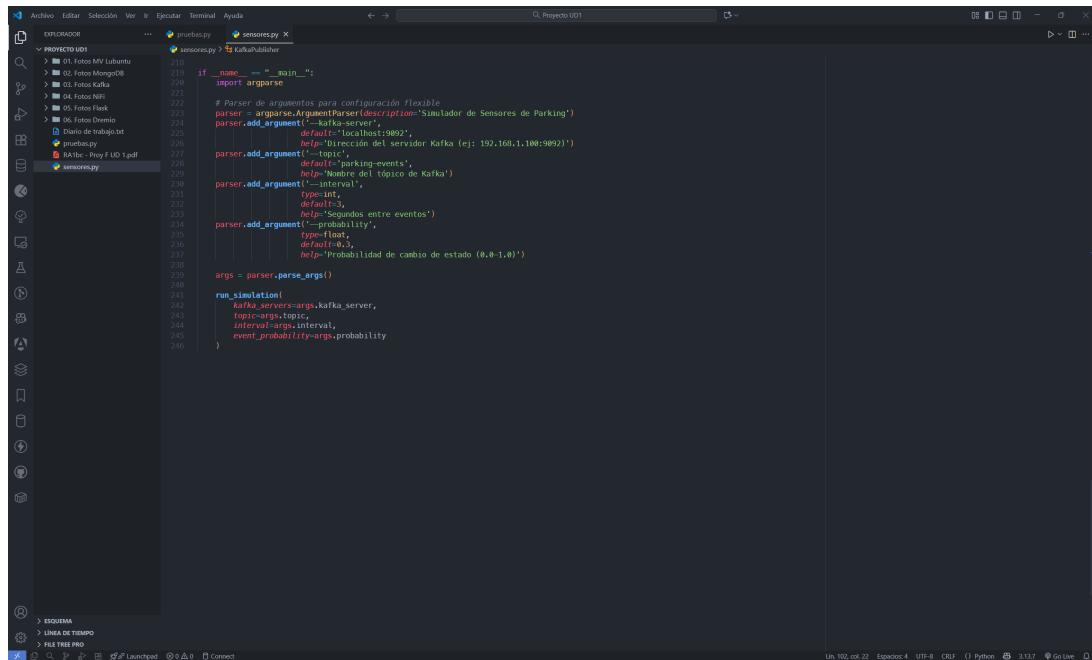
        def run_simulation(self, kafka_servers='localhost:9092', topic='parking-events', interval=1, event_probability=0.3):
            """Ejecuta la simulación del parking
            Args:
                kafka_servers: Servidor(es) de Kafka
                topic: Tópico de Kafka
                interval: Segundos entre actualizaciones
                event_probability: Probabilidad de que ocurra un cambio de estado
            """
            print("-" * 60)
            print("SmartParking Flow - Simulador de Sensores")
            print(f"Kafka Servers: {kafka_servers}")
            print(f"Topic: {topic}")

```

Figura C.23: Código del publicador de Kafka: 'KafkaPublisher'.

```
142     def run_simulation(kafka_servers='localhost:9092',
143                         topic='parking-events',
144                         interval=3,
145                         event_probability=0.3):
146         """
147             Ejecuta la simulación del parking
148
149         Args:
150             kafka_servers: Servidor(es) de Kafka
151             topic: Tópico de Kafka
152             interval: Segundos entre actualizaciones
153             event_probability: Probabilidad de que ocurra un cambio de estado
154         """
155         print("=" * 60)
156         print("SmartParking Flow - Simulador de Sensores")
157         print("=" * 60)
158         print(f"Kafka Servers: {kafka_servers}")
159         print(f"Topic: {topic}")
160         print(f"Intervalo: {interval} segundos")
161         print("=" * 60)
162
163         # Inicializar simulador y publisher
164         simulator = ParkingSensorSimulator()
165         publisher = KafkaPublisher(bootstrap_servers=kafka_servers, topic=topic)
166
167         print("\nEstado inicial del parking:")
168         status = simulator.get_parking_status()
169         print(f" Total plazas: {status['total_bays']}")
170         print(f" Ocupadas: {status['occupied']}")
171         print(f" Libres: {status['free']}")
172         print(f" Tasa ocupación: {status['occupancy_rate']}%")
173         print("\n" + "=" * 60)
174         print("Iniciando simulación... (Ctrl+C para detener)")
175         print("=" * 60 + "\n")
176
177     try:
178         iteration = 0
179         while True:
180             iteration += 1
181
182             # Decidir si hay movimiento de vehículos
183             if random.random() < event_probability:
184                 bay_id = simulator.simulate_vehicle_movement()
185                 event = simulator.generate_event(bay_id)
186
187                 # Publicar en Kafka
188                 if publisher.publish_event(event):
189                     action = "ENTRADA" if event["occupied"] else "SALIDA"
190                     print(f"[{iteration:04d}] {action} en {bay_id} "
191                         f"(Temp: {event['metrics']['temperature_c']}°C, "
192                         f"Batería: {event['metrics']['battery_pct']}%)")
193                 else:
194                     print(f"[{iteration:04d}] ERROR al publicar evento para {bay_id}")
195
196             # Cada 10 iteraciones, mostrar estadísticas
197             if iteration % 10 == 0:
198                 status = simulator.get_parking_status()
199                 print(f"\n--- Estado del parking (iteración {iteration}) ---")
200                 print(f"Ocupadas: {status['occupied']}/{status['total_bays']} "
201                     f"({status['occupancy_rate']}%)")
202                 print("-" * 50 + "\n")
203
204             time.sleep(interval)
205
206     except KeyboardInterrupt:
207         print("\n\n" + "=" * 60)
208         print("Simulación detenida por el usuario")
209         print("=" * 60)
210     finally:
211         publisher.close()
212         print("\nEstadísticas finales:")
213         status = simulator.get_parking_status()
214         print(f" Ocupadas: {status['occupied']}/{status['total_bays']} ")
215         print(f" Libres: {status['free']} ")
216         print(f" Tasa ocupación: {status['occupancy_rate']}%")
```

Figura C.24: Código de la función 'run_simulation' (lógica principal del productor).



```

# Archivo: sensors.py
# Proyecto: SmartParking
# Descripción: Simulador de Sensores de Parking
# Autor: [REDACTED]
# Fecha: [REDACTED]

if __name__ == "__main__":
    import argparse

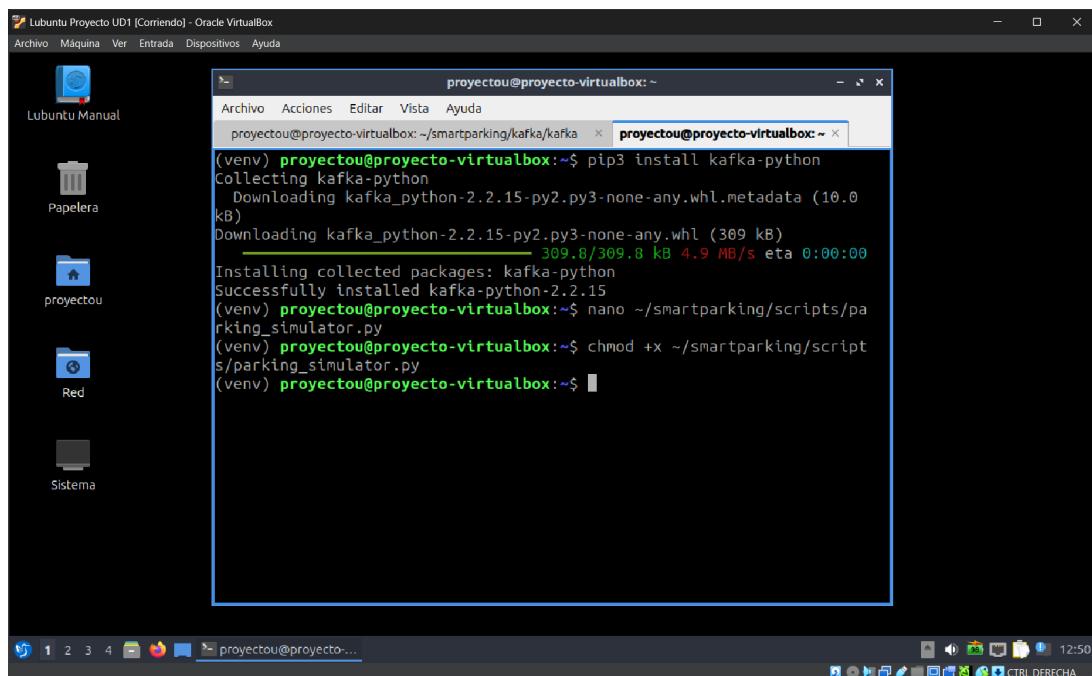
    # Parser de argumentos para configuración flexible
    parser = argparse.ArgumentParser(description='Simulador de Sensores de Parking')
    parser.add_argument('--kafka-server',
                       default='localhost:9092',
                       help='Dirección del servidor Kafka (ej: 192.168.1.100:9092)')
    parser.add_argument('--topic',
                       default='parking-events',
                       help='Nombre del tema de Kafka')
    parser.add_argument('--interval',
                       type=int,
                       default=3,
                       help='Intervalo entre eventos')
    parser.add_argument('--probability',
                       type=float,
                       default=0.5,
                       help='Probabilidad de cambio de estado (0.0-1.0)')

    args = parser.parse_args()

    run_simulation(
        kafka_servers=args.kafka_server,
        topic=args.topic,
        interval=args.interval,
        event_probability=args.probability
    )

```

Figura C.25: Código del 'argparse' para configurar el simulador desde la terminal.



```

proyectou@projeto-virtualbox:~$ pip3 install kafka-python
Collecting kafka-python
  Downloading kafka_python-2.2.15-py2.py3-none-any.whl.metadata (10.0 kB)
  Downloading kafka_python-2.2.15-py2.py3-none-any.whl (309 kB)
    309.8/309.8 kB 4.9 MB/s eta 0:00:00
Installing collected packages: kafka-python
Successfully installed kafka-python-2.2.15
proyectou@projeto-virtualbox:~$ nano ~/smartparking/scripts/parking_simulator.py
proyectou@projeto-virtualbox:~$ chmod +x ~/smartparking/scripts/parking_simulator.py
proyectou@projeto-virtualbox:~$ 

```

Figura C.26: Dando permisos de ejecución ('chmod +x') al script del simulador.

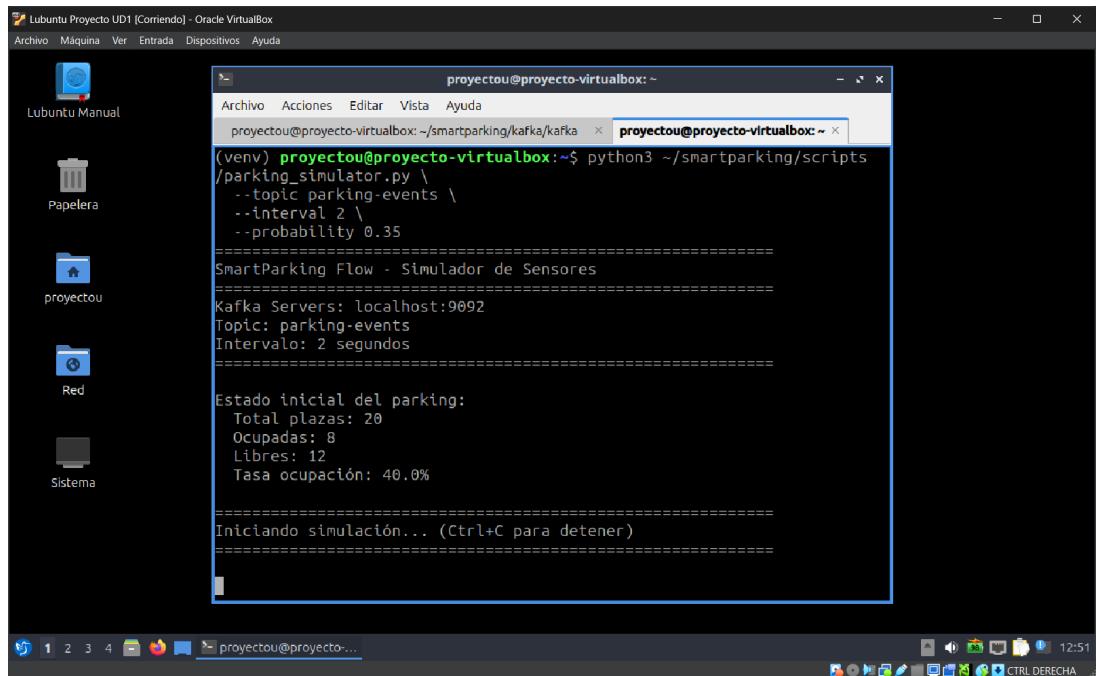


Figura C.27: Ejecución del script simulador de sensores ('parking_simulator.py').

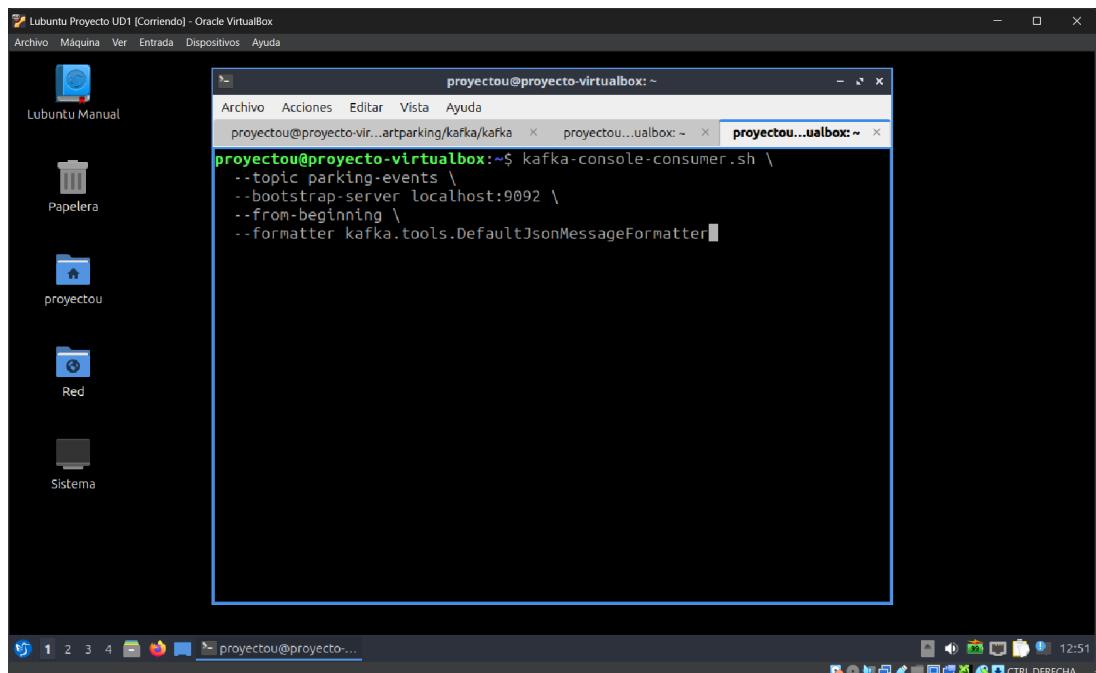


Figura C.28: Comando para iniciar un consumidor de consola en formato JSON.

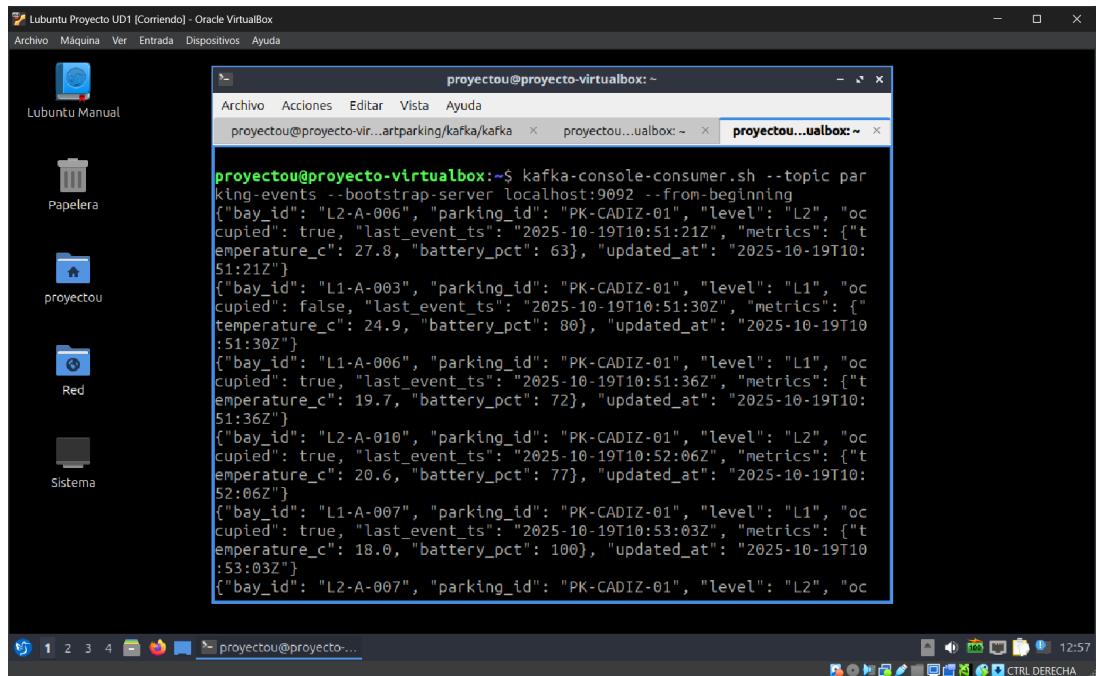
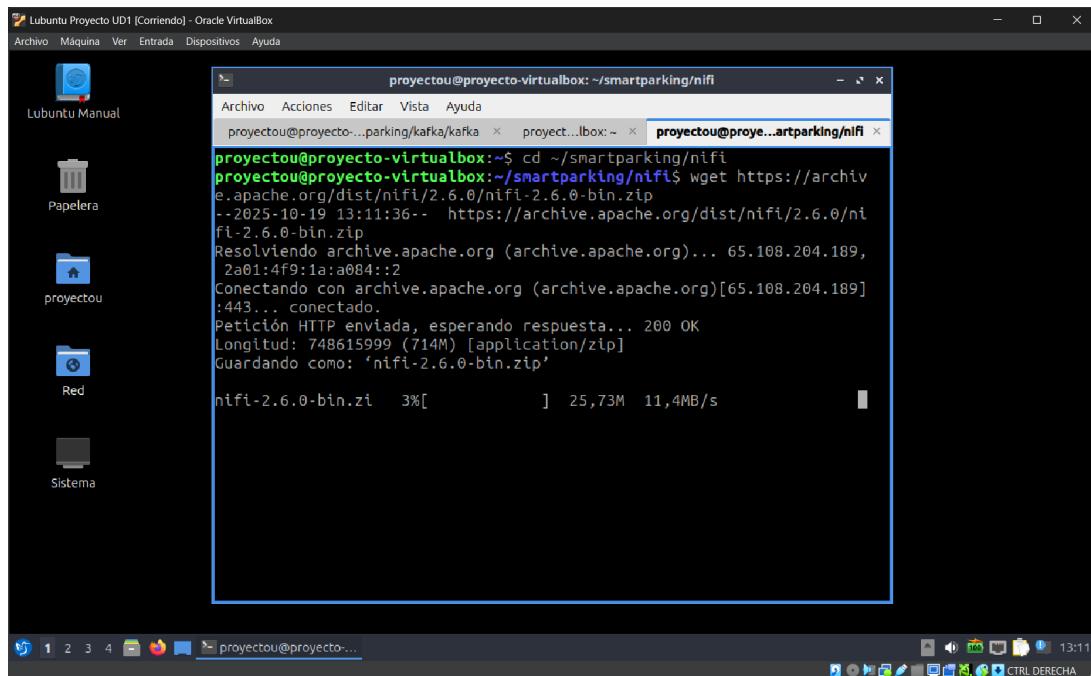


Figura C.29: Consumidor de consola de Kafka mostrando los mensajes JSON entrantes.

Apéndice D

Anexo D: Configuración del Flujo de Ni-Fi

Instalación, configuración y diseño del *pipeline* de datos en Apache NiFi para procesar y enrutar los eventos de Kafka a MongoDB.

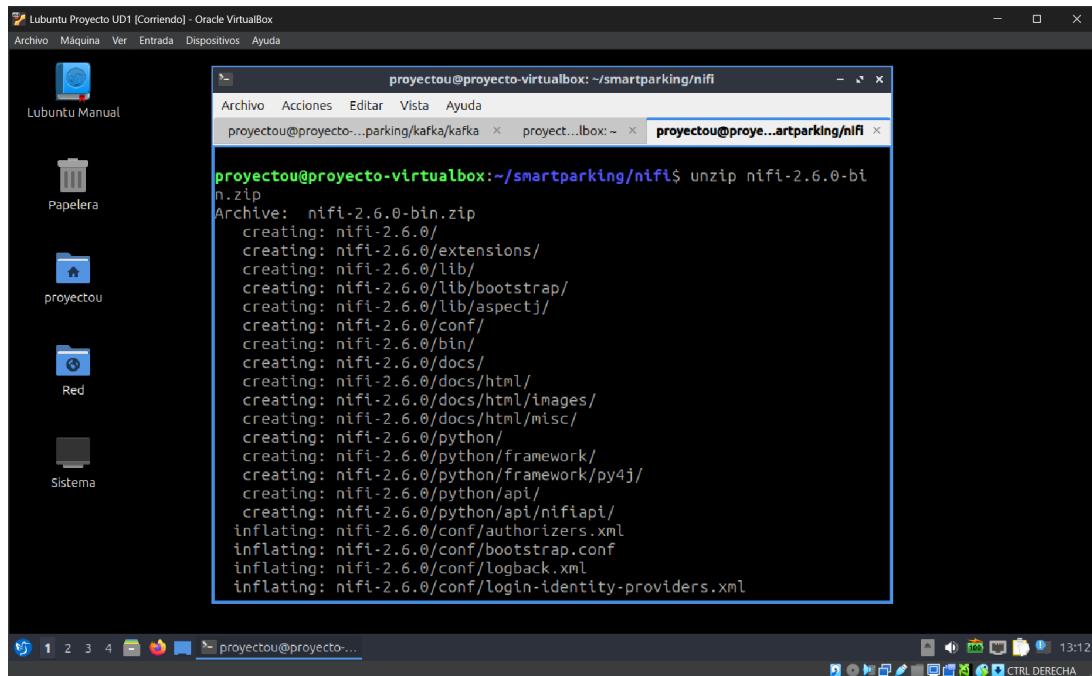


The screenshot shows a terminal window titled "proyecto@projeto-virtualbox: ~/smartparking/nifi". The window is part of a desktop environment with a sidebar containing icons for Lubuntu Manual, Papelera, proyecto, Red, and Sistema. The terminal content is as follows:

```
proyecto@projeto-virtualbox:~$ cd ~/smartparking/nifi
proyecto@projeto-virtualbox:~/smartparking/nifi$ wget https://archive.apache.org/dist/nifi/2.6.0/nifi-2.6.0-bin.zip
--2025-10-19 13:11:36-- https://archive.apache.org/dist/nifi/2.6.0/nifi-2.6.0-bin.zip
Resolviendo archive.apache.org (archive.apache.org)... 65.108.204.189,
2a01:4f9:1a::084::2
Conectando con archive.apache.org (archive.apache.org)[65.108.204.189]:443... conectado.
Petición HTTP enviada, esperando respuesta... 200 OK
Longitud: 748615999 (714M) [application/zip]
Guardando como: 'nifi-2.6.0-bin.zip'

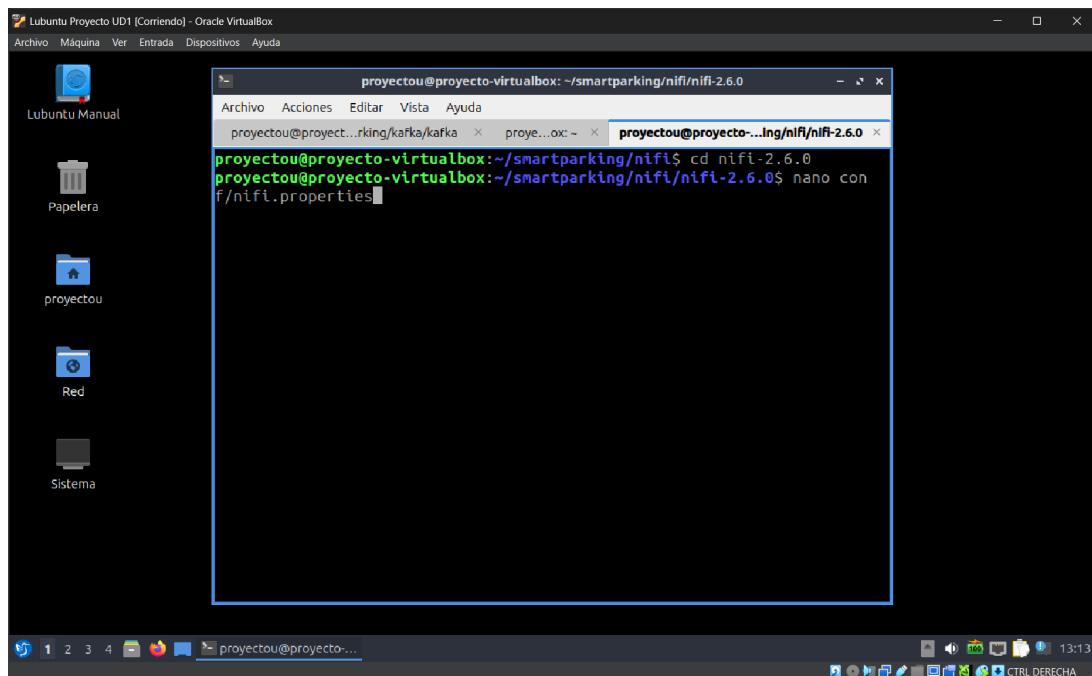
nifi-2.6.0-bin.zip 3%[          ] 25,73M 11,4MB/s
```

Figura D.1: Descarga del binario de Apache NiFi ('wget').



A screenshot of a terminal window titled "proyecto@projeto-virtualbox: ~/smartparking/nifi". The window shows the command "unzip nifi-2.6.0-bin.zip" being run, which extracts files from the "nifi-2.6.0-bin.zip" archive into the current directory. The extracted files include various subdirectories and configuration files like "bin", "conf", "docs", "extensions", "lib", and "logback.xml". The terminal is running on a Lubuntu desktop environment.

Figura D.2: Descompresión del archivo ('unzip').



A screenshot of a terminal window titled "proyecto@projeto-virtualbox: ~/smartparking/nifi/nifi-2.6.0". The user has navigated to the "nifi-2.6.0" directory. The command "nano conf/nifi.properties" is being run to edit the configuration file. The terminal is running on a Lubuntu desktop environment.

Figura D.3: Accediendo al directorio de configuración ('conf').

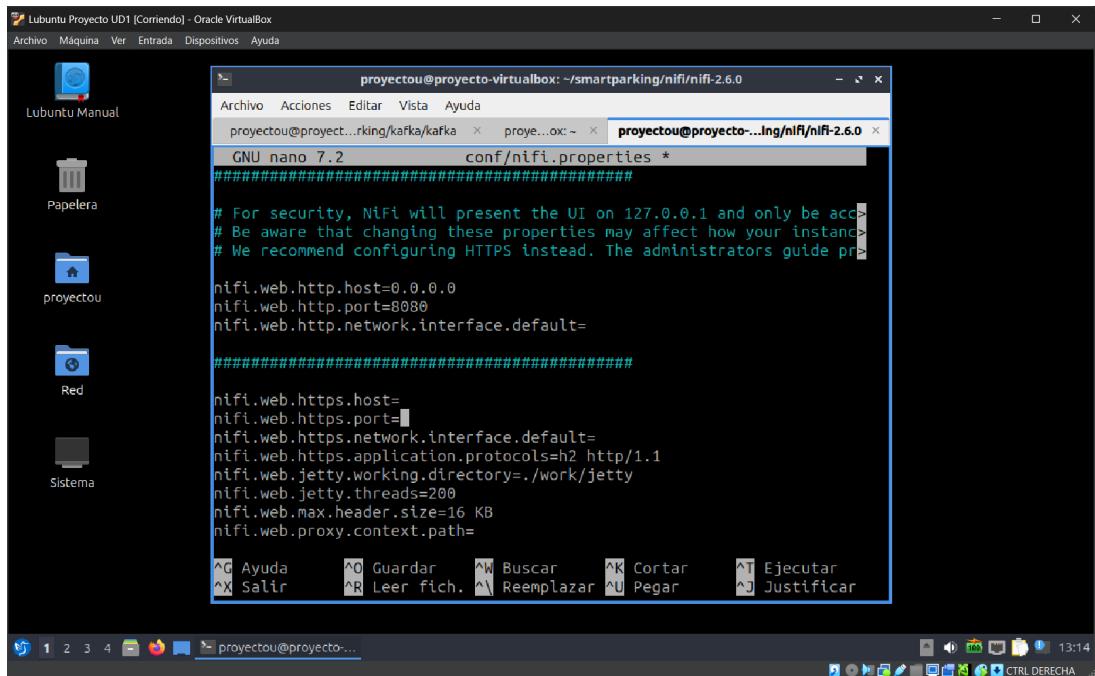


Figura D.4: Editando 'nifi.properties': 'nifi.web.http.host=0.0.0.0'.

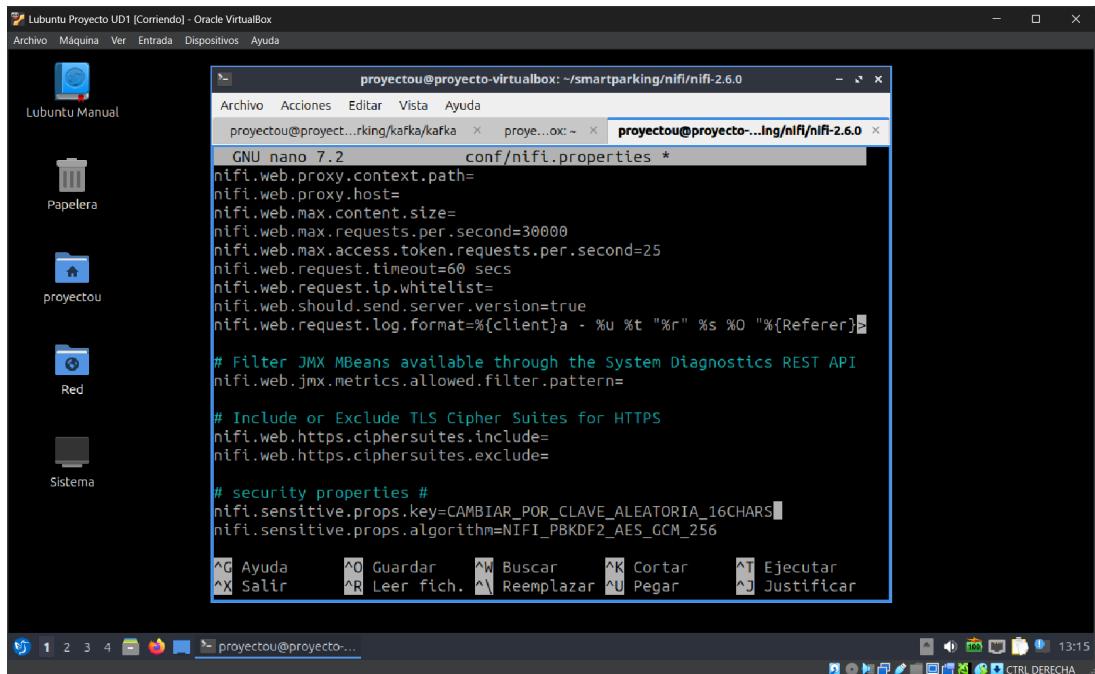
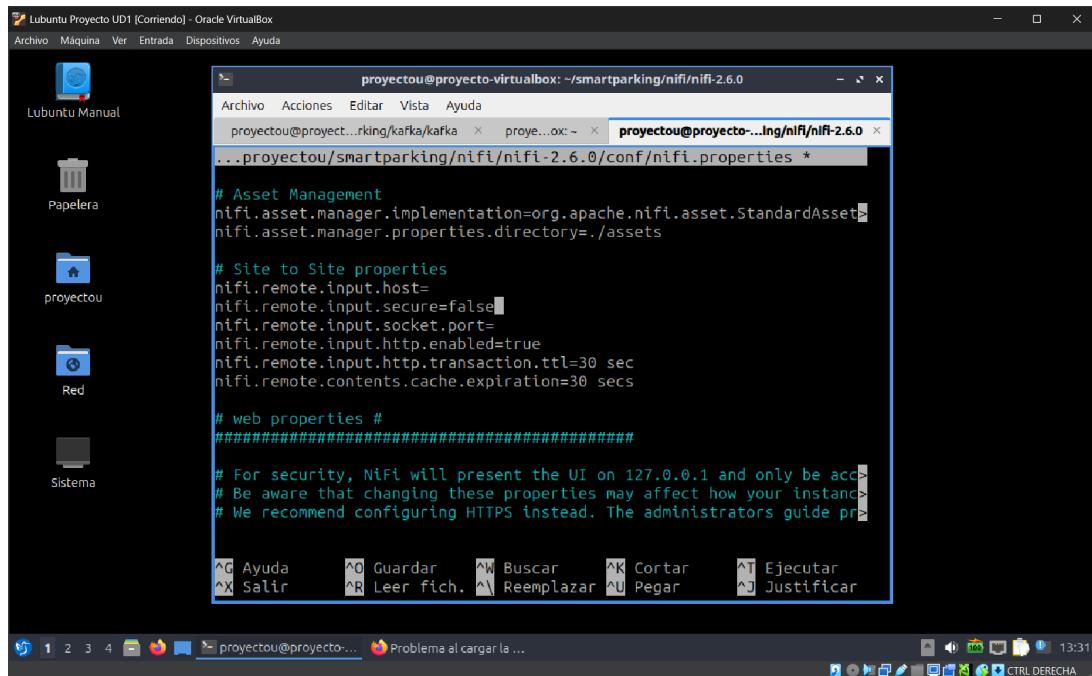


Figura D.5: Editando 'nifi.properties': 'nifi.sensitive.props.key'.



```
# Asset Management
nifi.asset.manager.implementation=org.apache.nifi.asset.StandardAsset
nifi.asset.manager.properties.directory=./assets

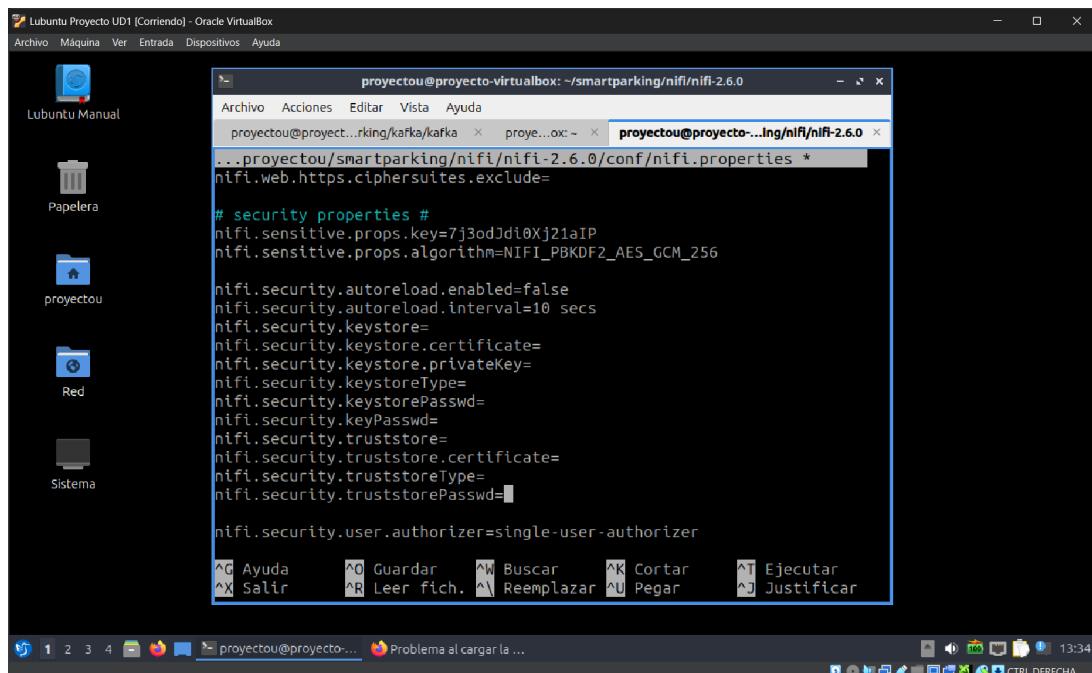
# Site to Site properties
nifi.remote.input.host=
nifi.remote.input.secure=false
nifi.remote.input.socket.port=
nifi.remote.input.http.enabled=true
nifi.remote.input.http.transaction.ttl=30 sec
nifi.remote.contents.cache.expiration=30 secs

# web properties #
#####
# For security, NiFi will present the UI on 127.0.0.1 and only be accessible via https
# Be aware that changing these properties may affect how your instance behaves
# We recommend configuring HTTPS instead. The administrators guide provides more information

nifi.web.https.ciphersuites.exclude=
nifi.sensitive.props.key=7j3odJdi0Xj21aIP
nifi.sensitive.props.algorithm=NIFI_PBKDF2_AES_GCM_256

nifi.security.autoreload.enabled=false
nifi.security.autoreload.interval=10 secs
nifi.security.keystore=
nifi.security.keystore.certificate=
nifi.security.keystore.privateKey=
nifi.security.keystore.type=
nifi.security.keystorePasswd=
nifi.security.keyPasswd=
nifi.security.truststore=
nifi.security.truststore.certificate=
nifi.security.truststore.type=
nifi.security.truststorePasswd=
```

Figura D.6: Editando 'nifi.properties': 'nifi.remote.input.host'.



```
nifi.web.https.ciphersuites.exclude=
nifi.sensitive.props.key=7j3odJdi0Xj21aIP
nifi.sensitive.props.algorithm=NIFI_PBKDF2_AES_GCM_256

nifi.security.autoreload.enabled=false
nifi.security.autoreload.interval=10 secs
nifi.security.keystore=
nifi.security.keystore.certificate=
nifi.security.keystore.privateKey=
nifi.security.keystore.type=
nifi.security.keystorePasswd=
nifi.security.keyPasswd=
nifi.security.truststore=
nifi.security.truststore.certificate=
nifi.security.truststore.type=
nifi.security.truststorePasswd=
```

Figura D.7: Editando 'nifi.properties': 'nifi.security.user.authorizer'.

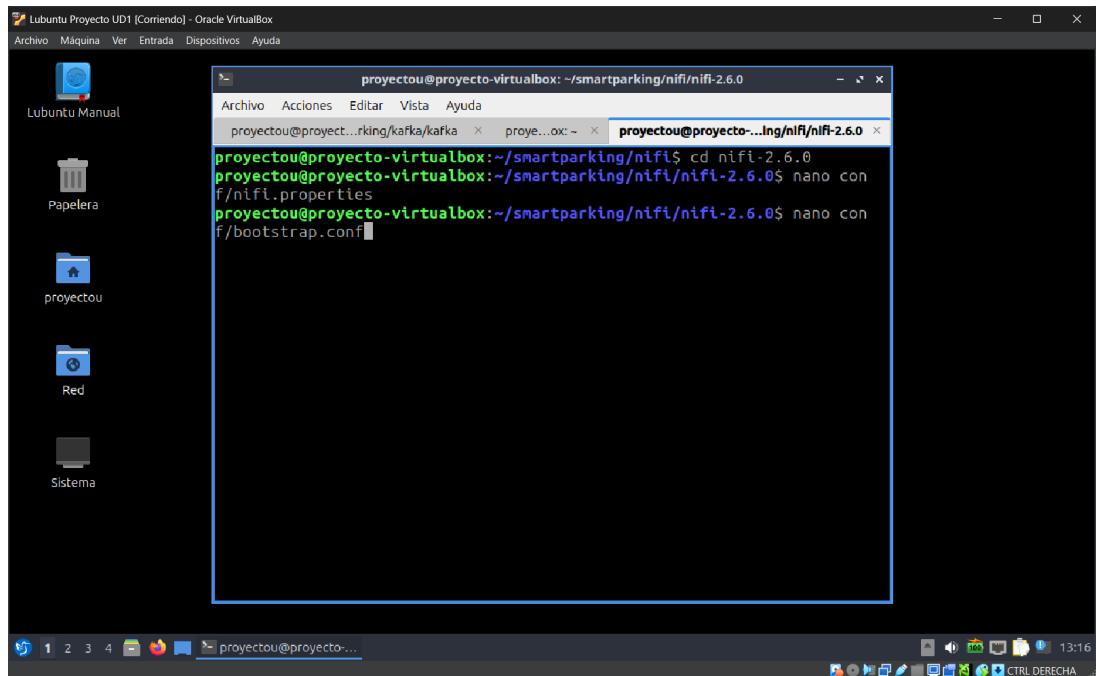


Figura D.8: Accediendo al archivo 'bootstrap.conf'.

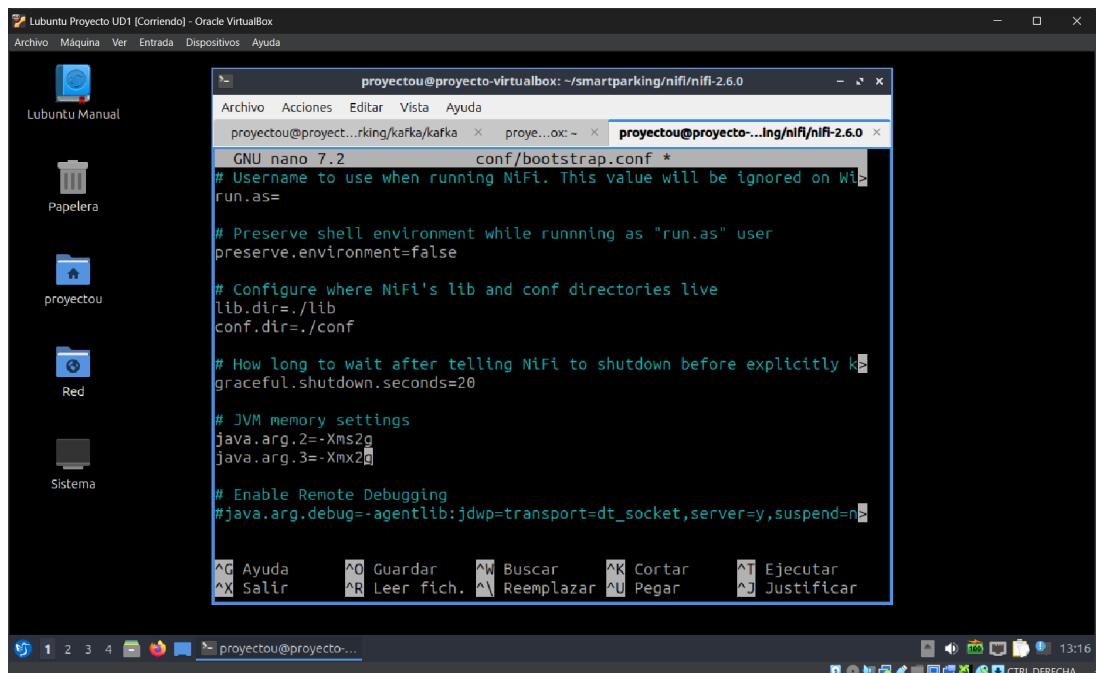


Figura D.9: Editando 'bootstrap.conf': Configuración de memoria JVM (Xms2g, Xmx2g).

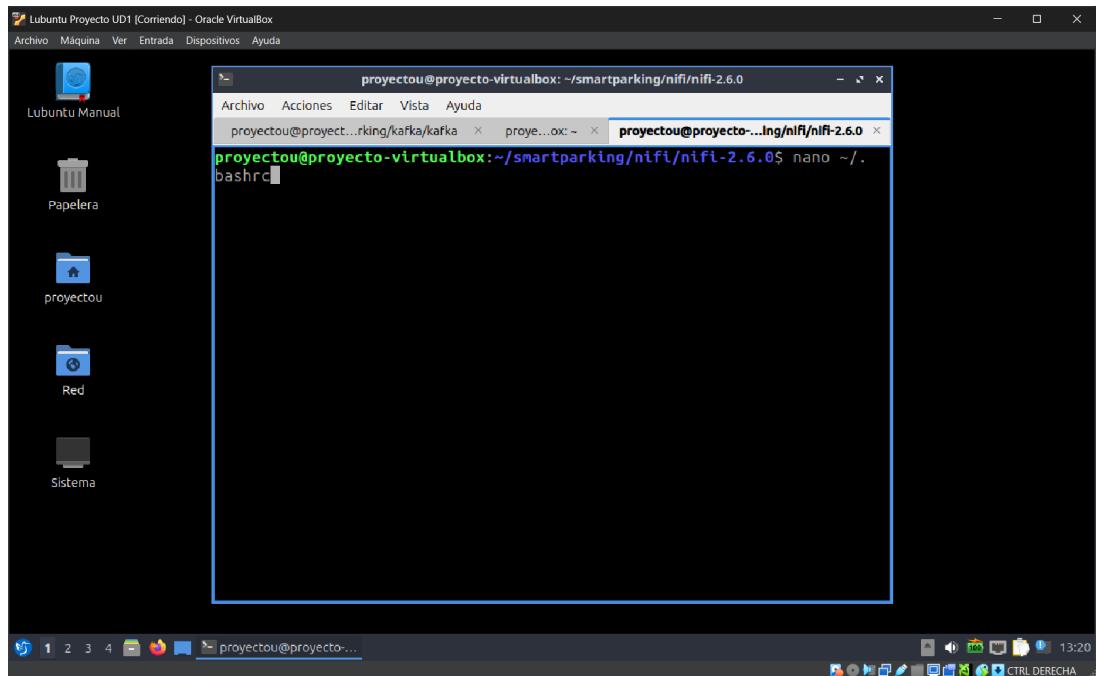


Figura D.10: Editando ' /.bashrc' para añadir 'NIFI_HOME'.

A screenshot of a Linux desktop environment, likely Ubuntu, running in Oracle VirtualBox. The desktop has a dark theme with icons for Lubuntu Manual, Papelera, proyecto, Red, and Sistema. A terminal window titled 'proyectou@proj...virtualbox:~/smartparking/nifi/nifi-2.6.0' is open, showing the contents of the .bashrc file. The file includes environment variable assignments for KAFKA_HOME, PATH, and NIFI_HOME, along with other bash startup logic. The terminal prompt shows the user is in their home directory under the 'nifi' folder.

```
GNU nano 7.2 /home/proyectou/.bashrc *
# ~/.bashrc: executed by bash(1) for non-login shells.
# see /usr/share/doc/bash/examples/startup-files (in the package bash)
# for examples

export KAFKA_HOME=/home/proyectou/smartparking/kafka/kafka
export PATH=$KAFKA_HOME/bin
export NIFI_HOME=/home/proyectou/smartparking/nifi/nifi-2.6.0
export PATH=$PATH:$NIFI_HOME/bin

# If not running interactively, don't do anything
case $- in
    *i*) ;;
    *) return;;
esac

# don't put duplicate lines or lines starting with space in the history
# See bash(1) for more options
HISTCONTROL=ignoreboth

# append to the history file, don't overwrite it
^G Ayuda      ^O Guardar     ^W Buscar      ^K Cortar      ^T Ejecutar
^X Salir      ^R Leer fich.  ^\ Reemplazar  ^U Pegar       ^J Justificar
```

Figura D.11: Añadiendo 'NIFI_HOME' y actualizando el 'PATH' en ' /.bashrc'.

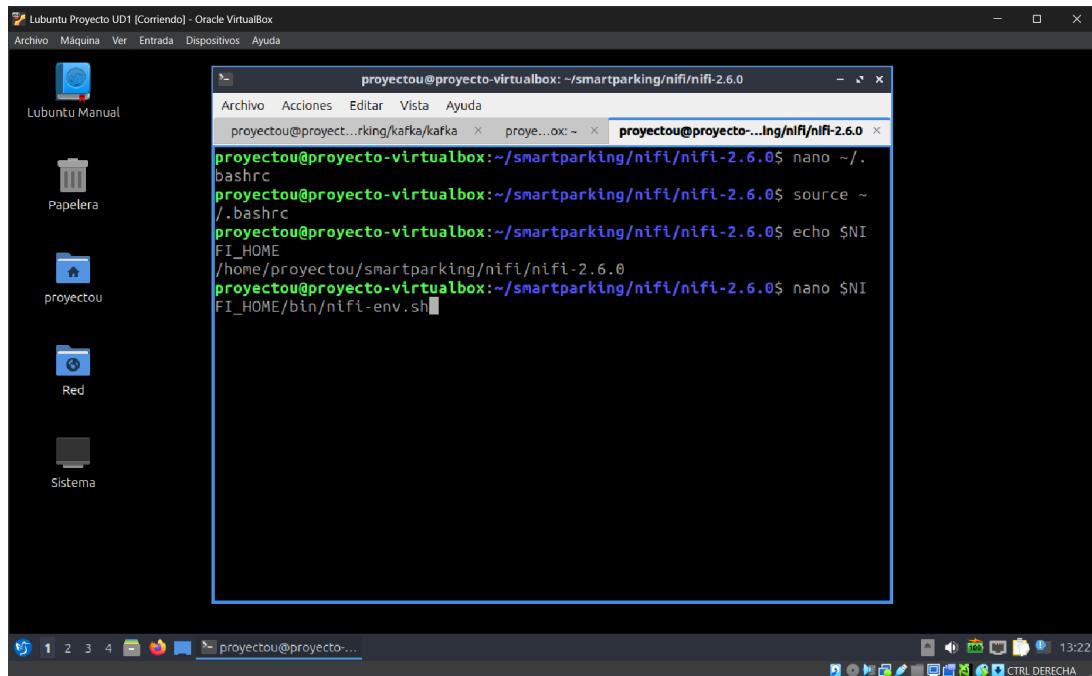


Figura D.12: Accediendo al script de entorno 'nifi-env.sh'.

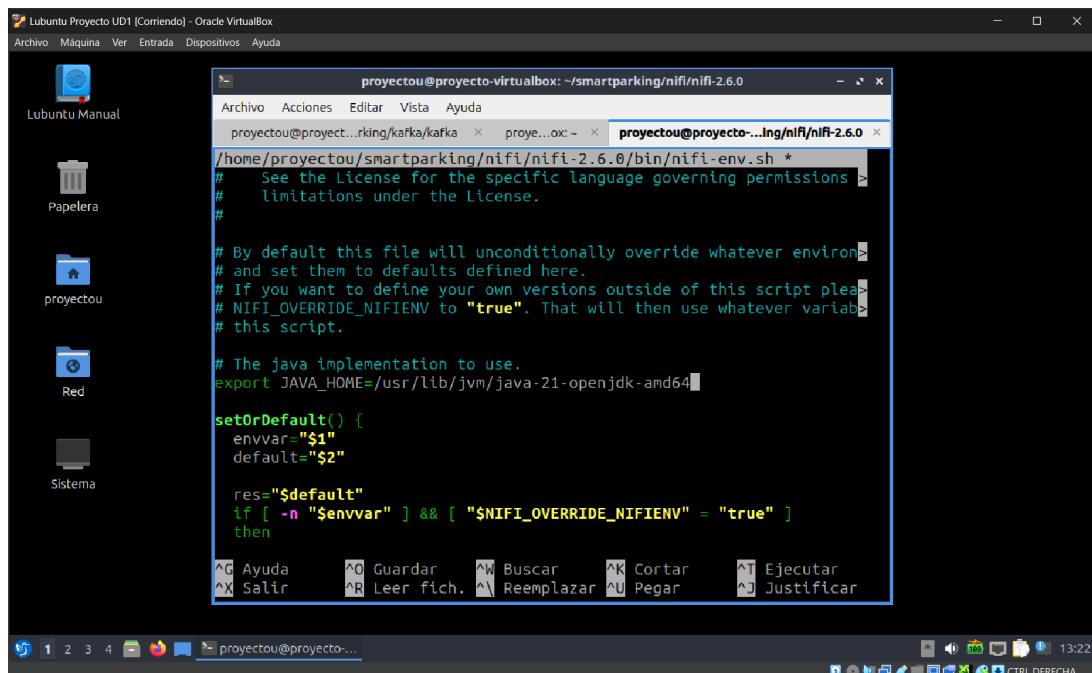


Figura D.13: Estableciendo 'JAVA_HOME' en 'nifi-env.sh'.

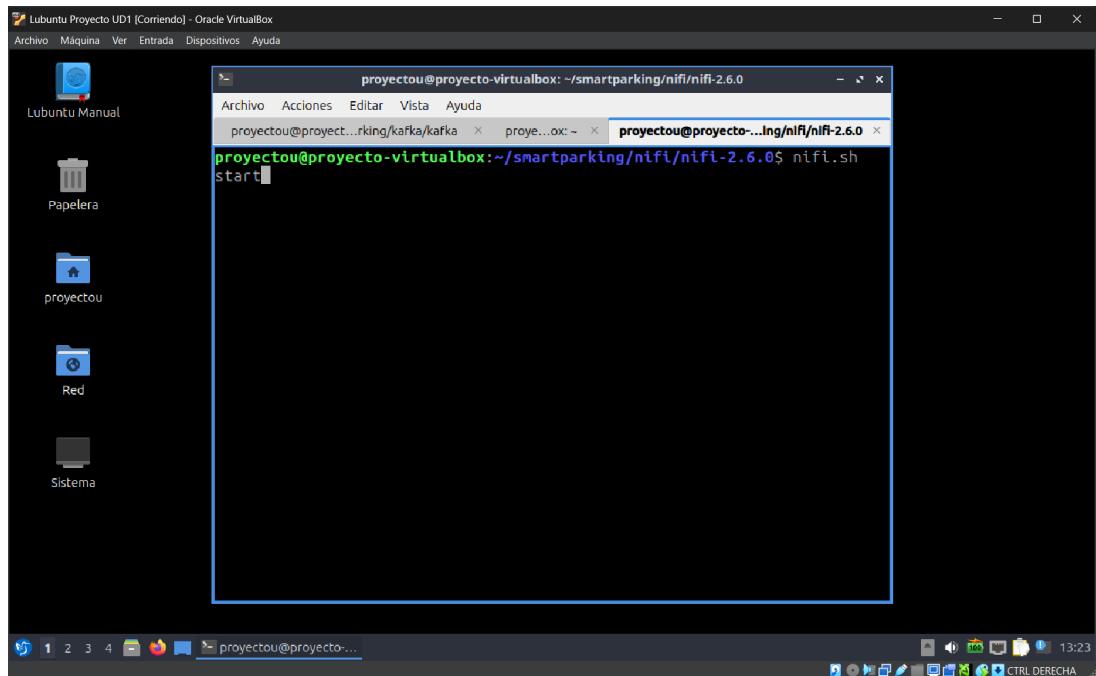


Figura D.14: Iniciando el servicio de NiFi ('nifi.sh start').

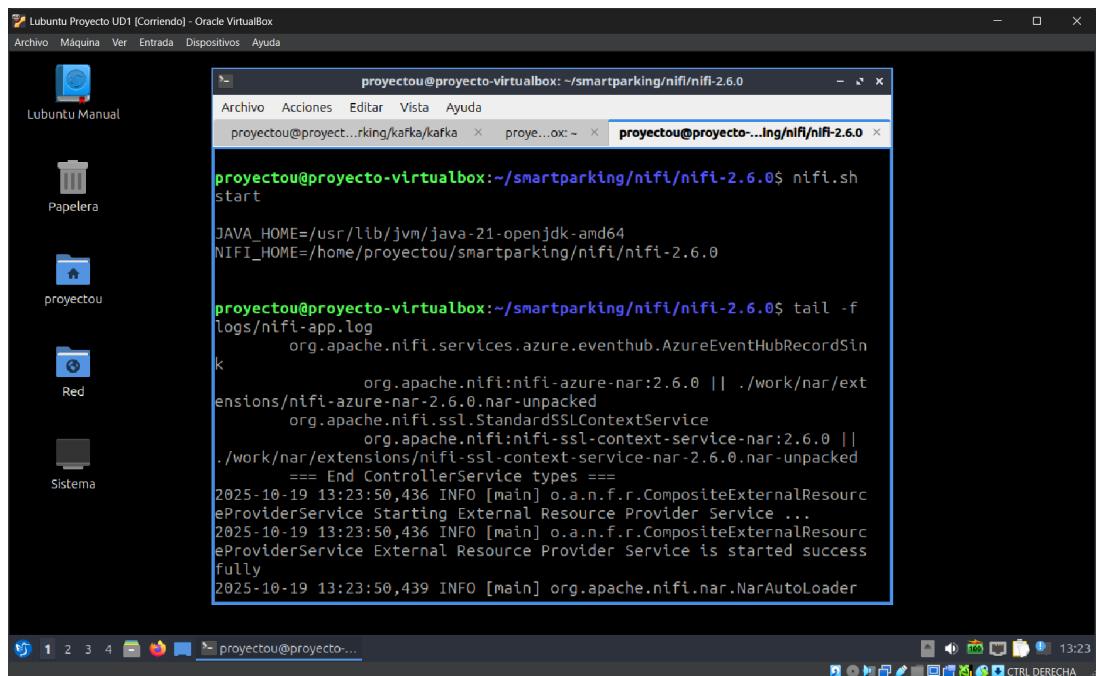


Figura D.15: Comprobando los logs de inicio de NiFi ('tail -f nifi-app.log').

SmartParking Flow — Monitorización Inteligente

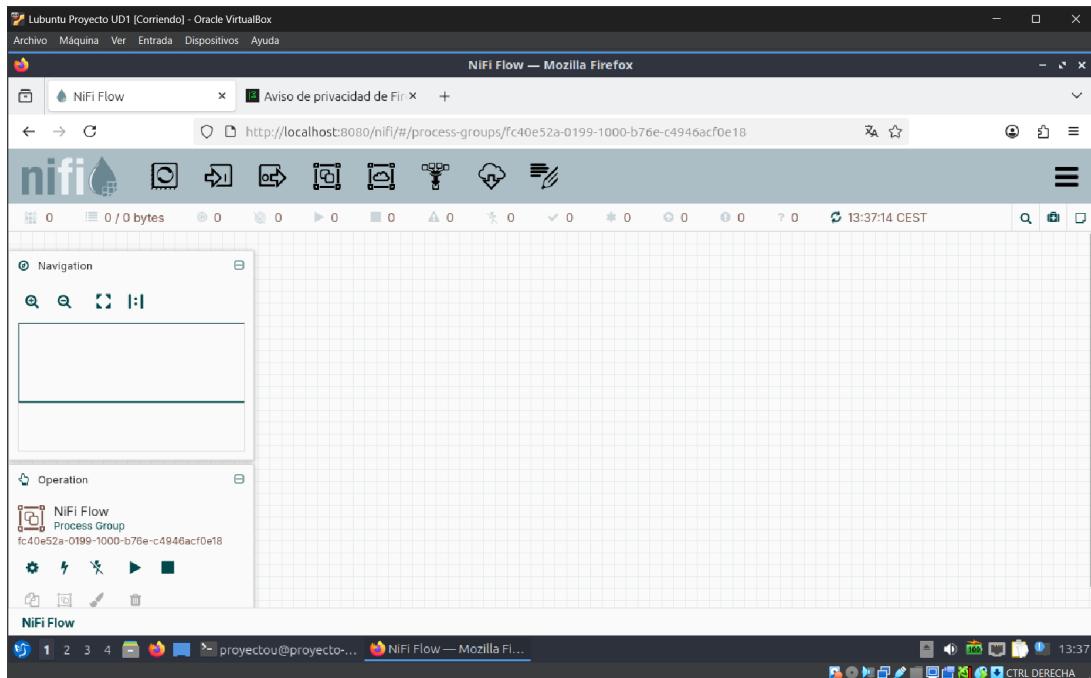


Figura D.16: Accediendo a la interfaz web de NiFi (localhost:8080/nifi).

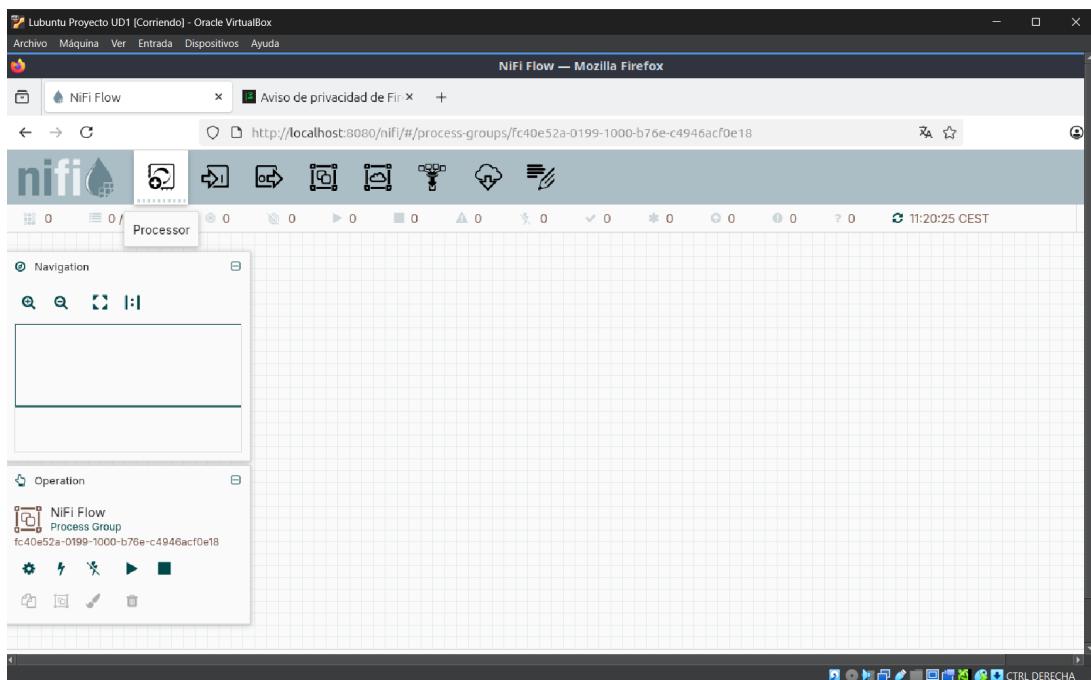


Figura D.17: Arrastrando un nuevo procesador al canvas.

SmartParking Flow — Monitorización Inteligente

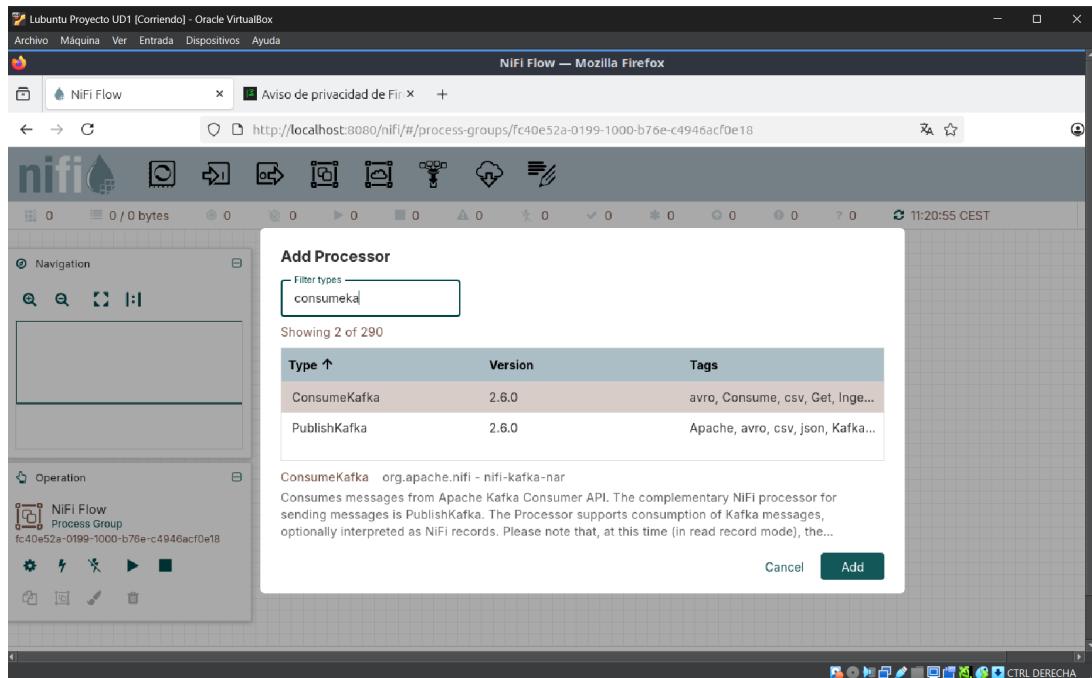


Figura D.18: Buscando el procesador 'ConsumeKafka'.

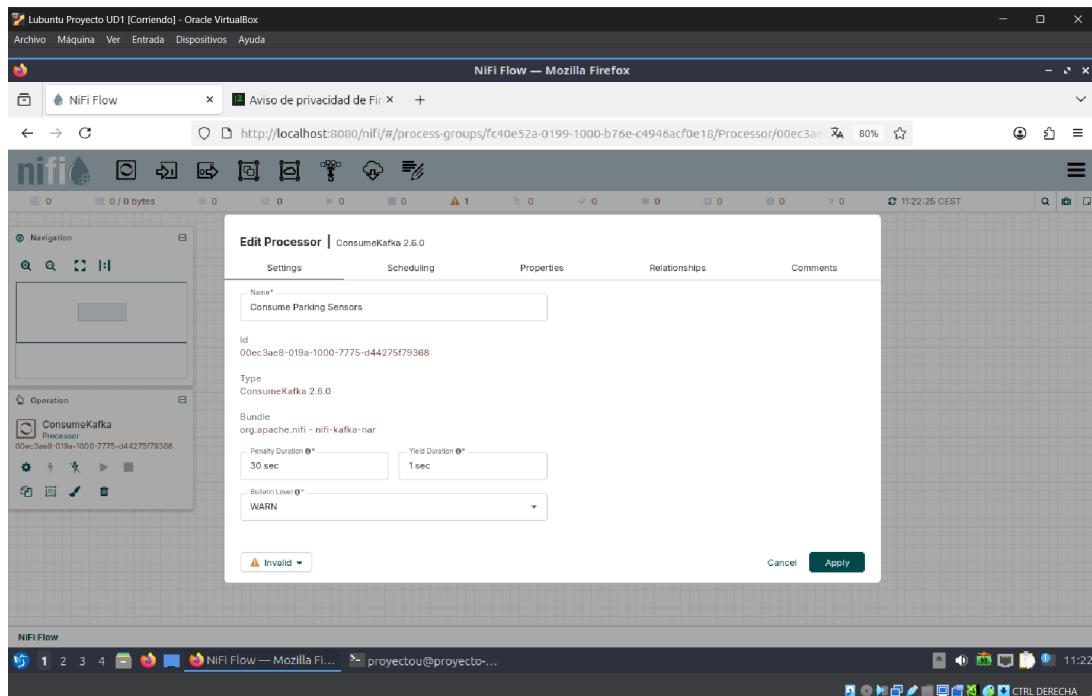


Figura D.19: Configuración (Settings) del procesador 'ConsumeKafka'.

SmartParking Flow — Monitorización Inteligente

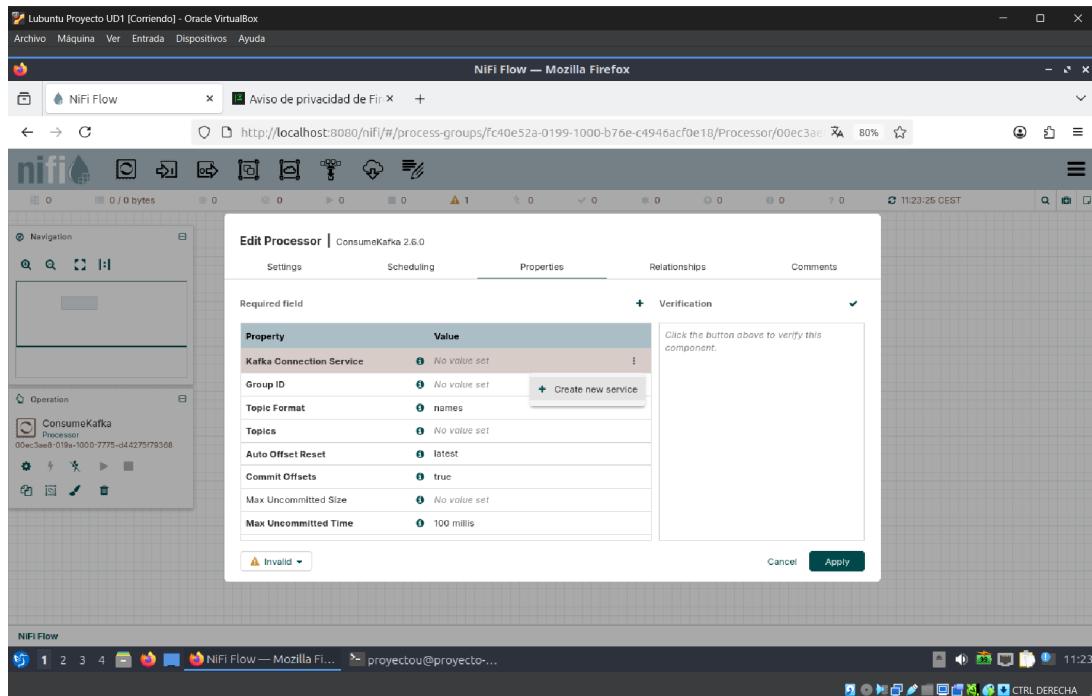


Figura D.20: Configuración (Properties) del procesador 'ConsumeKafka'.

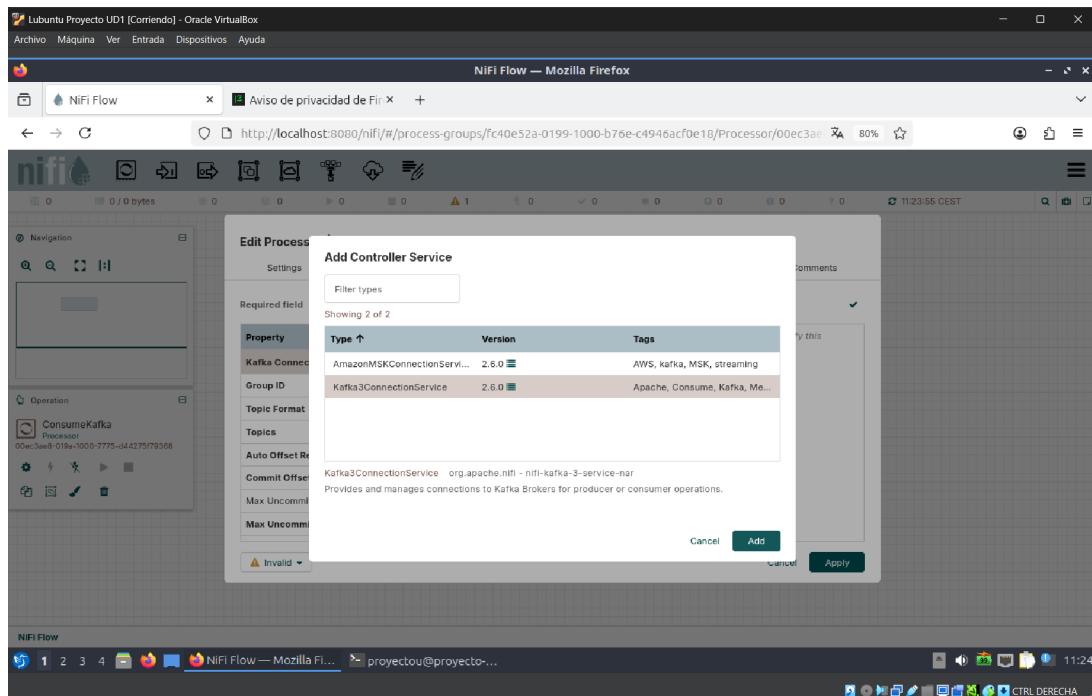


Figura D.21: Creación de un nuevo Controller Service: 'Kafka3ConnectionService'.

SmartParking Flow — Monitorización Inteligente

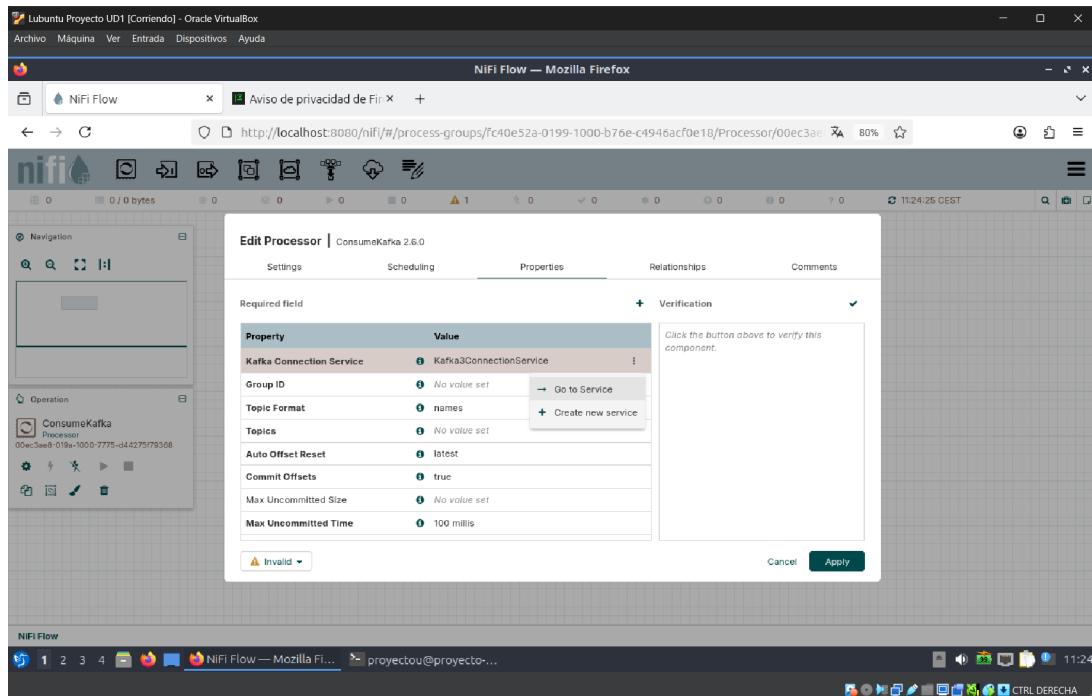


Figura D.22: Asignando el 'Kafka3ConnectionService' al procesador.

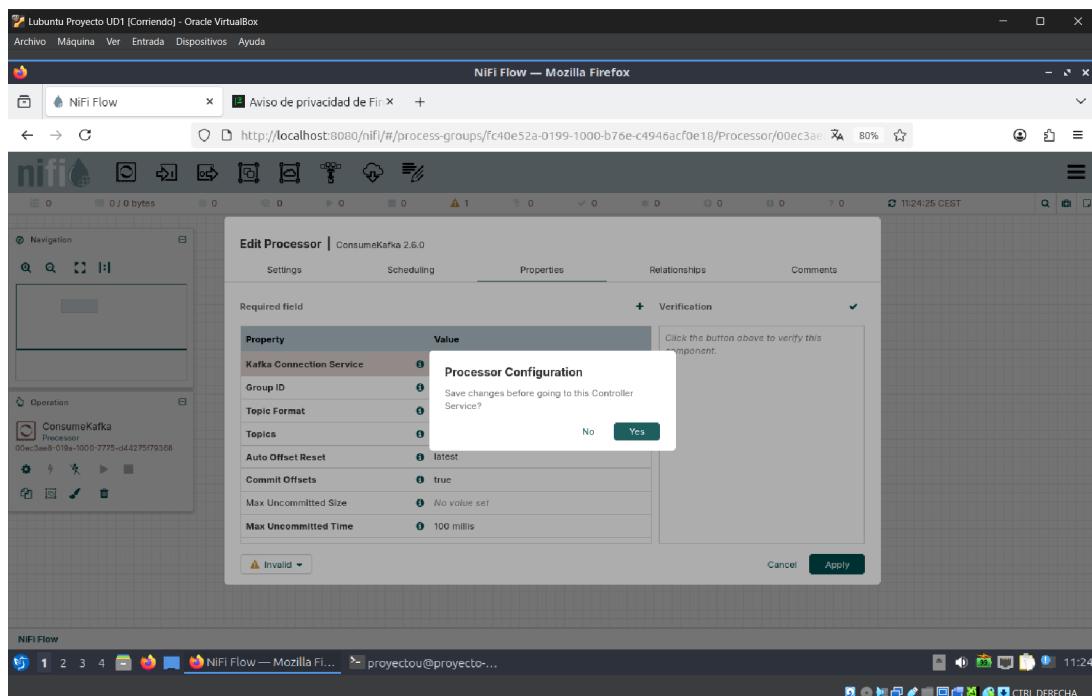


Figura D.23: Guardando cambios en el procesador.

SmartParking Flow — Monitorización Inteligente

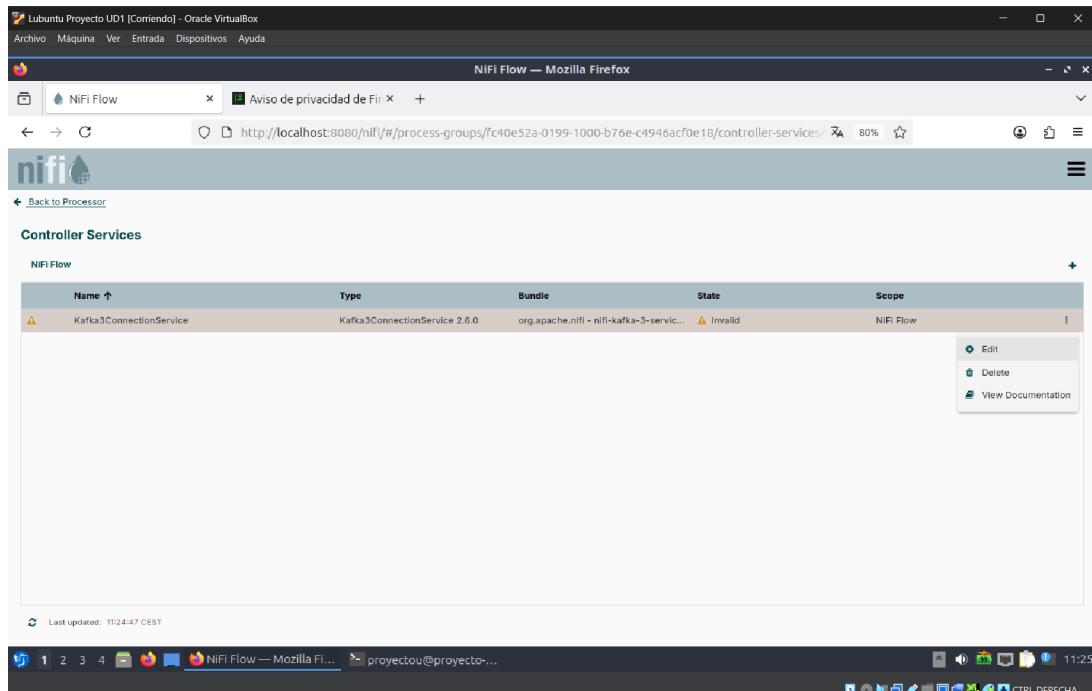


Figura D.24: Accediendo a la configuración del Controller Service ('Kafka3ConnectionService').

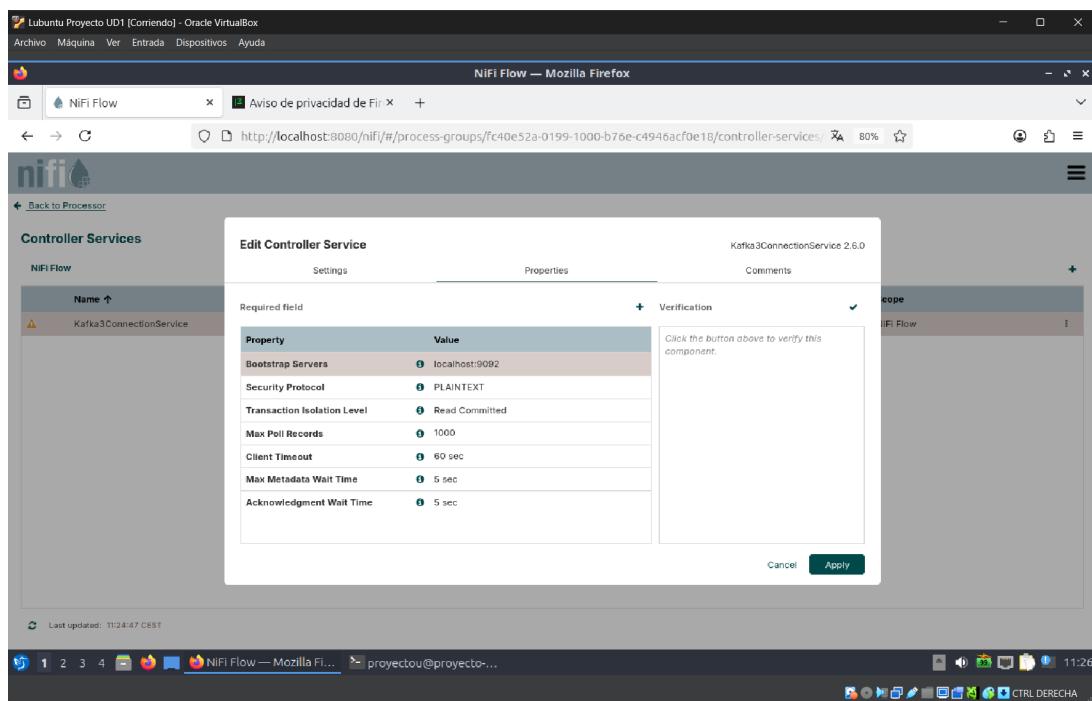


Figura D.25: Configurando el 'Kafka3ConnectionService' (Bootstrap Servers: localhost:9092).

SmartParking Flow — Monitorización Inteligente

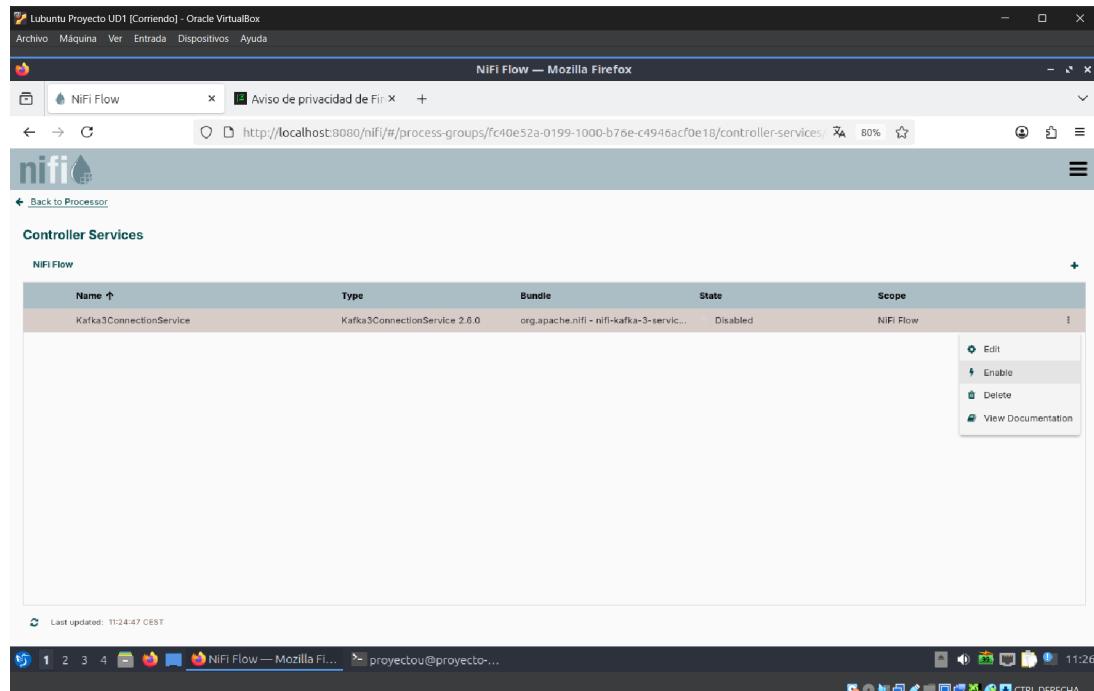


Figura D.26: Controller Service 'Kafka3ConnectionService' en estado "Disabled".

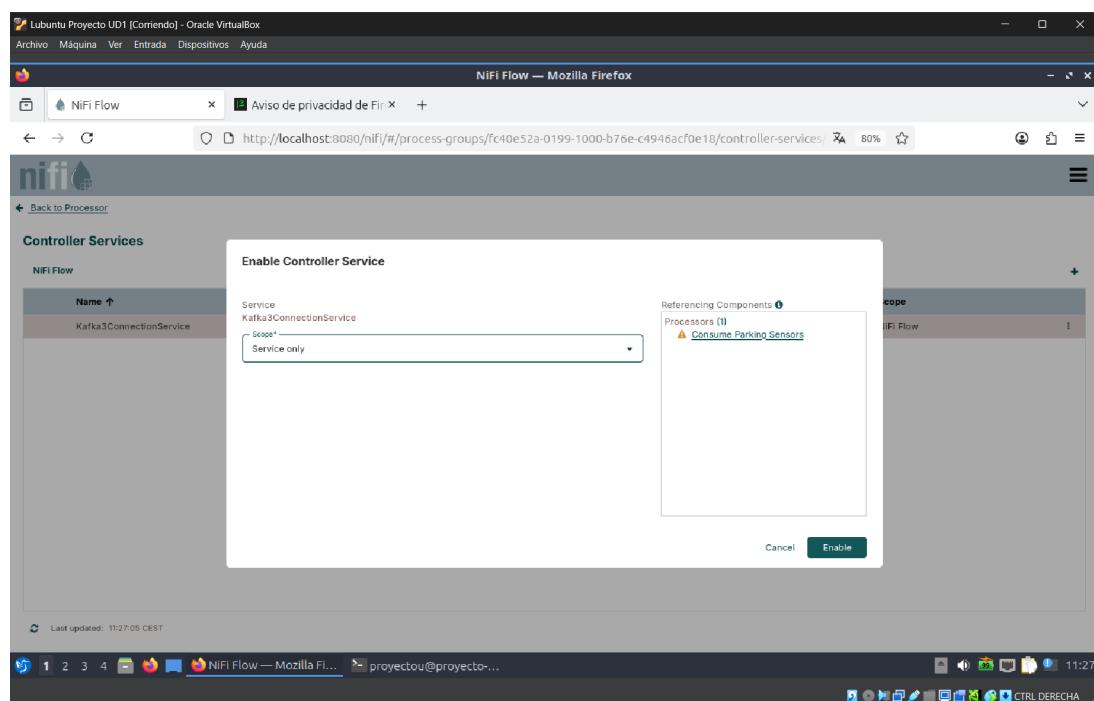


Figura D.27: Habilitando el 'Kafka3ConnectionService'.

SmartParking Flow — Monitorización Inteligente

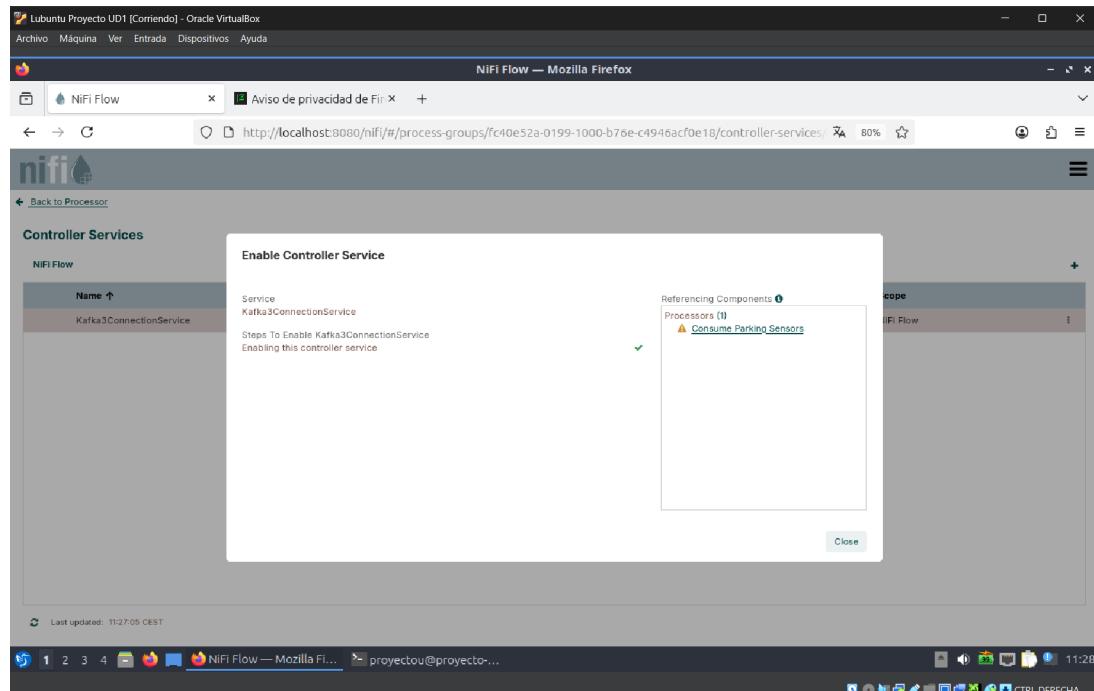


Figura D.28: Controller Service 'Kafka3ConnectionService' siendo habilitado.

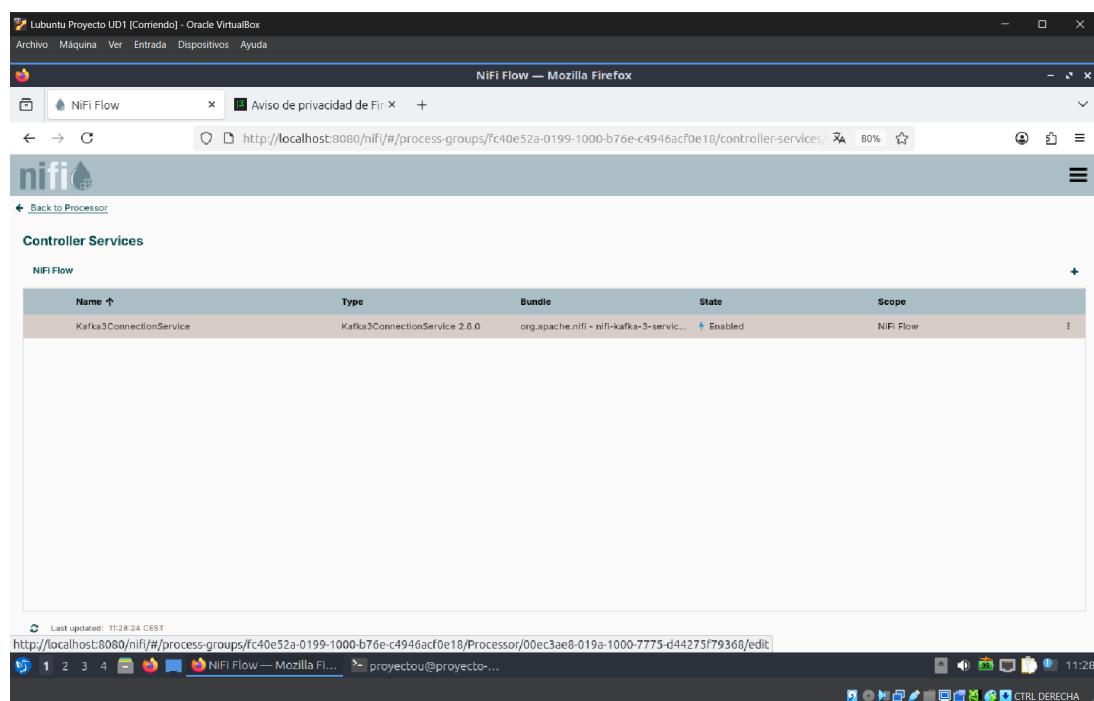


Figura D.29: Controller Service 'Kafka3ConnectionService' en estado ".Enabled".

SmartParking Flow — Monitorización Inteligente

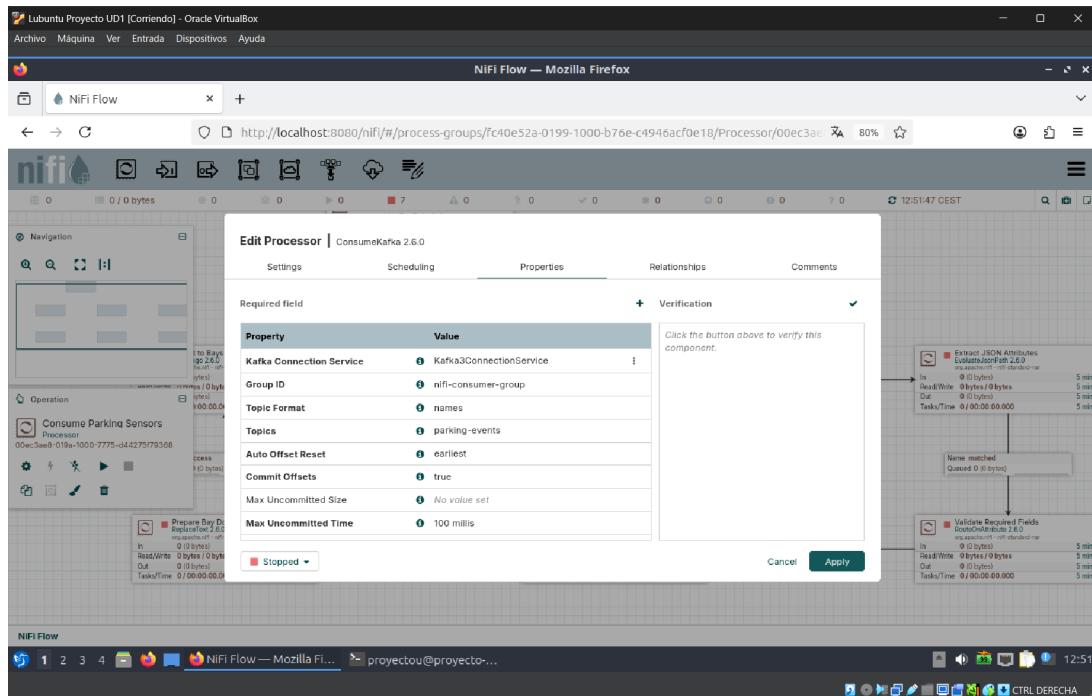


Figura D.30: Configuración final del procesador 'ConsumeKafka' (Tópico: parking-events).

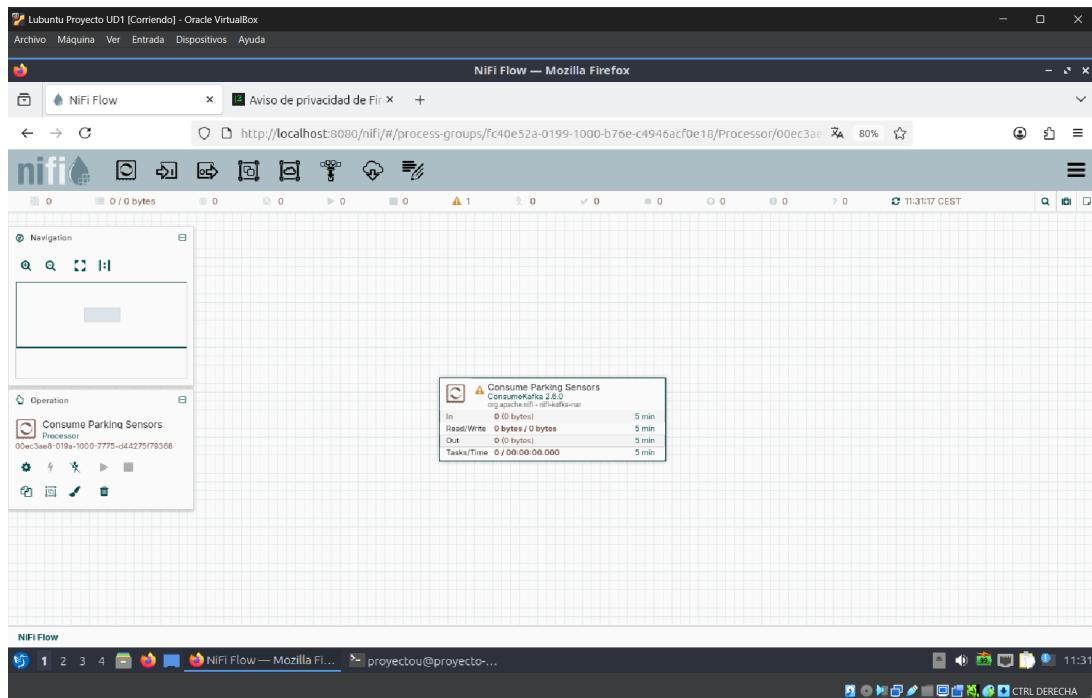


Figura D.31: Procesador 'Consume Parking Sensors' en el canvas.

SmartParking Flow — Monitorización Inteligente

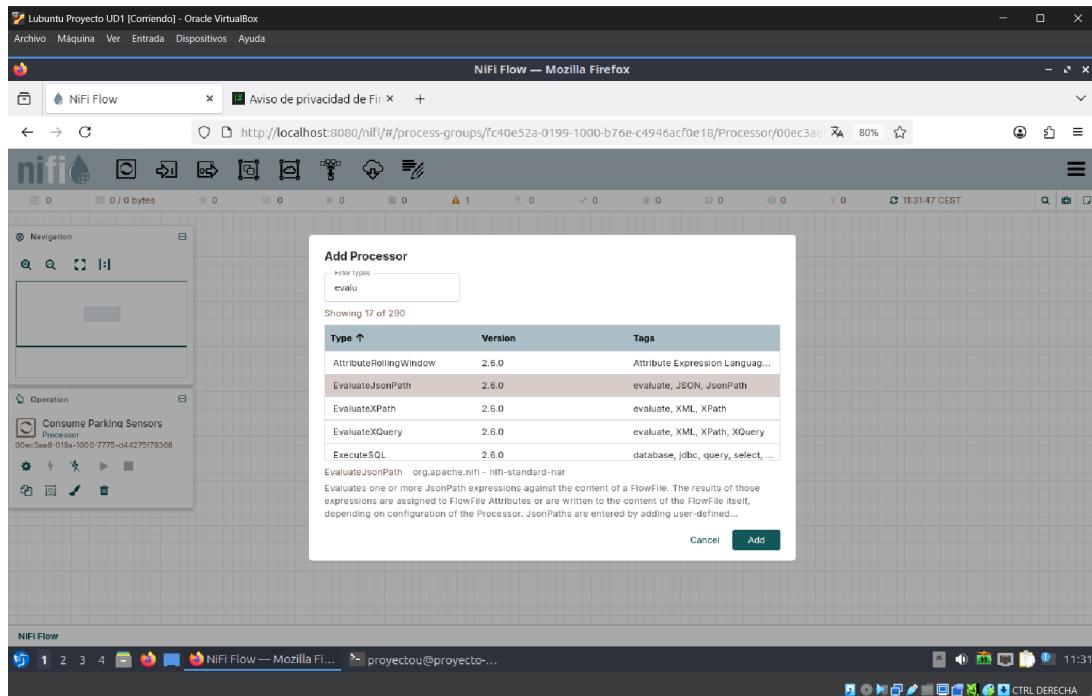


Figura D.32: Añadiendo el procesador 'EvaluateJsonPath'.

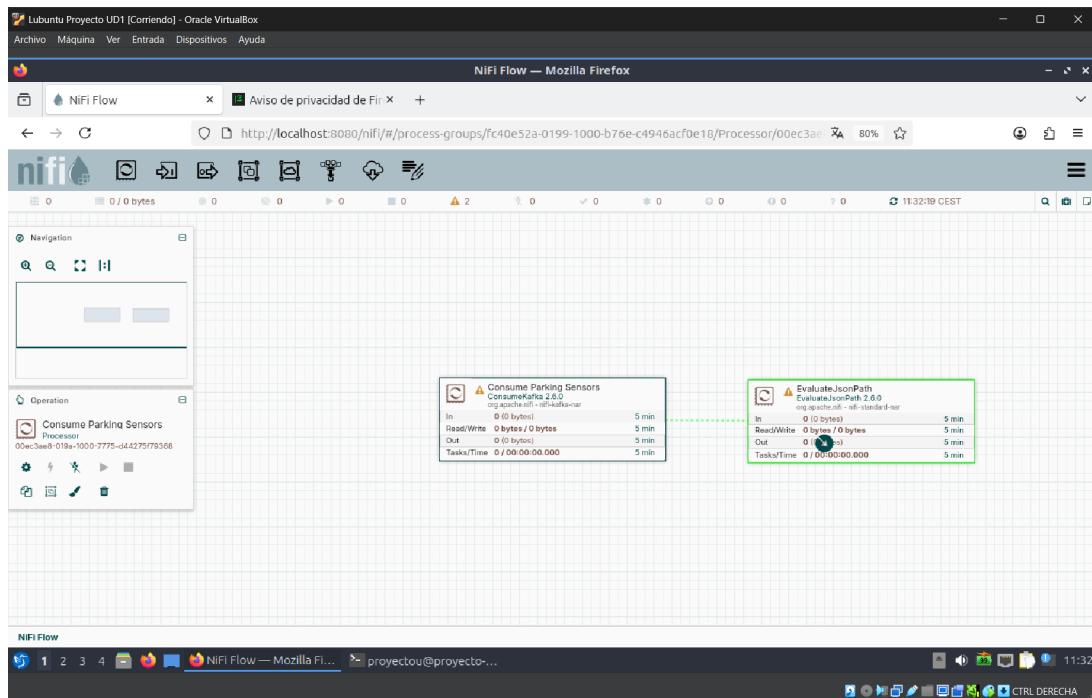


Figura D.33: Procesadores 'Consume Parking Sensors' y 'EvaluateJsonPath' en el canvas.

SmartParking Flow — Monitorización Inteligente

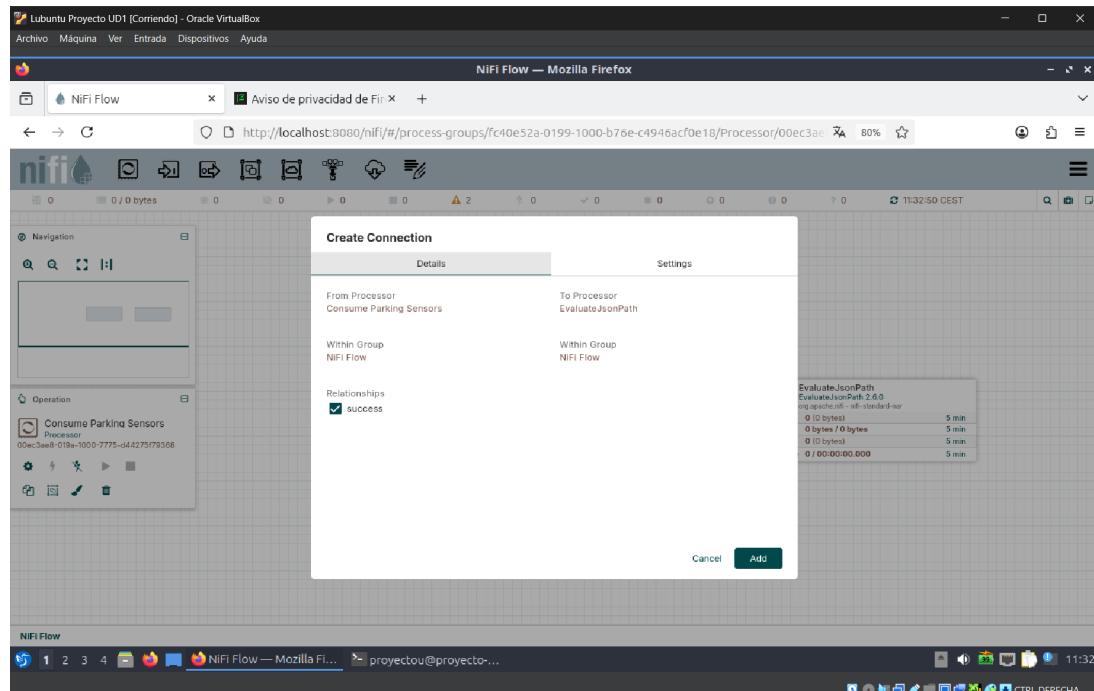


Figura D.34: Creando conexión "success" entre 'ConsumeKafka' y 'EvaluateJsonPath'.

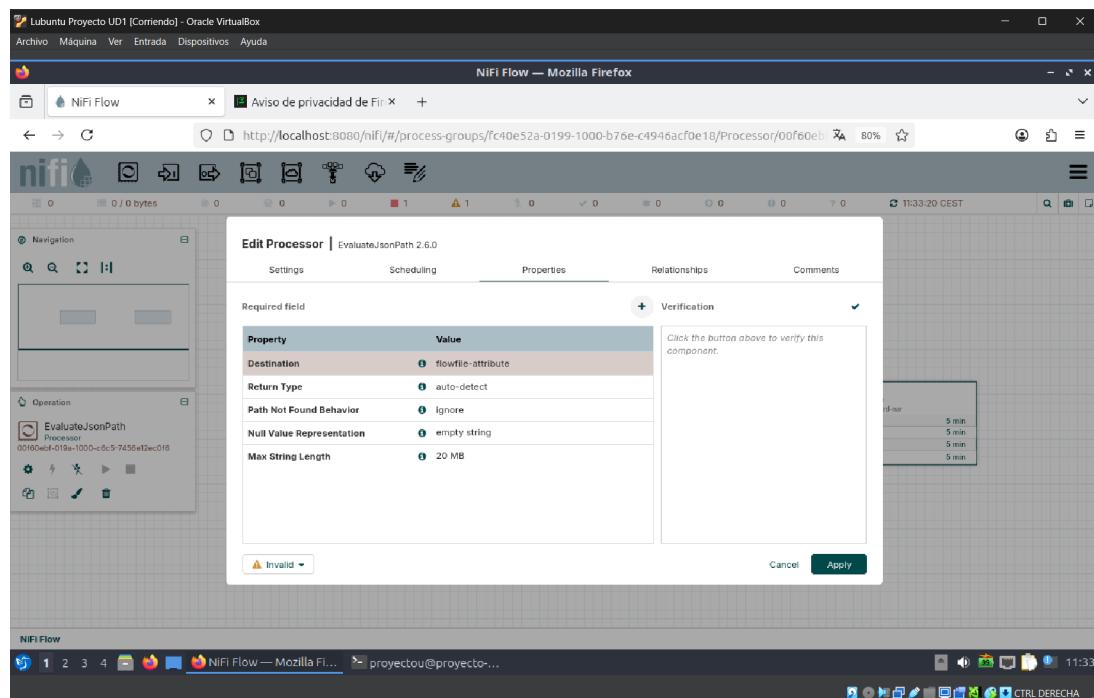


Figura D.35: Configuración (Properties) de 'EvaluateJsonPath': 'Destination: flowfile-attribute'.

SmartParking Flow — Monitorización Inteligente

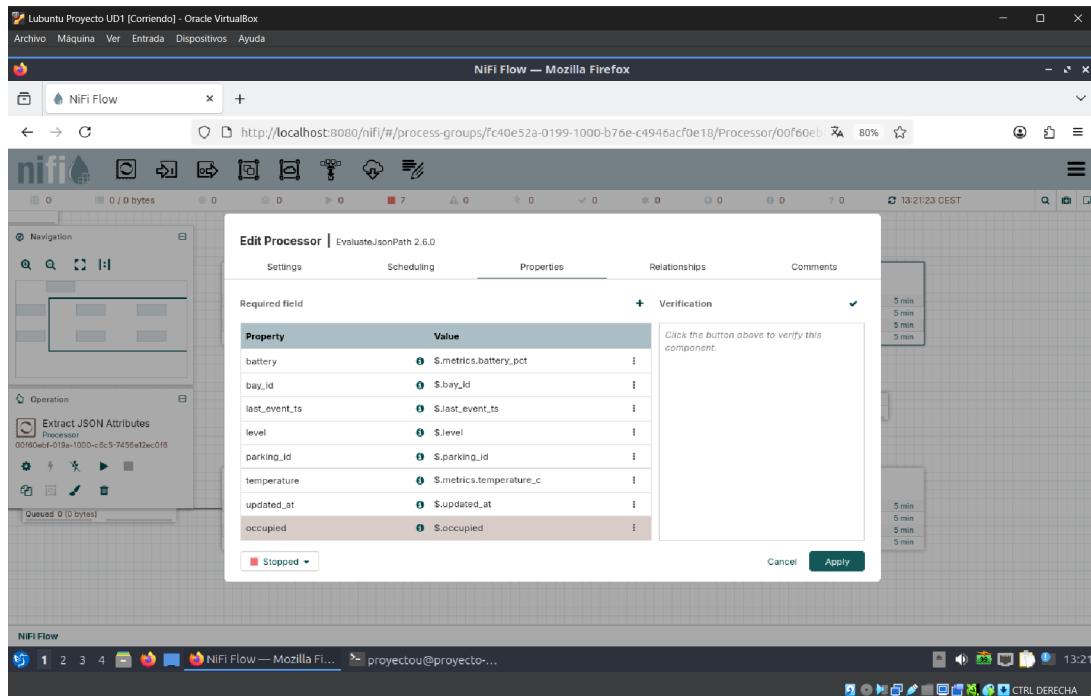


Figura D.36: Configuración (Properties) de 'EvaluateJsonPath': Extracción de atributos (battery, bay_id, ...).

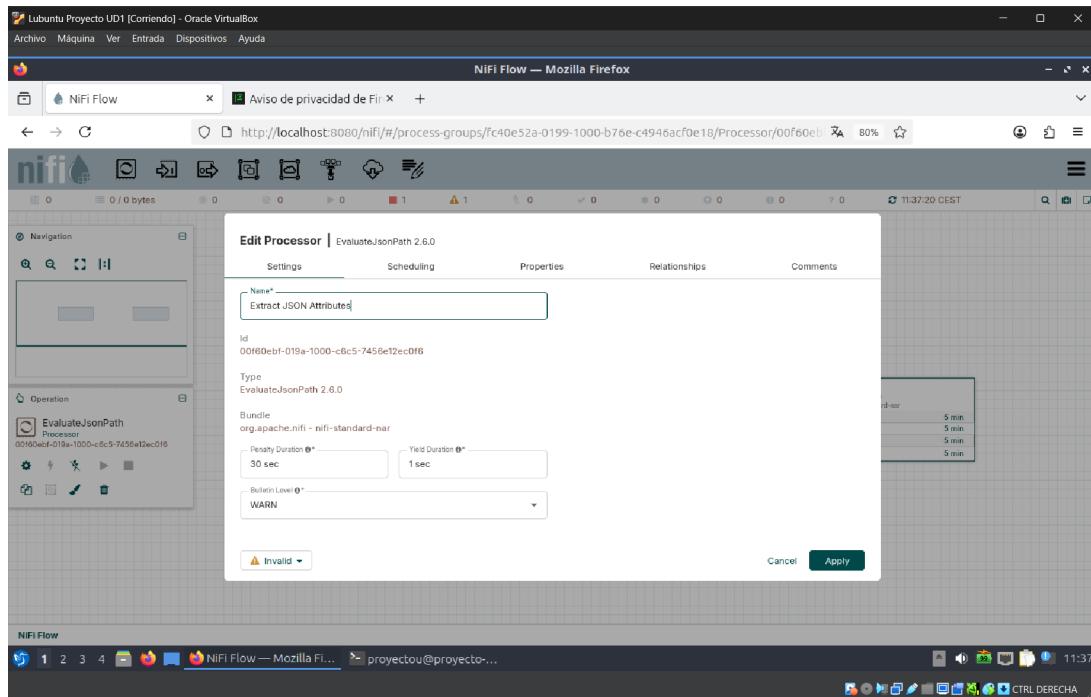


Figura D.37: Configuración (Settings) de 'EvaluateJsonPath': 'Name: Extract JSON Attributes'.

SmartParking Flow — Monitorización Inteligente

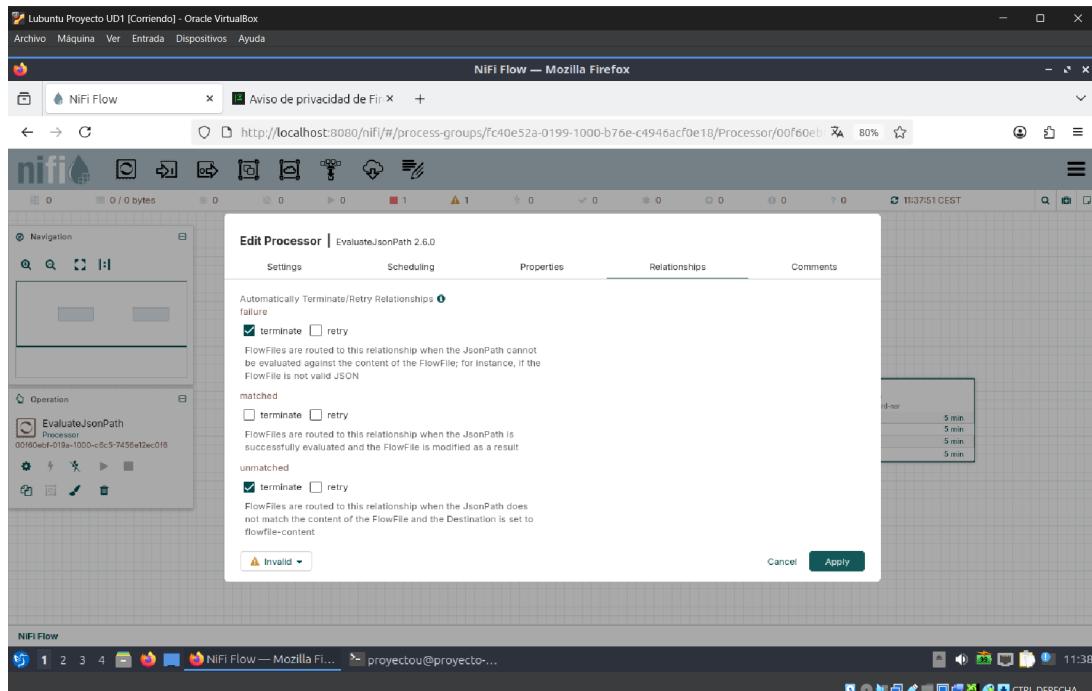


Figura D.38: Configuración (Relationships) de 'EvaluateJsonPath': Terminar 'unmatched'.

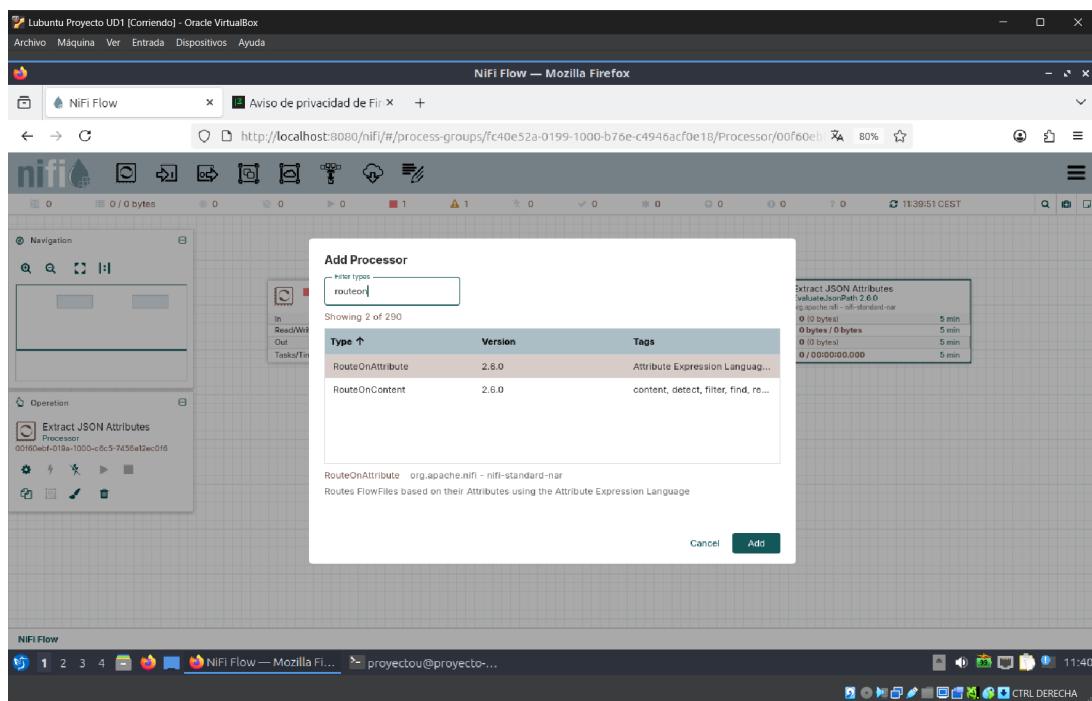


Figura D.39: Añadiendo el procesador 'RouteOnAttribute'.

SmartParking Flow — Monitorización Inteligente

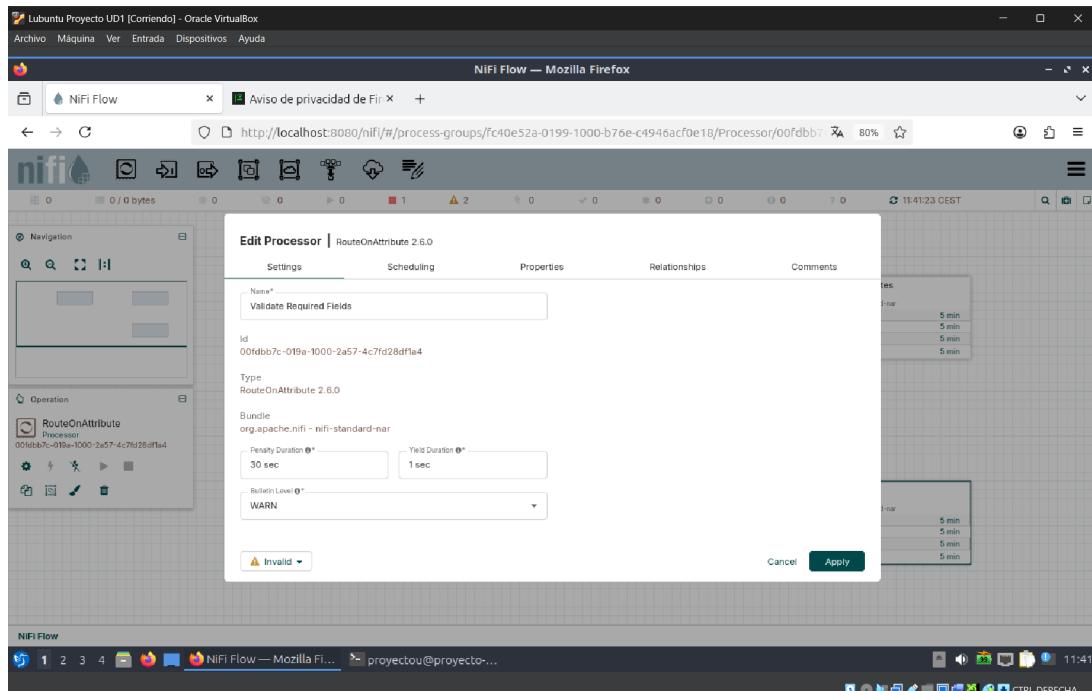


Figura D.40: Configuración (Settings) de 'RouteOnAttribute': 'Name: Validate Required Fields'.

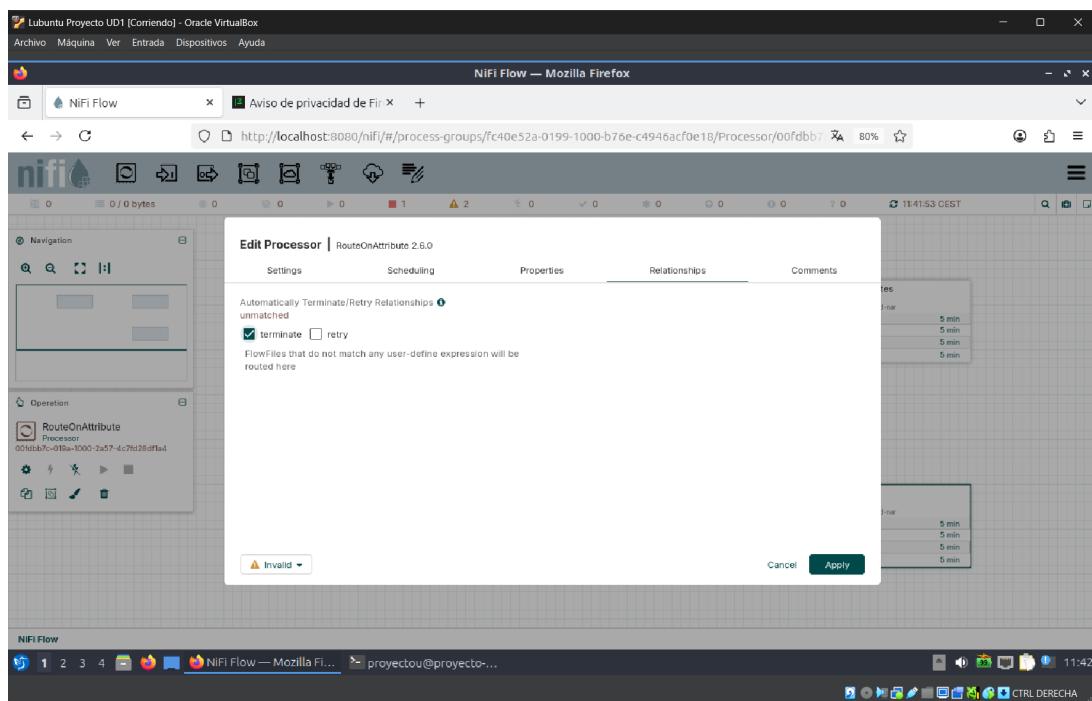


Figura D.41: Configuración (Relationships) de 'RouteOnAttribute': Terminar ünmatched".

SmartParking Flow — Monitorización Inteligente

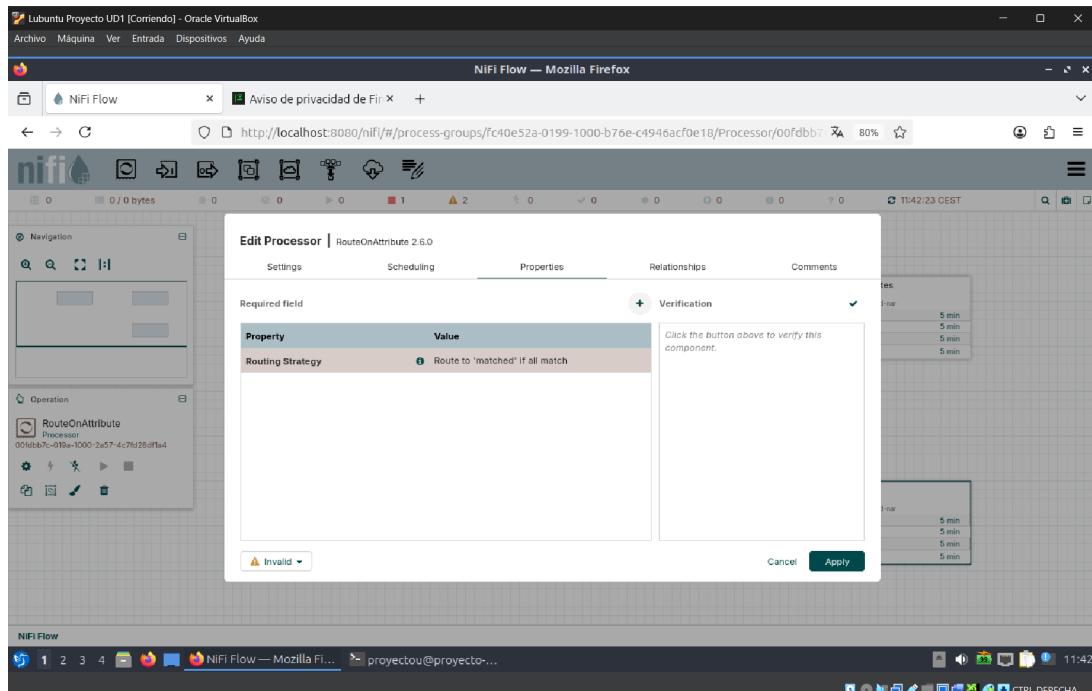


Figura D.42: Configuración (Properties) de 'RouteOnAttribute': 'Routing Strategy'.

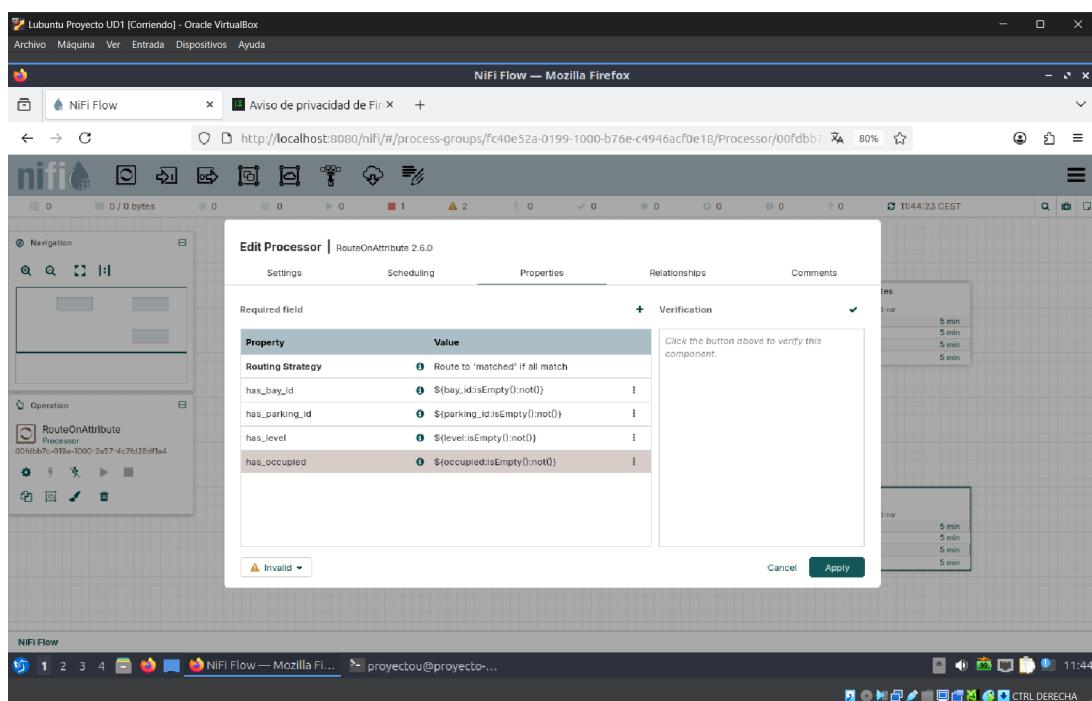


Figura D.43: Configuración (Properties) de 'RouteOnAttribute': Añadiendo propiedades de validación.

SmartParking Flow — Monitorización Inteligente

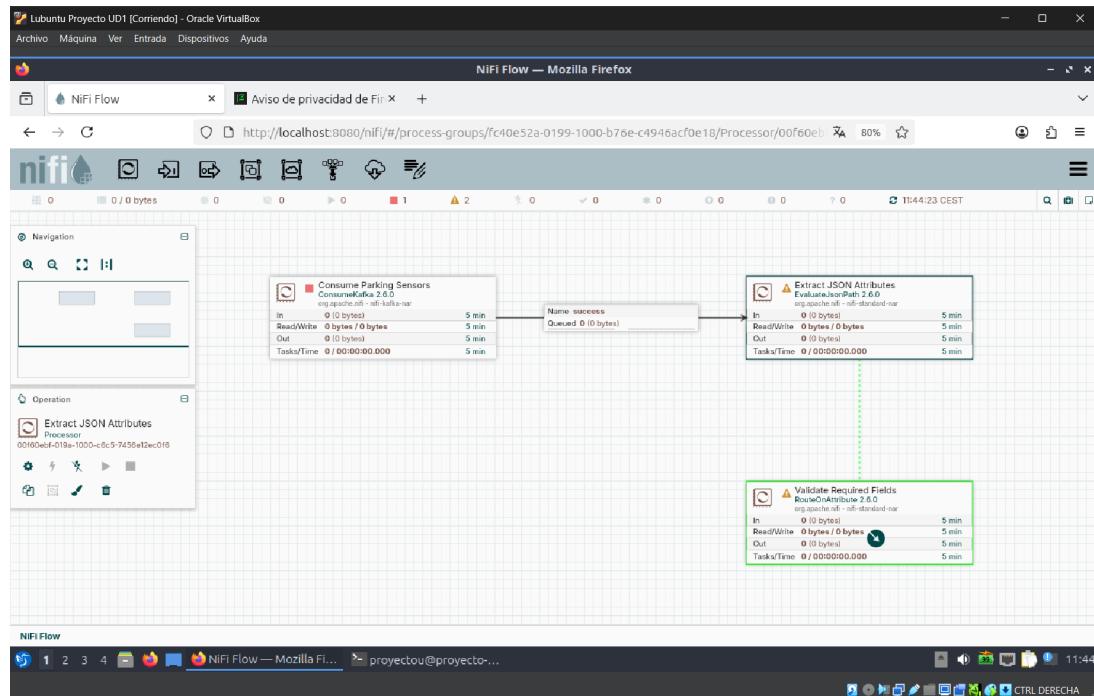


Figura D.44: Flujo con los tres procesadores iniciales.

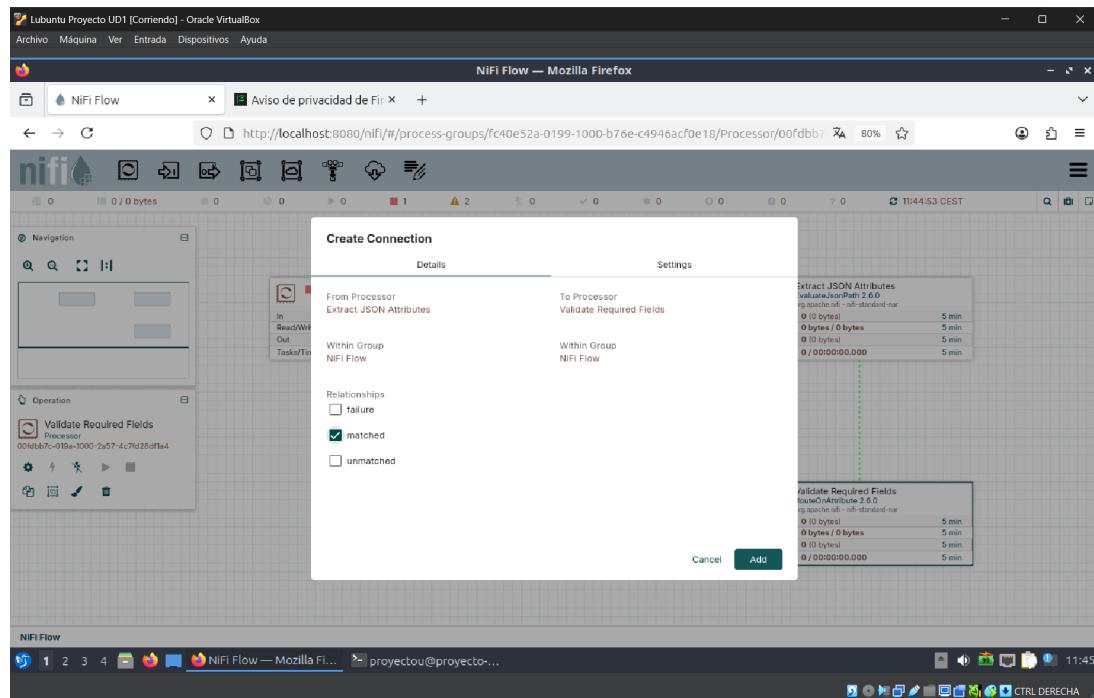


Figura D.45: Creando conexión "matched" entre 'EvaluateJsonPath' y 'RouteOnAttribute'.

SmartParking Flow — Monitorización Inteligente

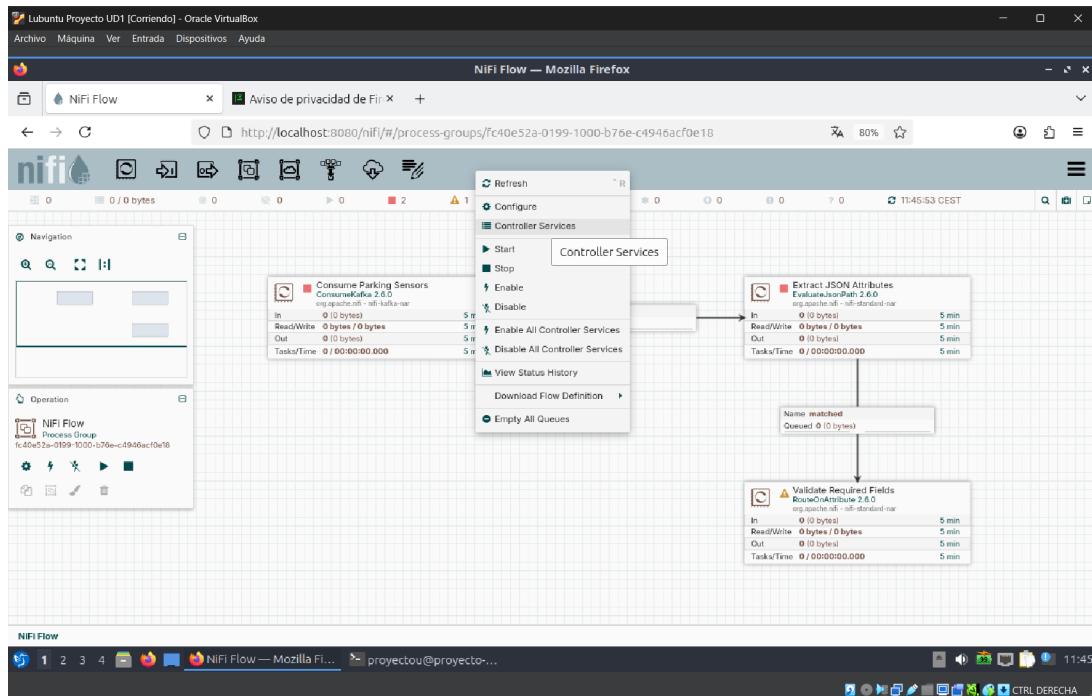


Figura D.46: Accediendo a la configuración de Controller Services del Process Group.

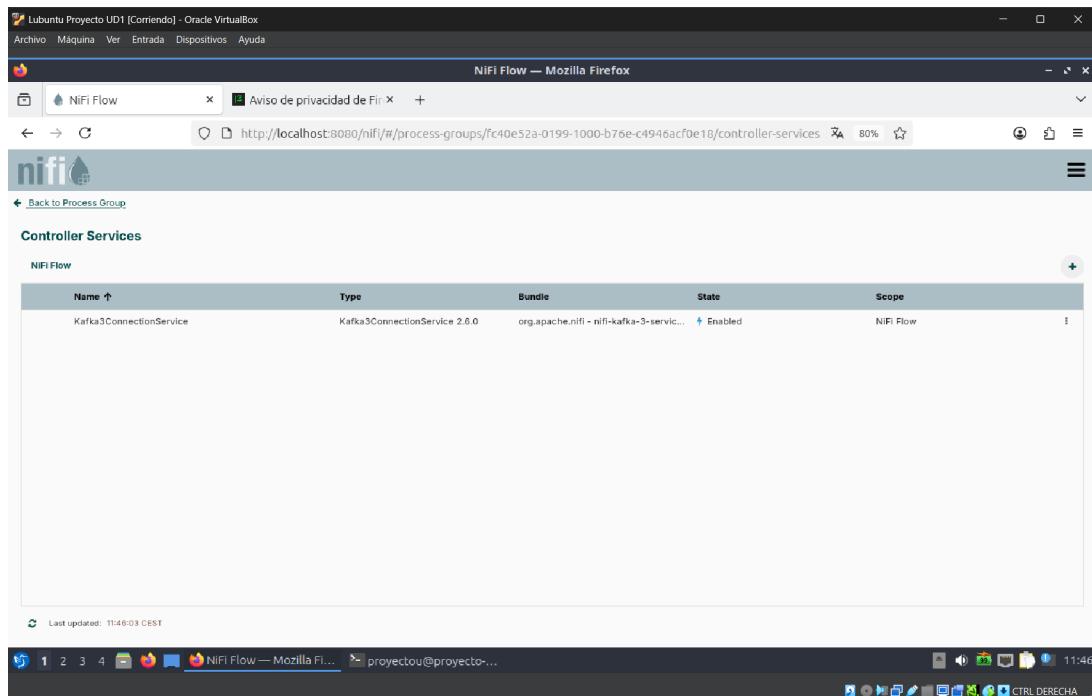


Figura D.47: Vista de 'Kafka3ConnectionService' habilitado.

SmartParking Flow — Monitorización Inteligente

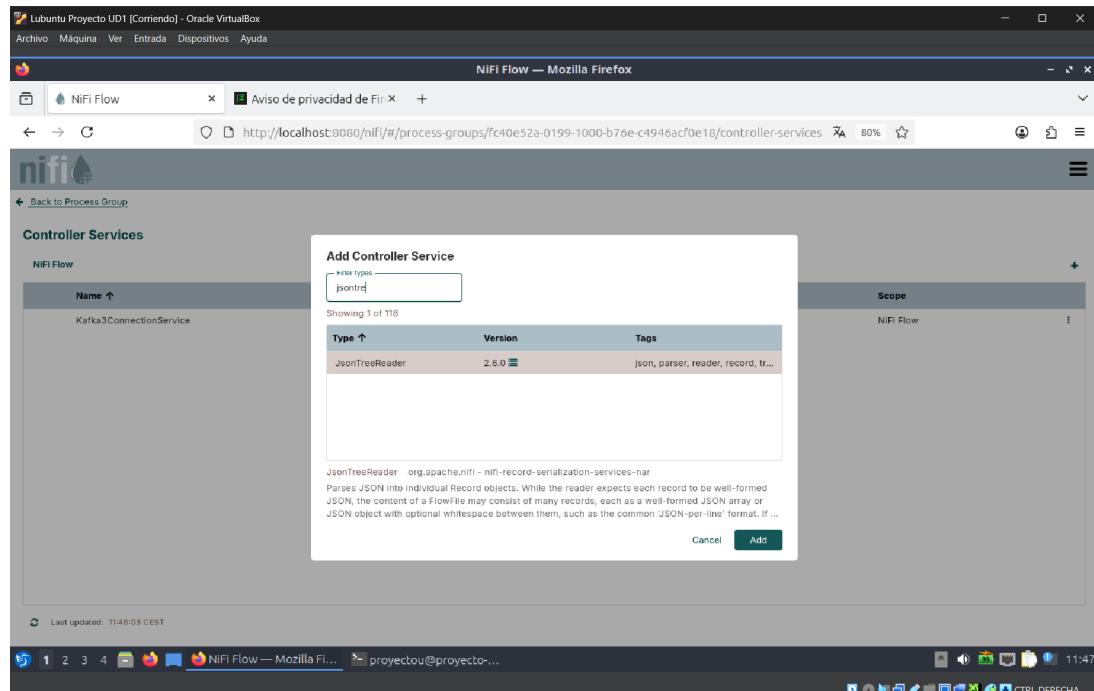


Figura D.48: Añadiendo un nuevo Controller Service: 'JsonTreeReader'.

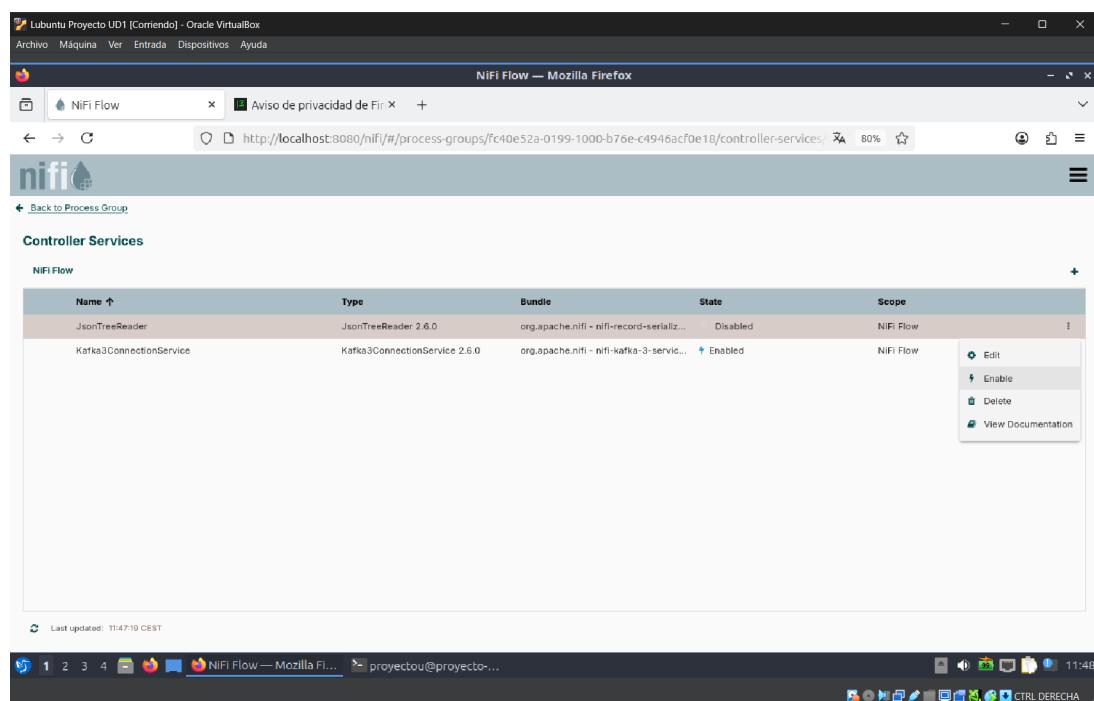


Figura D.49: Controller Service 'JsonTreeReader' en estado "Disabled".

SmartParking Flow — Monitorización Inteligente

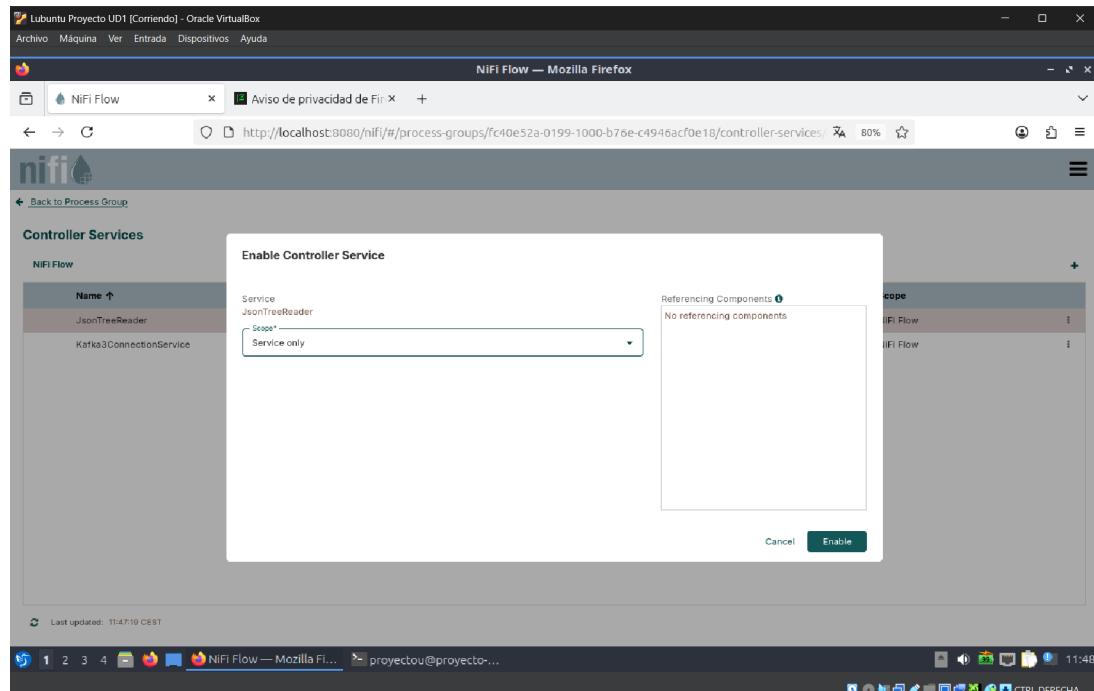


Figura D.50: Habilitando el 'JsonTreeReader'.

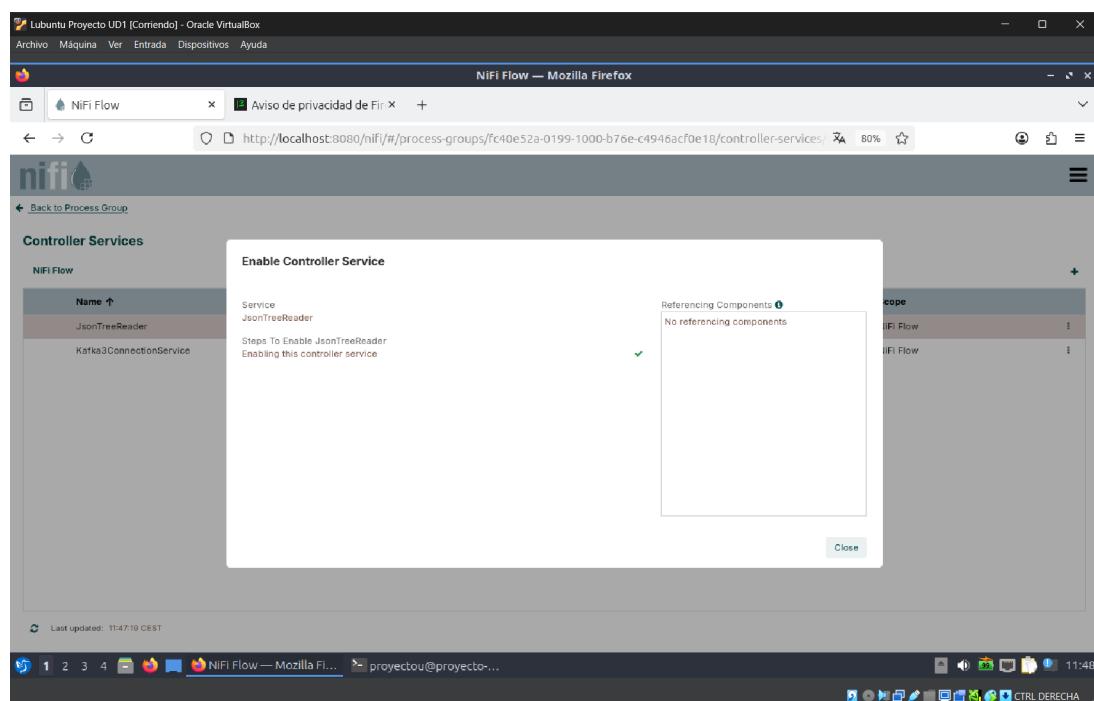


Figura D.51: Confirmación de habilitación del 'JsonTreeReader'.

SmartParking Flow — Monitorización Inteligente

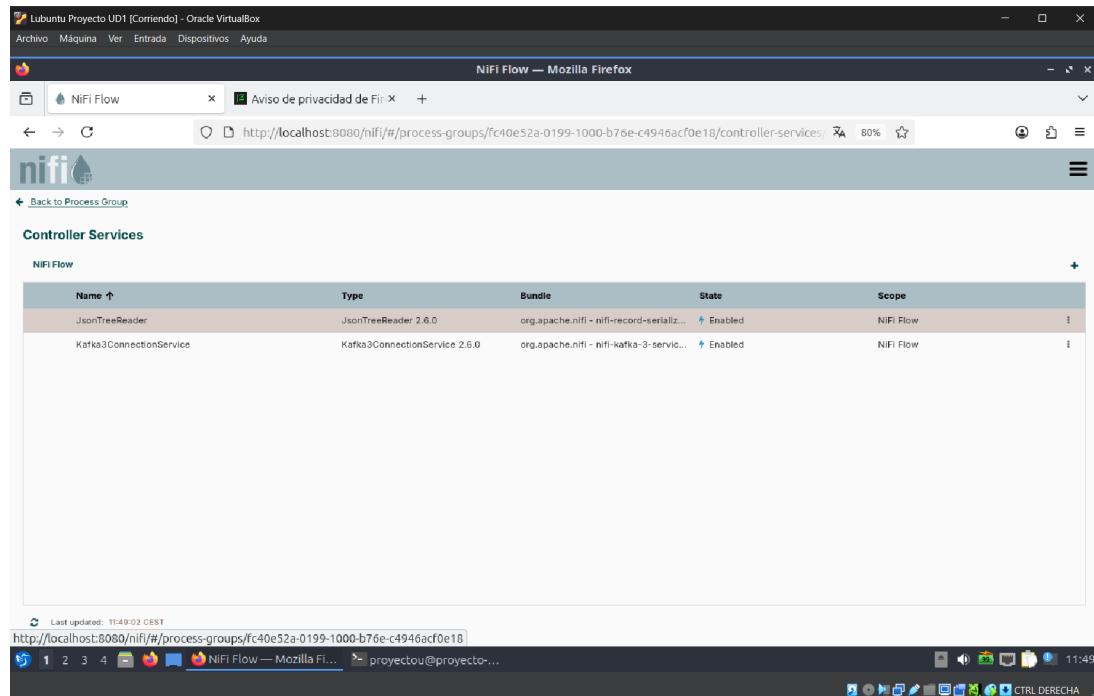


Figura D.52: Ambos Controller Services ('JsonTreeReader' y 'Kafka3ConnectionService') habilitados.

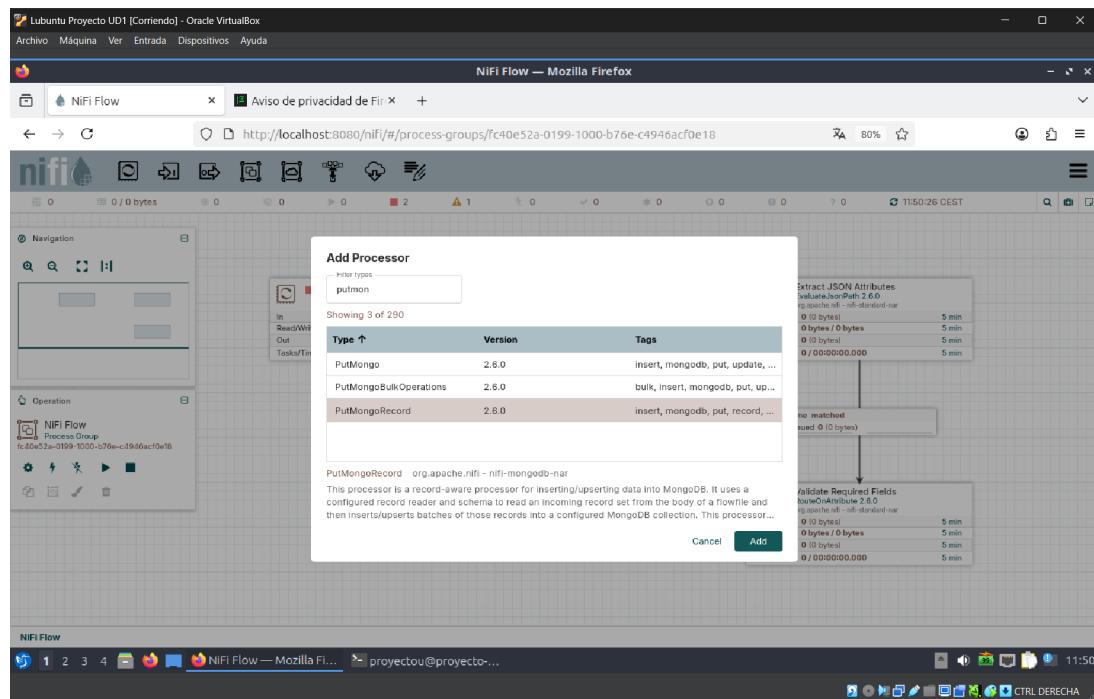


Figura D.53: Añadiendo el procesador 'PutMongoRecord'.

SmartParking Flow — Monitorización Inteligente

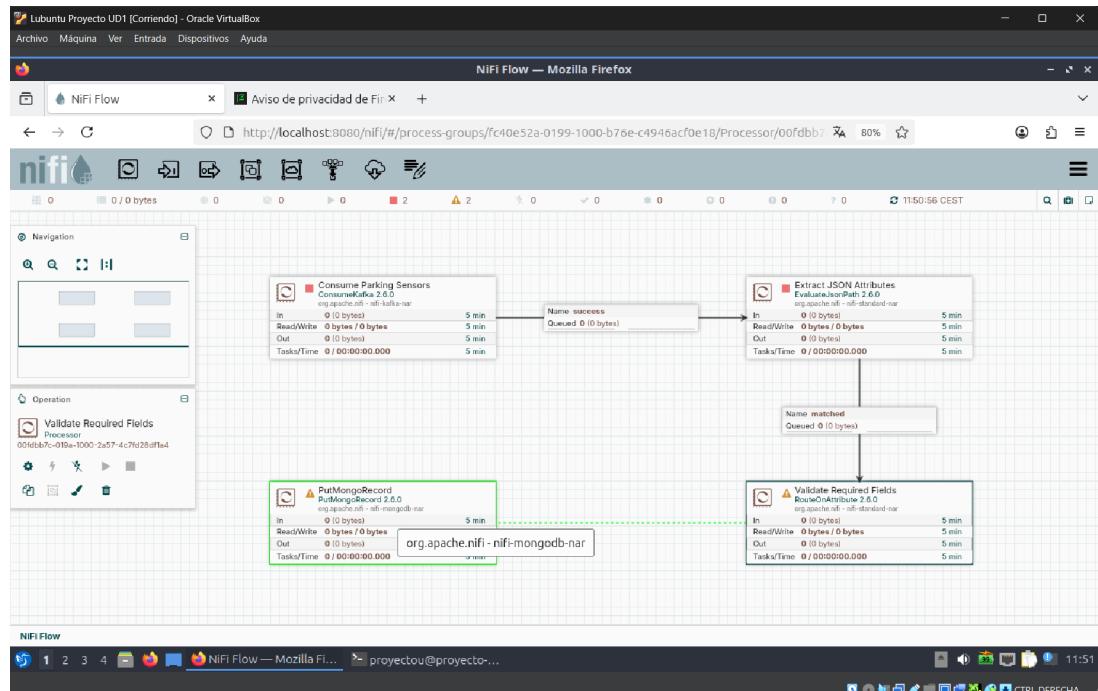


Figura D.54: Flujo con el procesador 'PutMongoRecord' añadido.

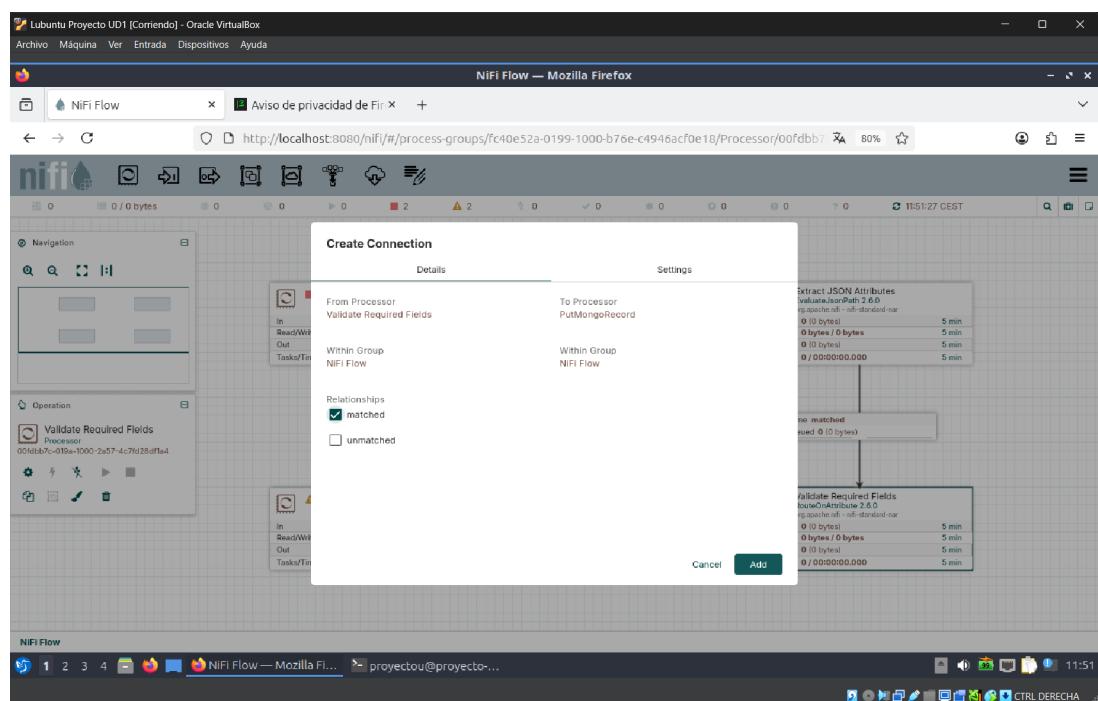


Figura D.55: Creando conexión "matched" entre 'RouteOnAttribute' y 'PutMongoRecord'.

SmartParking Flow — Monitorización Inteligente

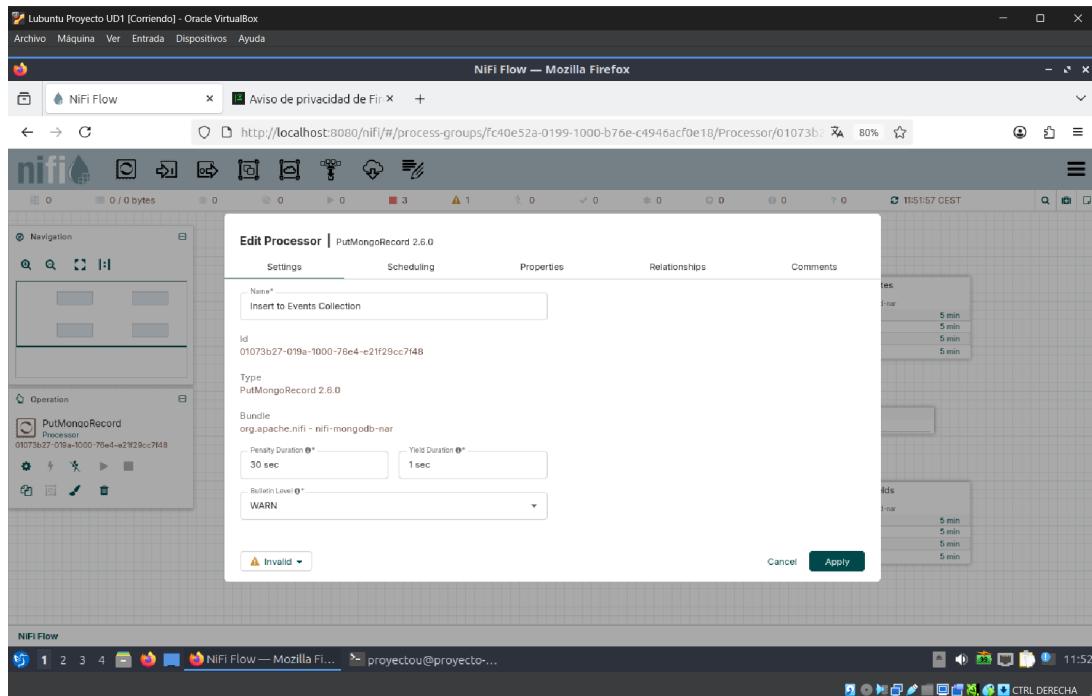


Figura D.56: Configuración (Settings) de 'PutMongoRecord': 'Name: Insert to Events Collection'.

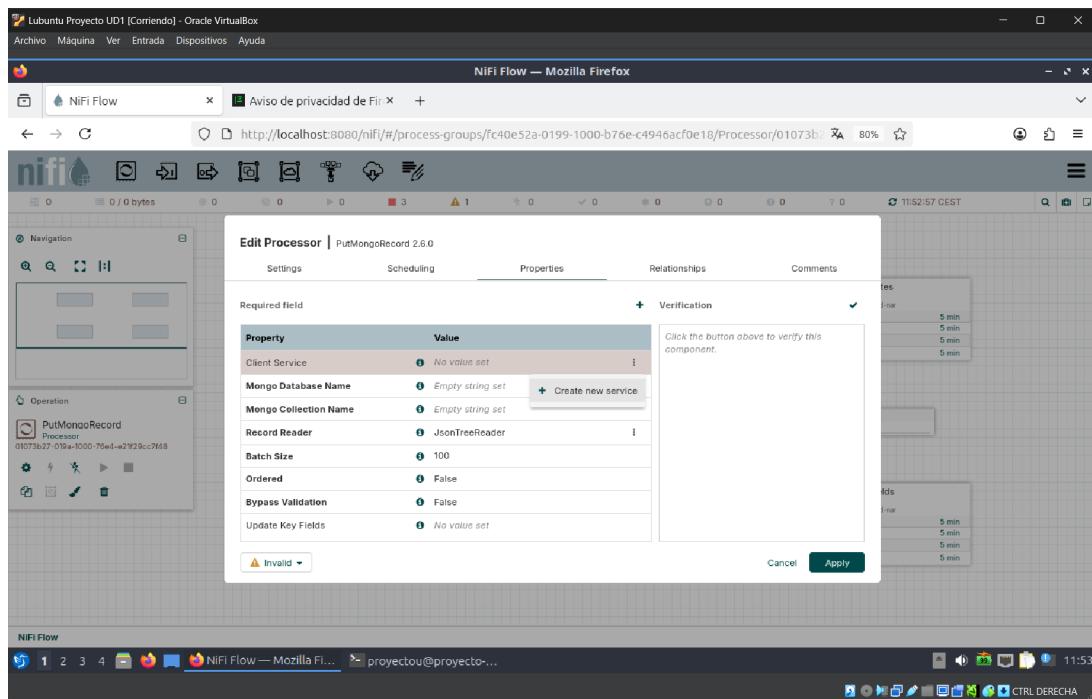


Figura D.57: Configuración (Properties) de 'PutMongoRecord' (vacía).

SmartParking Flow — Monitorización Inteligente

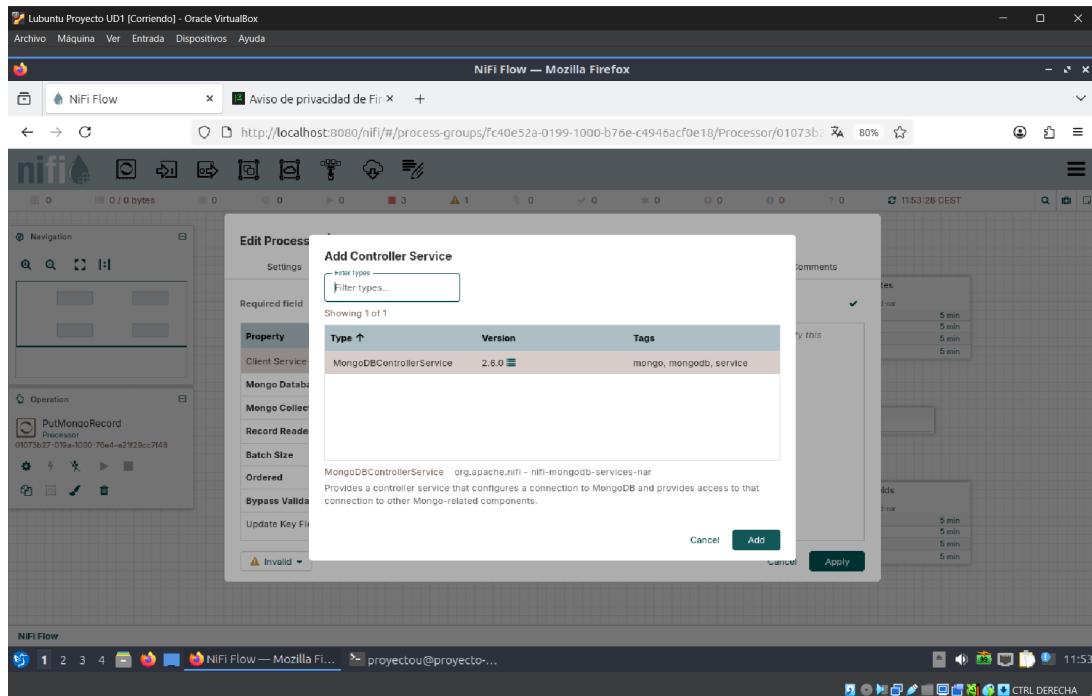


Figura D.58: Creando un nuevo Controller Service: 'MongoDBCPService' (obsoleto, se usará otro).

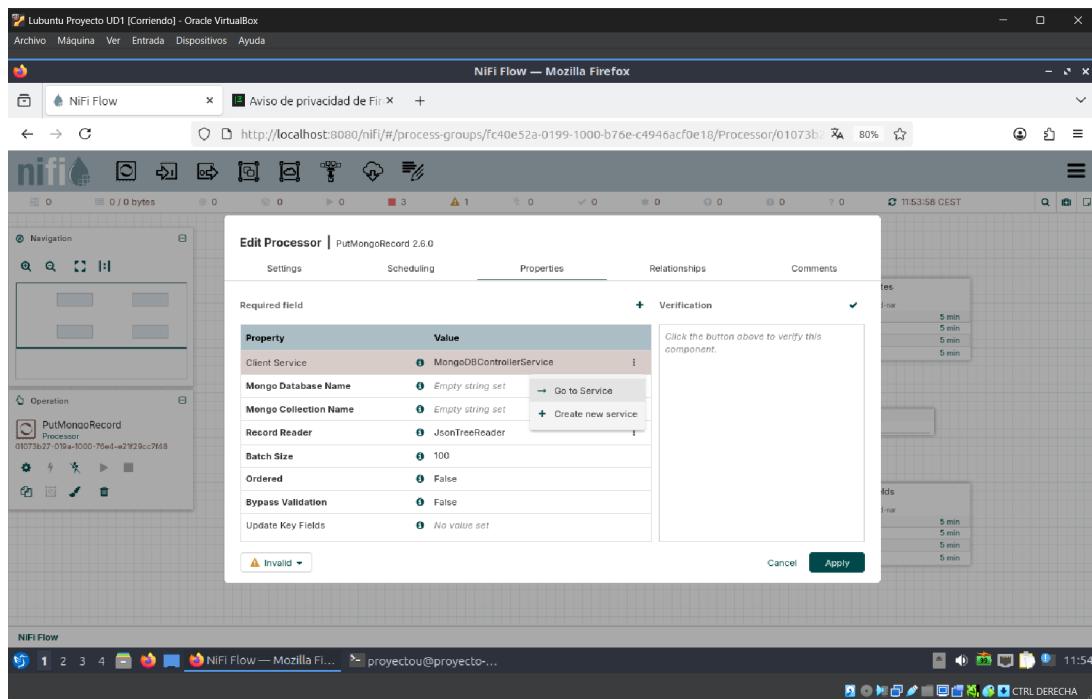


Figura D.59: Configuración de 'PutMongoRecord' (Properties) - se usará 'MongoDBControllerService'.

SmartParking Flow — Monitorización Inteligente

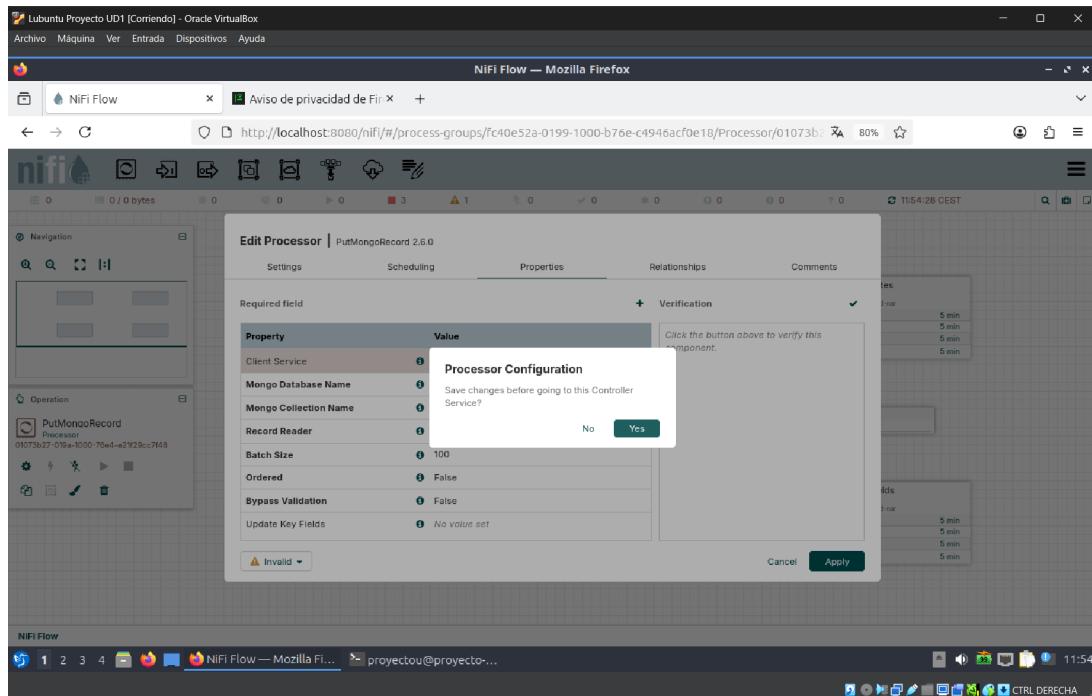


Figura D.60: Aviso de guardado de Controller Service.

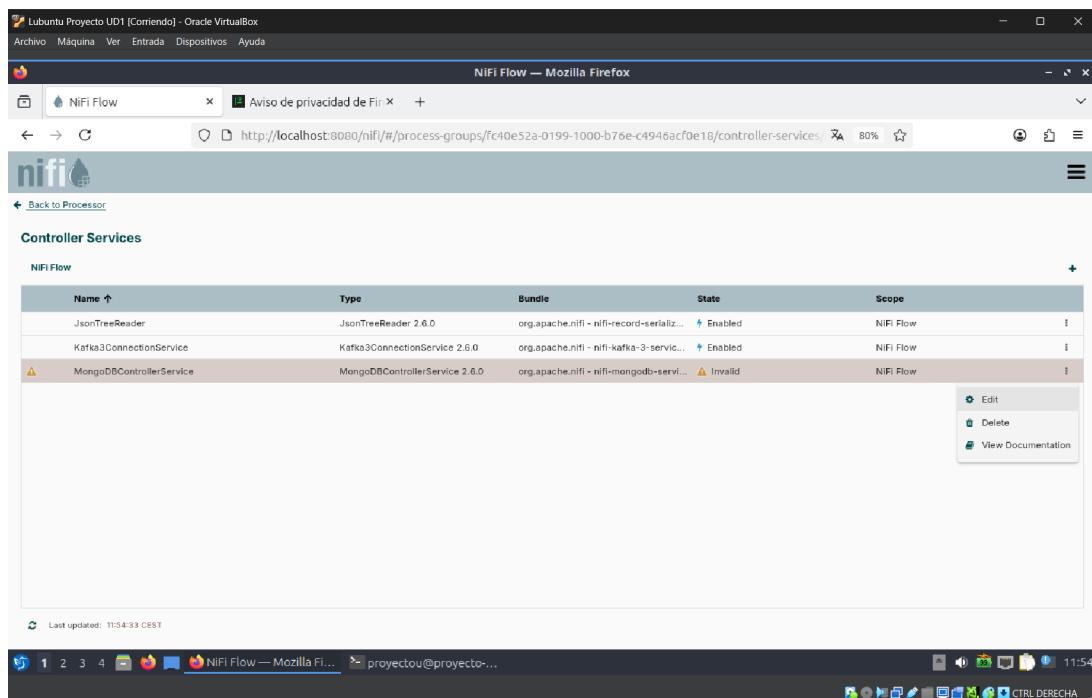


Figura D.61: Accediendo a la configuración del 'MongoDBControllerService'.

SmartParking Flow — Monitorización Inteligente

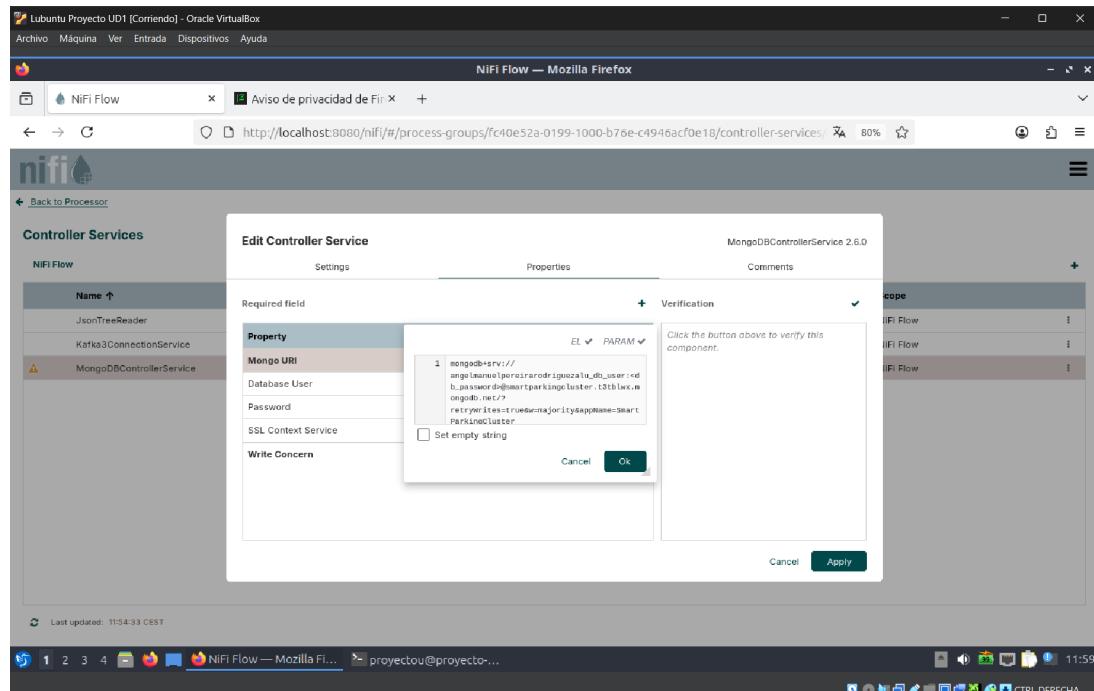


Figura D.62: Editando 'MongoDBControllerService': Pegando la URI de conexión de Atlas.

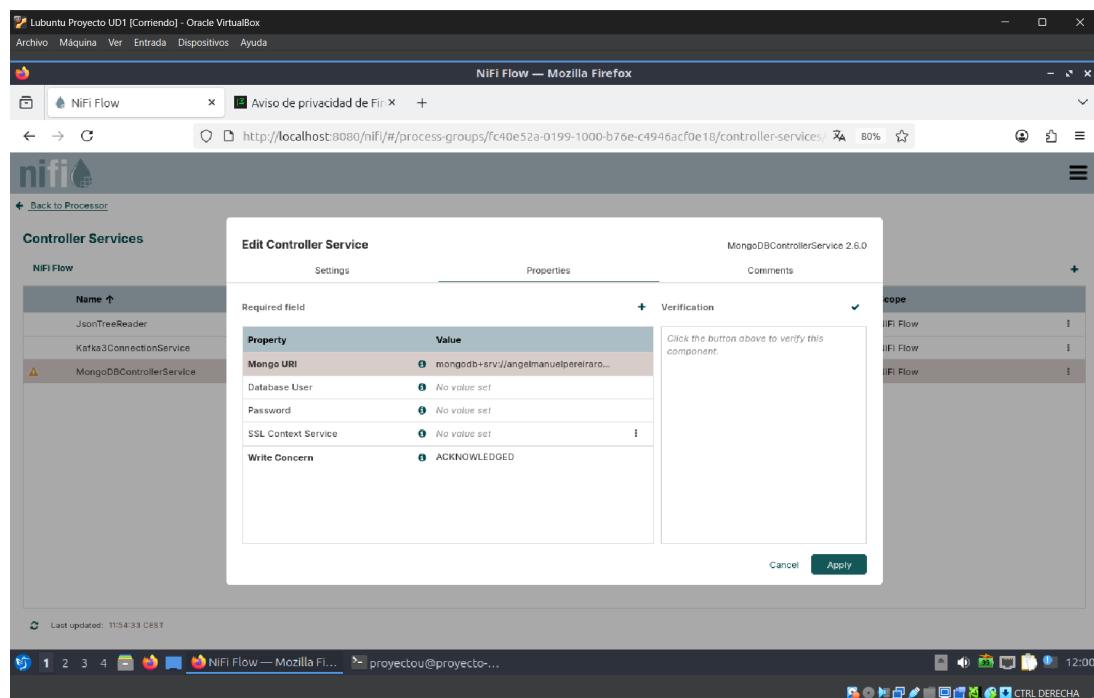


Figura D.63: Configuración (Properties) de 'MongoDBControllerService' con la URI.

SmartParking Flow — Monitorización Inteligente

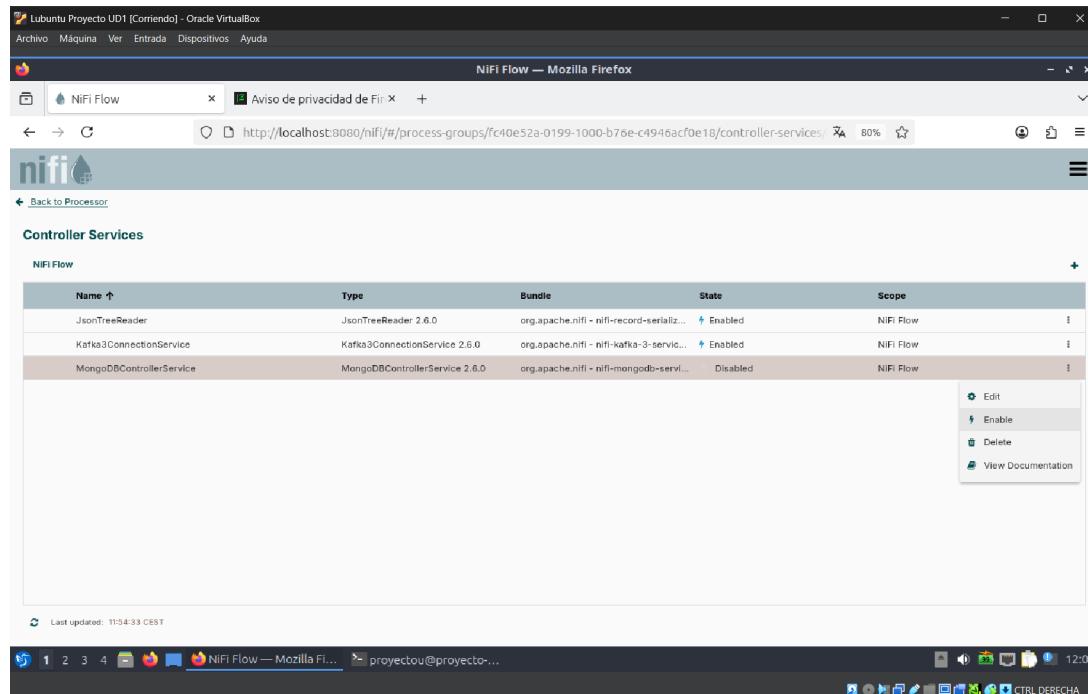


Figura D.64: Habilitando el 'MongoDBControllerService'.

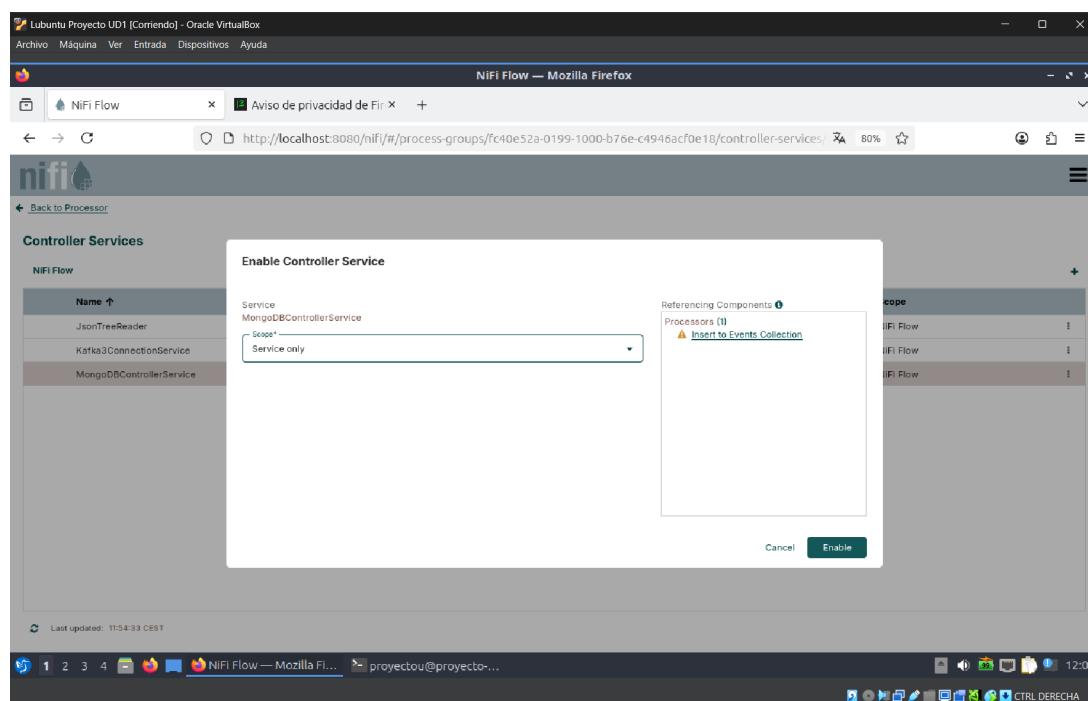


Figura D.65: Confirmación de habilitación del 'MongoDBControllerService'.

SmartParking Flow — Monitorización Inteligente

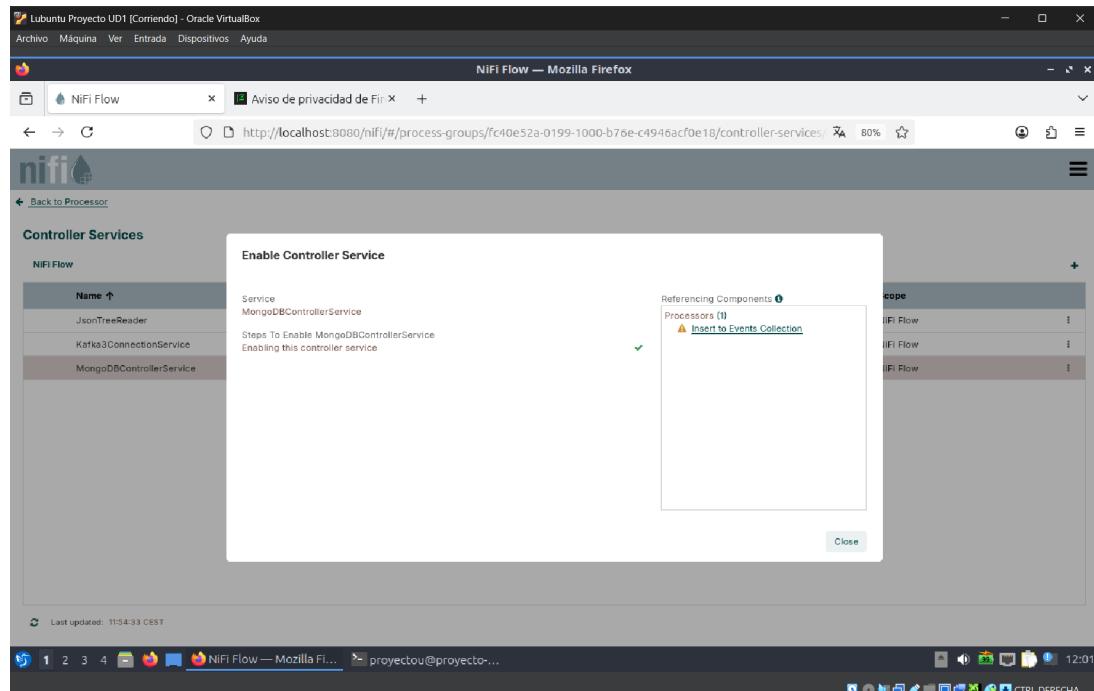


Figura D.66: Habilitando el 'MongoDBControllerService'.

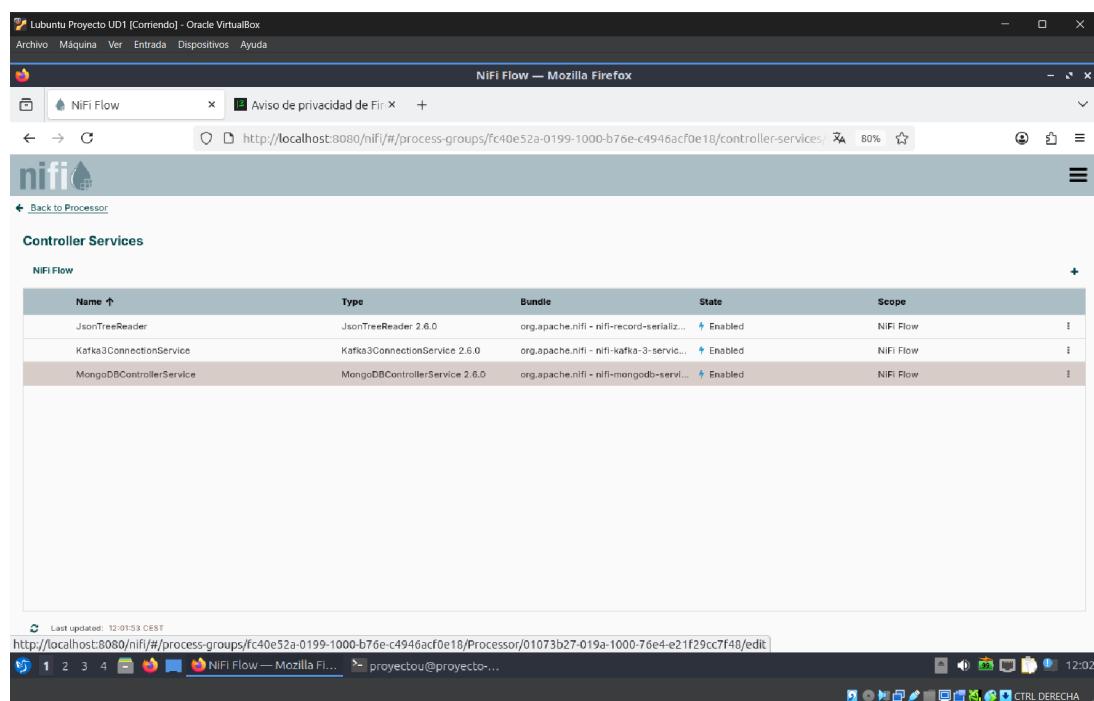


Figura D.67: Todos los Controller Services habilitados.

SmartParking Flow — Monitorización Inteligente

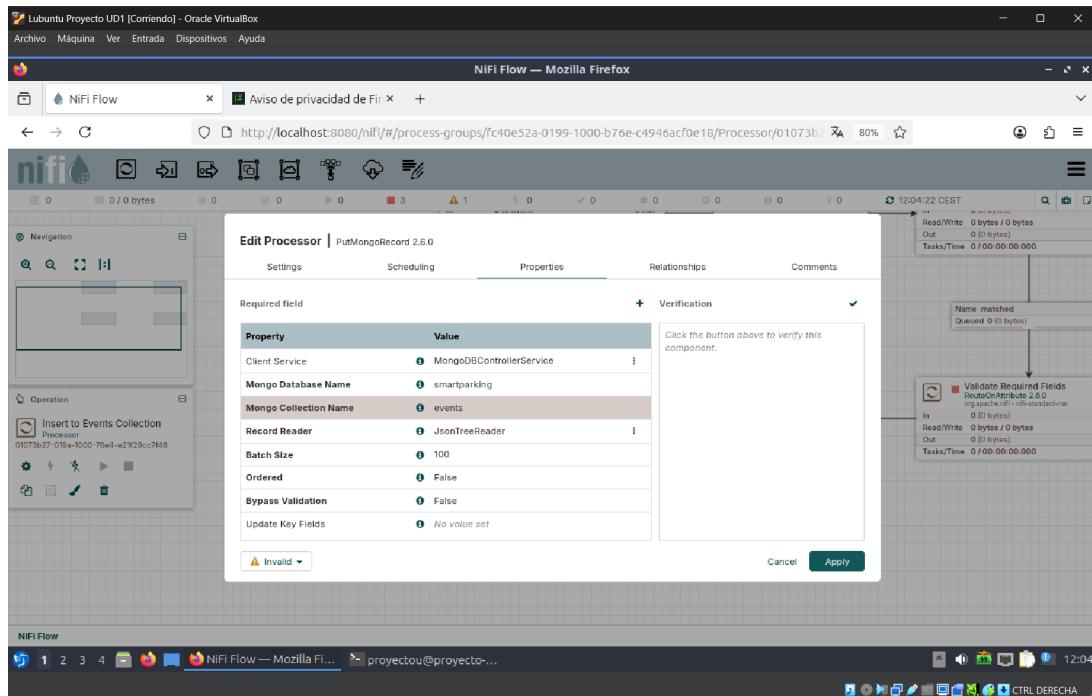


Figura D.68: Configuración de 'PutMongoRecord': 'Mongo Database Name: smartparking', 'Collection: events'.

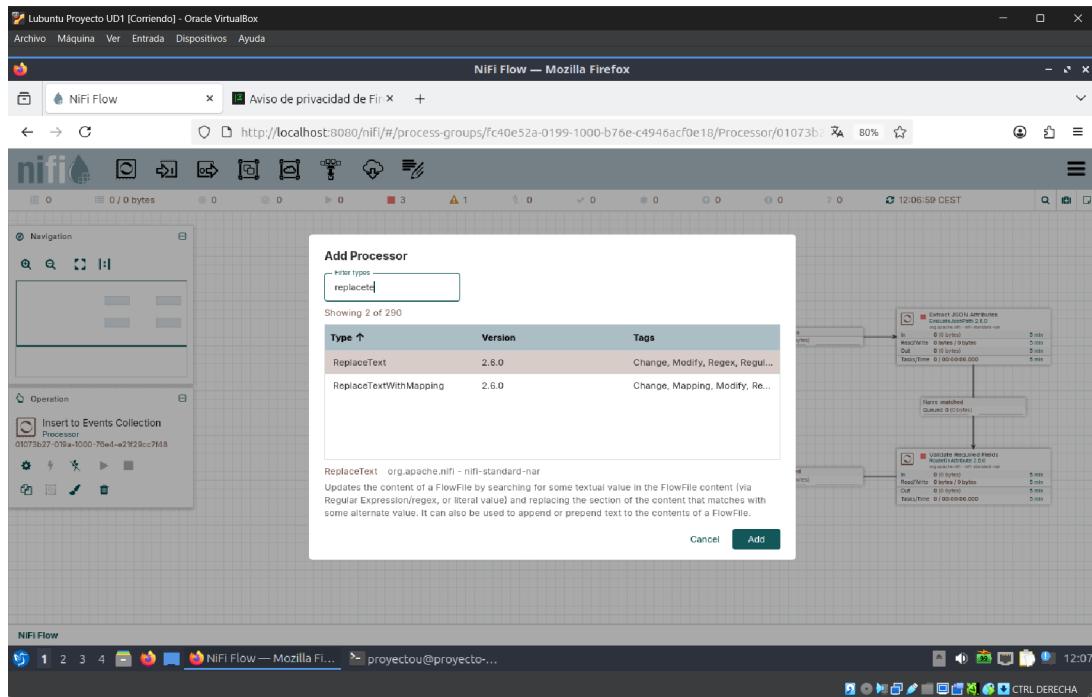


Figura D.69: Añadiendo el procesador 'ReplaceText'.

SmartParking Flow — Monitorización Inteligente

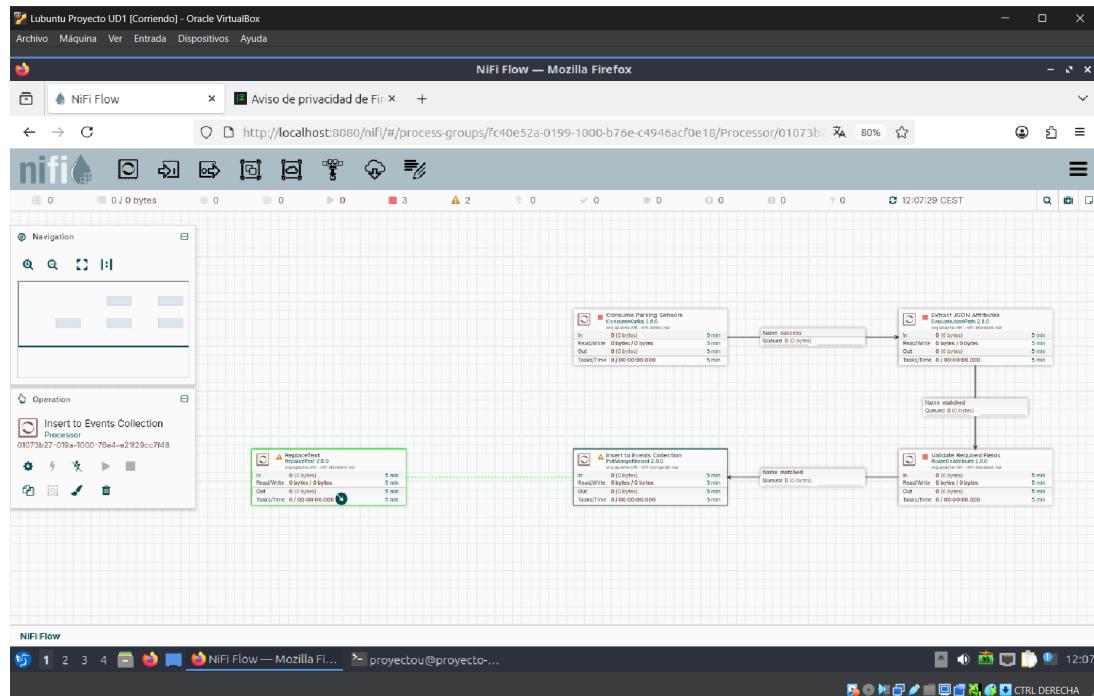


Figura D.70: Flujo con el procesador 'ReplaceText' añadido.

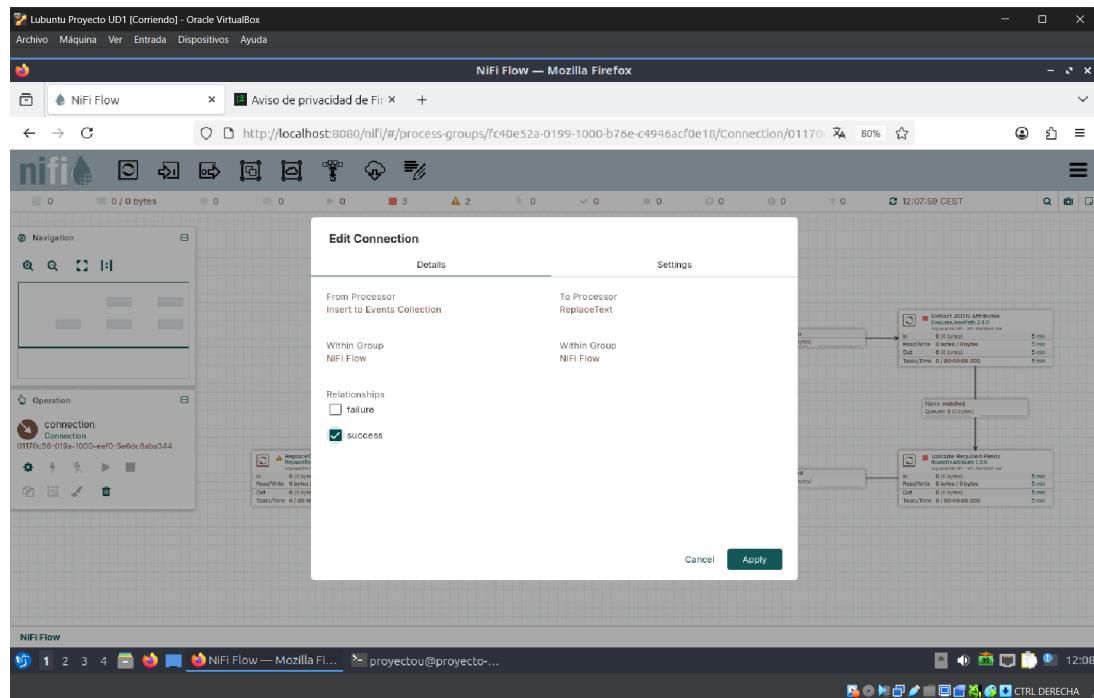


Figura D.71: Creando conexión "success" entre 'Insert to Events Collection' y 'ReplaceText'.

SmartParking Flow — Monitorización Inteligente

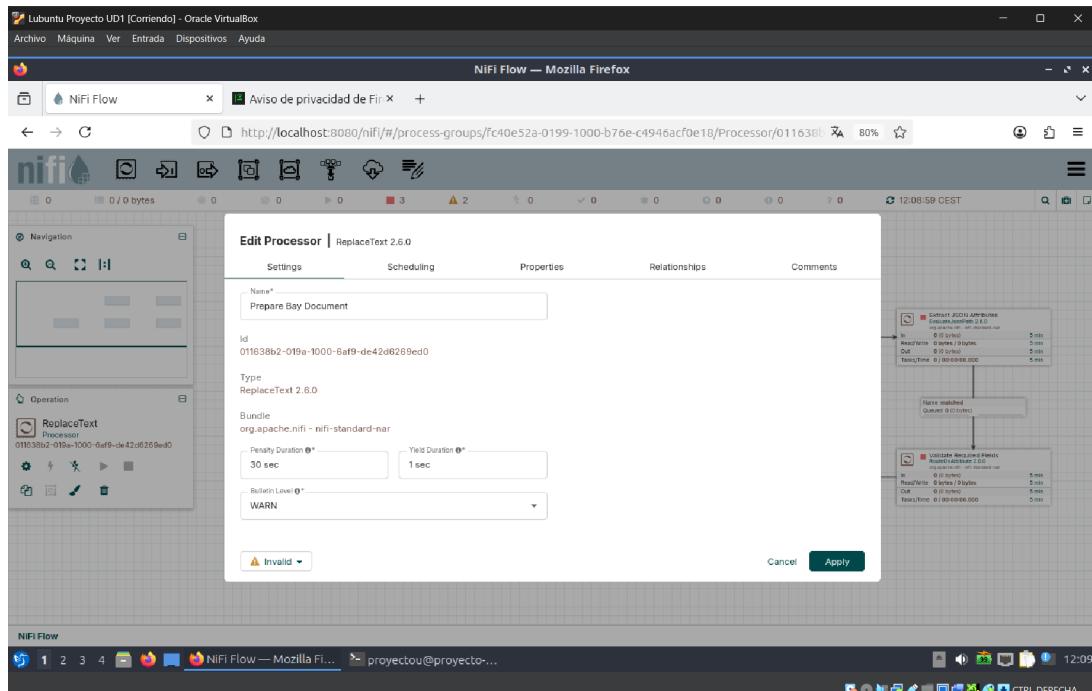


Figura D.72: Configuración (Settings) de 'ReplaceText': 'Name: Prepare Bay Document'.

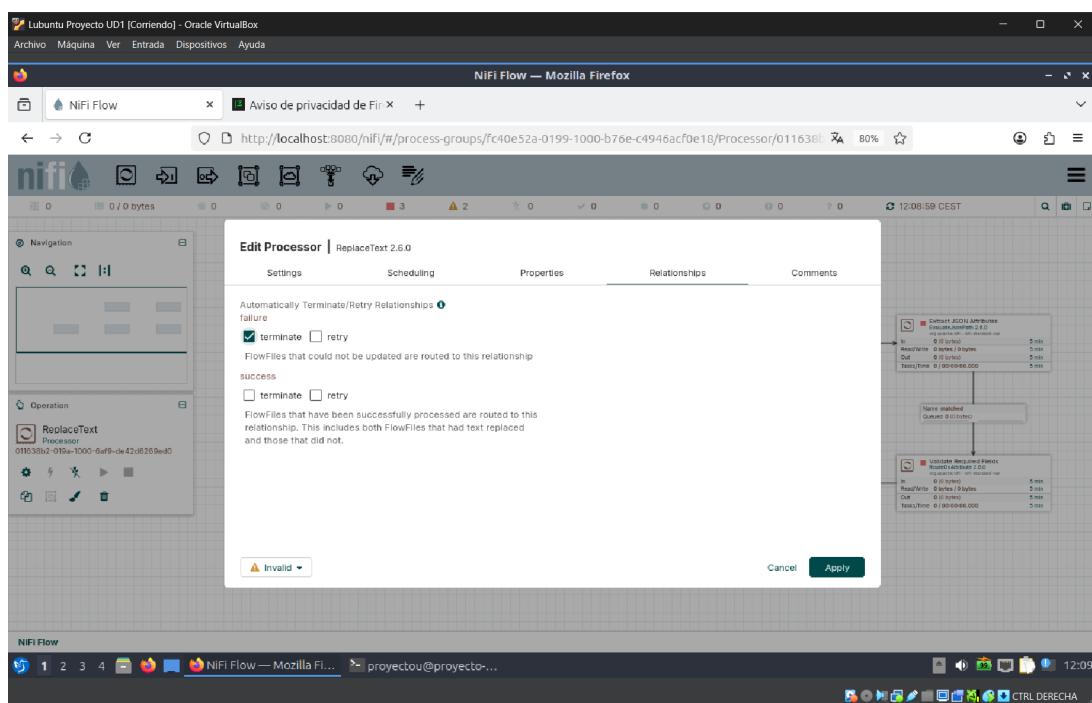


Figura D.73: Configuración (Relationships) de 'ReplaceText': Terminar 'failure'.

SmartParking Flow — Monitorización Inteligente

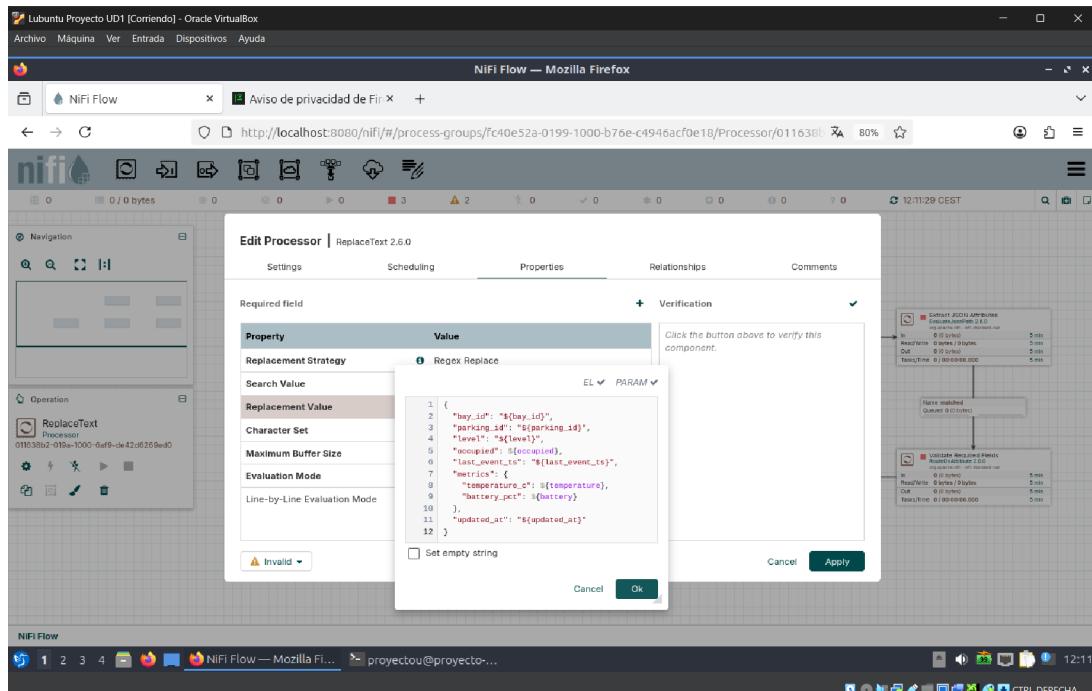


Figura D.74: Configuración (Properties) de ReplaceText: Replacement Value (JSON del documento bays).

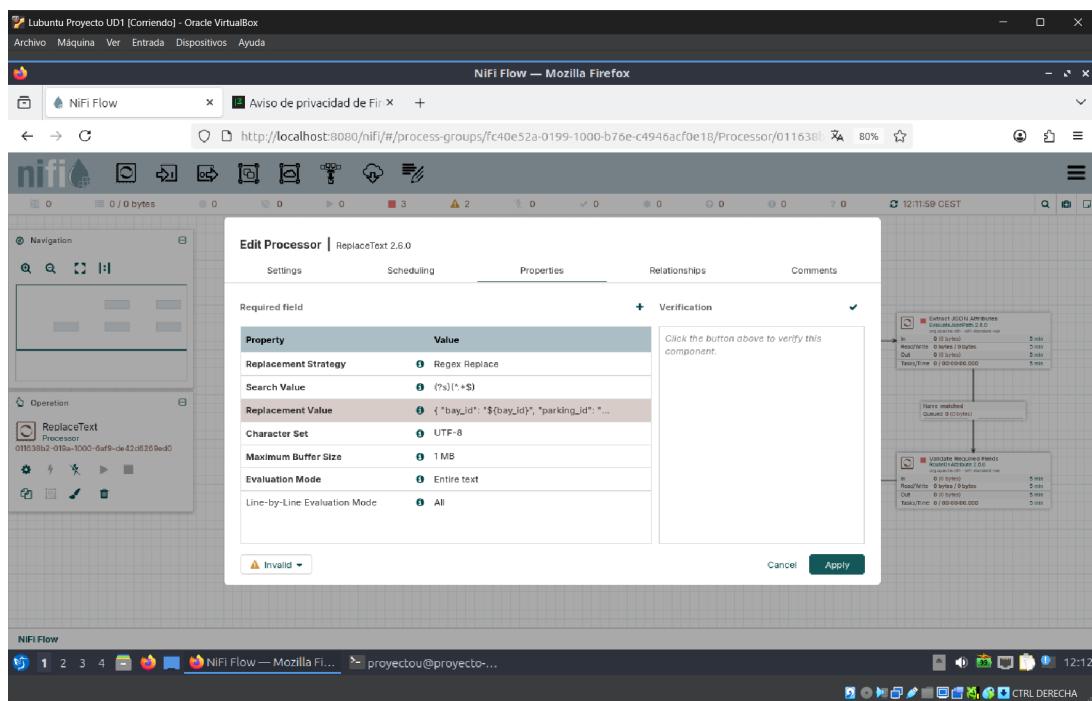


Figura D.75: Configuración (Properties) de ReplaceText: Search Value: (?s)(^.*\$).

SmartParking Flow — Monitorización Inteligente

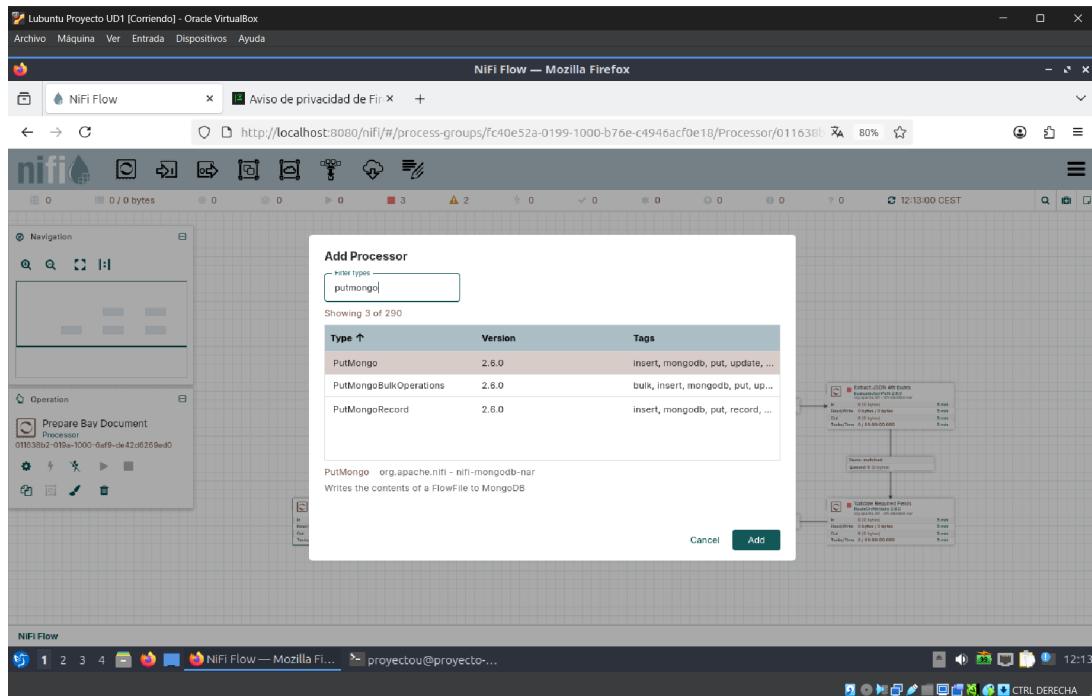


Figura D.76: Añadiendo el procesador 'PutMongo' (para el upsert).

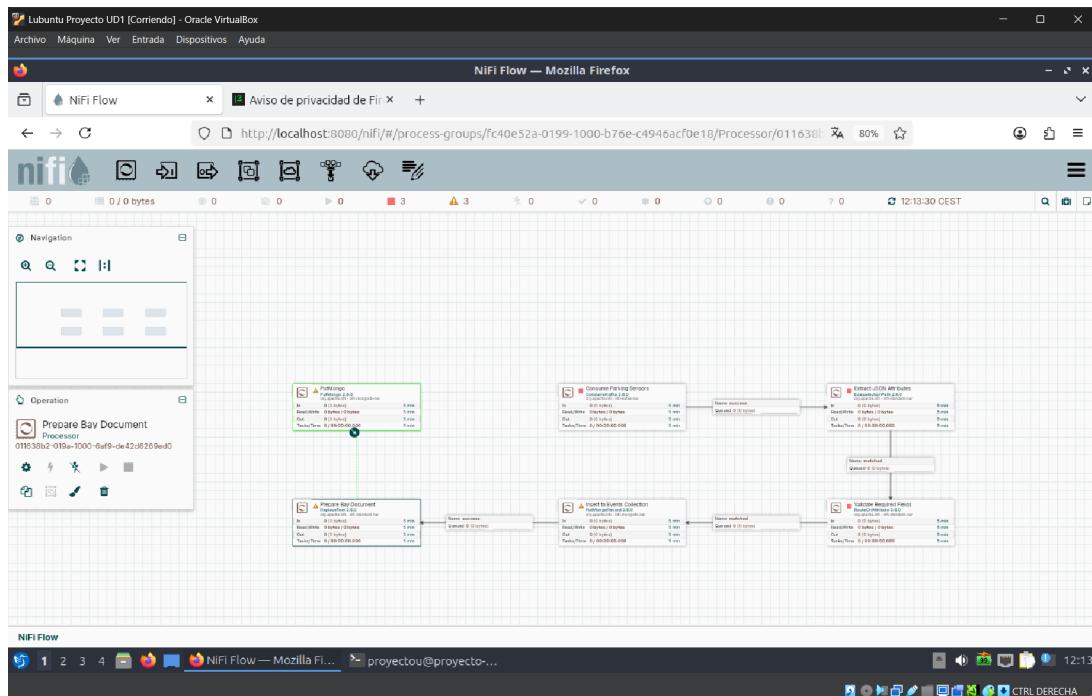


Figura D.77: Flujo con el procesador 'PutMongo' añadido.

SmartParking Flow — Monitorización Inteligente

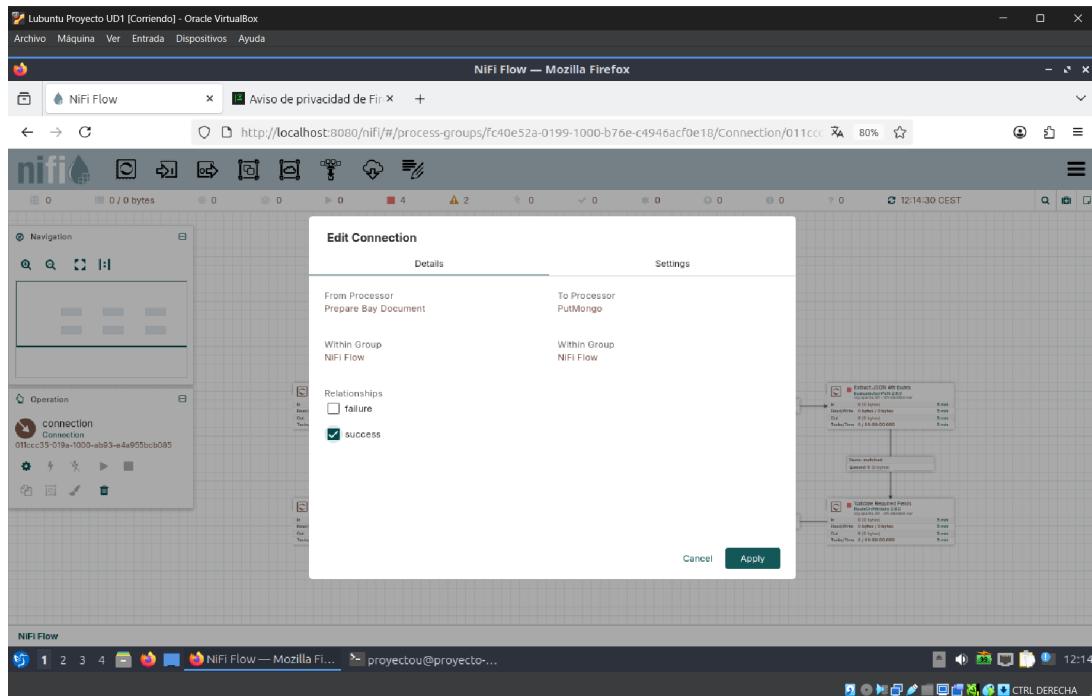


Figura D.78: Creando conexión "success." entre 'ReplaceText' y 'PutMongo'.

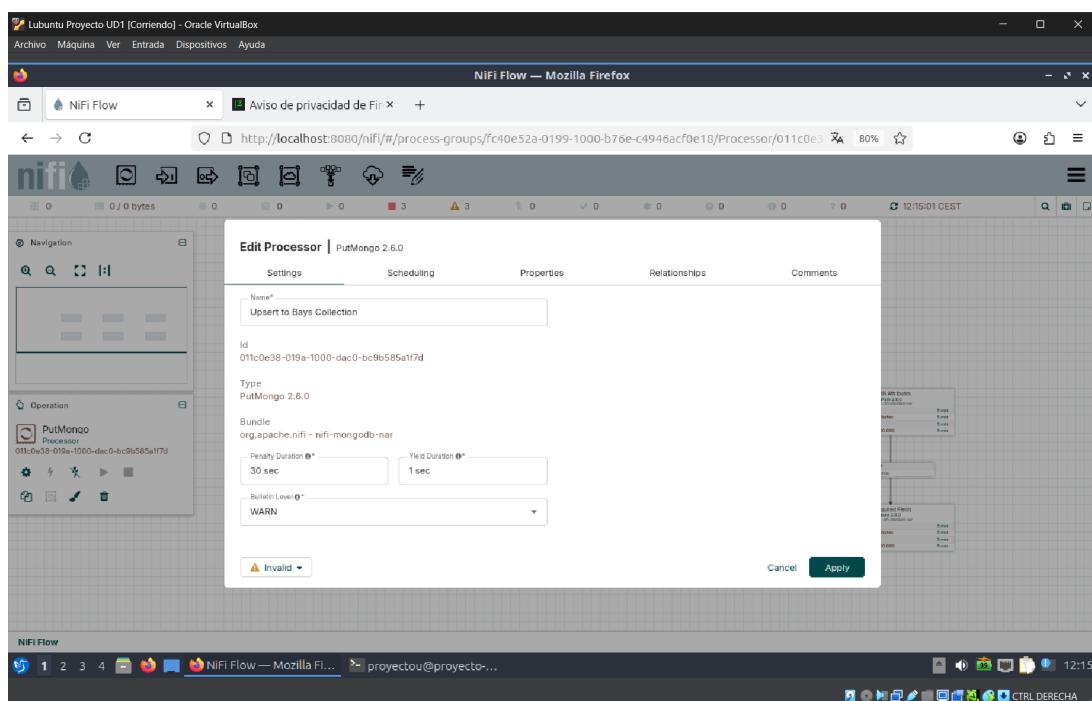


Figura D.79: Configuración (Settings) de 'PutMongo': 'Name: Upsert to Bays Collection'.

SmartParking Flow — Monitorización Inteligente

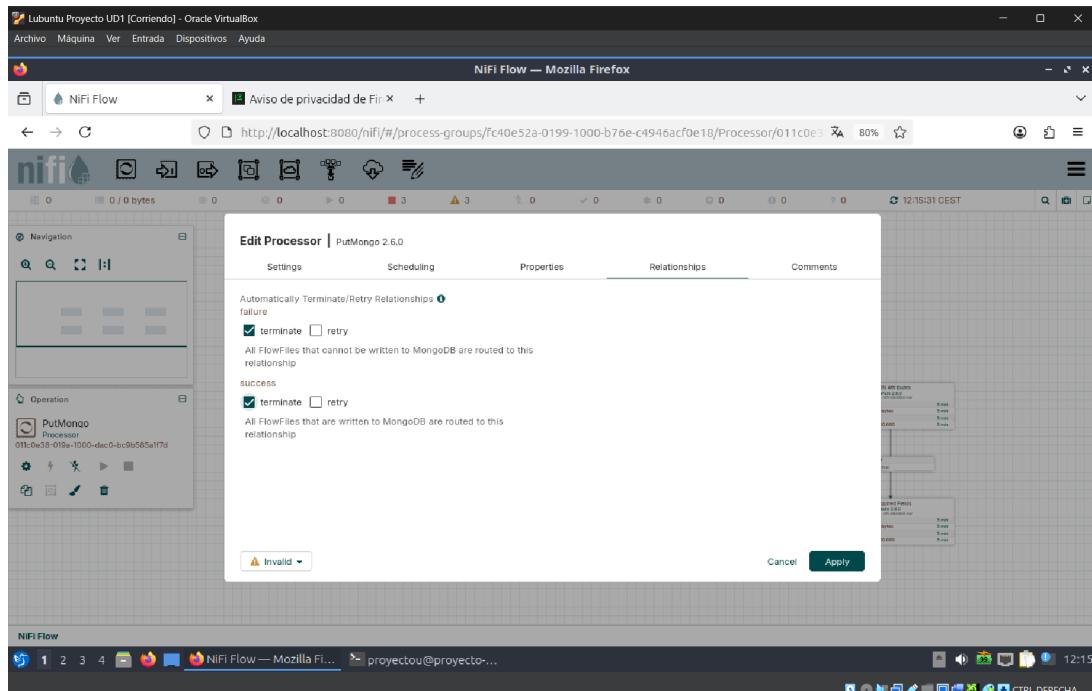


Figura D.80: Configuración (Relationships) de 'PutMongo': Terminar "failurez" "success".

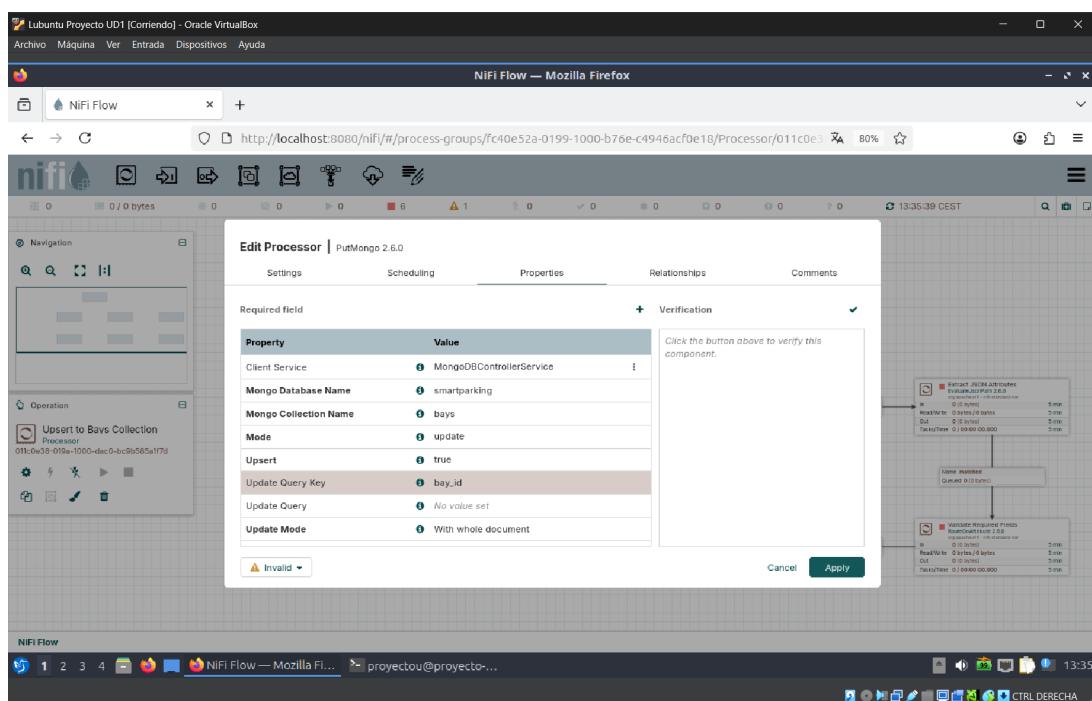


Figura D.81: Configuración (Properties) de 'PutMongo': 'Collection: bays', 'Mode: upsert', 'Key: bay_id'.

SmartParking Flow — Monitorización Inteligente

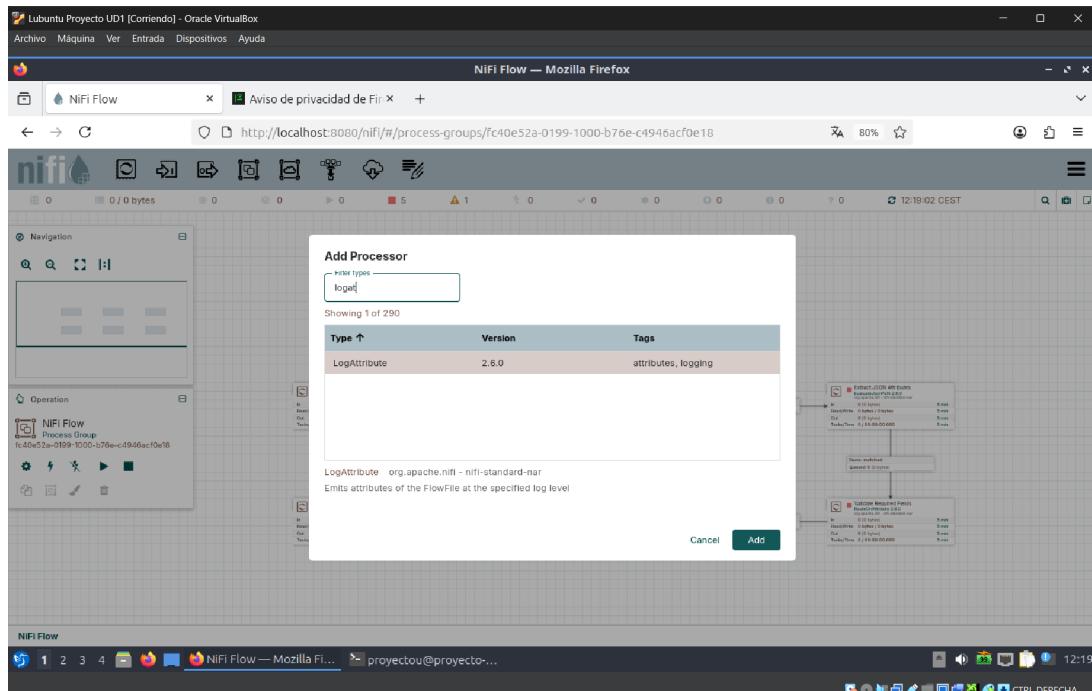


Figura D.82: Añadiendo el procesador 'LogAttribute' (para errores).

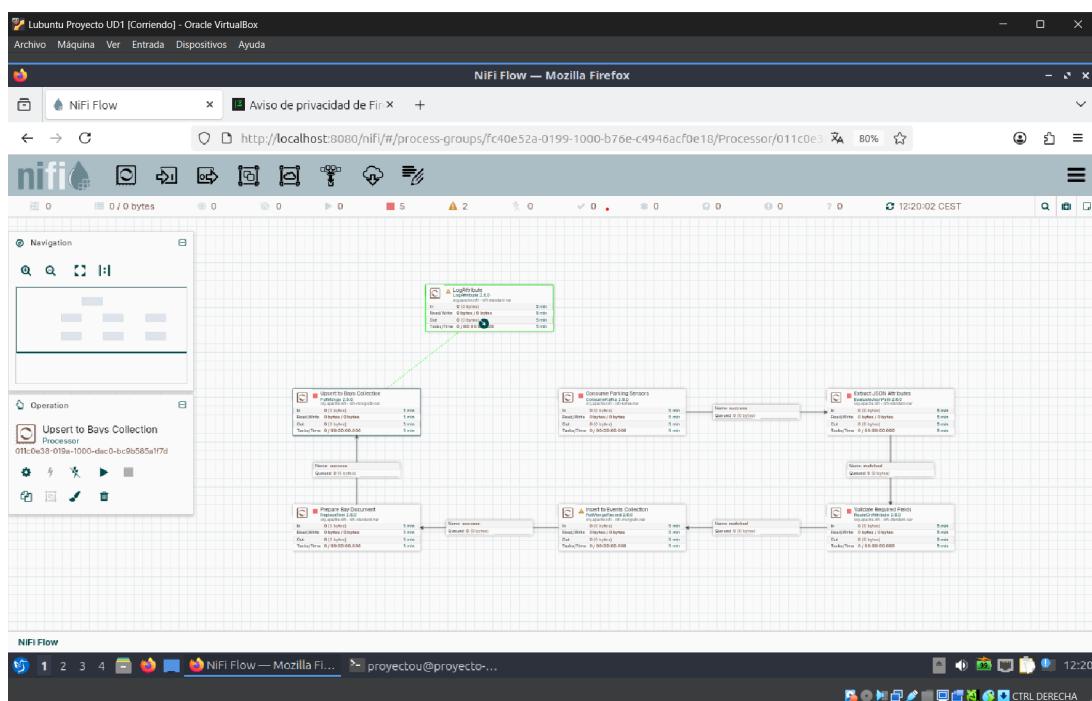


Figura D.83: Flujo con el procesador 'LogAttribute' añadido.

SmartParking Flow — Monitorización Inteligente

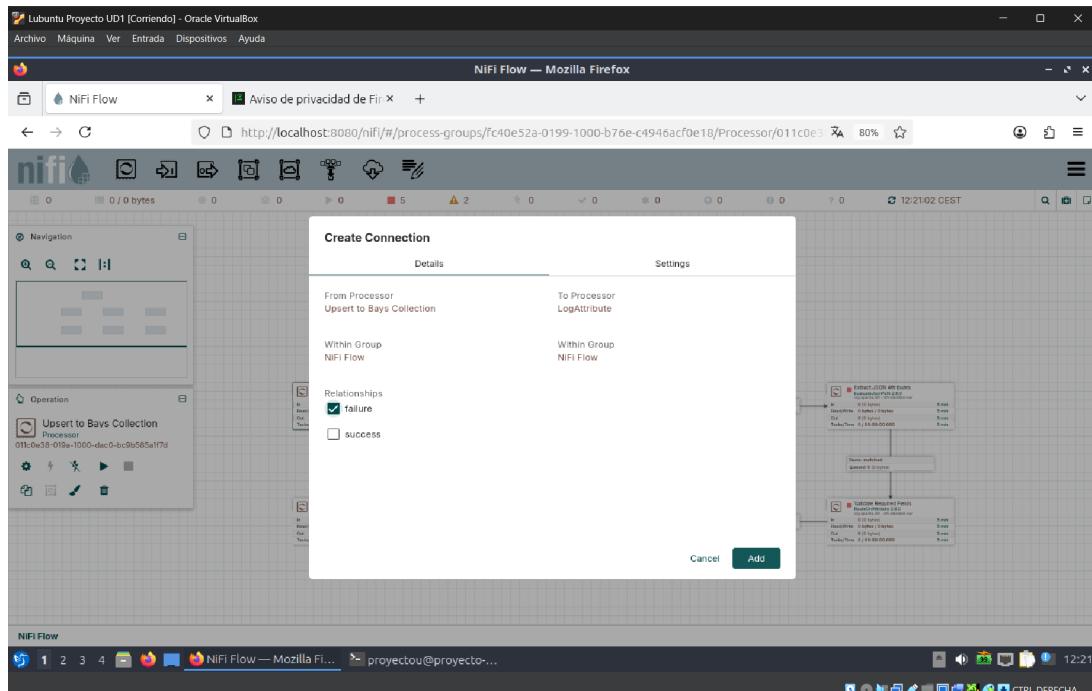


Figura D.84: Creando conexión "failure." entre 'Upsert to Bays Collection' y 'Log Errors'.

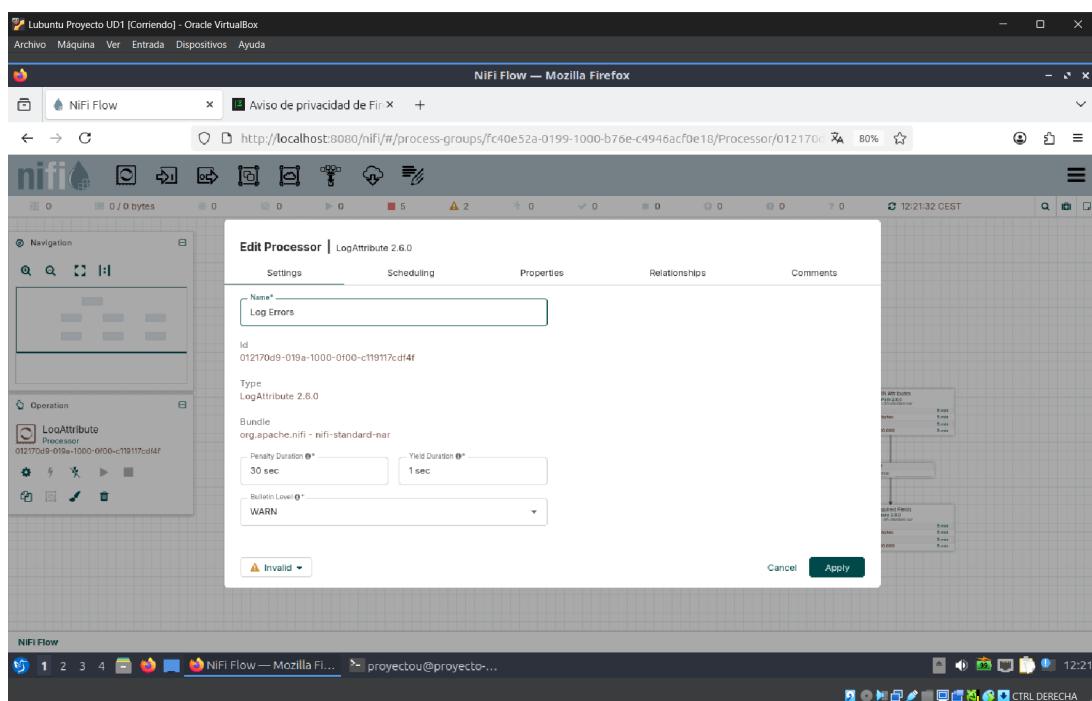


Figura D.85: Configuración (Settings) de 'LogAttribute': 'Name: Log Errors'.

SmartParking Flow — Monitorización Inteligente

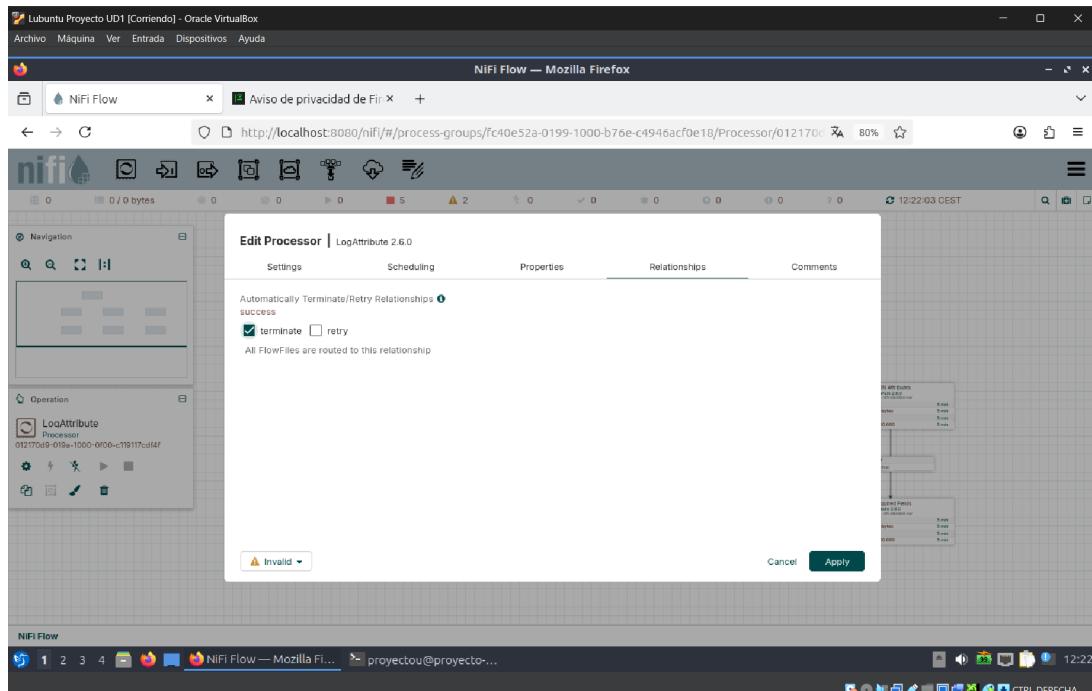


Figura D.86: Configuración (Relationships) de 'LogAttribute': Terminar "success".

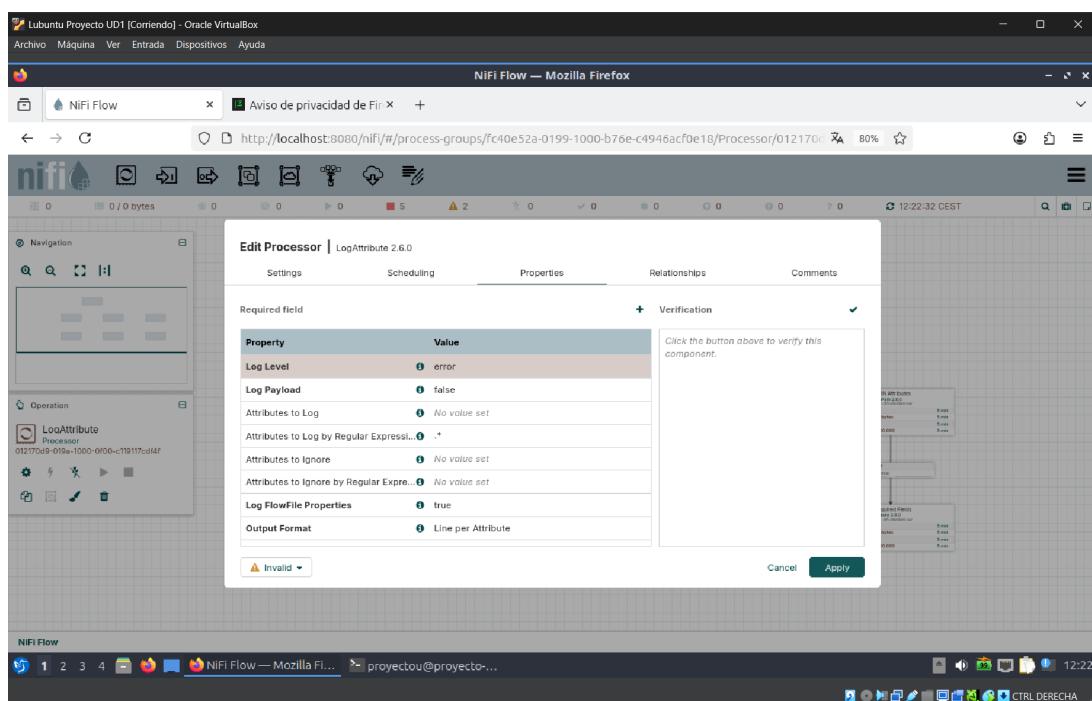


Figura D.87: Configuración (Properties) de 'LogAttribute': 'Log Level: error'.

SmartParking Flow — Monitorización Inteligente

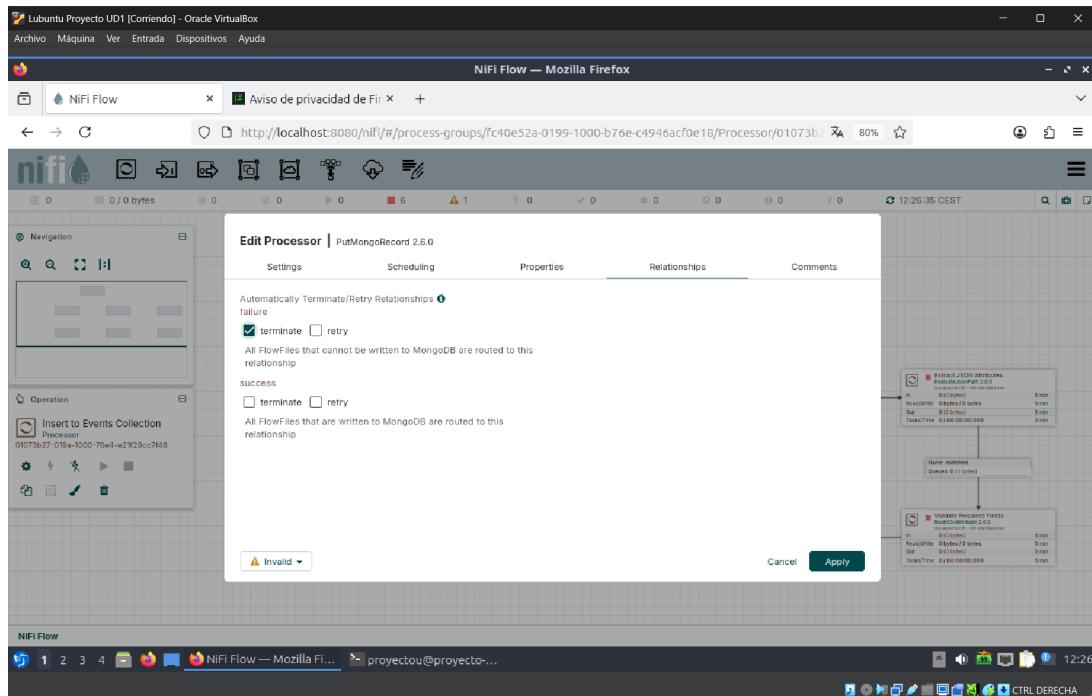


Figura D.88: Configuración (Relationships) de 'Insert to Events Collection': Terminar "failurez" "success".

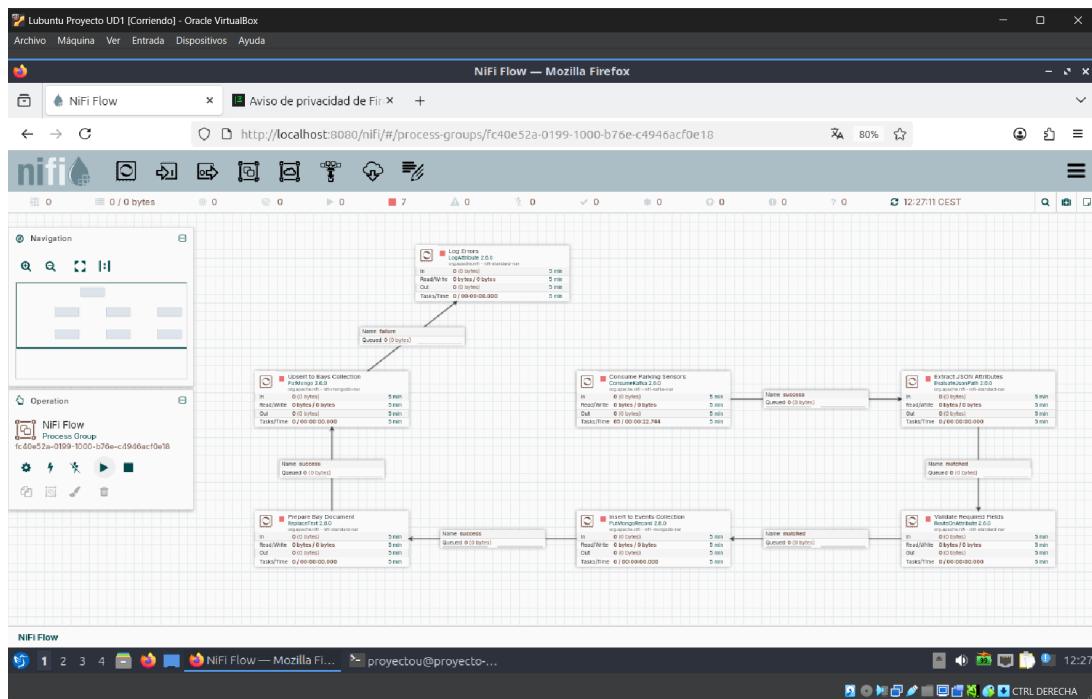


Figura D.89: Vista general del flujo de NiFi (incompleta).

SmartParking Flow — Monitorización Inteligente

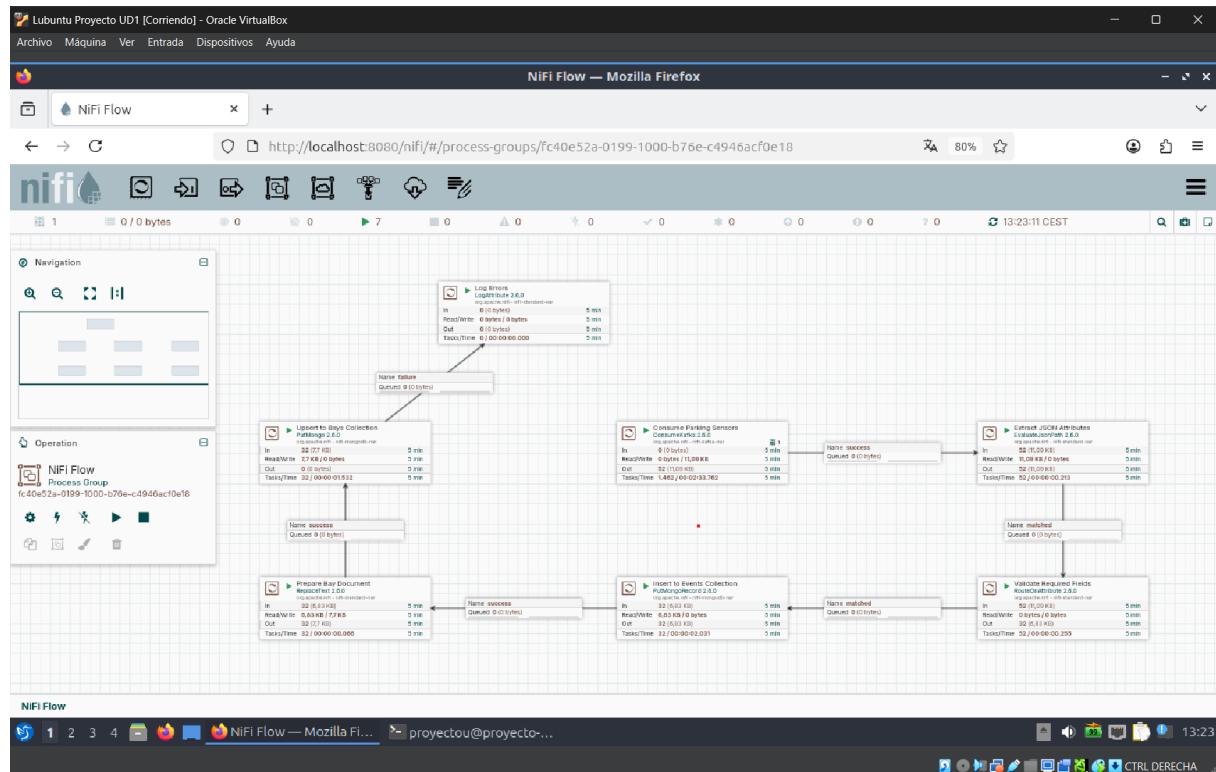


Figura D.90: Vista general del flujo de datos completo en Apache NiFi.

The screenshot shows the MongoDB Atlas Data Explorer interface. It is connected to a cluster named 'SmartParkingCluster' in the 'smartparking' database. The 'events' collection is selected, showing 72K documents. The interface provides a search bar, filter options, and a preview of the query results. The results show three documents with the following schema:

```

{
  "_id": "64f61ba9420d738ba92f12b",
  "bay_id": "L1-A-001",
  "parking_id": "PK-C01Z-01",
  "level": "L1",
  "occupied": true,
  "last_event_ts": "2025-10-20T11:28:29Z",
  "metrics": {
    "0": 0
  },
  "updated_at": "2025-10-20T11:20:30Z"
}

{
  "_id": "64f61ba9420d738ba92f12c",
  "bay_id": "L1-A-004",
  "parking_id": "PK-C01Z-01",
  "level": "L1",
  "occupied": true,
  "last_event_ts": "2025-10-20T11:28:39Z",
  "metrics": {
    "0": 0
  },
  "updated_at": "2025-10-20T11:20:39Z"
}

{
  "_id": "64f61ba9420d738ba92f12d",
  "bay_id": "L2-A-005",
  "parking_id": "PK-C01Z-01",
  "level": "L2",
  "occupied": true,
  "last_event_ts": "2025-10-20T11:28:45Z",
  "metrics": {
    "0": 0
  },
  "updated_at": "2025-10-20T11:20:45Z"
}

```

Figura D.91: Datos de la colección 'events' vistos desde MongoDB Atlas.

SmartParking Flow — Monitorización Inteligente

The screenshot shows the MongoDB Atlas Data Explorer interface. The left sidebar has sections for Cluster, Overview, Data Explorer (selected), Real Time, Cluster Metrics, Query Insights, Performance Advisor, Online Archive, Command Line Tools, Infrastructure as Code, and SHORTCUTS. The main area shows the 'smartparking' cluster with two databases: 'sample_mflix' and 'smartparking'. Under 'smartparking', there are collections 'bays' and 'events'. The 'bays' collection is selected, displaying its storage size (36KB), logical data size (4.4KB), total documents (20), and index size (72KB). A search bar at the top right says 'Search namespaces' with a placeholder 'sample_mflix'. Below it, a query builder allows generating queries from natural language in Compose. The results section shows 'QUERY RESULTS: 1-20 OF 20' with three document snippets. Each snippet includes fields like '_id', 'bay_id', 'parking_id', 'level', 'type', 'occupied', 'last_event_ts', 'metrics' (an object), and 'updated_at'. The bottom status bar says 'System Status: All Good'.

Figura D.92: Datos de la colección 'bays' vistos desde MongoDB Atlas.

Apéndice E

Anexo E: Aplicación Flask (API y Visualización)

Despliegue y funcionamiento de la aplicación web de visualización en tiempo real, incluyendo el código fuente principal de la aplicación.

E.1 Código de la Aplicación (app.py)

A continuación, se muestra el código completo de la aplicación Flask, `app.py`, que gestiona la API REST y la conexión con MongoDB.

```
1 """
2 SmartParking Flask Application
3 Visualizacion en tiempo real del estado del parking inteligente
4 Conecta a MongoDB Atlas para obtener datos actualizados
5 """
6
7 from flask import Flask, render_template, jsonify
8 from pymongo import MongoClient
9 from pymongo.errors import ConnectionFailure,
10    ServerSelectionTimeoutError
11 import os
12 from dotenv import load_dotenv
13 import logging
14 from datetime import datetime
15
16 # Configurar logging
17 logging.basicConfig(
18     level=logging.INFO,
19     format='%(asctime)s - %(name)s - %(levelname)s - %(message)s',
20     handlers=[
21         logging.FileHandler('smartparking.log'),
22         logging.StreamHandler()
23     ]
24 )
25 logger = logging.getLogger(__name__)
26
27 # Cargar variables de entorno
28 load_dotenv()
```

```
28 # Crear aplicacion Flask
29 app = Flask(__name__)
30 app.config['SECRET_KEY'] = os.getenv('SECRET_KEY', 'dev-secret-key-
    change-in-production')
31 app.config['JSON_SORT_KEYS'] = False
32
33 # Configuracion de MongoDB
34 MONGO_URI = os.getenv('MONGO_URI')
35 MONGO_DB = os.getenv('MONGO_DB', 'smartparking')
36
37 # Conectar a MongoDB Atlas
38 db = None
39 try:
40     client = MongoClient(
41         MONGO_URI,
42         serverSelectionTimeoutMS=5000,
43         connectTimeoutMS=10000,
44         socketTimeoutMS=10000
45     )
46     # Verificar conexion
47     client.admin.command('ping')
48     db = client[MONGO_DB]
49     logger.info(f"V Conectado exitosamente a MongoDB Atlas - Base de
      datos: {MONGO_DB}")
50 except ConnectionFailure as e:
51     logger.error(f"X Error de conexion a MongoDB Atlas: {e}")
52     db = None
53 except Exception as e:
54     logger.error(f"X Error inesperado conectando a MongoDB: {e}")
55     db = None
56
57
58 # ===== RUTAS DE LA APLICACION =====
59
60 @app.route('/')
61 def index():
62     """
63         Pagina principal - Mapa visual del parking
64     """
65     return render_template('index.html')
66
67
68 @app.route('/api/health')
69 def health_check():
70     """
71         Endpoint de health check para verificar estado del servicio
72
73         Returns:
74             JSON con estado del servicio y conexion a BD
75     """
76     if db is None:
77         return jsonify({
78             "status": "unhealthy",
79             "database": "disconnected",
80             "timestamp": datetime.now().isoformat()
81         })
```

```
82     }),  503
83
84     try:
85         db.command('ping')
86         total_bays = db.bays.count_documents({})
87         total_events = db.events.count_documents({})
88
89         return jsonify({
90             "status": "healthy",
91             "database": "connected",
92             "total_bays": total_bays,
93             "total_events": total_events,
94             "timestamp": datetime.now().isoformat()
95         }),  200
96
97     except Exception as e:
98         logger.error(f"Error en health check: {e}")
99         return jsonify({
100             "status": "unhealthy",
101             "error": str(e),
102             "timestamp": datetime.now().isoformat()
103         }),  503
104
105
106 @app.route('/api/bays')
107 def get_bays():
108     """
109     Obtener todas las plazas del parking
110
111     Returns:
112         JSON con array de plazas ordenadas por bay_id
113     """
114     if db is None:
115         logger.error("Base de datos no disponible")
116         return jsonify({"error": "Database connection not available"}), 503
117
118     try:
119         # Query optimizada con proyección
120         bays = list(db.bays.find(
121             {},
122             {
123                 '_id': 0,
124                 'bay_id': 1,
125                 'parking_id': 1,
126                 'level': 1,
127                 'occupied': 1,
128                 'last_event_ts': 1,
129                 'metrics': 1,
130                 'updated_at': 1
131             }
132         ).sort('bay_id', 1))
133
134         logger.info(f"API /api/bays: Retornadas {len(bays)} plazas")
135
136         return jsonify({
```

```
137         "success": True,
138         "count": len(bays),
139         "data": bays,
140         "timestamp": datetime.now().isoformat()
141     }, 200
142
143 except Exception as e:
144     logger.error(f"Error en /api/bays: {e}")
145     return jsonify({
146         "success": False,
147         "error": str(e)
148     }), 500
149
150
151 @app.route('/api/stats')
152 def get_stats():
153     """
154     Obtener estadísticas generales del parking
155
156     Returns:
157         JSON con total, ocupadas, libres, porcentaje y estadísticas por
158         nivel
159     """
160
161     if db is None:
162         logger.error("Base de datos no disponible")
163         return jsonify({"error": "Database connection not available"}), 503
164
165     try:
166         # Estadísticas generales
167         total = db.bays.count_documents({})
168         occupied = db.bays.count_documents({"occupied": True})
169         free = total - occupied
170         occupancy_rate = round((occupied / total * 100), 2) if total > 0
171         else 0
172
173         # Estadísticas por nivel usando agregación
174         pipeline = [
175             {
176                 "$group": {
177                     "_id": "$level",
178                     "total": {"$sum": 1},
179                     "occupied": {
180                         "$sum": {"$cond": ["$occupied", 1, 0]}
181                     },
182                     "free": {
183                         "$sum": {"$cond": ["$occupied", 0, 1]}
184                     },
185                     "avg_temperature": {"$avg": "$metrics.temperature_c"
186                 },
187                     "avg_battery": {"$avg": "$metrics.battery_pct"},
188                     "low_battery_count": {
189                         "$sum": {
190                             "$cond": [
191                                 {"$lt": ["$metrics.battery_pct", 20]}, 1,
```

```
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
```

```
          0
        ]
      }
    }
  },
  {"$sort": {"_id": 1}}
]

by_level = list(db.bays.aggregate(pipeline))

# Formatear resultados por nivel
levels_stats = []
for level in by_level:
    levels_stats.append({
        "level": level["_id"],
        "total": level["total"],
        "occupied": level["occupied"],
        "free": level["free"],
        "occupancy_rate": round((level["occupied"] / level["total"]) * 100, 2) if level["total"] > 0 else 0,
        "avg_temperature": round(level["avg_temperature"], 1) if level["avg_temperature"] else None,
        "avg_battery": round(level["avg_battery"], 1) if level["avg_battery"] else None,
        "low_battery_sensors": level["low_battery_count"]
    })

stats = {
    "success": True,
    "total": total,
    "occupied": occupied,
    "free": free,
    "occupancy_rate": occupancy_rate,
    "levels": levels_stats,
    "timestamp": datetime.now().isoformat()
}

logger.info(f"API /api/stats: {occupied}/{total} ocupadas ({occupancy_rate}%)")

return jsonify(stats), 200

except Exception as e:
    logger.error(f"Error en /api/stats: {e}")
    return jsonify({
        "success": False,
        "error": str(e)
}), 500

@app.route('/api/bays/level/<level>')
def get_bays_by_level(level):
    """
    Obtener plazas de un nivel específico

```

```
241     Args:
242         level (str): Nivel del parking (L1, L2, L3, etc.)
243
244     Returns:
245         JSON con array de plazas del nivel solicitado
246     """
247     if db is None:
248         return jsonify({"error": "Database connection not available"}), 503
249
250     try:
251         level_upper = level.upper()
252
253         bays = list(db.bays.find(
254             {"level": level_upper},
255             {"_id": 0}
256         ).sort('bay_id', 1))
257
258         if not bays:
259             logger.warning(f"No se encontraron plazas para nivel {level_upper}")
260             return jsonify({
261                 "success": True,
262                 "level": level_upper,
263                 "count": 0,
264                 "data": [],
265                 "message": f"No hay plazas en el nivel {level_upper}"
266             }), 200
267
268             logger.info(f"API /api/bays/level/{level_upper}: {len(bays)} plazas")
269
270         return jsonify({
271             "success": True,
272             "level": level_upper,
273             "count": len(bays),
274             "data": bays
275         }), 200
276
277     except Exception as e:
278         logger.error(f"Error en /api/bays/level/{level}: {e}")
279         return jsonify({
280             "success": False,
281             "error": str(e)
282         }), 500
283
284
285 @app.route('/api/maintenance/low-battery')
286 def get_low_battery_bays():
287     """
288     Obtener plazas con bateria baja que requieren mantenimiento
289
290     Returns:
291         JSON con plazas que tienen bateria < 20 %
292     """
293     if db is None:
```

```
294         return jsonify({"error": "Database connection not available"}),  
295     503  
296  
297     try:  
298         low_battery_threshold = 20  
299  
300         low_battery_bays = list(db.bays.find(  
301             {"metrics.battery_pct": {"$lt": low_battery_threshold}},  
302             {'_id': 0}  
303         ).sort('metrics.battery_pct', 1))  
304  
305         # Clasificar por prioridad  
306         urgent = [b for b in low_battery_bays if b['metrics'][  
307             'battery_pct'] < 10]  
308         high = [b for b in low_battery_bays if 10 <= b['metrics'][  
309             'battery_pct'] < 20]  
310  
311         logger.warning(f"Mantenimiento: {len(low_battery_bays)} sensores  
312         con bateria baja")  
313  
314         return jsonify({  
315             "success": True,  
316             "total_count": len(low_battery_bays),  
317             "urgent_count": len(urgent),  
318             "high_priority_count": len(high),  
319             "urgent": urgent,  
320             "high_priority": high,  
321             "threshold": low_battery_threshold  
322         }), 200  
323  
324     except Exception as e:  
325         logger.error(f"Error en /api/maintenance/low-battery: {e}")  
326         return jsonify({  
327             "success": False,  
328             "error": str(e)  
329         }), 500  
330  
331 # ===== MANEJADORES DE ERRORES =====  
332  
333 @app.errorhandler(404)  
334 def not_found(error):  
335     """Manejador para errores 404"""  
336     return jsonify({  
337         "success": False,  
338         "error": "Endpoint no encontrado",  
339         "code": 404  
340     }), 404  
341  
342 @app.errorhandler(500)  
343 def internal_error(error):  
344     """Manejador para errores 500"""  
345     logger.error(f"Error interno del servidor: {error}")  
346     return jsonify({  
347         "success": False,
```

```
346         "error": "Error interno del servidor",
347         "code": 500
348     }, 500
349
350
351 @app.errorhandler(503)
352 def service_unavailable(error):
353     """Manejador para errores 503"""
354     return jsonify({
355         "success": False,
356         "error": "Servicio no disponible",
357         "code": 503
358     }), 503
359
360
361 # ===== PUNTO DE ENTRADA =====
362
363 if __name__ == '__main__':
364     logger.info("=" * 60)
365     logger.info("Iniciando SmartParking Flask Application")
366     logger.info("=" * 60)
367     logger.info(f"Base de datos: {MONGO_DB}")
368     logger.info(f"Servidor: http://localhost:5000")
369     logger.info("=" * 60)
370
371     # Ejecutar aplicacion
372     app.run(
373         host='localhost',
374         port=5000,
375         debug=True
376     )
```

Listing E.1: Código fuente de `app.py`.

E.2 Despliegue y Funcionamiento

```

1 # ===== SMARTPARKING FLASK APPLICATION =====
2 # Dependencias necesarias para ejecutar la aplicación
3
4 # Framework web
5 Flask==3.0.0
6 Werkzeug==3.0.1
7
8 # MongoDB driver con soporte para Atlas (mongodb+srv://)
9 pymongo[srv]==4.1
10
11 # Resolución DNS para MongoDB Atlas
12 dnspython==2.4.2
13
14 # Variables de entorno
15 python-dotenv==1.0.0
16

```

PROBLEMAS SALIDA CONSOLA DE DEPURACIÓN TERMINAL PUERTOS POSTMAN CONSOLE

PS C:\Users\amp2\Documents\FormacionProfesional\CFGS\IABD\BDA\Proyecto UD1\SmartParking Flow> python -m venv venv
PS C:\Users\amp2\Documents\FormacionProfesional\CFGS\IABD\BDA\Proyecto UD1\SmartParking Flow> \venv\Scripts\activate
(venv) PS C:\Users\amp2\Documents\FormacionProfesional\CFGS\IABD\BDA\Proyecto UD1\SmartParking Flow> pip install -r requirements.txt
Requirement already satisfied: Flask==3.0.0 in c:\users\amp2\documents\formacionprofesionalcfgs\iabd\bda\proyecto ud1\smartparking flow\venv\lib\site-packages (from -r requirements.txt (line 5)) (3.0.0)
Requirement already satisfied: Werkzeug==3.0.1 in c:\users\amp2\documents\formacionprofesionalcfgs\iabd\bda\proyecto ud1\smartparking flow\venv\lib\site-packages (from -r requirements.txt (line 6)) (3.0.1)
Requirement already satisfied: pymongo[srv]==4.1 in c:\users\amp2\documents\formacionprofesionalcfgs\iabd\bda\proyecto ud1\smartparking flow\venv\lib\site-packages (from -r requirements.txt (line 9)) (4.1.0)
Requirement already satisfied: dnspython==2.4.2 in c:\users\amp2\documents\formacionprofesionalcfgs\iabd\bda\proyecto ud1\smartparking flow\venv\lib\site-packages (from -r requirements.txt (line 12)) (2.4.2)
Requirement already satisfied: python-dotenv==1.0.0 in c:\users\amp2\documents\formacionprofesionalcfgs\iabd\bda\proyecto ud1\smartparking flow\venv\lib\site-packages (from -r requirements.txt (line 15)) (1.0.0)
Requirement already satisfied: click==8.1.3 in c:\users\amp2\documents\formacionprofesionalcfgs\iabd\bda\proyecto ud1\smartparking flow\venv\lib\site-packages (from Flask==3.0.0->r requirements.txt (line 5)) (8.1.3)
Requirement already satisfied: MarkupSafe==2.1.1 in c:\users\amp2\documents\formacionprofesionalcfgs\iabd\bda\proyecto ud1\smartparking flow\venv\lib\site-packages (from Werkzeug==3.0.1->r requirements.txt (line 6)) (2.1.1)
Requirement already satisfied: colorama in c:\users\amp2\documents\formacionprofesionalcfgs\iabd\bda\proyecto ud1\smartparking flow\venv\lib\site-packages (from click==8.1.3->Flask==3.0.0->r requirements.txt (line 5)) (0.4.6)

[notice] A new release of pip available: 22.2.2 > 25.3
[notice] To update, run: python -m pip install --upgrade pip
(venv) PS C:\Users\amp2\Documents\FormacionProfesional\CFGS\IABD\BDA\Proyecto UD1\SmartParking Flow>

Figura E.1: Instalación de dependencias de Python (`requirements.txt`).

```

1 # ===== MONGODB ATLAS CONFIGURATION =====
2 # Reemplaza los siguientes valores con tu información real de MongoDB Atlas
3
4 # Connection String de MongoDB Atlas
5 # Formato: mongodb+srv://:usuario:password@cluster.xxxxx.mongodb.net/?retryWrites=true&w=majority
6 MONGO_URI=mongodb+srv://:smartparking_user:TU_PASSWORD_AQUI@smartparkingcluster.xxxxx.mongodb.net/?retryWrites=true&w=majority&appName=SmartParkingCluster
7
8 # Nombre de la base de datos
9 MONGO_DB=smartparking
10
11 # ===== FLASK CONFIGURATION =====
12
13 # Entorno de Flask (development o production)
14 FLASK_ENV=development
15
16 # Modo debug (True o False)
17 FLASK_DEBUG=True
18
19 # Clave secreta para sesiones (genera una clave aleatoria segura)
20 SECRET_KEY=tu_clave_secreta_super_segura_cambia_en_produccion
21
22 # ===== NOTAS =====
23 # 1. Obtén tu MONGO_URI desde MongoDB Atlas:
24 #   - Ve a tu cluster → Connect → Drivers
25 #   - Copia la connection string
26 #   - Reemplaza <password> con tu contraseña real
27 #
28 # 2. Para generar una SECRET_KEY segura en Python:
29 #   python -c "import secrets; print(secrets.token_hex(32))"
30 #
31 # 3. NUNCA subas este archivo a Git con credenciales reales
32 #   Añade .env a tu .gitignore
33

```

Figura E.2: Contenido del archivo de variables de entorno (`'.env.template'`).

PROBLEMAS SALIDA CONSOLA DE DEPURACIÓN TERMINAL PUERTOS POSTMAN CONSOLE

(venv) PS C:\Users\amp2\Documents\FormacionProfesional\CFGS\IABD\BDA\Proyecto UD1\SmartParking Flow> python app.py
2025-10-26 12:36:24,003 - __main__ - INFO - V Conectado exitosamente a MongoDB Atlas - Base de datos: smartparking
2025-10-26 12:36:24,006 - __main__ - INFO -
2025-10-26 12:36:24,006 - __main__ - INFO - Iniciando SmartParking Flask Application
2025-10-26 12:36:24,006 - __main__ - INFO -
2025-10-26 12:36:24,006 - __main__ - INFO - Base de datos: smartparking
2025-10-26 12:36:24,007 - __main__ - INFO - Servidor: http://localhost:5000
2025-10-26 12:36:24,007 - __main__ - INFO -
* Serving Flask app 'app'
* Debug mode: on
2025-10-26 12:36:24,032 - werkzeug - INFO - WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
* Running on http://localhost:5000

Figura E.3: Ejecución del servidor Flask ('`python app.py`').

SmartParking Flow — Monitorización Inteligente

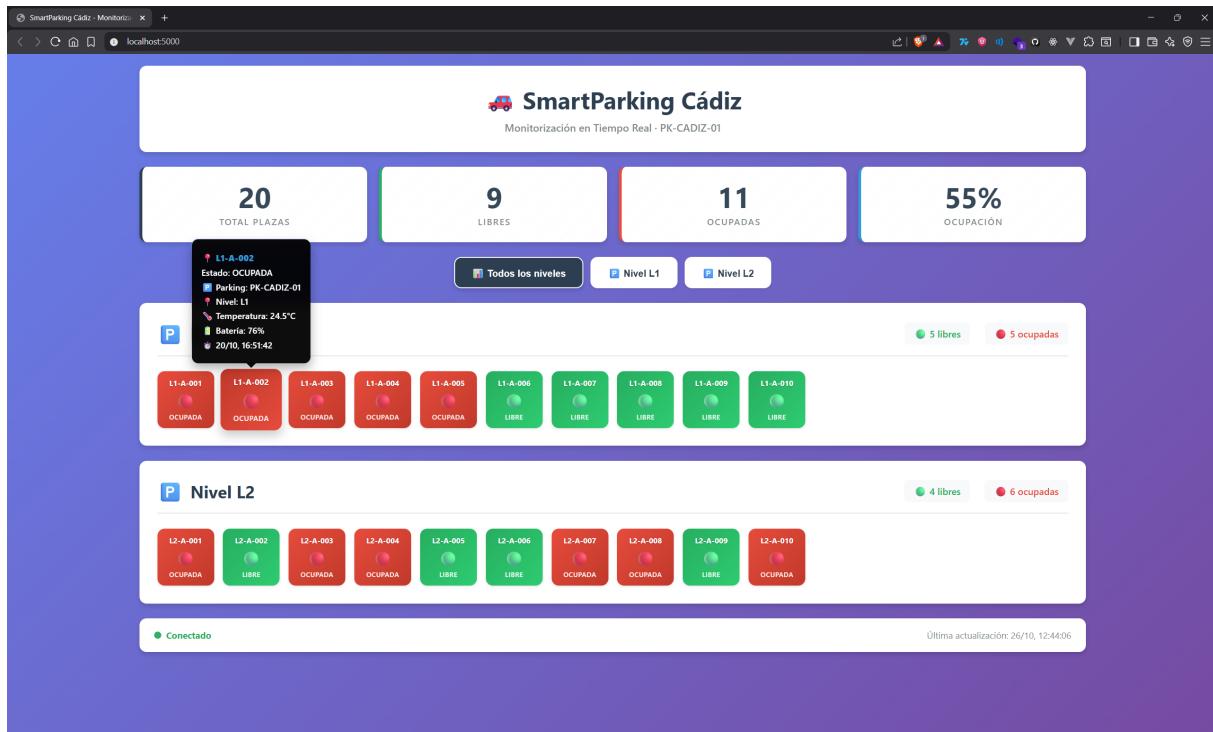


Figura E.4: Interfaz web de SmartParking mostrando el estado en tiempo real.

Apéndice F

Anexo F: Consultas de Análisis en Dremio

Instalación, configuración y conexión de Dremio a la fuente de datos de MongoDB Atlas para la ejecución de consultas SQL de análisis descriptivo.

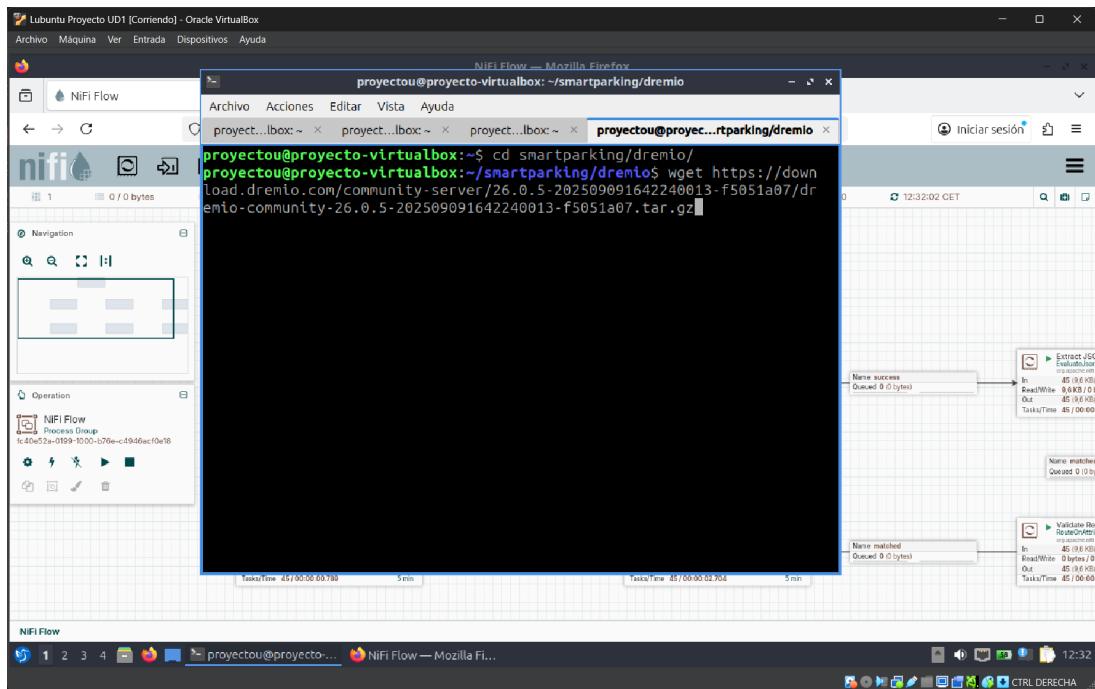


Figura F.1: Descarga de Dremio Community ('wget').

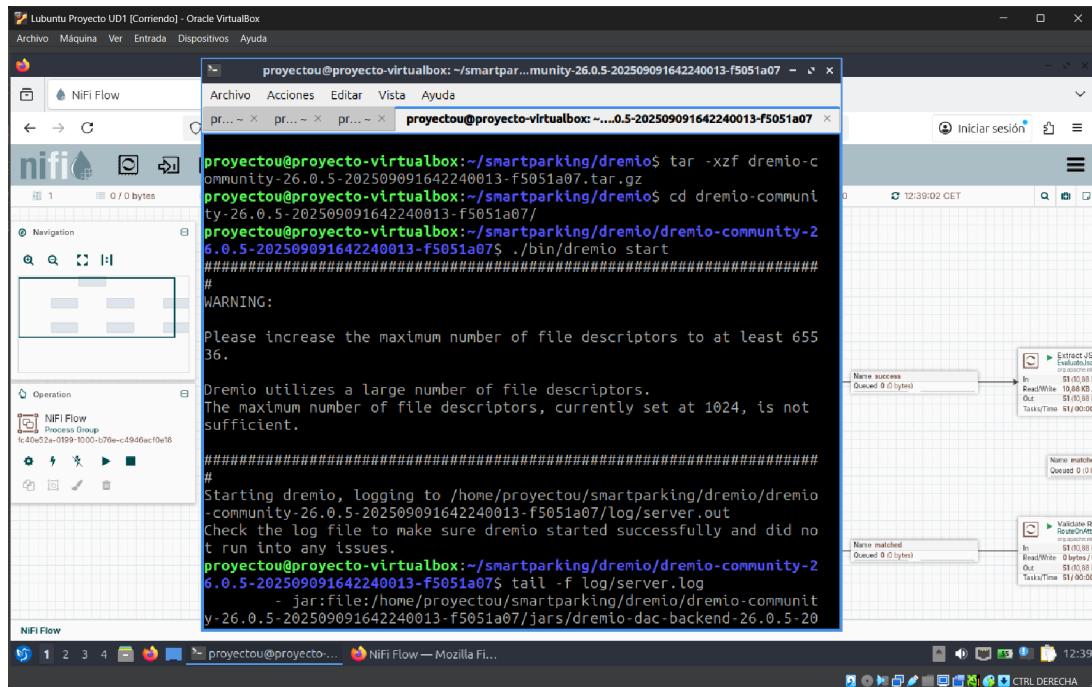


Figura F.2: Descompresión, inicio del servicio ('dremio start') y comprobación de logs.

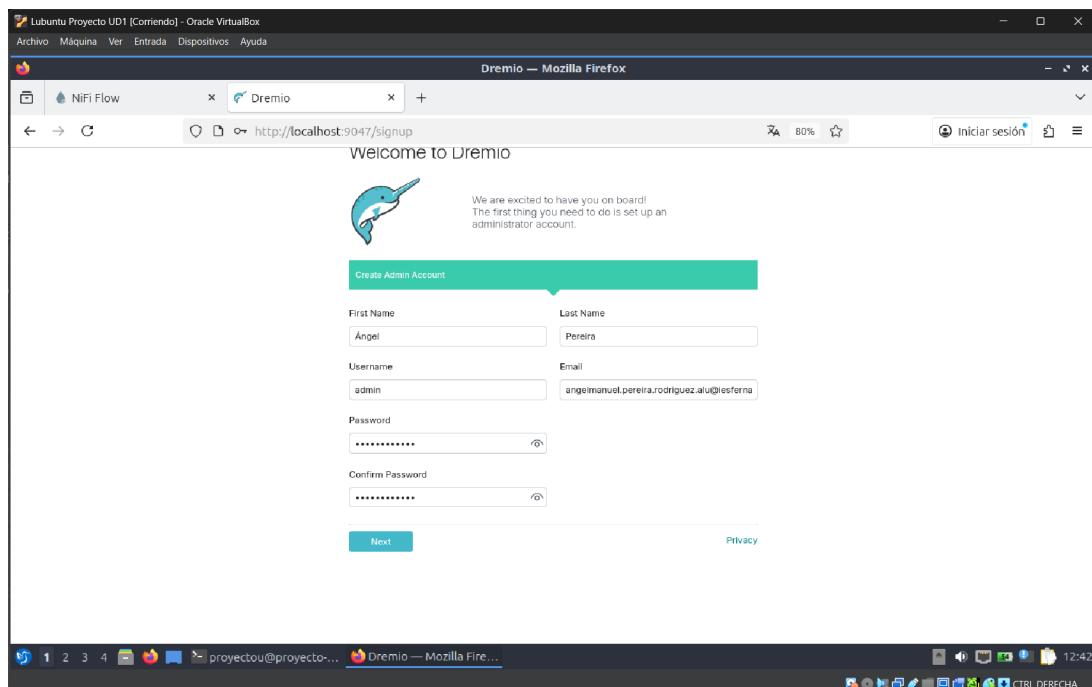


Figura F.3: Creación de la cuenta de administrador en la interfaz web de Dremio.

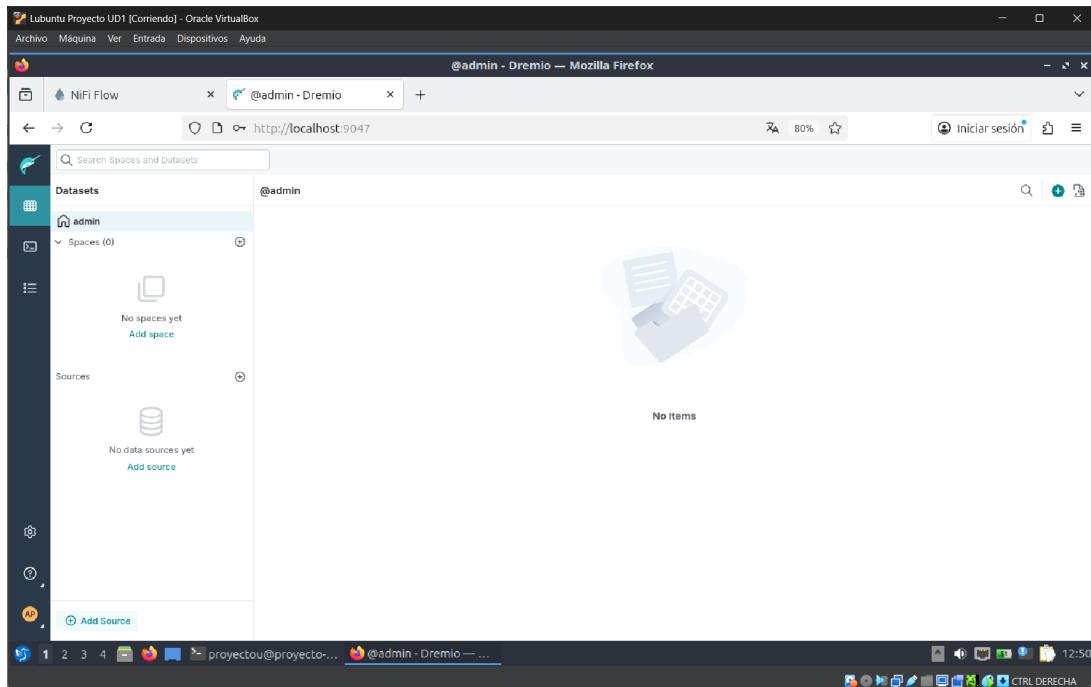


Figura F.4: Panel principal de Dremio (Datasets).

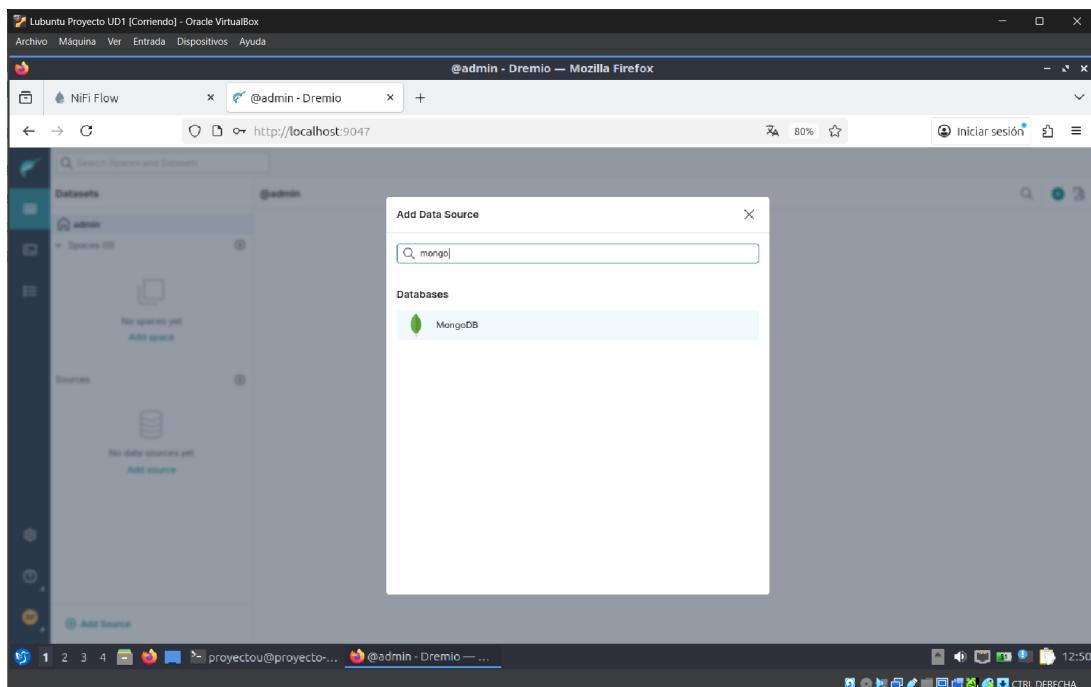


Figura F.5: Añadiendo una nueva fuente de datos (Add Data Source): MongoDB.

SmartParking Flow — Monitorización Inteligente

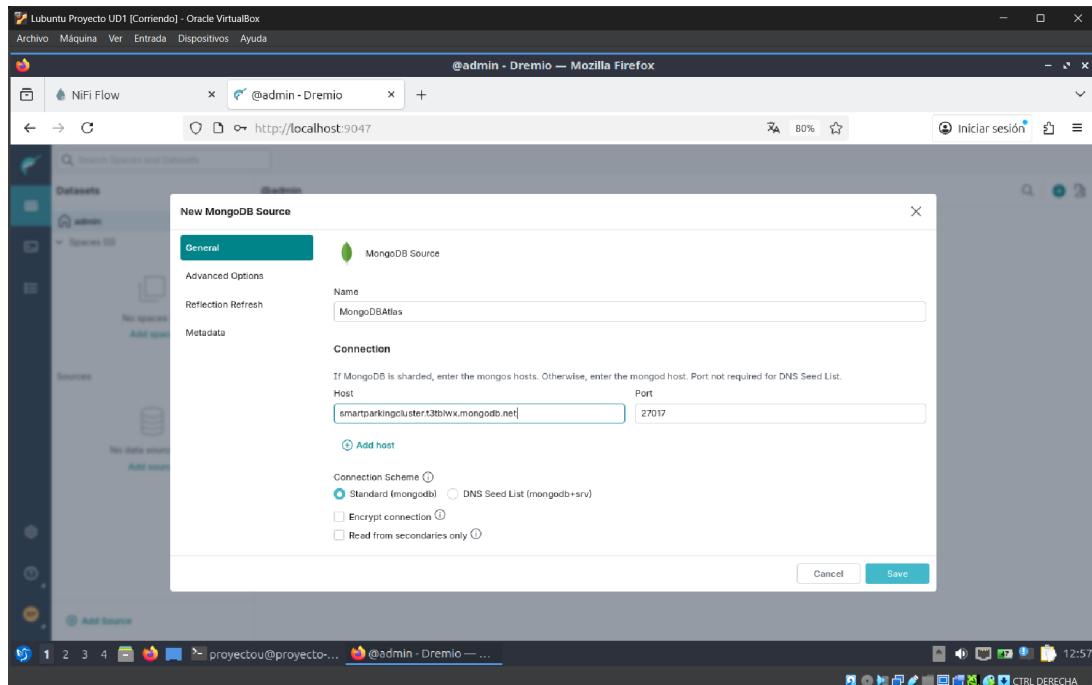


Figura F.6: Configuración de la fuente "MongoDB Source": Host y Puerto.

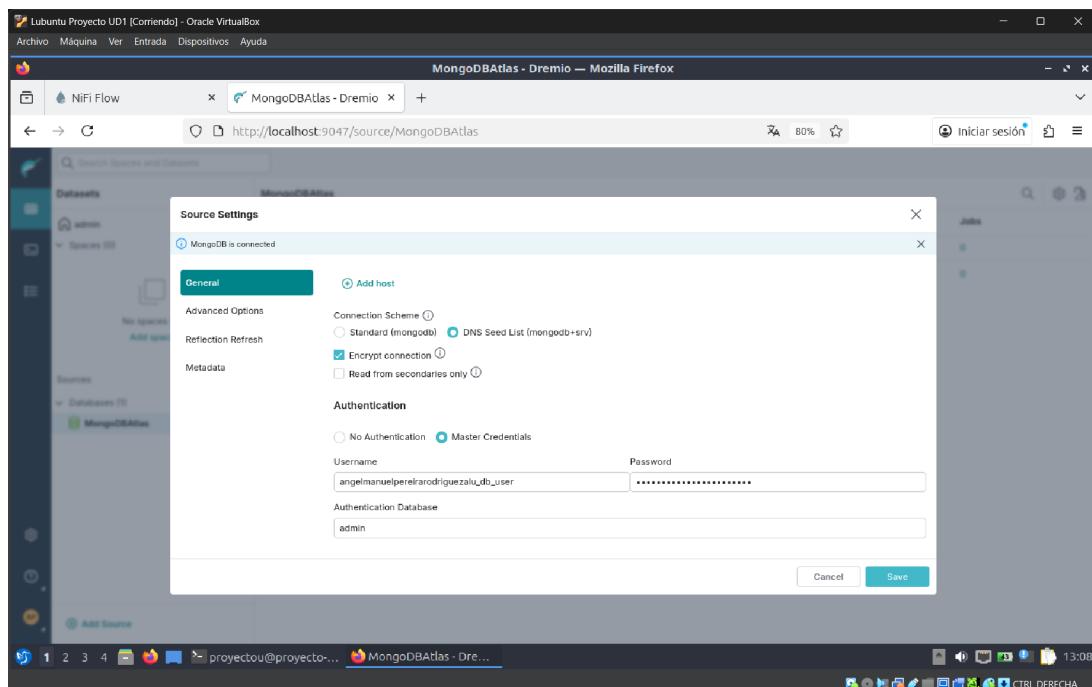


Figura F.7: Configuración de la autenticación de la fuente "MongoDB Source".

SmartParking Flow — Monitorización Inteligente

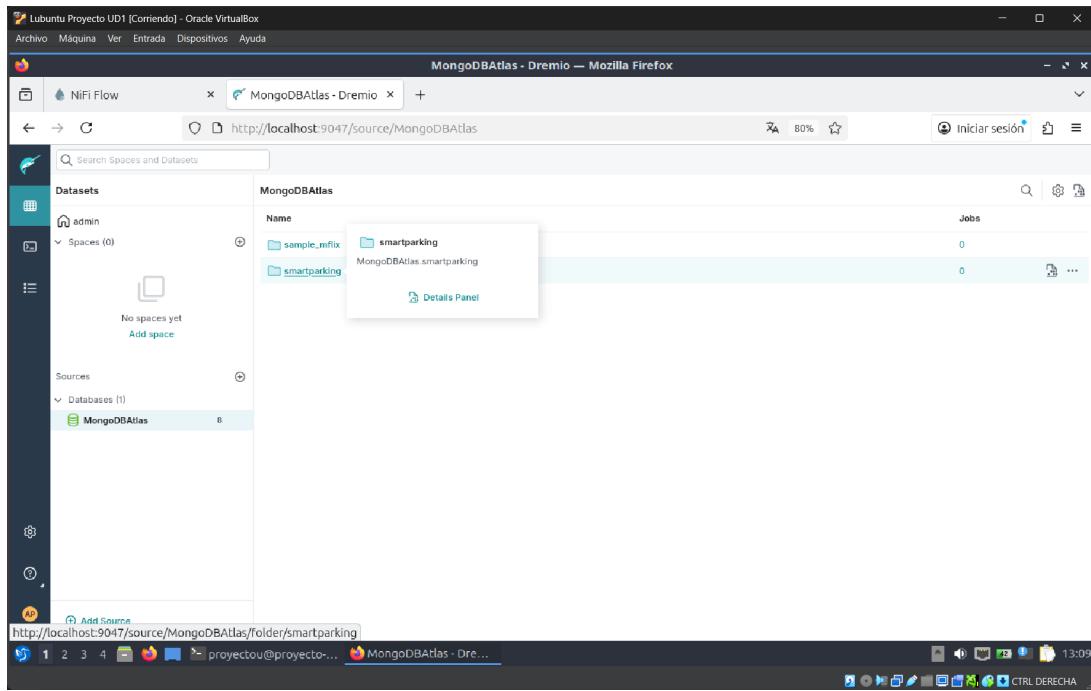


Figura F.8: Fuente de datos "MongoDBAtlas" conectada, mostrando la base de datos 'smartparking'.

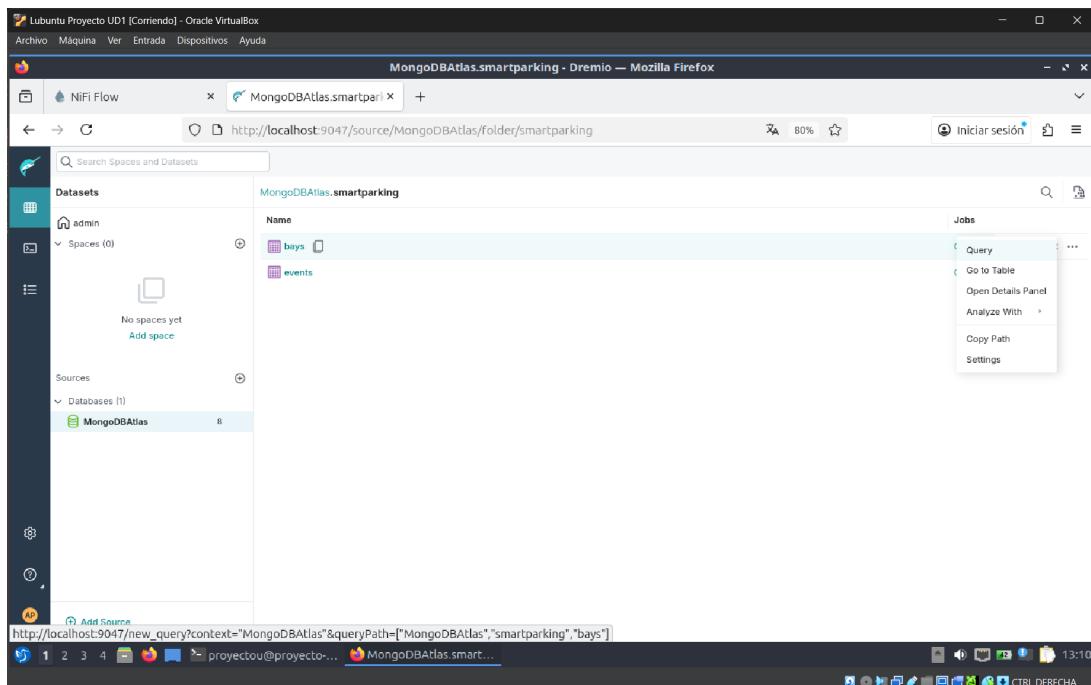
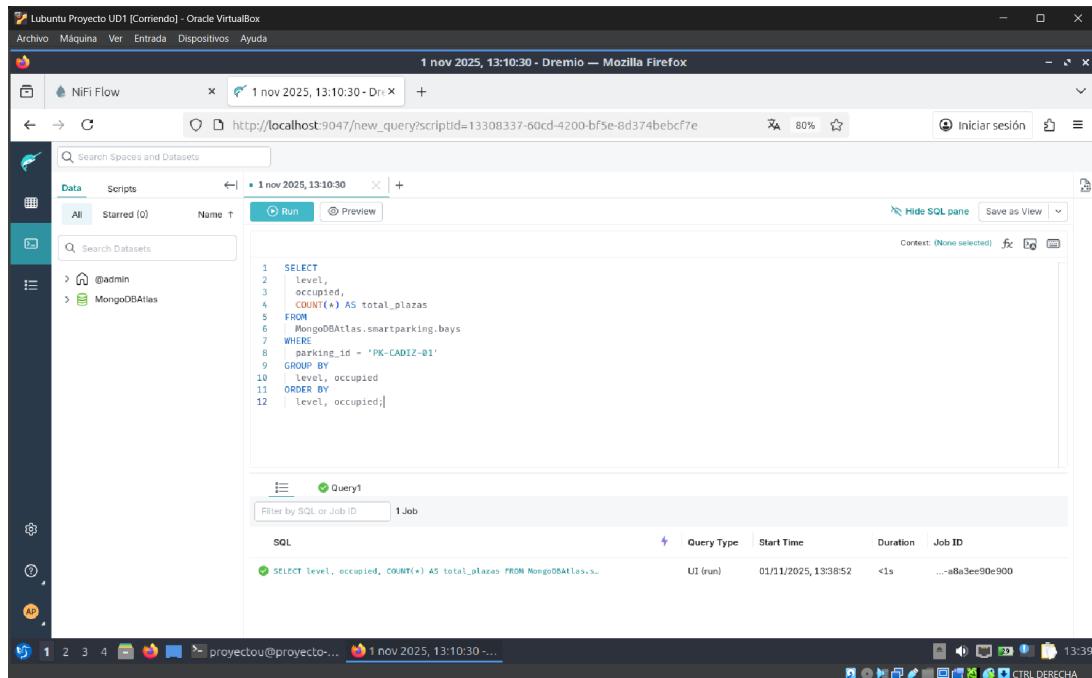


Figura F.9: Navegando en las colecciones 'bays' y 'events' dentro de Dremio.

SmartParking Flow — Monitorización Inteligente



The screenshot shows the Dremio SQL interface. The main area displays a SQL query:

```
1 SELECT
2     level,
3     occupied,
4     COUNT(*) AS total_plazas
5 FROM
6     MongoDBAtlas.smartparking.bays
7 WHERE
8     parking_id = 'PK-CADIZ-01'
9 GROUP BY
10    level, occupied
11 ORDER BY
12    level, occupied;
```

Below the query, a table titled "Job" shows the execution details:

SQL	Query Type	Start Time	Duration	Job ID
SELECT level, occupied, COUNT(*) AS total_plazas FROM MongoDBAtlas.s...	UI (run)	01/11/2025, 13:38:52	<1s	...a8a3ee90e900

Figura F.10: Consulta SQL 1: Ocupación actual por nivel.

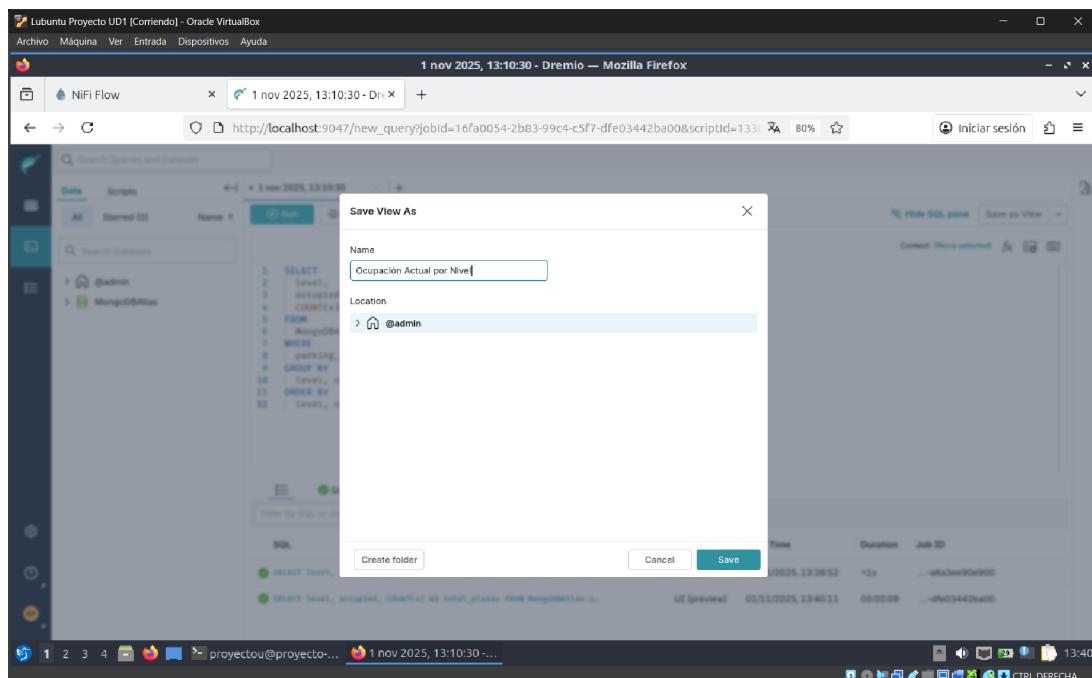
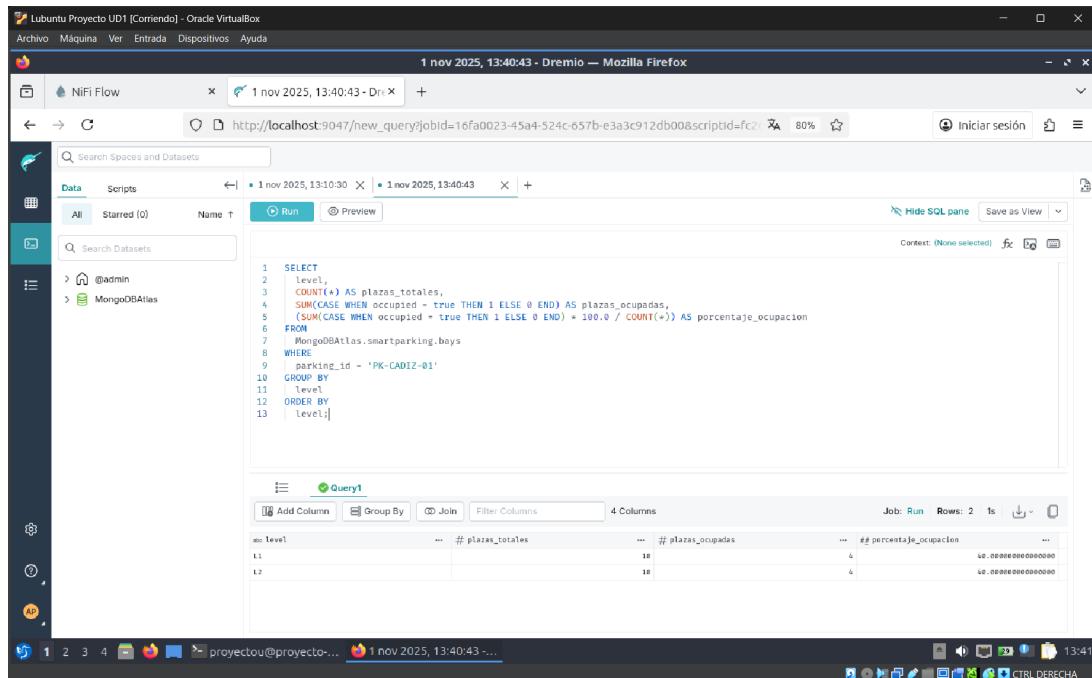


Figura F.11: Guardando la consulta (View) como ".Ocupacion Actual por Nivel".

SmartParking Flow — Monitorización Inteligente

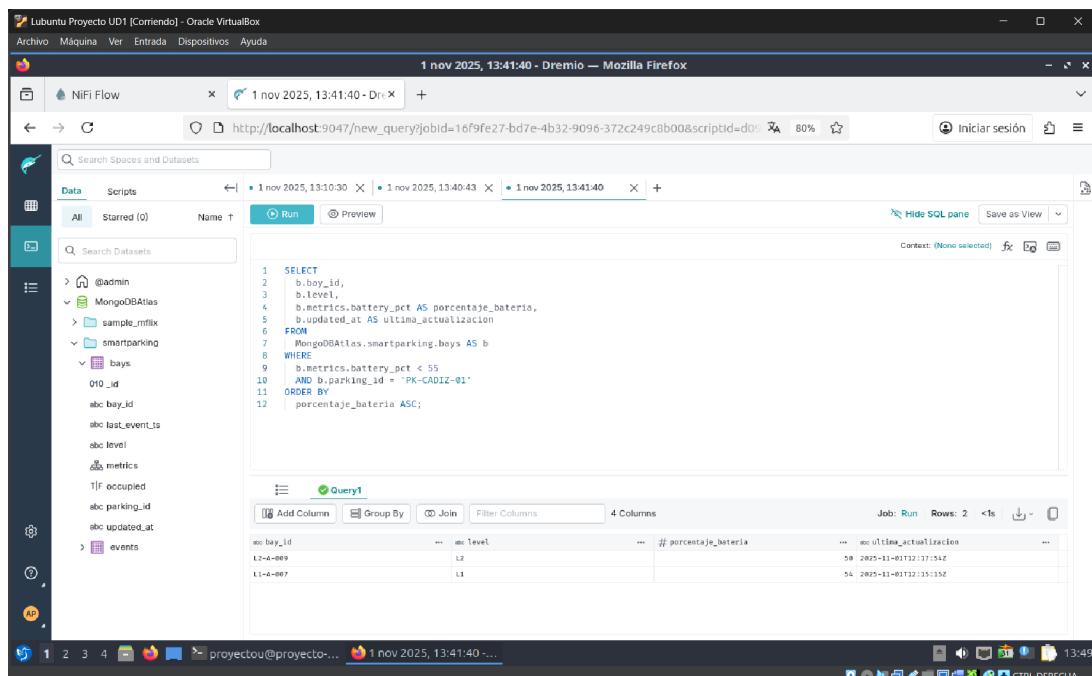


The screenshot shows the Dremio SQL interface within a Firefox browser window. The URL is http://localhost:9047/new_query?jobId=16fa0023-45a4-524c-657b-e3a3c912db00&scriptId=fca2e03f-0000-4000-8000-000000000000. The query is:

```
1 SELECT
2     level,
3     COUNT(*) AS plazas_totales,
4     SUM(CASE WHEN occupied = true THEN 1 ELSE 0 END) AS plazas_ocupadas,
5     (SUM(CASE WHEN occupied = true THEN 1 ELSE 0 END) * 100.0 / COUNT(*)) AS porcentaje_ocupacion
6   FROM
7     MongoDBAtlas.smartparking.bays
8   WHERE
9     parking_id = 'PK-CADIZ-01'
10  GROUP BY
11    level
12  ORDER BY
13    level;
```

The results table has four columns: 'level' (L1, L2), 'plazas_totales' (10, 10), 'plazas_ocupadas' (4, 4), and 'porcentaje_ocupacion' (40.0, 40.0). The job status is 'Run' with 2 rows.

Figura F.12: Consulta SQL 2: Porcentaje de ocupación por nivel.



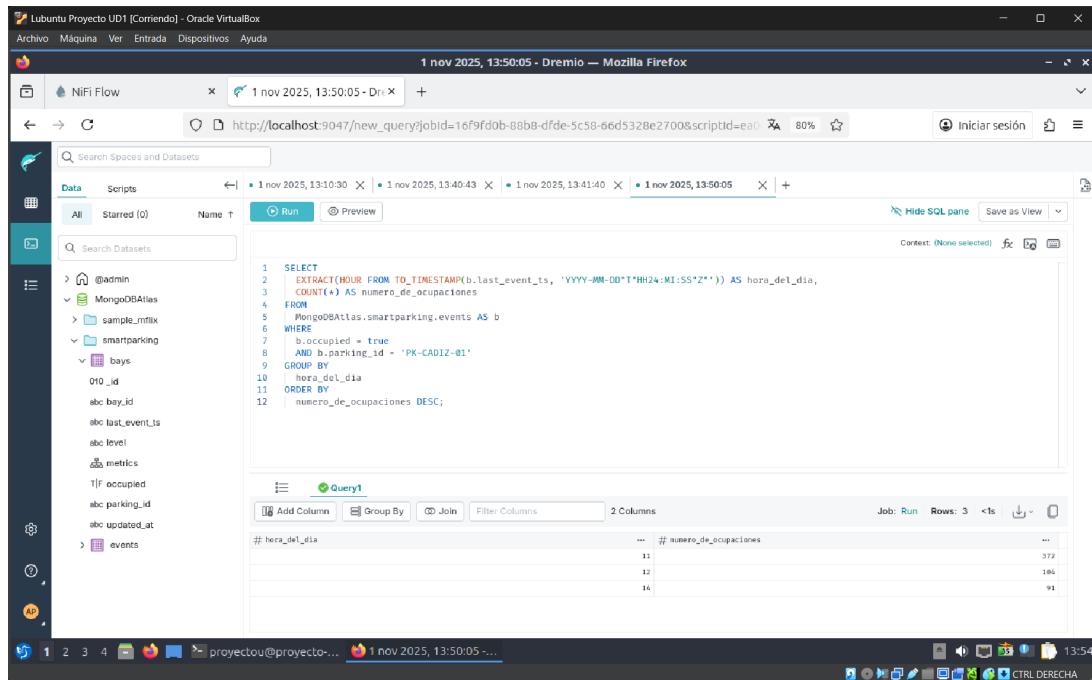
The screenshot shows the Dremio SQL interface within a Firefox browser window. The URL is http://localhost:9047/new_query?jobId=16f9fe27-bd7e-4b32-9096-372c249c8b00&scriptId=d05a2a00-0000-4000-8000-000000000000. The query is:

```
1 SELECT
2     b.bay_id,
3     b.level,
4     b.metrics.battery_pct AS porcentaje_bateria,
5     b.updated_at AS ultima_actualizacion
6   FROM
7     MongoDBAtlas.smartparking.bays AS b
8   WHERE
9     b.metrics.battery_pct < 55
10    AND b.parking_id = 'PK-CADIZ-01'
11  ORDER BY
12    porcentaje_bateria ASC;
```

The results table has four columns: 'bay_id' (L1-A-009, L1-A-007), 'level' (L2, L1), 'porcentaje_bateria' (50, 54), and 'ultima_actualizacion' (2025-11-01T12:17:54Z, 2025-11-01T12:15:15Z). The job status is 'Run' with 2 rows.

Figura F.13: Consulta SQL 3: Plazas con nivel bajo de batería (mantenimiento).

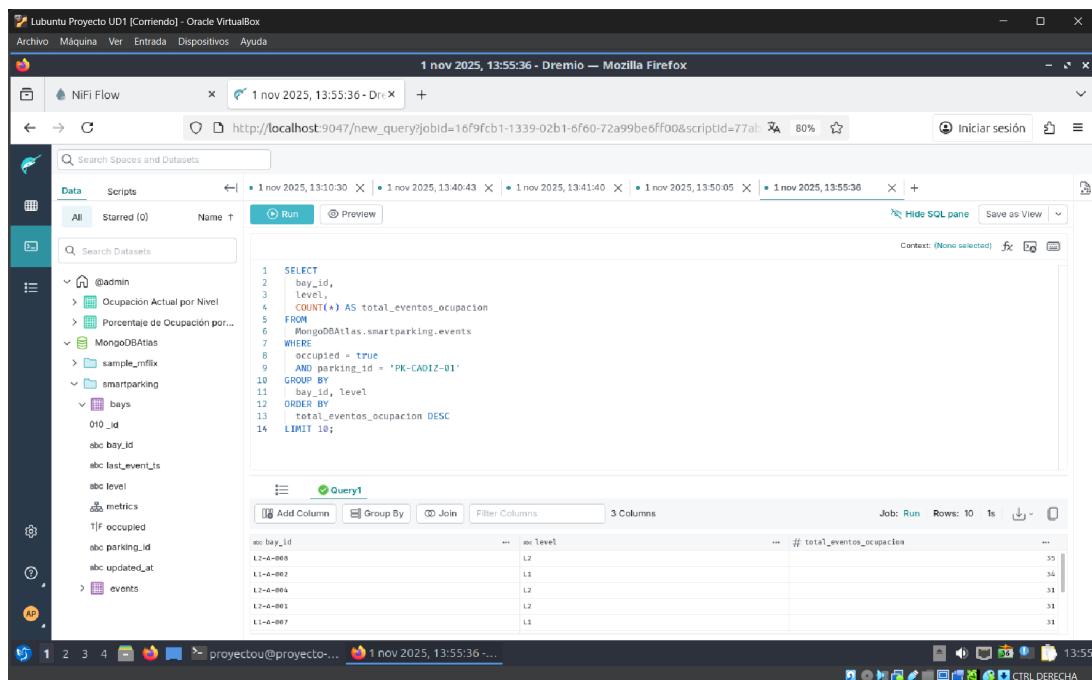
SmartParking Flow — Monitorización Inteligente



```
1 SELECT
2   EXTRACT(HOUR FROM TO_TIMESTAMP(b.last_event_ts, 'YYYY-MM-DD'T'HH24:MI:SS'Z')) AS hora_del_dia,
3   COUNT(*) AS numero_de_ocupaciones
4   FROM
5     MongoDBAtlas.smartparking.events AS b
6   WHERE
7     b.occupied = true
8     AND b.parking_id = 'PK-CADIZ-01'
9   GROUP BY
10    hora_del_dia
11   ORDER BY
12    numero_de_ocupaciones DESC;
```

hora_del_dia	numero_de_ocupaciones
11	572
12	186
14	91

Figura F.14: Consulta SQL 4: Horas del día con mayor número de eventos de ocupación.



```
1 SELECT
2   bay_id,
3   level,
4   COUNT(*) AS total_eventos_ocupacion
5   FROM
6     MongoDBAtlas.smartparking.events
7   WHERE
8     occupied = true
9     AND parking_id = 'PK-CADIZ-01'
10  GROUP BY
11    bay_id, level
12  ORDER BY
13    total_eventos_ocupacion DESC
14  LIMIT 10;
```

bay_id	level	total_eventos_ocupacion
L2-A-008	L2	35
L1-A-002	L1	34
L2-A-004	L2	33
L2-A-001	L2	33
L1-A-007	L1	31
L1-A-003	L1	31
L1-A-005	L1	31
L1-A-006	L1	31
L1-A-009	L1	31
L1-A-010	L1	31

Figura F.15: Consulta SQL 5: Plazas más ocupadas (Top 10).

```

1 SELECT
2     bay_id,
3     level,
4     COUNT(*) AS total_eventos_ocupacion
5 FROM
6     MongoDBAtlas.smartparking.events
7     WHERE
8     occupied = true
9     AND parking_id = 'PK-CADIZ-01'
10    GROUP BY
11        bay_id, level
12    ORDER BY
13        total_eventos_ocupacion DESC
14    LIMIT 10;

```

bay_id	level	total_eventos_ocupacion
L1-A-001	L2	35
L1-A-002	L1	34
L1-A-003	L2	31
L1-A-004	L2	31
L1-A-005	L2	31
L1-A-006	L2	31
L1-A-007	L1	31
L1-A-008	L1	31

Figura F.16: Resultado de la consulta SQL 5, mostrando las plazas con más eventos.