

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
5. Fase de Implementación	12
5.1. Entorno de Desarrollo (Máquina Virtual)	12
5.2. Almacenamiento de Datos (MongoDB Atlas)	12
5.3. Broker de Eventos (Apache Kafka)	13
5.4. Orquestación del Flujo (Apache NiFi)	13
5.5. Aplicación Web y API REST (Flask)	14
5.5.1. Backend (<code>app.py</code>)	14
5.5.2. Frontend (<code>index.html</code>)	14
5.6. Capa de Análisis (Dremio)	15

6. Fase de Pruebas	16
6.1. Pruebas de Componentes e Integración	16
6.2. Validación de Requisitos Funcionales (RF)	17
6.3. Validación de Requisitos No Funcionales (RNF)	18
6.4. Conclusión de las Pruebas	18
7. Conclusiones y Líneas Futuras	19
7.1. Conclusiones	19
7.2. Líneas Futuras	20
Bibliografía	21
8. Diario de Trabajo (Anexo)	22
A. Anexo A: Configuración de la Máquina Virtual (Lubuntu)	24
B. Anexo B: Configuración de MongoDB Atlas	41
C. Anexo C: Configuración de Apache Kafka	67
D. Anexo D: Configuración del Flujo de NiFi	83
E. Anexo E: Aplicación Flask (API y Visualización)	130
E.1. Código de la Aplicación (app.py)	130
E.2. Plantilla de la Interfaz (index.html)	139
E.3. Despliegue y Funcionamiento	155
F. Anexo F: Consultas de Análisis en Dremio	157

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.	24
A.2. Menú de arranque (GRUB) de la ISO de Lubuntu.	25
A.3. Inicio del instalador de Lubuntu.	25
A.4. Selección de 'Install Lubuntu'.	26
A.5. Selección de idioma (Español).	26
A.6. Selección de ubicación (Madrid).	27
A.7. Selección de teclado (Spanish).	27
A.8. Selección de tipo de instalación (Normal).	28
A.9. Configuración de particiones (Borrar disco).	28
A.10. Creación del usuario y nombre de equipo.	29
A.11. Resumen de la instalación.	29
A.12. Confirmación de inicio de instalación.	30
A.13. Proceso de instalación.	30
A.14. Instalación finalizada.	31
A.15. Extracción del disco de instalación virtual al reiniciar.	31
A.16. Escritorio de Lubuntu y apertura de QTerminal.	32
A.17. Actualización de paquetes ('sudo apt update').	32
A.18. Insertando imagen de CD de las 'Guest Additions' de VirtualBox.	33
A.19. Aviso de medio extraíble (Guest Additions).	33
A.20. Navegación al directorio de las Guest Additions.	34
A.21. Ejecución del script de instalación <code>VBoxLinuxAdditions.run</code>	34
A.22. Extracción del disco virtual de las Guest Additions.	35
A.23. Aviso de forzar desmontaje del disco óptico.	35
A.24. Habilitando portapapeles bidireccional en VirtualBox.	36
A.25. Terminal de Lubuntu post-reinicio.	36
A.26. Comprobación de la dirección IP de la MV ('ip addr') - 192.168.1.73.	37
A.27. Búsqueda de Símbolo del sistema (CMD) en el anfitrión (Windows).	38
A.28. Prueba de conectividad (ping) desde el anfitrión (Windows) a la MV.	39
A.29. Instalación de paquetes base en Lubuntu ('curl', 'wget', 'git', 'python3-pip').	39
A.30. Creación de directorios del proyecto y comprobación de versiones ('python3' y 'pip').	40
B.1. Búsqueda inicial de MongoDB Atlas.	41
B.2. Formulario de registro en MongoDB Atlas.	42
B.3. Paso de verificación de correo electrónico.	42

B.4.	Recepción del correo de verificación.	43
B.5.	Contenido del correo 'Verify Your MongoDB Email Address'	43
B.6.	Confirmación de email verificado.	44
B.7.	Configuración opcional de Multi-Factor Authentication (MFA).	44
B.8.	Recepción del código de verificación de MongoDB.	45
B.9.	Pantalla de configuración de seguridad de la cuenta.	45
B.10.	Pantalla de bienvenida a la nueva navegación de Atlas.	46
B.11.	Panel de 'All Projects' inicial.	46
B.12.	Renombrando el proyecto a 'SmartParking Flow'.	47
B.13.	Panel principal del proyecto 'SmartParking Flow'.	47
B.14.	Configuración del cluster gratuito (M0).	48
B.15.	Verificación Captcha para la creación del cluster.	48
B.16.	Creación del usuario de la base de datos.	49
B.17.	Confirmación de creación de usuario.	49
B.18.	Selección del método de conexión al cluster.	50
B.19.	Cargando datos de ejemplo (sample data).	50
B.20.	Vista del cluster tras finalizar la carga de datos de ejemplo.	51
B.21.	Configuración de 'IP Access List' (IP actual).	51
B.22.	Añadiendo una nueva entrada a 'IP Access List'.	52
B.23.	Añadiendo '0.0.0.0/0' (Allow Access From Anywhere).	52
B.24.	'IP Access List' actualizada con acceso global.	53
B.25.	Vista del cluster con datos de ejemplo cargados (74.12 MB).	53
B.26.	Explorador de datos (Data Explorer) viendo 'sample_mflix.comments'.	54
B.27.	Creación de la base de datos 'smartparking' y la colección 'events'.	54
B.28.	Vista de la colección 'events' vacía.	55
B.29.	Creación de la colección 'bays'.	55
B.30.	Vista de las colecciones 'bays' y 'events' creadas.	56
B.31.	Creación del índice para la colección 'bays'.	56
B.32.	Confirmación de creación de índice en 'bays' sobre 'bay_id'.	57
B.33.	Índice 'bay_id_1' creado y listado en la colección 'bays'.	57
B.34.	Vista de la pestaña 'Indexes' de la colección 'events'.	58
B.35.	Definición del índice compuesto para la colección 'events'.	58
B.36.	Confirmación de creación de índice en 'events'.	59
B.37.	Índice 'bay_event_ts_desc_idx' creado en la colección 'events'.	59
B.38.	Vista del cluster (Data Size: 116.16 MB).	60
B.39.	Opciones de conexión al cluster.	60
B.40.	Obteniendo la cadena de conexión (Connection String) para Python.	61
B.41.	Instalando 'pymongo[srv]' en el anfitrión (Windows).	61
B.42.	Script de prueba <code>pruebas.py</code> en VS Code (Windows).	62
B.43.	Resultado de la ejecución del script de prueba en Windows.	62
B.44.	Documento de prueba 'TEST-001' insertado en 'smartparking.bays'.	63
B.45.	Documento 'TEST-001' marcado para eliminación parte 1.	63
B.46.	Documento 'TEST-001' marcado para eliminación parte 2.	64
B.47.	Colección 'smartparking.bays' vacía tras borrado.	64
B.48.	Instalando 'python3-venv' en la MV de Lubuntu.	65
B.49.	Instalando 'pymongo[srv]' en el entorno virtual de Lubuntu.	65

B.50.Editando el script de prueba <code>pruebas.py</code> en Lubuntu (nano).	66
B.51.Resultado de la ejecución del script de prueba en Lubuntu.	66
C.1. Descarga de Apache Kafka ('wget').	67
C.2. Descompresión del archivo ('tar -xzf') y renombrado de la carpeta.	68
C.3. Creación del directorio de datos ('/opt/kafka-data') y edición de 'server.properties'.	68
C.4. Configuración de 'server.properties': 'node.id' y 'controller.quorum.bootstrap.servers'.	69
C.5. Configuración de 'server.properties': 'listeners' y 'advertised.listeners'.	69
C.6. Configuración de 'server.properties': 'log.dirs'.	70
C.7. Configuración de 'server.properties': Políticas de retención de logs.	70
C.8. Instalación de Java (OpenJDK 11 y 21) en la MV.	71
C.9. Selección de la versión de Java (JDK 11) usando 'update-alternatives'.	71
C.10.Selección de la versión de 'javac' (JDK 11).	72
C.11.Editando el script 'kafka-run-class.sh'.	72
C.12.Añadiendo 'JAVA_HOME' al script 'kafka-run-class.sh'.	73
C.13.Formateo del directorio de almacenamiento de Kafka ('kafka-storage.sh for- mat').	73
C.14.Inicio del servidor Kafka ('kafka-server-start.sh').	74
C.15.Editando el archivo '/.bashrc' para añadir variables de entorno.	74
C.16.Añadiendo 'KAFKA_HOME' y actualizando el 'PATH' en '/.bashrc'.	75
C.17.Recargando la configuración de la terminal ('source /.bashrc').	75
C.18.Comando para crear un nuevo tópico de Kafka.	76
C.19.Creación del tópico <code>parking-events</code> y listado de tópicos.	76
C.20.Instalación de la librería 'kafka-python' con 'pip3'.	77
C.21.Código del simulador de sensores: 'ParkingSensorSimulator.__init__'.	77
C.22.Código del simulador de sensores: 'ParkingSensorSimulator.generate_event'.	78
C.23.Código del publicador de Kafka: 'KafkaPublisher'.	78
C.24.Código de la función 'run_simulation' (lógica principal del productor).	79
C.25.Código del 'argparse' para configurar el simulador desde la terminal.	80
C.26.Dando permisos de ejecución ('chmod +x') al script del simulador.	80
C.27.Ejecución del script simulador de sensores ('parking_simulator.py').	81
C.28.Comando para iniciar un consumidor de consola en formato JSON.	81
C.29.Consumidor de consola de Kafka mostrando los mensajes JSON entrantes.	82
D.1. Descarga del binario de Apache NiFi ('wget').	83
D.2. Descompresión del archivo ('unzip').	84
D.3. Accediendo al directorio de configuración ('conf').	84
D.4. Editando 'nifi.properties' parte 1.	85
D.5. Editando 'nifi.properties' parte 2.	85
D.6. Editando 'nifi.properties' parte 3.	86
D.7. Editando 'nifi.properties' parte 4.	86
D.8. Accediendo al archivo 'bootstrap.conf'.	87
D.9. Editando 'bootstrap.conf': Configuración de memoria JVM (Xms2g, Xmx2g).	87
D.10.Editando '/.bashrc' para añadir 'NIFI_HOME'.	88
D.11.Añadiendo 'NIFI_HOME' y actualizando el 'PATH' en '/.bashrc'.	88
D.12.Accediendo al script de entorno 'nifi-env.sh'.	89
D.13.Estableciendo 'JAVA_HOME' en 'nifi-env.sh'.	89

D.14.Iniciando el servicio de NiFi ('nifi.sh start').	90
D.15.Comprobando los logs de inicio de NiFi ('tail -f nifi-app.log')	90
D.16.Accediendo a la interfaz web de NiFi (localhost:8080/nifi).	91
D.17.Arrastrando un nuevo procesador al canvas.	91
D.18.Buscando el procesador 'ConsumeKafka'.	92
D.19.Configuración (Settings) del procesador 'ConsumeKafka'.	92
D.20.Configuración (Properties) del procesador 'ConsumeKafka'.	93
D.21.Creación de un nuevo Controller Service: 'Kafka3ConnectionService'.	93
D.22.Asignando el 'Kafka3ConnectionService' al procesador.	94
D.23.Guardando cambios en el procesador.	94
D.24.Accediendo a la configuración del Controller Service ('Kafka3ConnectionService').	95
D.25.Configurando el 'Kafka3ConnectionService' (Bootstrap Servers: localhost:9092).	95
D.26.Controller Service 'Kafka3ConnectionService' en estado 'Disabled'.	96
D.27.Habilitando el 'Kafka3ConnectionService'.	96
D.28.Controller Service 'Kafka3ConnectionService' siendo habilitado.	97
D.29.Controller Service 'Kafka3ConnectionService' en estado 'Enabled'.	97
D.30.Configuración final del procesador 'ConsumeKafka' (Tópico: parking-events).	98
D.31.Procesador 'Consume Parking Sensors' en el canvas.	98
D.32.Añadiendo el procesador 'EvaluateJsonPath'.	99
D.33.Procesadores 'Consume Parking Sensors' y 'EvaluateJsonPath' en el canvas.	99
D.34.Creando conexión 'success' entre 'ConsumeKafka' y 'EvaluateJsonPath'. . .	100
D.35.Configuración (Properties) de 'EvaluateJsonPath': 'Destination: flowfile-attribute'.	100
D.36.Configuración (Properties) de 'EvaluateJsonPath': Extracción de atributos (battery, bay_id, ...).	101
D.37.Configuración (Settings) de 'EvaluateJsonPath': 'Name: Extract JSON Attributes'.	101
D.38.Configuración (Relationships) de 'EvaluateJsonPath': Terminar 'unmatched'. .	102
D.39.Añadiendo el procesador 'RouteOnAttribute'.	102
D.40.Configuración (Settings) de 'RouteOnAttribute': 'Name: Validate Required Fields'.	103
D.41.Configuración (Relationships) de 'RouteOnAttribute': Terminar 'unmatched'. .	103
D.42.Configuración (Properties) de 'RouteOnAttribute': 'Routing Strategy'. . .	104
D.43.Configuración (Properties) de 'RouteOnAttribute': Añadiendo propiedades de validación.	104
D.44.Flujo con los tres procesadores iniciales.	105
D.45.Creando conexión 'matched' entre 'EvaluateJsonPath' y 'RouteOnAttribute'. .	105
D.46.Accediendo a la configuración de Controller Services del Process Group. .	106
D.47.Vista de 'Kafka3ConnectionService' habilitado.	106
D.48.Añadiendo un nuevo Controller Service: 'JsonTreeReader'.	107
D.49.Controller Service 'JsonTreeReader' en estado 'Disabled'.	107
D.50.Habilitando el 'JsonTreeReader'.	108
D.51.Confirmación de habilitación del 'JsonTreeReader'.	108
D.52.Ambos Controller Services ('JsonTreeReader' y 'Kafka3ConnectionService') habilitados.	109
D.53.Añadiendo el procesador 'PutMongoRecord'.	109

D.54.Flujo con el procesador 'PutMongoRecord' añadido.	110
D.55.Creando conexión 'matched' entre 'RouteOnAttribute' y 'PutMongoRecord'.	110
D.56.Configuración (Settings) de 'PutMongoRecord': 'Name: Insert to Events Collection'.	111
D.57.Configuración (Properties) de 'PutMongoRecord' (vacía).	111
D.58.Creando un nuevo Controller Service: 'MongoDBControllerService'.	112
D.59.Configuración de 'PutMongoRecord' (Properties) - se usará 'MongoDB-ControllerService'.	112
D.60.Aviso de guardado de Controller Service.	113
D.61.Accediendo a la configuración del 'MongoDBControllerService'.	113
D.62.Editando 'MongoDBControllerService': Pegando la URI de conexión de Atlas.	114
D.63.Configuración (Properties) de 'MongoDBControllerService' con la URI.	114
D.64.Habilitando el 'MongoDBControllerService'.	115
D.65.Confirmación de habilitación del 'MongoDBControllerService'.	115
D.66.Habilitando el 'MongoDBControllerService'.	116
D.67.Todos los Controller Services habilitados.	116
D.68.Configuración de 'PutMongoRecord': 'Mongo Database Name: smartparking', 'Collection: events'.	117
D.69.Añadiendo el procesador 'ReplaceText'.	117
D.70.Flujo con el procesador 'ReplaceText' añadido.	118
D.71.Creando conexión 'success' entre 'Insert to Events Collection' y 'ReplaceText'.	118
D.72.Configuración (Settings) de 'ReplaceText': 'Name: Prepare Bay Document'.	119
D.73.Configuración (Relationships) de 'ReplaceText': Terminar 'failure'.	119
D.74.Configuración (Properties) de ReplaceText: Replacement Value (JSON del documento bays).	120
D.75.Configuración (Properties) de ReplaceText: Search Value: (?s)(*).	120
D.76.Añadiendo el procesador 'PutMongo' (para el upsert).	121
D.77.Flujo con el procesador 'PutMongo' añadido.	121
D.78.Creando conexión 'success' entre 'ReplaceText' y 'PutMongo'.	122
D.79.Configuración (Settings) de 'PutMongo': 'Name: Upsert to Bays Collection'.	122
D.80.Configuración (Relationships) de 'PutMongo': Terminar 'failure' y 'success'.	123
D.81.Configuración (Properties) de 'PutMongo': 'Collection: bays', 'Mode: upsert', 'Key: bay_id'.	123
D.82.Añadiendo el procesador 'LogAttribute' (para errores).	124
D.83.Flujo con el procesador 'LogAttribute' añadido.	124
D.84.Creando conexión 'failure' entre 'Upsert to Bays Collection' y 'Log Errors'.	125
D.85.Configuración (Settings) de 'LogAttribute': 'Name: Log Errors'.	125
D.86.Configuración (Relationships) de 'LogAttribute': Terminar 'success'.	126
D.87.Configuración (Properties) de 'LogAttribute': 'Log Level: error'.	126
D.88.Configuración (Relationships) de 'Insert to Events Collection': Terminar 'failure' y 'success'.	127
D.89.Vista general del flujo de NiFi (incompleta).	127
D.90.Vista general del flujo de datos completo en Apache NiFi.	128
D.91.Datos de la colección 'events' vistos desde MongoDB Atlas.	128
D.92.Datos de la colección 'bays' vistos desde MongoDB Atlas.	129

E.1. Instalación de dependencias de Python (<code>requirements.txt</code>)	155
E.2. Contenido del archivo de variables de entorno (' <code>.env.template</code> ')	155
E.3. Ejecución del servidor Flask (' <code>python app.py</code> ')	155
E.4. Interfaz web de SmartParking mostrando el estado en tiempo real.	156
F.1. Descarga de Dremio Community (' <code>wget</code> ')	157
F.2. Descompresión, inicio del servicio (' <code>dremio start</code> ') y comprobación de logs.	158
F.3. Creación de la cuenta de administrador en la interfaz web de Dremio.	158
F.4. Panel principal de Dremio (Datasets)	159
F.5. Añadiendo una nueva fuente de datos (Add Data Source): MongoDB.	159
F.6. Configuración de la fuente 'MongoDB Source': Host y Puerto.	160
F.7. Configuración de la autenticación de la fuente 'MongoDB Source'.	160
F.8. Fuente de datos 'MongoDBAtlas' conectada, mostrando la base de datos 'smartparking'	161
F.9. Navegando en las colecciones 'bays' y 'events' dentro de Dremio.	161
F.10. Consulta SQL 1: Ocupación actual por nivel.	162
F.11. Guardando la consulta (View) como 'Ocupacion Actual por Nivel'.	162
F.12. Consulta SQL 2: Porcentaje de ocupación por nivel.	163
F.13. Consulta SQL 3: Plazas con nivel bajo de batería (mantenimiento).	163
F.14. Consulta SQL 4: Horas del día con mayor número de eventos de ocupación.	164
F.15. Consulta SQL 5: Plazas más ocupadas (Top 10).	164
F.16. Muestra de todas las consultas guardadas (Views) en Dremio.	165

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. El detalle de las tareas acometidas y el tiempo invertido en cada fase se encuentra documentado en el Diario de Trabajo (Ver Anexo 8).

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 'ocupada') 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).

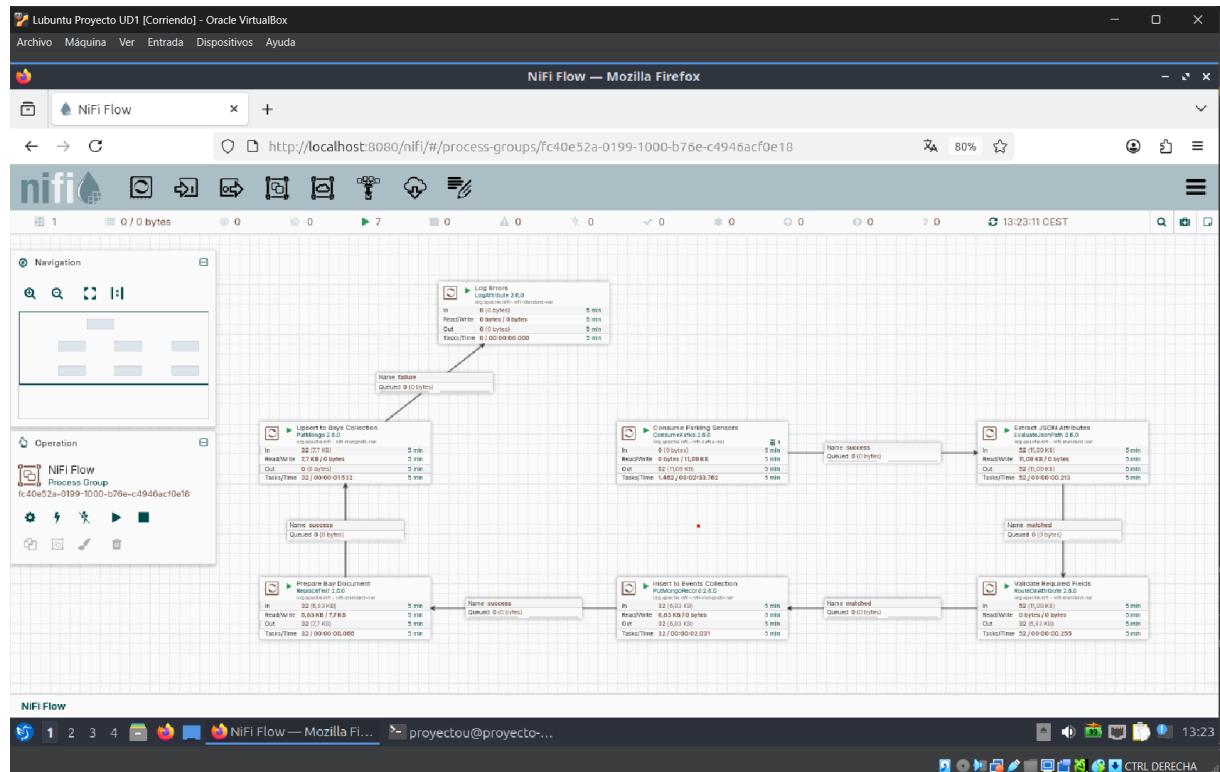


Figura 4.1: Diagrama de la arquitectura del sistema (Flujo de NiFi).

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`.

Capítulo 5

Fase de Implementación

En esta fase se detalla la ejecución práctica del diseño propuesto en el capítulo anterior. El proceso abarcó la configuración de la infraestructura base, la instalación y configuración de cada servicio del *stack* tecnológico, y el desarrollo de los scripts y aplicaciones necesarias.

El proceso completo se ha documentado exhaustivamente en los anexos (Anexo A a F), que proveen una guía visual paso a paso de cada hito de la implementación.

5.1 Entorno de Desarrollo (Máquina Virtual)

El primer paso consistió en preparar el entorno de desarrollo. Se configuró una máquina virtual (VM) en VirtualBox con **Lubuntu 24.04** como sistema operativo (Ver Anexo A, Figura A.1).

- Se realizó la instalación estándar del sistema (Figuras A.3 a A.14) y se aplicaron las actualizaciones de paquetes (`sudo apt update`, Figura A.17).
- Se instalaron las *VirtualBox Guest Additions* (Figura A.21) para mejorar la integración entre el sistema anfitrión y el huésped, habilitando el portapapeles bidireccional (Figura A.24).
- Se verificó la conectividad de red de la VM (obteniendo la IP 192.168.1.73, Figura A.26) y se comprobó la comunicación con el anfitrión mediante `ping` (Figura A.28).
- Finalmente, se instalaron las herramientas base necesarias para el proyecto, como `curl`, `wget`, `git` y `python3-pip` (Figura A.29).

El proceso completo se detalla en el Anexo A.

5.2 Almacenamiento de Datos (MongoDB Atlas)

Se optó por **MongoDB Atlas** como servicio gestionado de base de datos NoSQL para cumplir con los requisitos de escalabilidad y flexibilidad.

- Se creó una cuenta gratuita y un nuevo proyecto denominado 'SmartParking Flow' (Ver Anexo B, Figura B.12).

- Se desplegó un *cluster* gratuito (M0) (Figura B.14) y se configuró la seguridad: se creó un usuario de base de datos (Figura B.16) y se configuró la 'IP Access List' a 0.0.0.0/0 para permitir la conectividad (Figura B.23).
- Se creó la base de datos `smartparking` y las dos colecciones diseñadas: `events` (Figura B.27) y `bays` (Figura B.29).
- Se implementaron los índices diseñados: un índice único sobre `bay_id` en la colección `bays` (Figura B.33) y un índice compuesto descendente (`last_event_ts, bay_id`) en `events` (Figura B.37).

La conectividad se validó satisfactoriamente usando un script de Python con `pymongo` desde la MV (Ver Anexo B, Figura B.51).

5.3 Broker de Eventos (Apache Kafka)

Se instaló y configuró Apache Kafka en la MV de Lubuntu para actuar como el sistema nervioso central de la arquitectura.

- Se descargó el binario de Kafka (Ver Anexo C, Figura C.1) y se instaló la dependencia de Java (OpenJDK 11) (Figura C.8).
- Se modificó el archivo `server.properties` para establecer los `listeners` (Figura C.5) y la ruta de los logs (`log.dirs` en `/opt/kafka-data`, Figura C.6).
- Se formateó el almacenamiento (`kafka-storage.sh format`, Figura C.13) y se inició el servidor Kafka (Figura C.14).
- Se creó el tópico principal del proyecto: `parking-events` (Figura C.19).
- Paralelamente, se desarrolló el script **simulador de sensores** (`parking_simulator.py`) usando `kafka-python` (Figuras C.21 a C.25). Este script genera datos JSON aleatorios y los publica en el tópico.

Se validó la correcta producción (Figura C.27) y consumo de mensajes usando el simulador y el consumidor de consola de Kafka (Ver Anexo C, Figura C.29).

5.4 Orquestación del Flujo (Apache NiFi)

El motor ETL del proyecto se implementó con Apache NiFi, siguiendo el diseño de la arquitectura.

- Se instaló NiFi en la VM (Ver Anexo D, Figura D.1), configurando `nifi.properties` para que la interfaz web fuese accesible (`nifi.web.http.host=0.0.0.0`, Figura D.4). Se ajustó la memoria de la JVM a 2GB (Figura D.9).
- Se configuraron los **Controller Services** necesarios: `Kafka3ConnectionService` (Figura D.25), `JsonTreeReader` (Figura D.48) y `MongoDBControllerService` (con la URI de Atlas, Figura D.63).

- Se construyó el flujo de procesadores visual (Ver Figura D.90):
 1. **ConsumeKafkaRecord**: Suscrito al tópico `parking-events` (Figura D.30).
 2. **EvaluateJsonPath**: Para extraer los campos del JSON a atributos (Figura D.36).
 3. **RouteOnAttribute**: Para validar que los campos esenciales no estuvieran vacíos (Figura D.43).
 4. **PutMongoRecord (Rama Histórica)**: Inserta el FlowFile original en la colección `events` (Figura D.68).
 5. **ReplaceText (Rama Estado)**: Transforma el contenido del FlowFile para ajustarse al documento de estado (Figura D.74).
 6. **PutMongo (Rama Estado)**: Realiza una operación `upsert` en la colección `bays`, usando `bay_id` como clave (Figura D.81).

Tras iniciar el flujo, se verificó en MongoDB Atlas la correcta inserción en `events` (Figura D.91) y la actualización de estados en `bays` (Figura D.92).

5.5 Aplicación Web y API REST (Flask)

La capa de visualización se desarrolló con Flask, proporcionando tanto la API REST como la interfaz de usuario (Ver Anexo E).

5.5.1. Backend (app.py)

El archivo `app.py` (detallado en el Anexo E) implementa el servidor web.

- Se instalaron las dependencias de Python (Figura E.1).
- El script establece la conexión con MongoDB Atlas y define las rutas de la API.
- Define la ruta principal GET `/` que renderiza `index.html`.
- Expone GET `/api/bays` (para el estado) y GET `/api/stats` (para las estadísticas agregadas).

5.5.2. Frontend (index.html)

La plantilla `index.html` (detallada en el Anexo E) contiene el HTML, CSS y JavaScript.

- El **JavaScript** utiliza `fetch` para consumir los endpoints `/api/stats` y `/api/bays` al cargar.
- Genera dinámicamente la parrilla de plazas, asignando clases `.free` o `.occupied` según el estado.
- Utiliza `setInterval` para refrescar los datos automáticamente cada 20 segundos (RF-06).

El servidor se ejecutó con `python app.py` (Figura E.3) y se accedió a la interfaz web (`localhost:5000`), validando la visualización en tiempo real (Figura E.4).

5.6 Capa de Análisis (Dremio)

Finalmente, para cumplir con los requisitos de análisis (RF-08), se implementó la plataforma Dremio.

- Se descargó (Figura F.1) e inició el servicio Dremio (Ver Anexo F, Figura F.2).
- A través de la interfaz web, se añadió una nueva fuente de datos de tipo **MongoDB** (Figura F.5).
- Se configuró la conexión apuntando al *cluster* de MongoDB Atlas (Figuras F.6 y F.7).
- Una vez conectado, Dremio permitió navegar por las colecciones `bays` y `events` (Figuras F.8 y F.9).
- Se ejecutaron múltiples consultas SQL analíticas directamente sobre los datos NoSQL, como:
 - Ocupación actual por nivel (Figura F.10).
 - Listado de sensores con batería baja (Figura F.13).
 - Análisis de horas punta (Figura F.14).
 - Top 10 de plazas más utilizadas (Figura F.15).

Con la finalización de esta fase, todo el *pipeline* de datos, desde la generación simulada del sensor hasta la visualización y el análisis, quedó completamente funcional y automatizado.

Capítulo 6

Fase de Pruebas

Una vez completada la fase de implementación (Capítulo 5), se procede a la fase de pruebas. El objetivo de esta fase es verificar y validar que el sistema no solo es funcional, sino que cumple con todos los requisitos funcionales (RF) y no funcionales (RNF) definidos en la fase de análisis (Capítulo 3).

Las pruebas se han realizado de forma continua durante la implementación, y los anexos de este documento (Anexos A a F) sirven como evidencia principal de la validación de cada componente y del sistema en su conjunto.

6.1 Pruebas de Componentes e Integración

Se realizaron pruebas en cada etapa del *pipeline* para asegurar el correcto funcionamiento individual y la integración entre servicios.

- **Prueba de Conectividad de Red (MV):** Se validó que la máquina virtual Lubuntu tenía conectividad con el anfitrión (Anexo A, Figura A.28) y con internet, permitiendo la descarga de paquetes y la conexión a servicios cloud.
- **Prueba de Conexión a MongoDB Atlas:** Antes de integrar con NiFi, se ejecutó un script de Python (`pruebas.py`) desde la MV para verificar la conectividad, autenticación y permisos de escritura sobre el cluster de MongoDB Atlas. La prueba fue exitosa (Anexo B, Figura B.51).
- **Prueba del Broker (Kafka):** Se realizó una prueba unitaria del broker. Se ejecutó el script productor (`parking_simulator.py`) (Anexo C, Figura C.27) y simultáneamente se utilizó un consumidor de consola. Se verificó que los mensajes JSON se producían y consumían correctamente en el tópico `parking-events` (Anexo C, Figura C.29).
- **Prueba del Flujo ETL (NiFi):** Se probó el flujo de NiFi de forma incremental. Al iniciar el procesador `ConsumeKafkaRecord`, se observó cómo los FlowFiles eran procesados y enrutados por las dos ramas (histórica y de estado), validando la lógica de bifurcación (Anexo D, Figura D.90).
- **Prueba de Integración End-to-End:** La prueba de integración clave consistió en ejecutar todos los servicios simultáneamente:

1. El simulador de sensores (Anexo C) publica datos.
2. Kafka recibe los mensajes.
3. El flujo de NiFi (Anexo D) consume los mensajes, los procesa y los enruta.
4. Los datos aparecen correctamente en las colecciones `events` (Figura D.91) y `bays` (Figura D.92) en MongoDB Atlas.
5. El servidor Flask (Anexo E, Figura E.3) consulta la base de datos.
6. La interfaz web (Anexo E, Figura E.4) muestra los datos actualizados.
7. Dremio (Anexo F) puede consultar los datos persistidos (Figura F.10).

Esta prueba validó que el *pipeline* completo es funcional.

6.2 Validación de Requisitos Funcionales (RF)

Se verificó el cumplimiento de cada requisito funcional definido en el Capítulo 3:

- **RF-01 (Ingesta de eventos): Validado.** El procesador `ConsumeKafkaRecord` de NiFi está configurado y consume activamente del tópico `parking-events` (Anexo D, Figura D.30).
- **RF-02 (Almacenamiento histórico): Validado.** El procesador `PutMongoRecord` inserta cada evento en la colección `events`. La evidencia visual se encuentra en (Anexo D, Figura D.91), donde se observa el histórico de todos los mensajes recibidos.
- **RF-03 (Almacenamiento de estado actual): Validado.** El procesador `PutMongo` (Anexo D, Figura D.81) realiza la operación `upsert` sobre la colección `bays`. La evidencia se muestra en (Anexo D, Figura D.92), donde solo existe un documento por `bay_id`, reflejando el último estado conocido.
- **RF-04 (Visualización de estado): Validado.** La aplicación Flask, al ejecutarse (Anexo E, Figura E.3), sirve la plantilla `index.html` que renderiza el mapa visual del aparcamiento (Anexo E, Figura E.4).
- **RF-05 (Codificación por color): Validado.** En la interfaz web (Anexo E, Figura E.4), las plazas libres (`occupied: false`) se muestran en color verde y las ocupadas (`occupied: true`) en color rojo, según definen las clases CSS `.bay.free` y `.bay.occupied` en `index.html` (Anexo E).
- **RF-06 (Actualización automática): Validado.** El código JavaScript de la plantilla `index.html` (Anexo E) contiene la función `setInterval` que ejecuta la recarga de datos (`loadParkingData`) cada 20 segundos. Se verificó visualmente que la web refresca el estado sin intervención del usuario.
- **RF-07 (Panel de estadísticas): Validado.** La parte superior de la interfaz web (Anexo E, Figura E.4) muestra correctamente las tarjetas de 'Total Plazas', 'Libres', 'Ocupadas' y 'Ocupación', alimentadas por el endpoint `/api/stats`.

- **RF-08 (Análisis de datos): Validado.** La plataforma Dremio se conectó exitosamente a MongoDB Atlas (Anexo F, Figura F.8) y permitió la ejecución de múltiples consultas SQL sobre los datos NoSQL (Figuras F.10 a F.16), cumpliendo el requisito de análisis.

6.3 Validación de Requisitos No Funcionales (RNF)

- **RNF-01 (Rendimiento - Latencia): Validado.** Mediante observación directa, se comprobó que el tiempo transcurrido desde que el simulador envía un evento hasta que este se ve reflejado en la interfaz web (Anexo E, Figura E.4) es de pocos segundos, cumpliendo con la expectativa de 'casi tiempo real'.
- **RNF-02 (Fiabilidad): Validado.** La arquitectura basada en Kafka (Anexo C) como *buffer* de mensajes y las colas internas de NiFi (Anexo D) garantizan que, aunque el consumidor (NiFi) o la base de datos (MongoDB) se caigan temporalmente, los mensajes de los sensores no se pierden y se procesan al re establecerse el servicio.
- **RNF-03 (Escalabilidad): Validado.** El uso de tecnologías inherentemente escalables horizontalmente como Kafka, NiFi y MongoDB Atlas (Anexo B) satisface este requisito a nivel de diseño.
- **RNF-04 (Mantenibilidad): Validado.** El flujo de datos ETL se gestiona mediante una interfaz gráfica (Anexo D, Figura D.90), lo que permite una monitorización visual y una modificación sencilla del flujo (añadir/quitar procesadores) sin necesidad de reprogramar.
- **RNF-05 (Flexibilidad): Validado.** El uso de JSON como formato de datos (Anexo C, Figura C.29) y MongoDB como base de datos NoSQL (Anexo B) permite añadir nuevos campos a los mensajes (ej. 'nivel_ruido') sin necesidad de alterar la estructura de la base de datos (Schema-on-Read).

6.4 Conclusión de las Pruebas

Tras la ejecución de las pruebas de componentes, integración y validación de requisitos, se concluye que el sistema **SmartParking Flow** es plenamente funcional y cumple con todos los objetivos y requisitos establecidos en la fase de análisis. La evidencia recopilada en los anexos (A-F) respalda la correcta implementación de cada parte del *pipeline*.

Capítulo 7

Conclusiones y Líneas Futuras

Al finalizar la implementación y las pruebas del proyecto **SmartParking Flow**, se pueden extraer las siguientes conclusiones sobre los resultados obtenidos y las posibles vías de expansión del sistema.

7.1 Conclusiones

El proyecto ha culminado con la implementación exitosa de un *pipeline* de datos robusto y funcional para la monitorización inteligente de aparcamientos, cumpliendo con todos los objetivos principales establecidos.

- **Integración Tecnológica Exitosa:** Se ha validado la correcta integración del *stack* tecnológico seleccionado. **Apache Kafka** ha demostrado ser un *buffer* de eventos fiable; **Apache NiFi** ha orquestado el flujo ETL de forma visual y mantenible; **MongoDB Atlas** ha proporcionado una base de datos NoSQL flexible y escalable; **Flask** ha servido como un *backend* ligero y eficiente para la API REST; y **Dremio** ha facilitado el análisis de datos NoSQL mediante SQL.
- **Cumplimiento de Requisitos:** Como se demostró en la Fase de Pruebas (Capítulo 6), el sistema satisface todos los requisitos funcionales (RF) y no funcionales (RNF) definidos. El sistema ingiere datos (RF-01), mantiene un histórico (RF-02) y un estado actual (RF-03), proporciona una visualización en tiempo real (RF-04, RF-05) con actualización automática (RF-06), muestra estadísticas (RF-07) y permite el análisis de datos (RF-08).
- **Arquitectura Robusta (Doble Ingesta):** El diseño de la base de datos (Anexo B) y del flujo de NiFi (Anexo D) para separar el *histórico* (`events`) del *estado actual* (`bays`) ha sido un éxito. Esta arquitectura optimiza tanto las consultas de estado en tiempo real (requeridas por la web de Flask) como el análisis histórico a gran escala (requerido por Dremio), sin que un caso de uso penalice al otro.
- **Alta Mantenibilidad y Flexibilidad:** El uso de Apache NiFi (Anexo D, Figura D.90) para el ETL gráfico (RNF-04) y MongoDB para el almacenamiento (RNF-05) dota al proyecto de una gran facilidad de mantenimiento y evolución. Añadir

nuevas métricas de sensores o modificar la lógica de enrutamiento no requiere una reingeniería compleja.

- **Viabilidad del Proyecto:** Se ha demostrado la viabilidad de implementar una solución avanzada de *Smart City* utilizando herramientas *open-source* (con la excepción del *hosting* de MongoDB Atlas), lo que permite una solución escalable y potente con un coste controlado.

7.2 Líneas Futuras

El sistema actual sienta las bases para numerosas mejoras y expansiones. A continuación, se detallan las líneas futuras más relevantes:

- **Integración de Sensores Reales (IoT):** El paso más evidente es reemplazar el simulador de Python (`sensores.py`, Anexo C) por hardware real. Esto implicaría desplegar sensores físicos (ej. magnéticos, ultrasónicos) y establecer la capa de comunicación (ej. LoRaWAN, NB-IoT) que publique los datos en el tópico de Kafka.
- **Sistema de Alertas en Tiempo Real:** Expandir el flujo de NiFi o la aplicación Flask para incluir un sistema de alertas. Por ejemplo, si un sensor reporta un nivel de batería por debajo del 15 % (detectado en Dremio, Anexo F, Figura F.13), el sistema podría enviar automáticamente un correo electrónico o un mensaje de Telegram al equipo de mantenimiento.
- **Análisis Predictivo (Machine Learning):** Utilizar el histórico de datos almacenado en la colección `events` (Anexo D, Figura D.91) para entrenar modelos de *Machine Learning*. Estos modelos podrían predecir la ocupación futura por franjas horarias, facilitando la implementación de sistemas de tarificación dinámica (*dynamic pricing*).
- **Aplicación Móvil para Usuarios:** Desarrollar una aplicación móvil nativa (iOS/Android) que consuma la API REST de Flask. Esta aplicación no solo mostraría la disponibilidad, sino que podría guiar al usuario a la plaza libre más cercana dentro del aparcamiento.
- **Integración con Sistemas de Pago:** Conectar el sistema a una pasarela de pago. Al detectar un evento 'ocupado' (`occupied: true`) y 'libre' (`occupied: false`) para la misma `bay_id`, el sistema podría calcular el tiempo de estancia y generar el cobro automáticamente.
- **Escalado Multi-Parking:** Extender la arquitectura actual para dar soporte a múltiples aparcamientos dentro de la iniciativa *SmartCity Cádiz*. Esto implicaría usar el campo `parking_id` (definido en el modelo de datos) para filtrar y agregar datos a nivel de ciudad, en lugar de un único `parking`.
- **Despliegue en Entorno de Producción:** Migrar la configuración de desarrollo (ej. permisos de IP 0.0.0.0/0, servicios en `localhost`) a un entorno de producción seguro, aplicando cifrado SSL/TLS en todas las comunicaciones, redes privadas virtuales (VPC) y roles de seguridad granulares.

Bibliografía

- [1] Documentación oficial de Apache Kafka. *Apache Software Foundation*. Disponible en: <https://kafka.apache.org/documentation/>
- [2] Documentación oficial de Apache NiFi. *Apache Software Foundation*. Disponible en: <https://nifi.apache.org/docs/nifi-docs/>
- [3] Documentación oficial de MongoDB Atlas. *MongoDB, Inc.* Disponible en: <https://www.mongodb.com/docs/atlas/>
- [4] Documentación oficial de Flask. *Pallets*. Disponible en: <https://flask.palletsprojects.com/>
- [5] Documentación oficial de Dremio. *Dremio Corporation*. Disponible en: <https://docs.dremio.com/>

Capítulo 8

Diario de Trabajo (Anexo)

A continuación, se detalla la distribución del trabajo por jornadas, describiendo las tareas acometidas y el tiempo invertido en cada fase del proyecto.

Viernes, 17 de octubre de 2025 (2,5 horas) Jornada dedicada a la realización del *planning* detallado del proyecto. Se procedió también con la configuración inicial de la máquina virtual en VirtualBox, instalando el sistema operativo Lubuntu 24.04 y los paquetes base.

Sábado, 18 de octubre de 2025 (1,25 horas) Configuración de la plataforma en la nube MongoDB Atlas. Se creó la cuenta, el proyecto 'SmartParking Flow', el cluster M0 y los usuarios de la base de datos. Se definieron las colecciones 'bays' y 'events' con sus respectivos índices.

Domingo, 19 de octubre de 2025 (2,5 horas) Se finalizaron las pruebas de conectividad con MongoDB Atlas usando un script de Python. Se procedió a la instalación y configuración de Apache Kafka, incluyendo la creación del tema 'parking-events' y el desarrollo del script de simulación de sensores. Finalmente, se realizó la instalación base de Apache NiFi.

Lunes, 20 de octubre de 2025 (2,5 horas) Jornada centrada en Apache NiFi. Se configuraron los Controller Services y se diseñó el flujo completo para consumir datos de Kafka, procesarlos y enviarlos a las dos colecciones correspondientes en MongoDB Atlas (histórico en 'events' y estado en 'bays' mediante *upsert*).

Domingo, 28 de octubre de 2025 (2 horas) Instalación de Flask y las dependencias de Python. Se creó la estructura del proyecto web y se implementó el backend 'app.py' con la conexión a MongoDB y los endpoints de la API REST. Se validó el funcionamiento inicial sirviendo la página 'index.html'.

Sábado, 1 de noviembre de 2025 (2,5 horas) Sesión de *troubleshooting* y afinado del flujo de NiFi para optimizar el enrutamiento. Posteriormente, se instaló y configuró Dremio, conectándolo a la fuente de MongoDB Atlas y ejecutando las consultas de análisis requeridas.

Domingo, 2 de noviembre de 2025 (4 horas) Comienzo de la redacción de la memoria del proyecto ('memoria.tex'). Se estableció la estructura del documento, el índice y se redactaron los primeros apartados (Introducción, Marco Teórico).

Lunes, 3 de noviembre de 2025 (2,5 horas) Configuración del entorno de LaTeX en local para la compilación del documento. Se continuó intensivamente con la redacción de la memoria, completando la captura de imágenes y la redacción de todos los Anexos (A-F).

Martes, 4 de noviembre de 2025 (1 hora) Se ha dedicado esta jornada a la redacción de las secciones finales de la memoria (Implementación, Pruebas, Conclusiones, Bibliografía y este Diario de Trabajo) y a la revisión final y compilación del documento completo.

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.

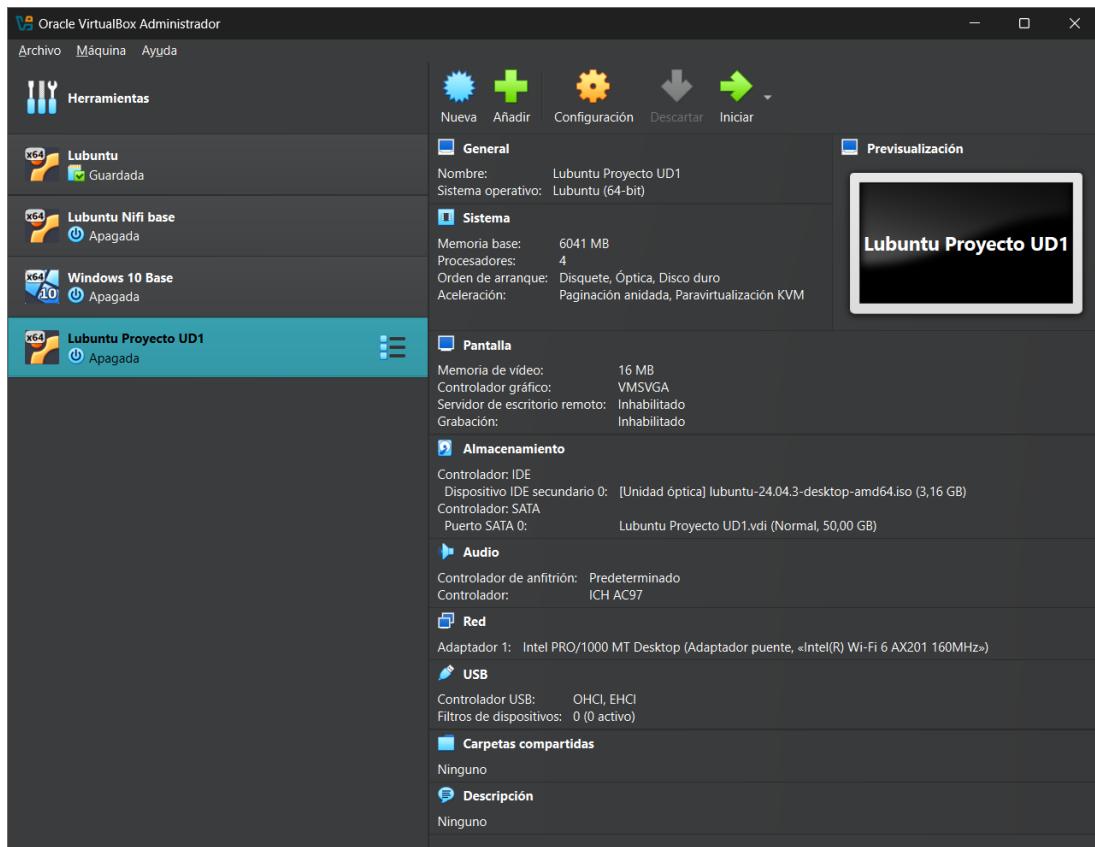


Figura A.1: Configuración inicial de la MV 'Lubuntu Proyecto UD1' en VirtualBox.

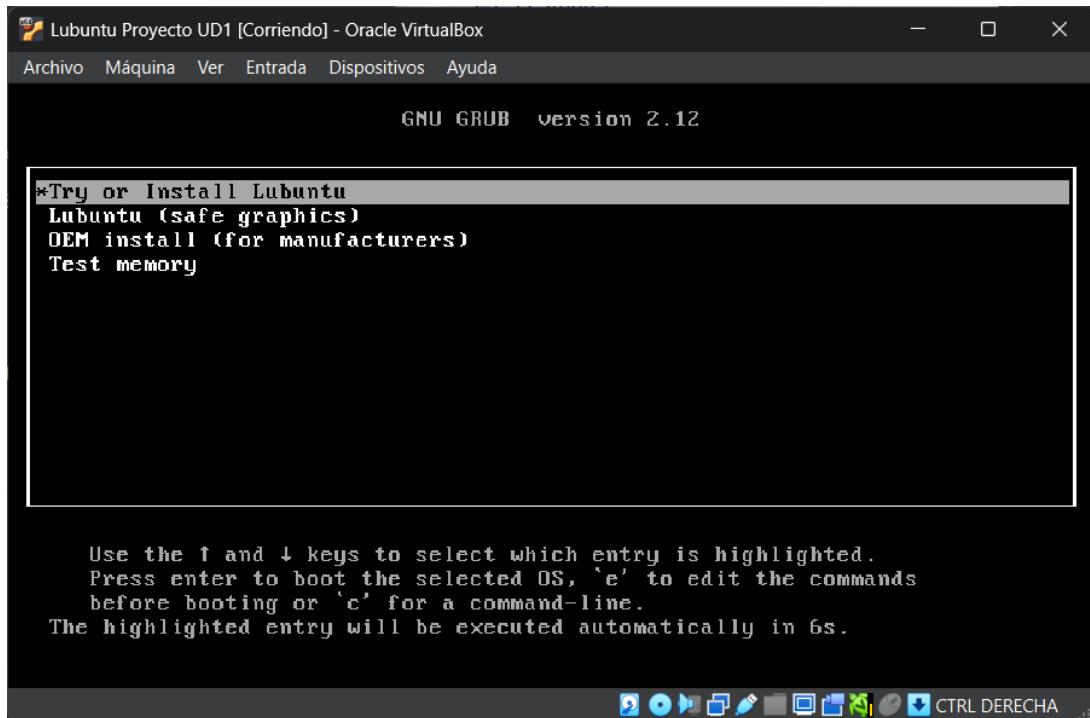


Figura A.2: Menú de arranque (GRUB) de la ISO de Lubuntu.



Figura A.3: Inicio del instalador de Lubuntu.

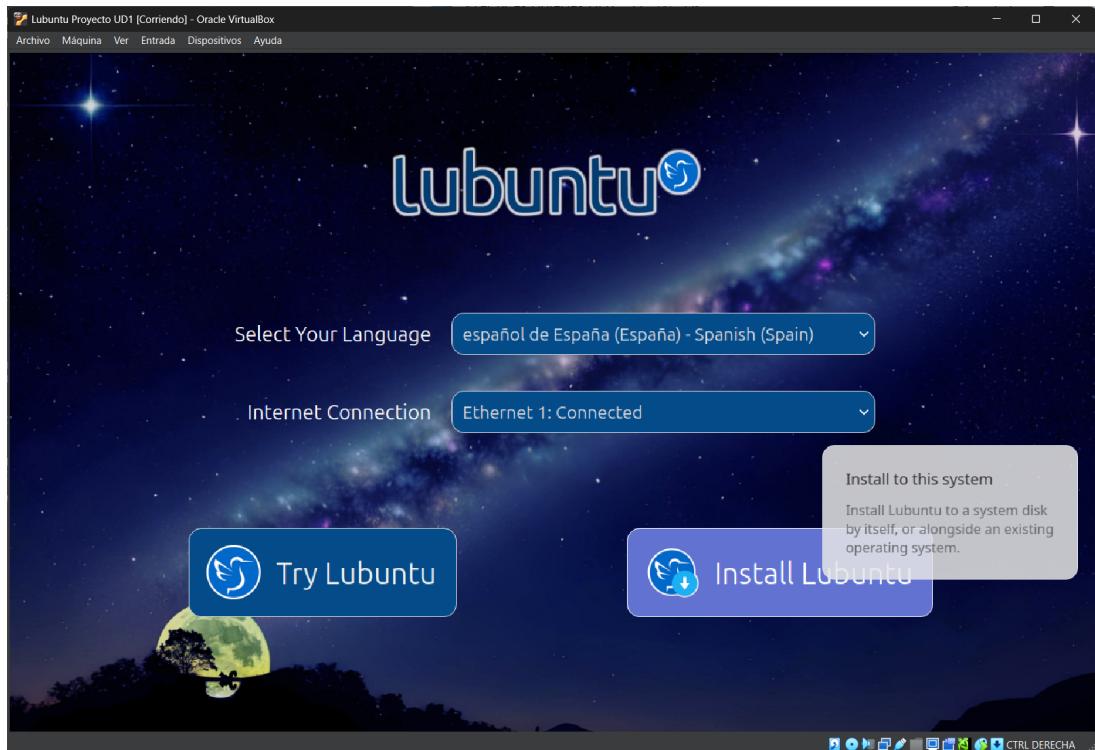


Figura A.4: Selección de 'Install Lubuntu'.

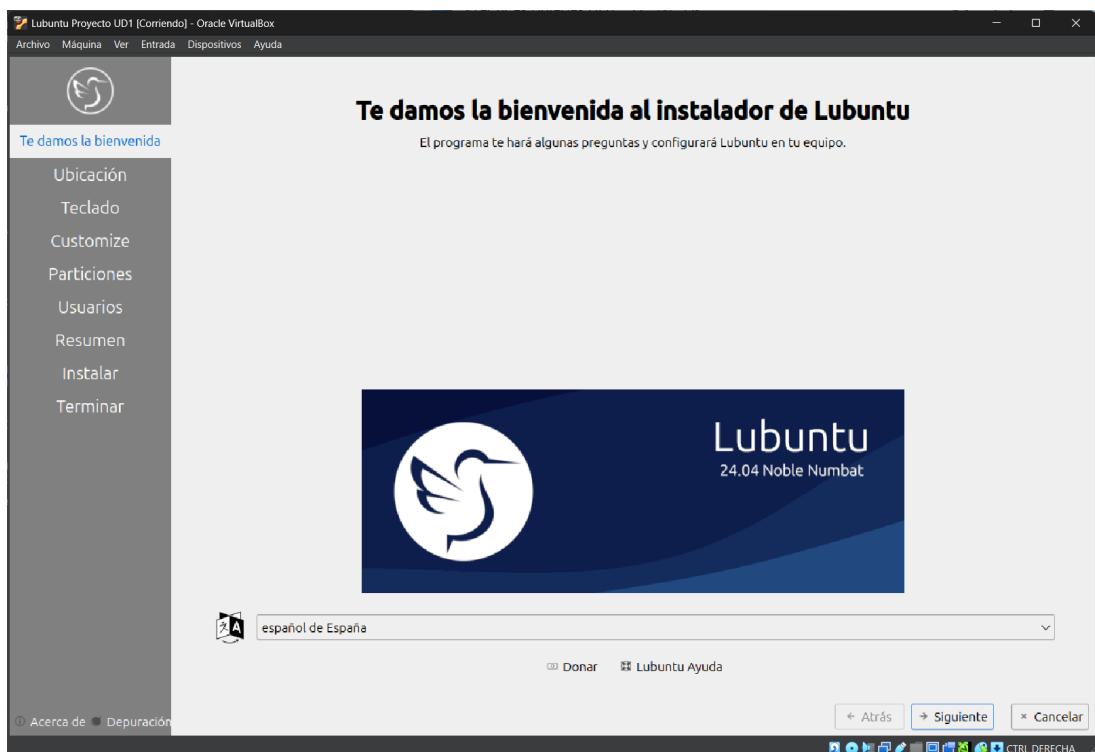


Figura A.5: Selección de idioma (Español).

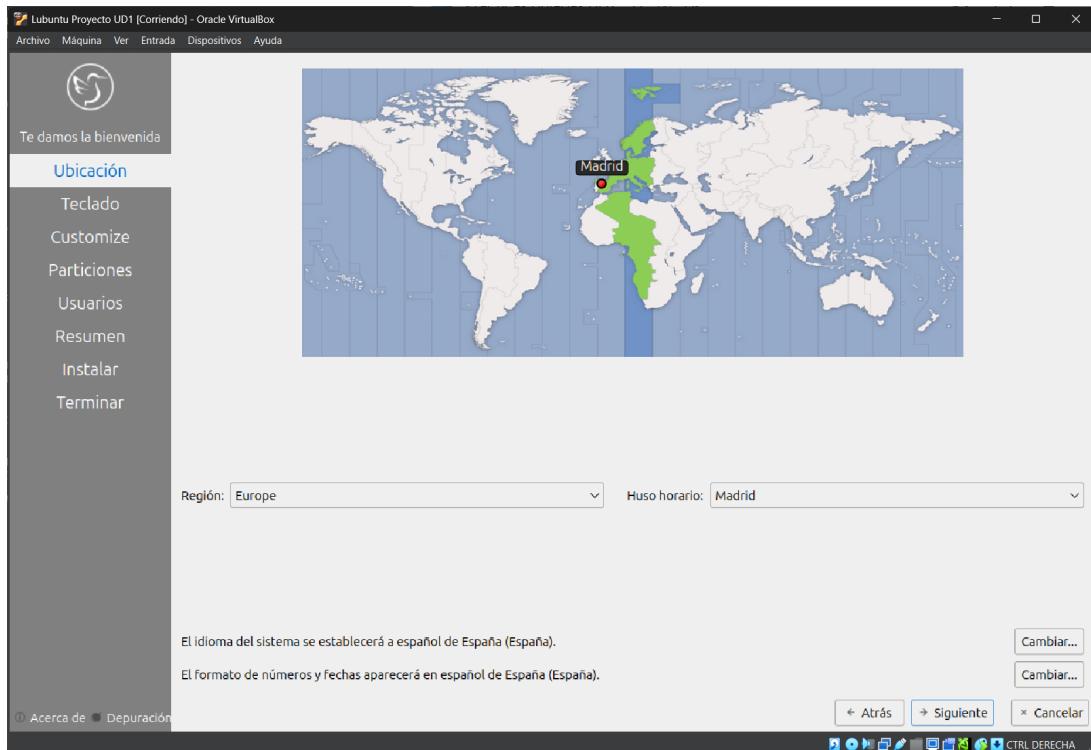


Figura A.6: Selección de ubicación (Madrid).

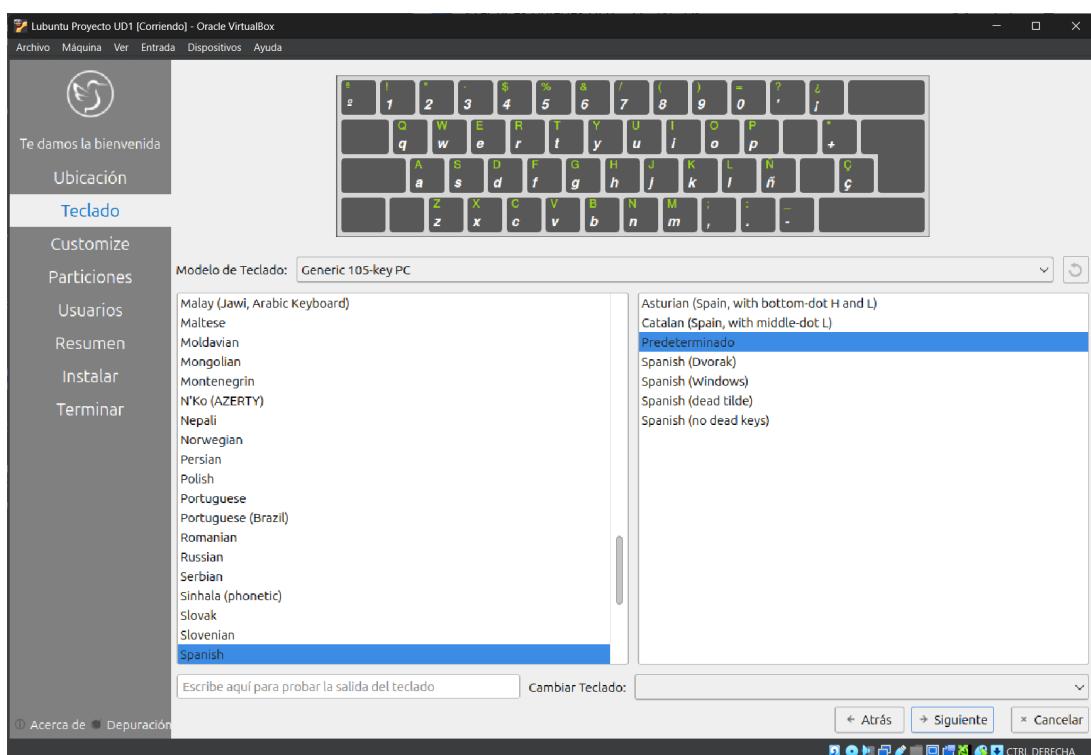


Figura A.7: Selección de teclado (Spanish).

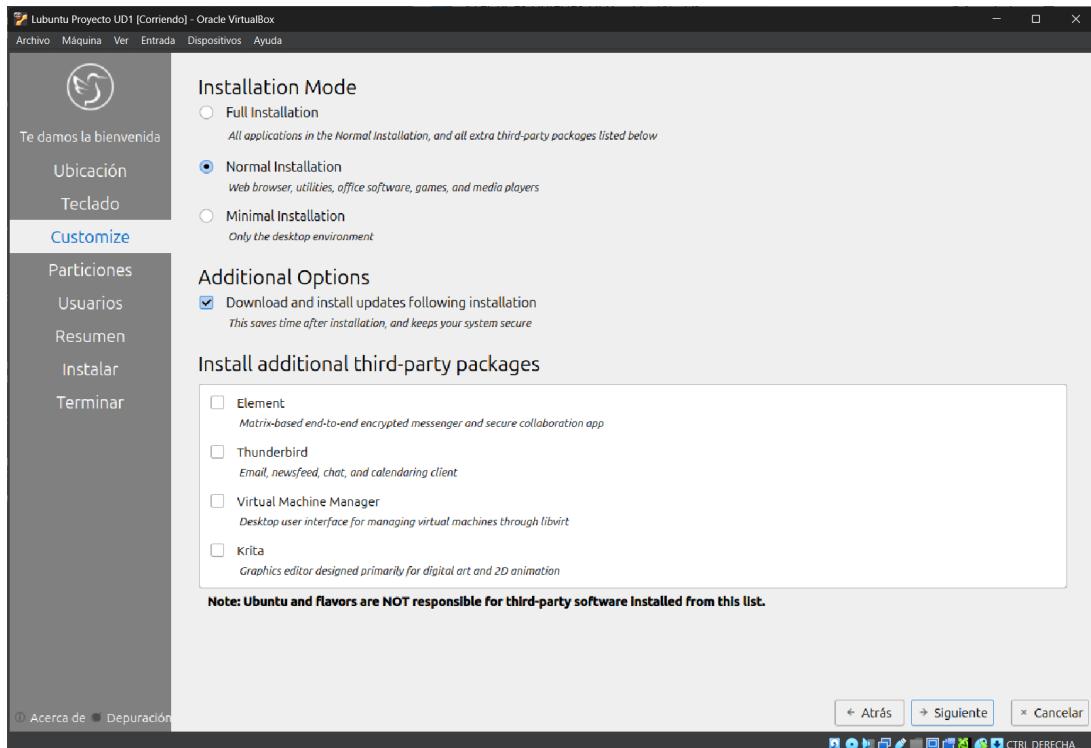


Figura A.8: Selección de tipo de instalación (Normal).

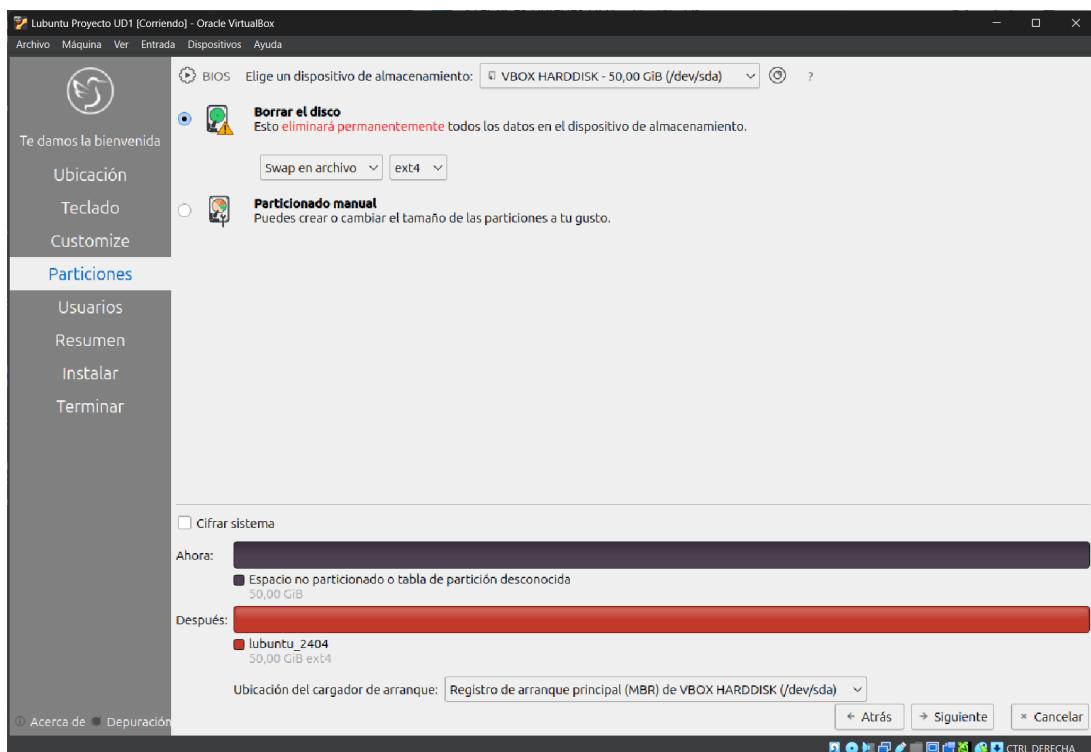


Figura A.9: Configuración de particiones (Borrar disco).

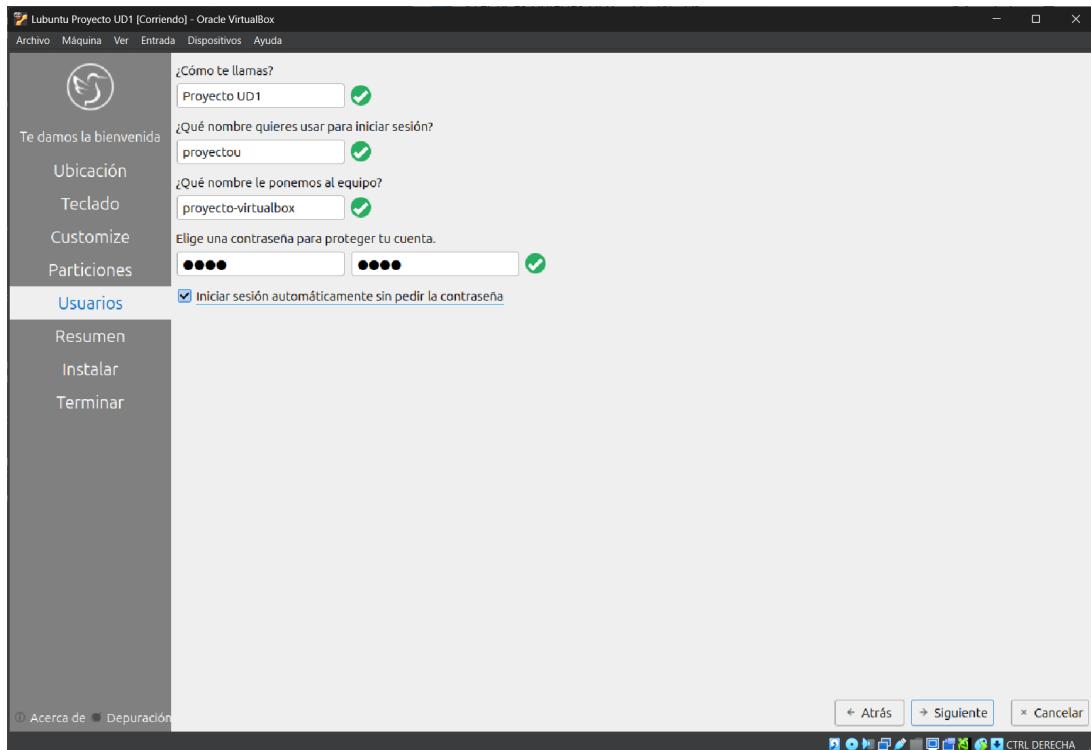


Figura A.10: Creación del usuario y nombre de equipo.

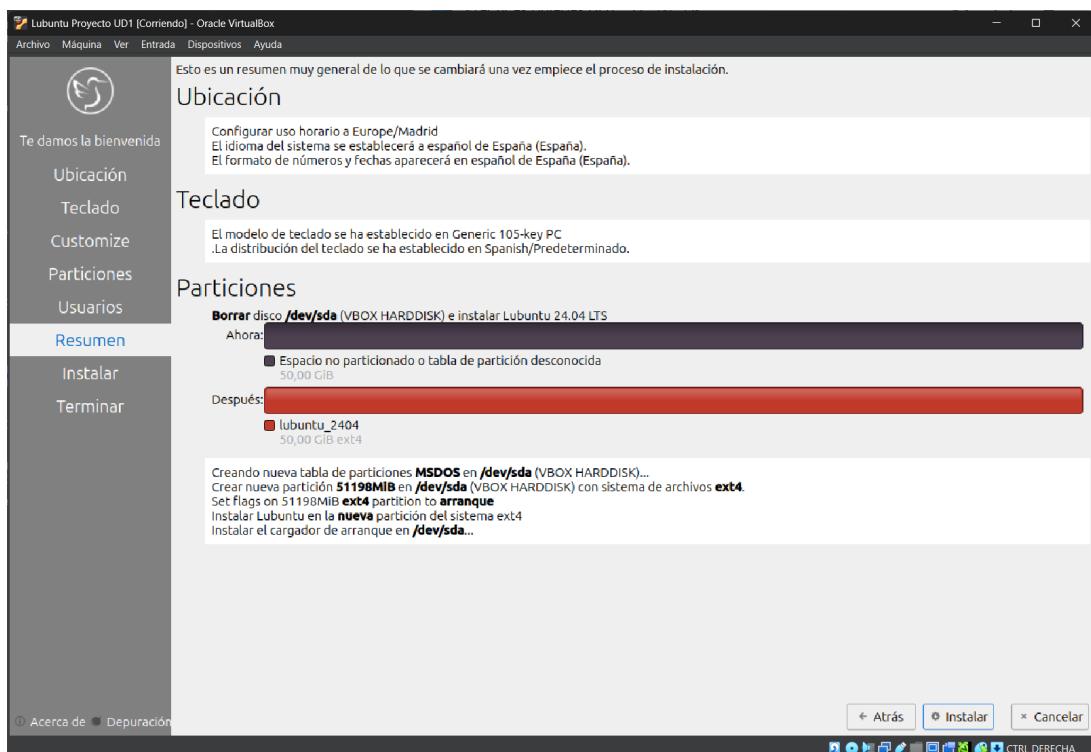


Figura A.11: Resumen de la instalación.

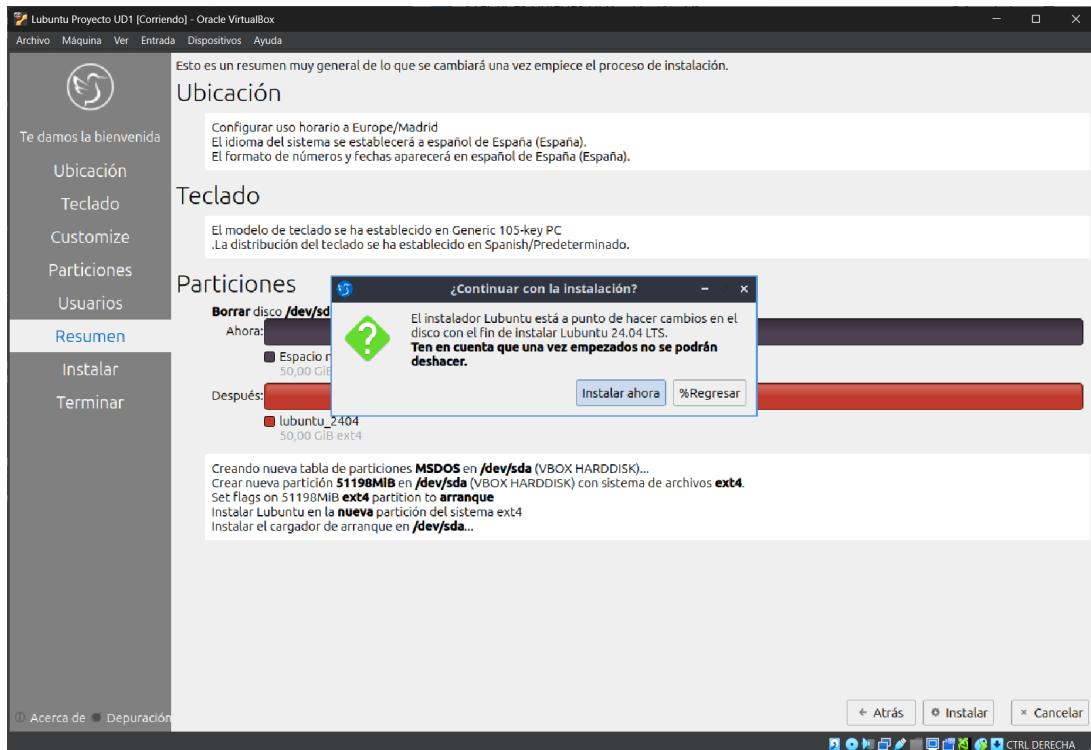


Figura A.12: Confirmación de inicio de instalación.

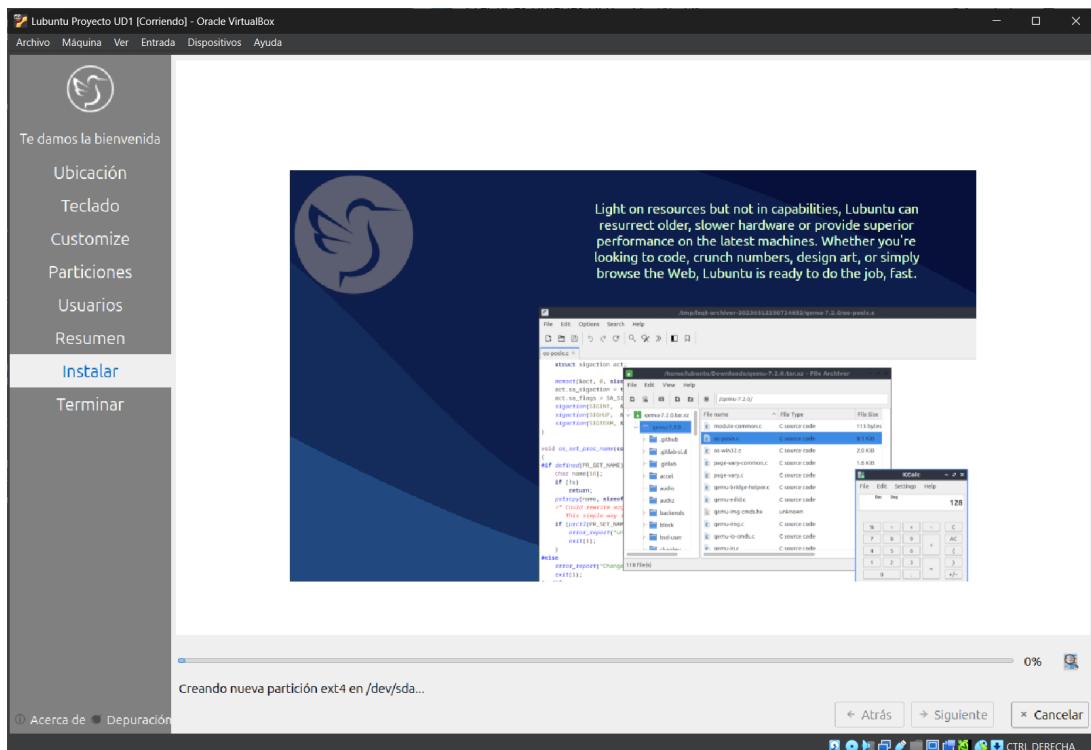


Figura A.13: Proceso de instalación.

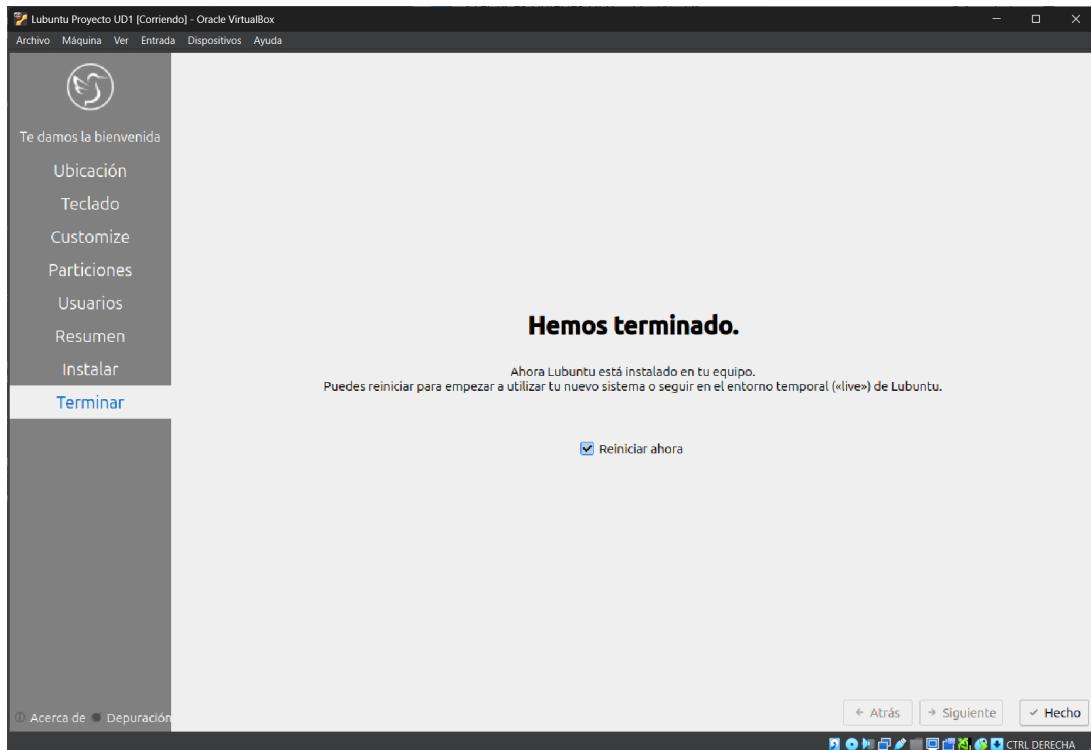


Figura A.14: Instalación finalizada.

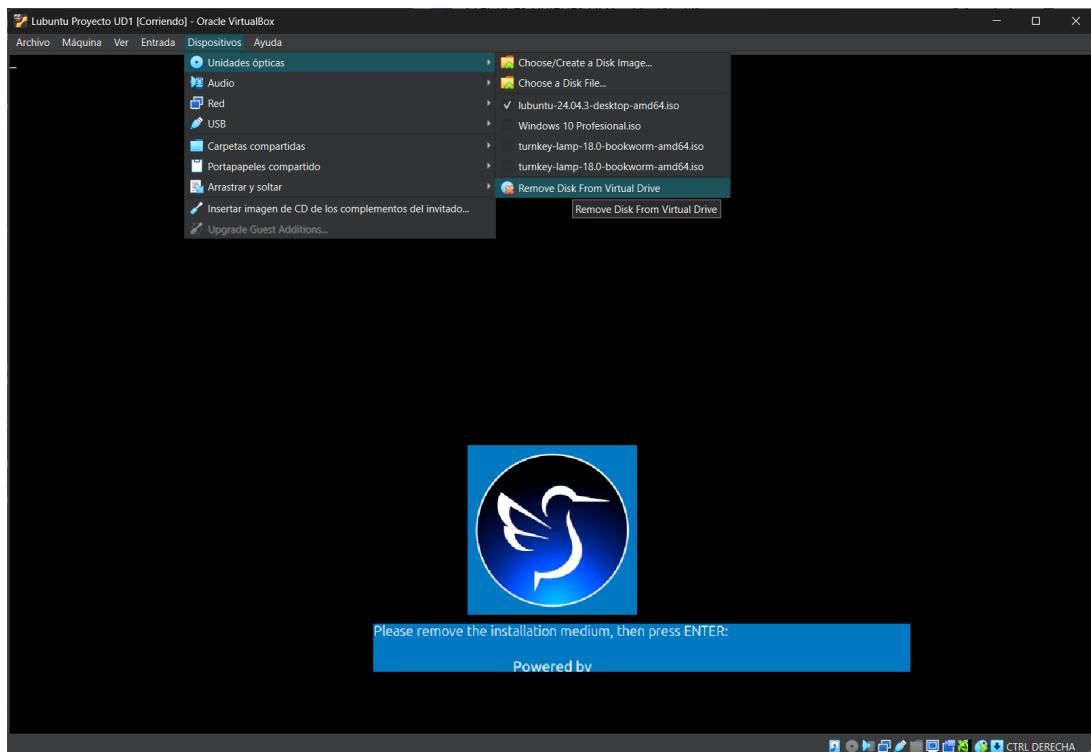


Figura A.15: Extracción del disco de instalación virtual al reiniciar.

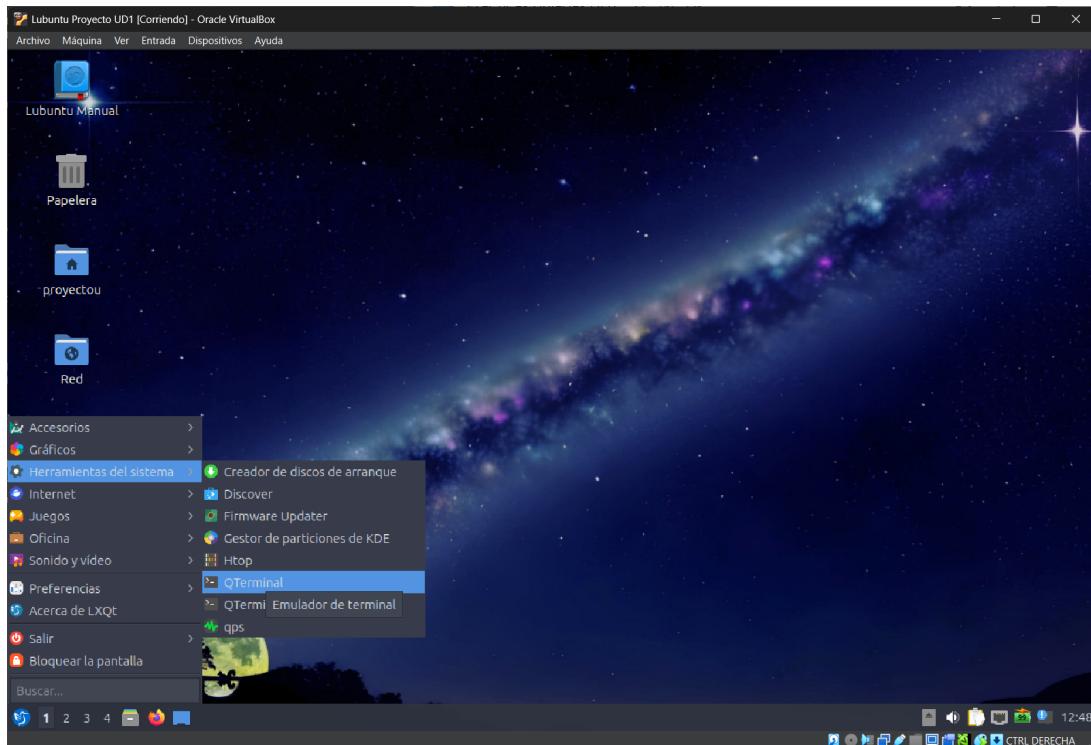


Figura A.16: Escritorio de Lubuntu y apertura de QTerminal.

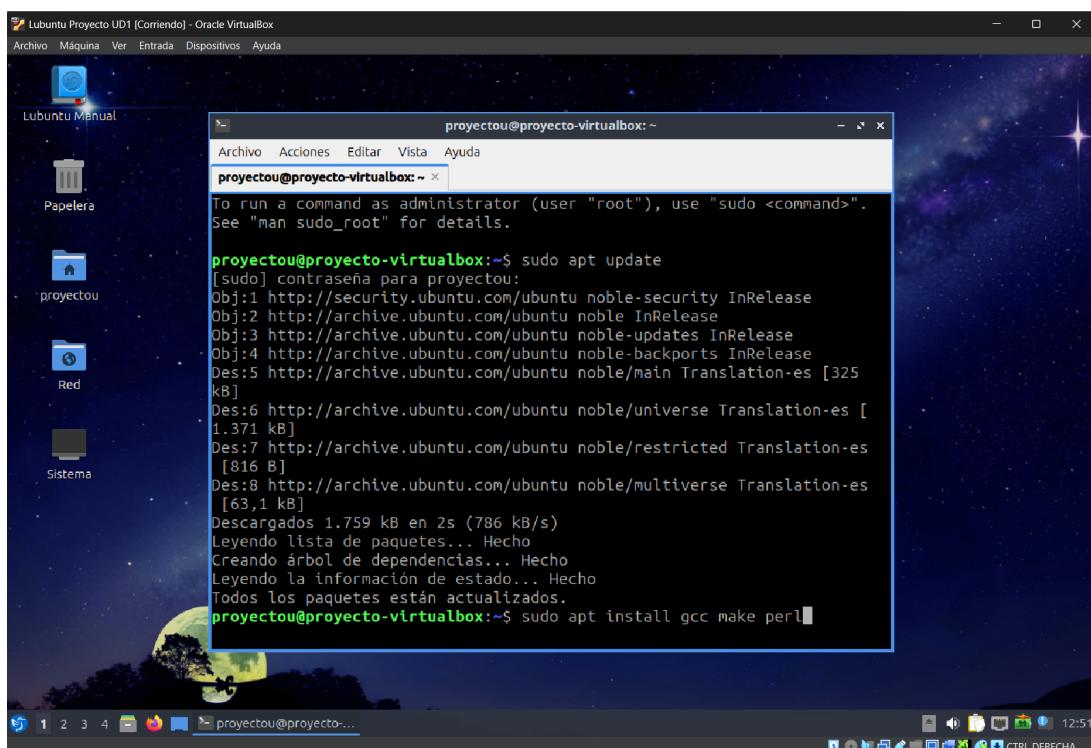


Figura A.17: Actualización de paquetes ('sudo apt update').

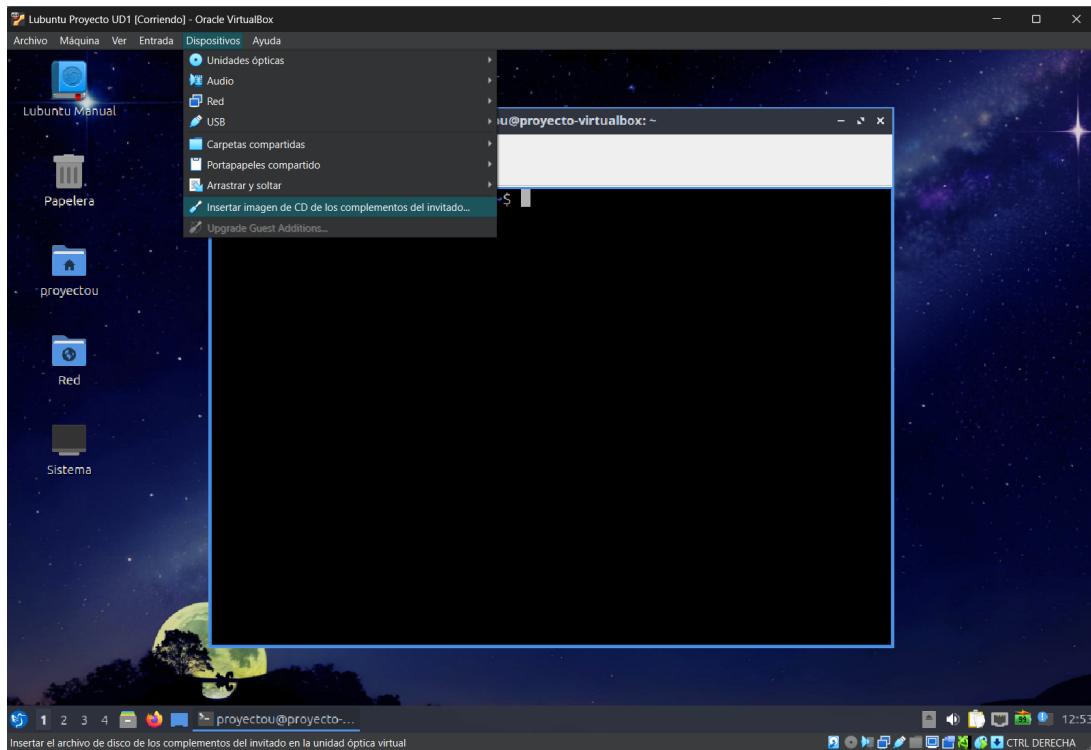


Figura A.18: Insertando imagen de CD de las 'Guest Additions' de VirtualBox.

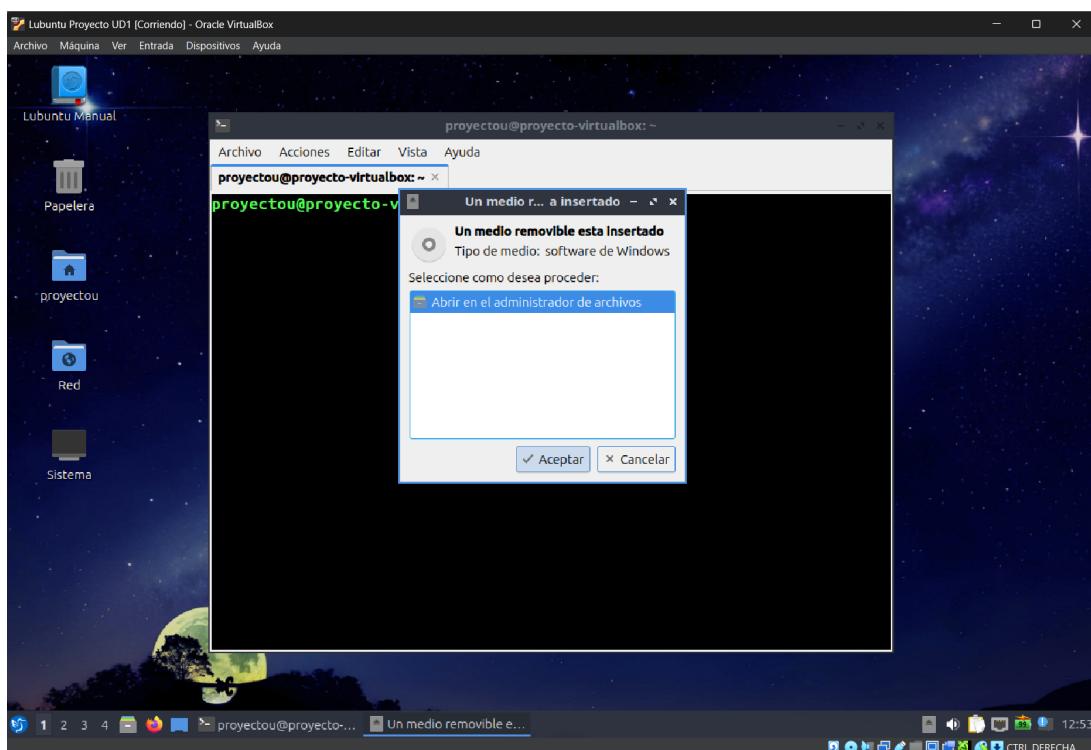


Figura A.19: Aviso de medio extraíble (Guest Additions).

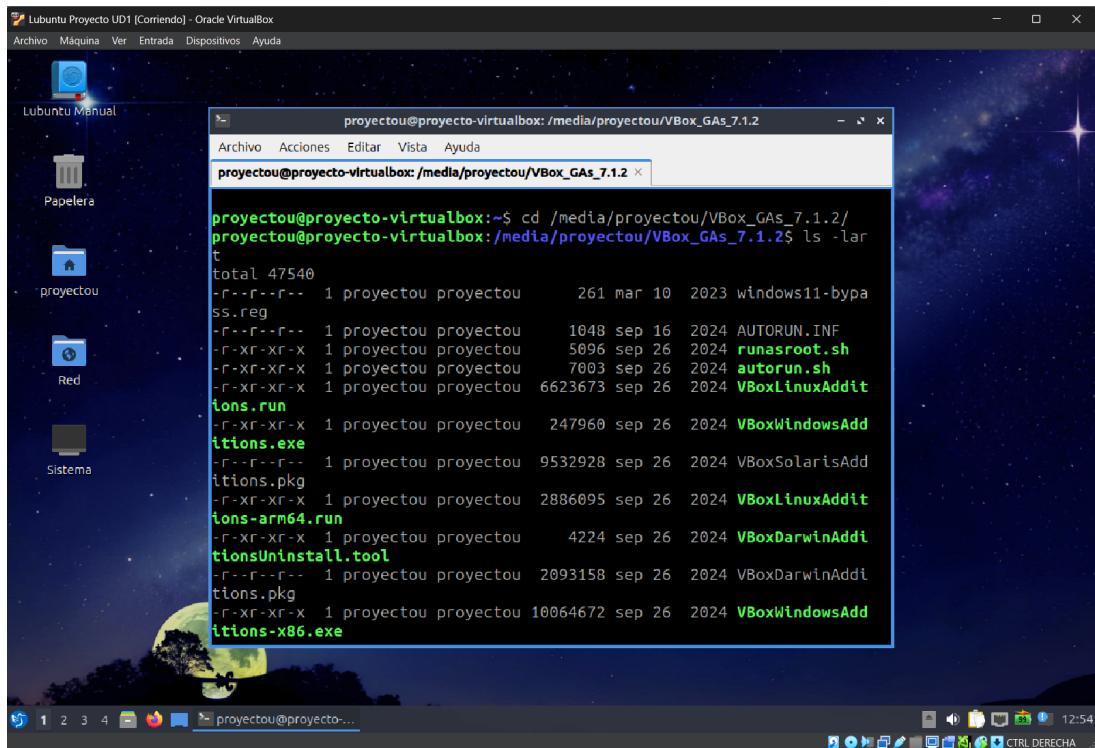


Figura A.20: Navegación al directorio de las Guest Additions.

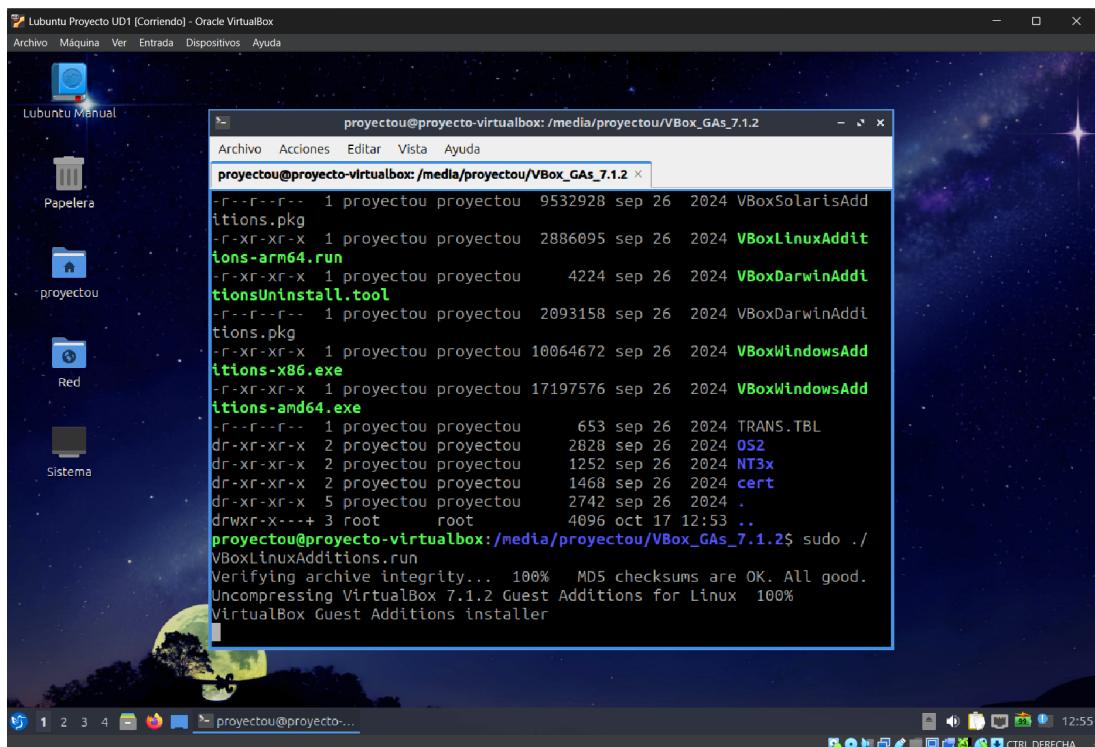


Figura A.21: Ejecución del script de instalación VBoxLinuxAdditions.run.

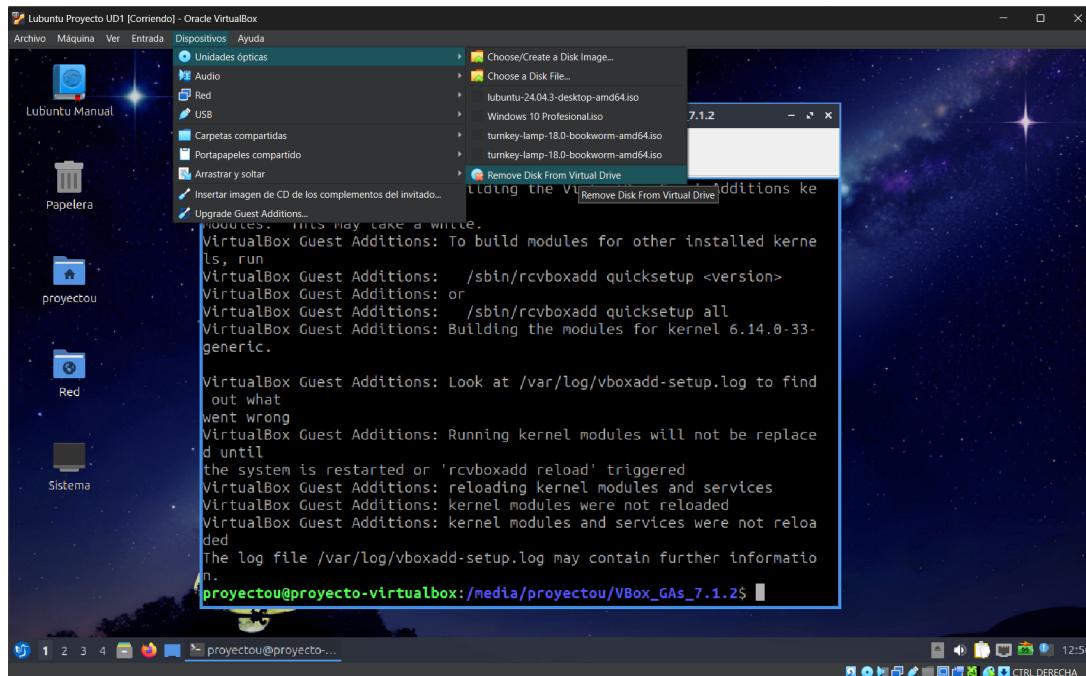


Figura A.22: Extracción del disco virtual de las Guest Additions.

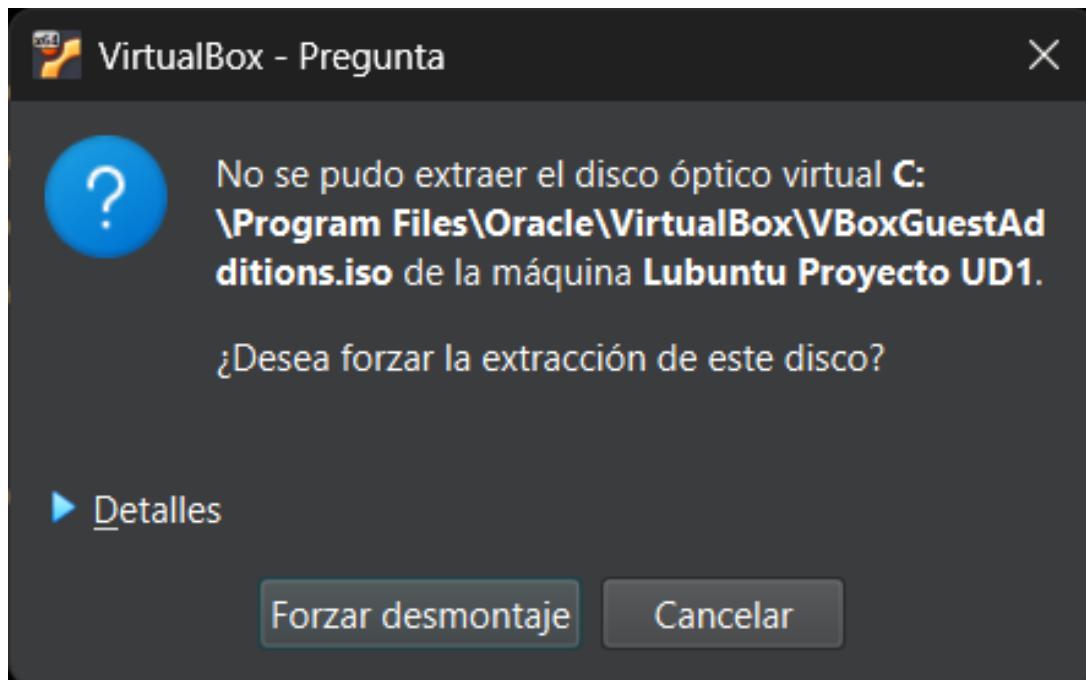


Figura A.23: Aviso de forzar desmontaje del disco óptico.

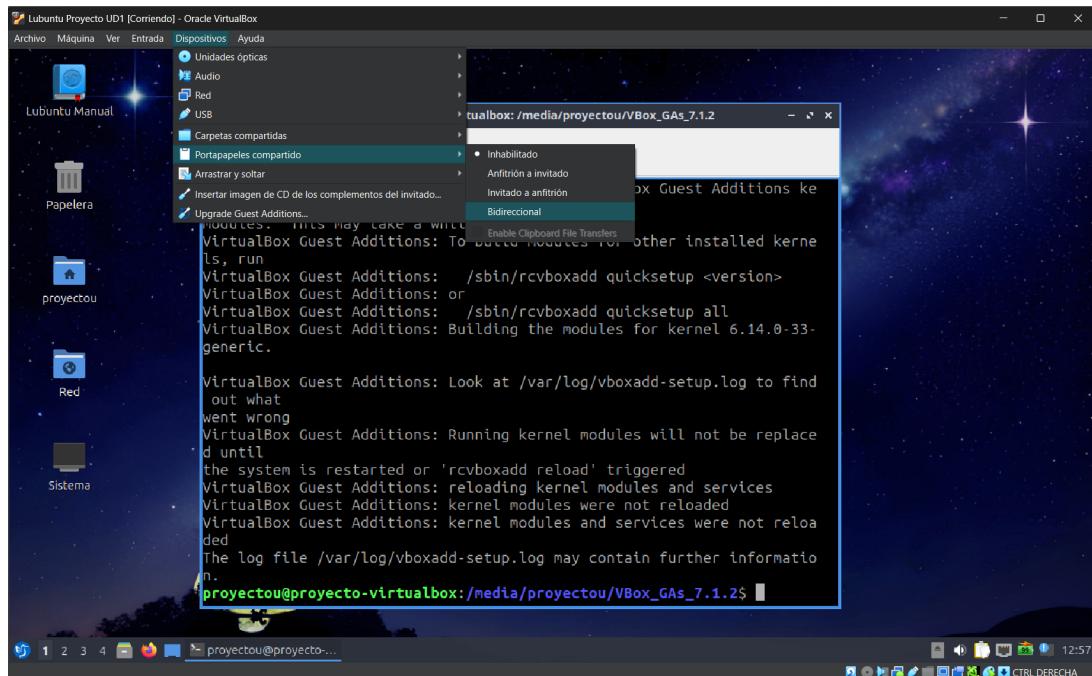


Figura A.24: Habilitando portapapeles bidireccional en VirtualBox.

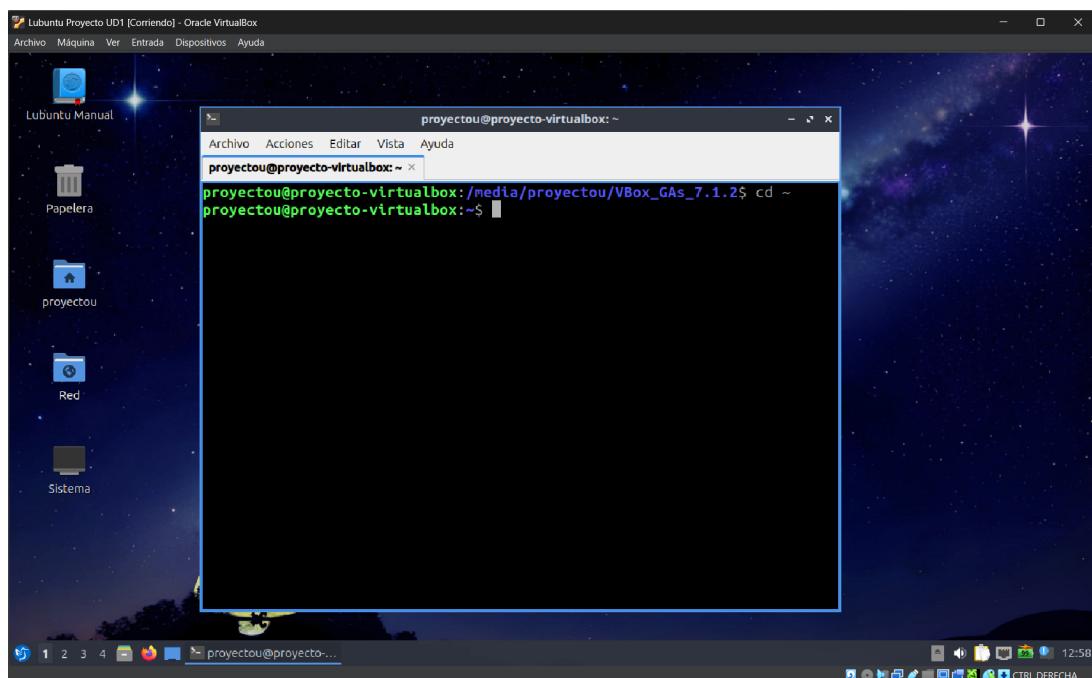


Figura A.25: Terminal de Lubuntu post-reinicio.

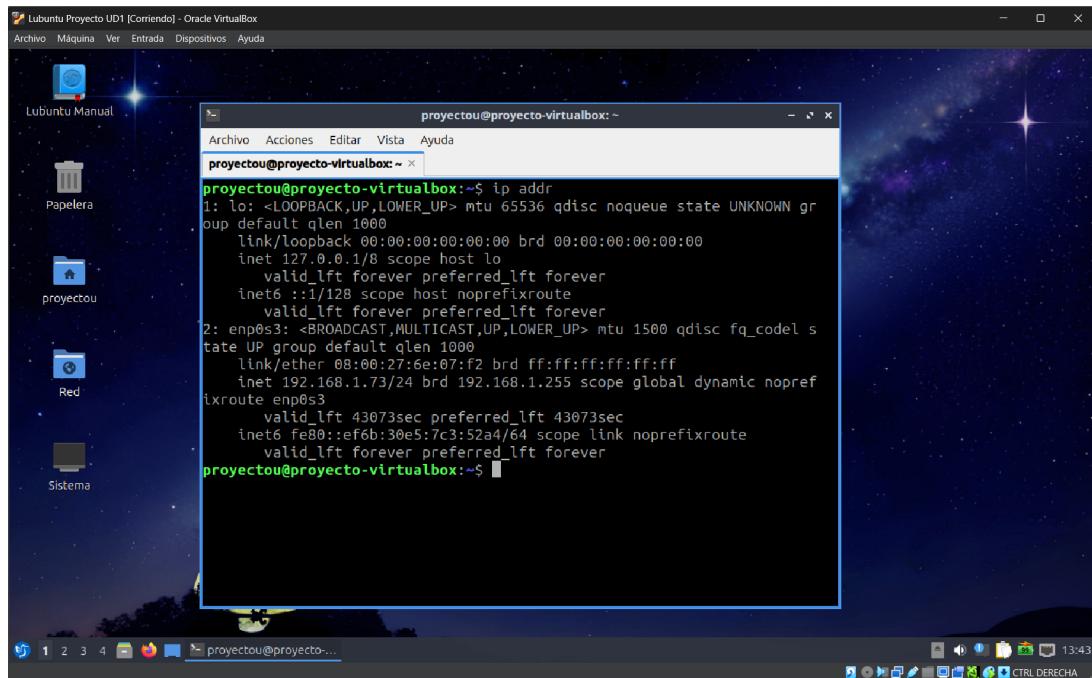


Figura A.26: Comprobación de la dirección IP de la MV ('ip addr') - 192.168.1.73.

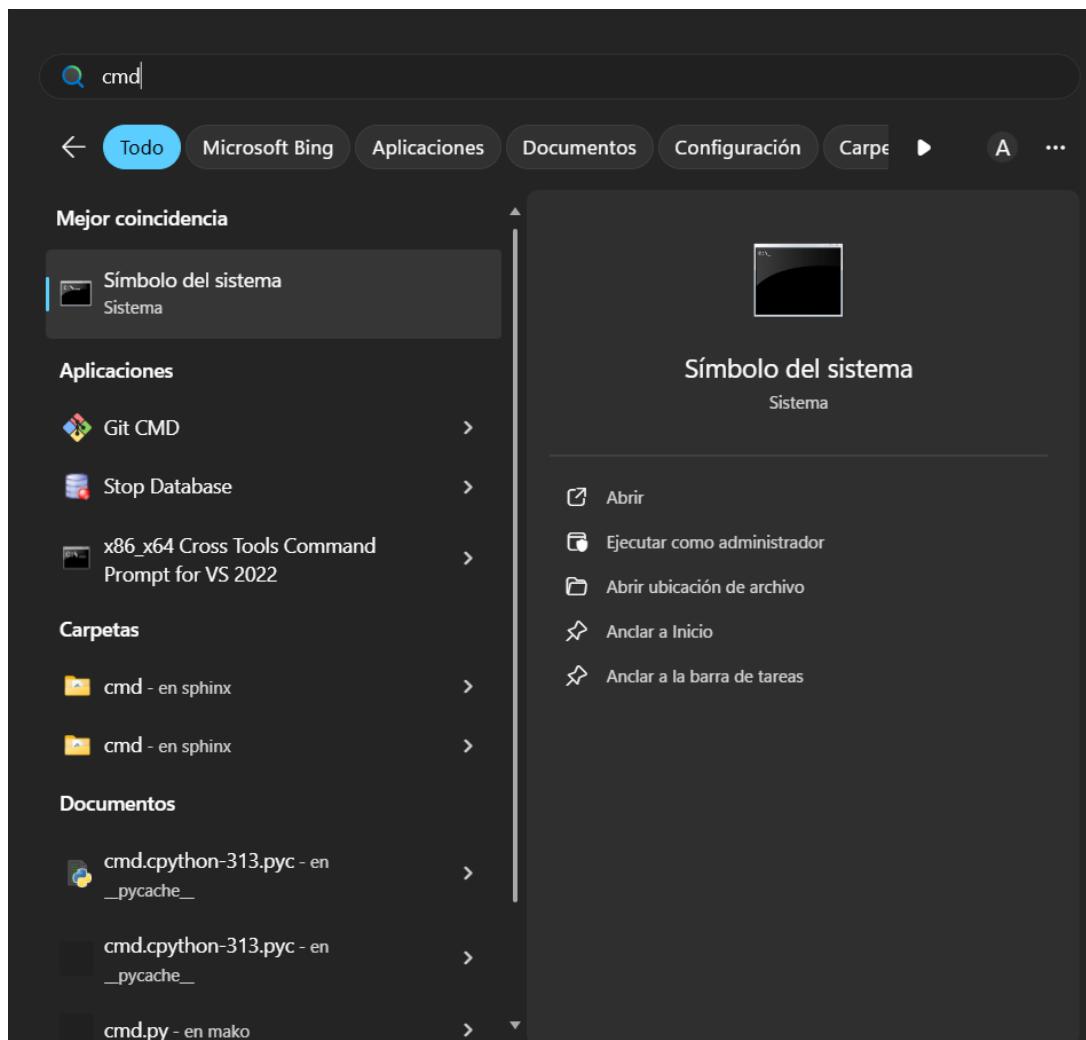
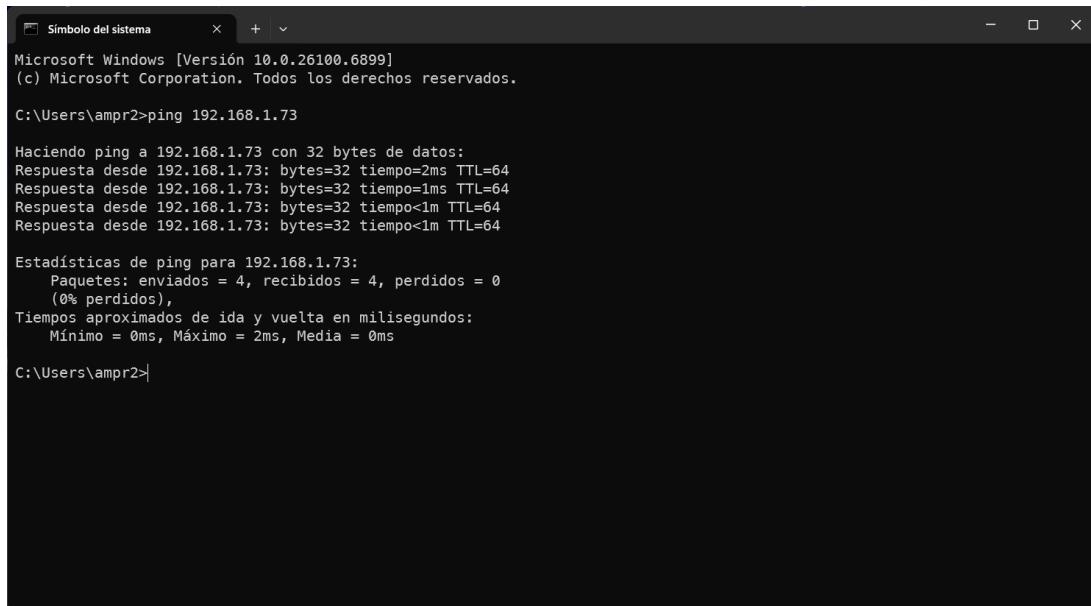


Figura A.27: Búsqueda de Símbolo del sistema (CMD) en el anfitrión (Windows).



```
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.

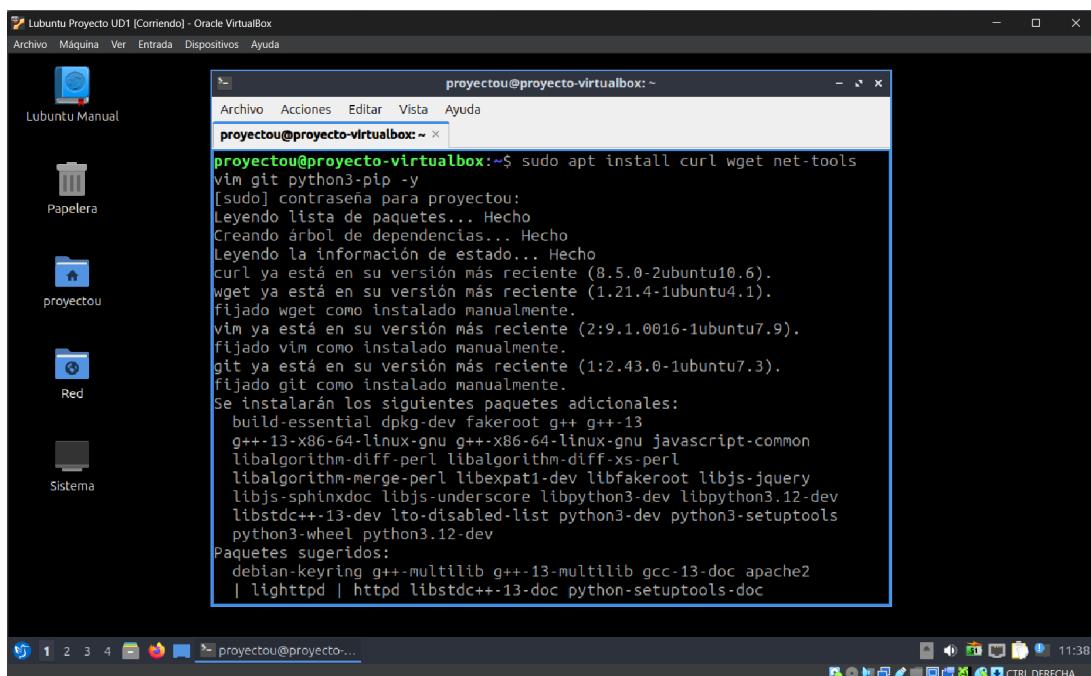


Figura A.29: Instalación de paquetes base en Lubuntu ('curl', 'wget', 'git', 'python3-pip').

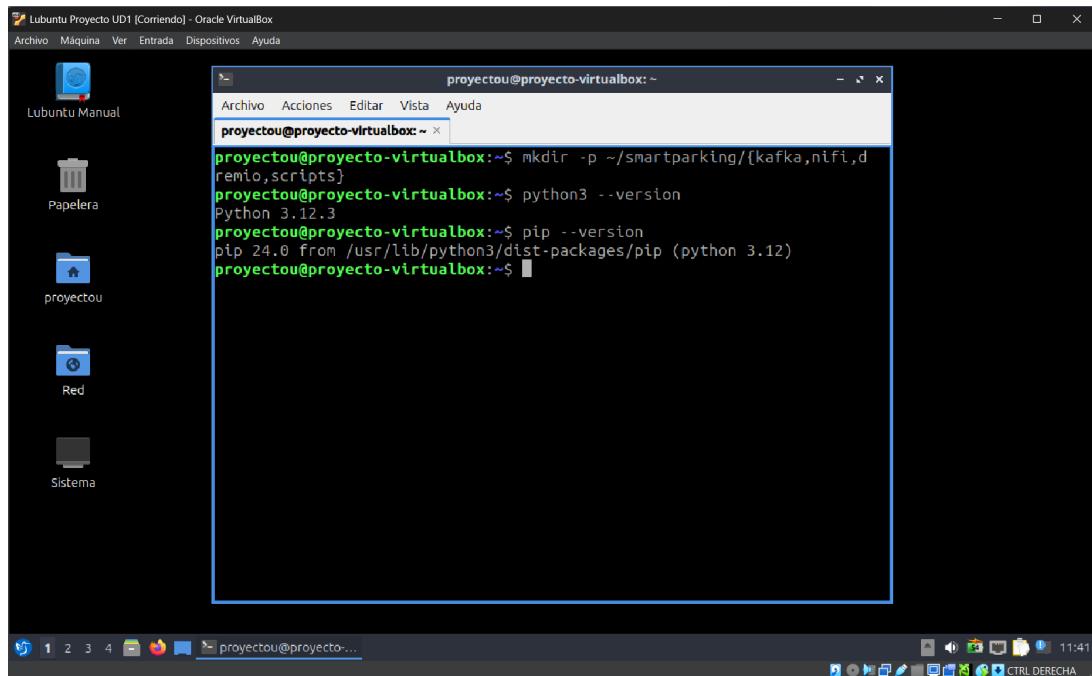


Figura A.30: Creación de directorios del proyecto y comprobación de versiones ('python3' y 'pip').

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.

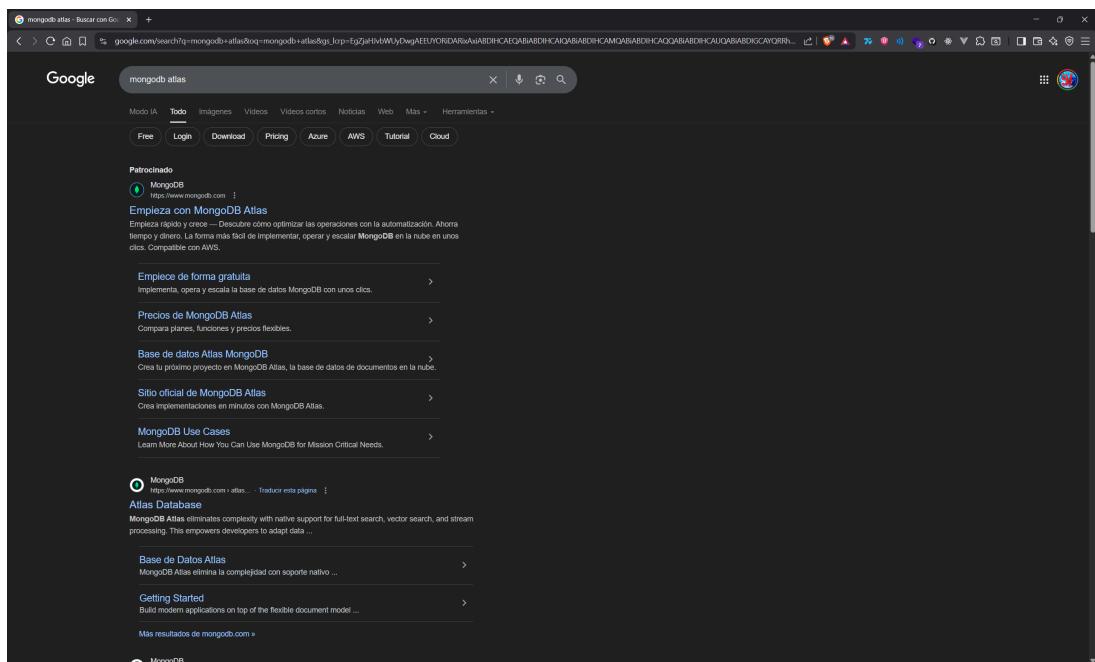


Figura B.1: Búsqueda inicial de MongoDB Atlas.

SmartParking Flow — Monitorización Inteligente

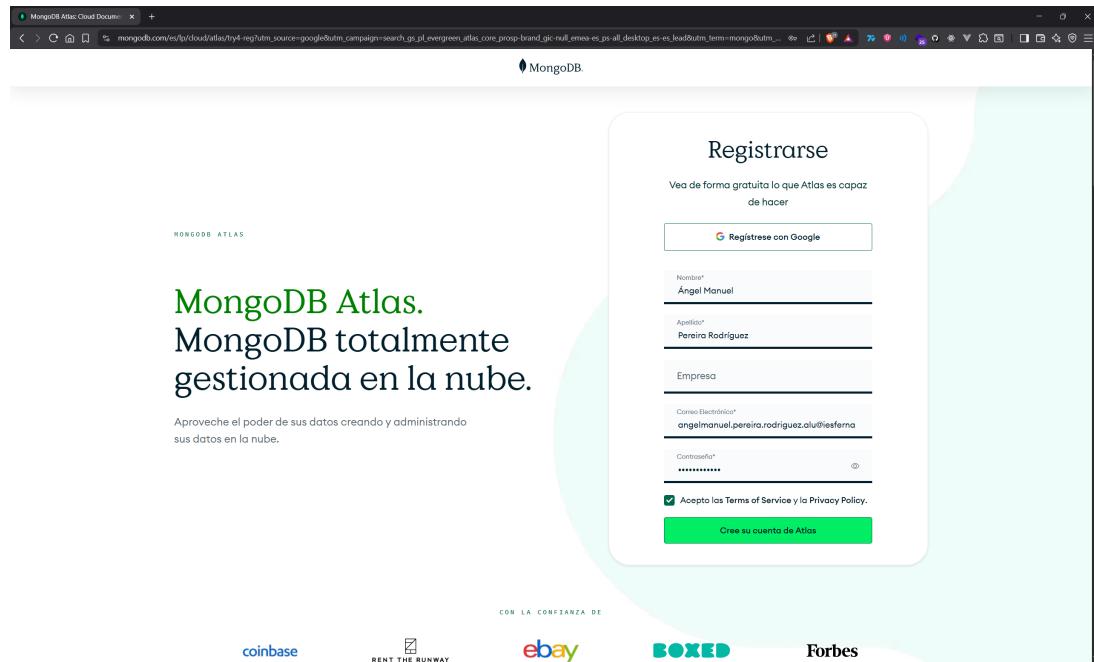


Figura B.2: Formulario de registro en MongoDB Atlas.

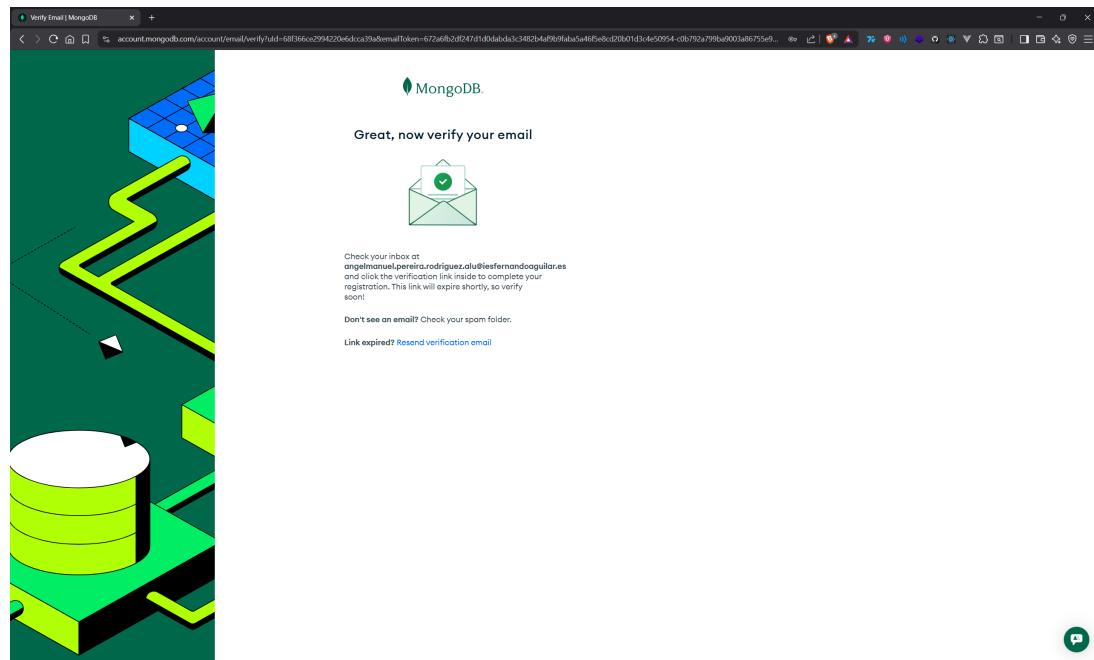


Figura B.3: Paso de verificación de correo electrónico.

SmartParking Flow — Monitorización Inteligente

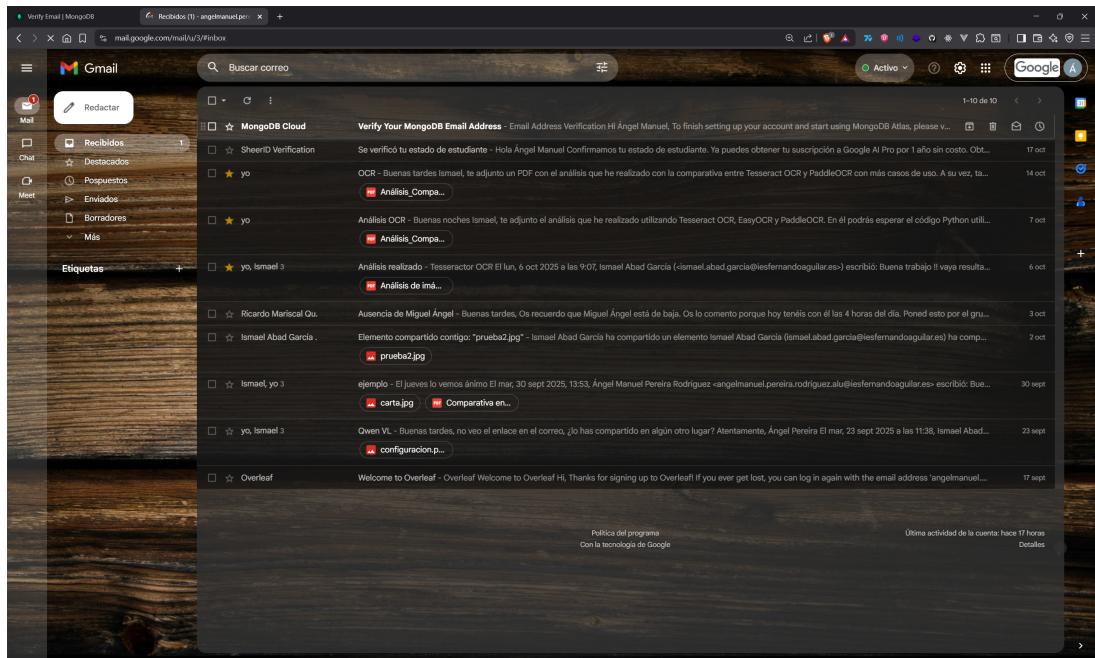


Figura B.4: Recepción del correo de verificación.

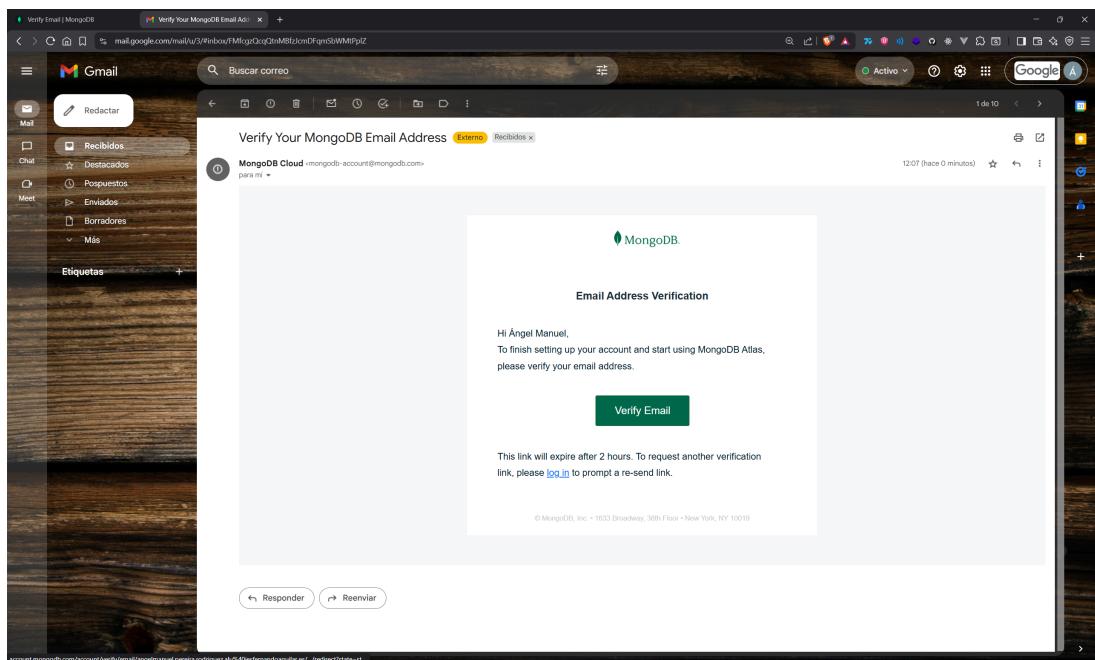


Figura B.5: Contenido del correo 'Verify Your MongoDB Email Address'.

SmartParking Flow — Monitorización Inteligente

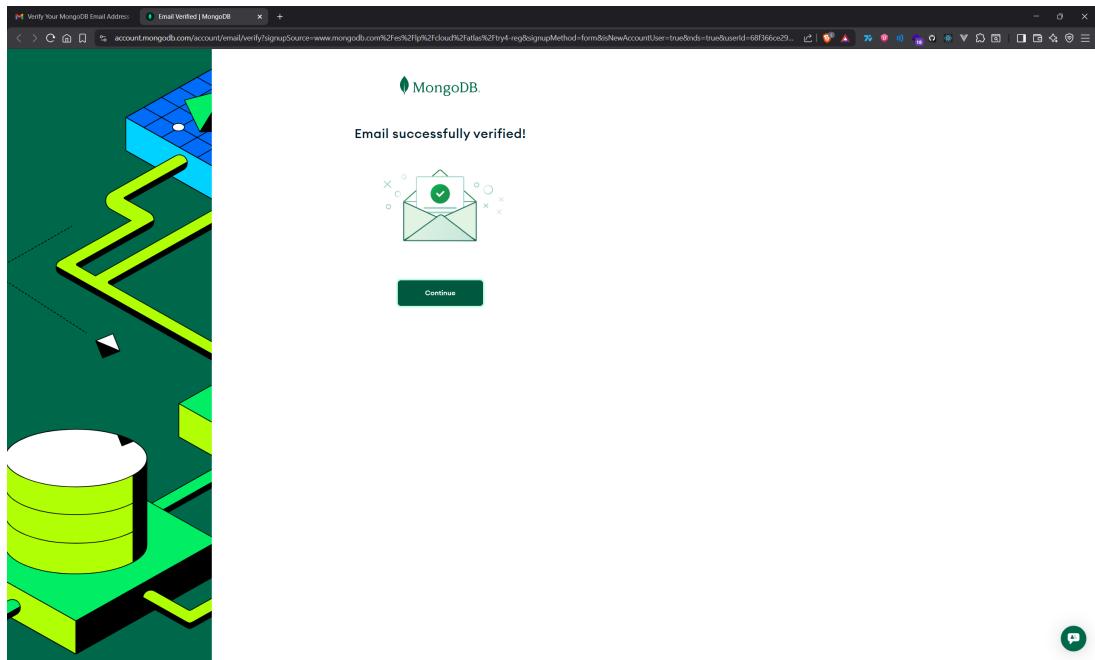


Figura B.6: Confirmación de email verificado.

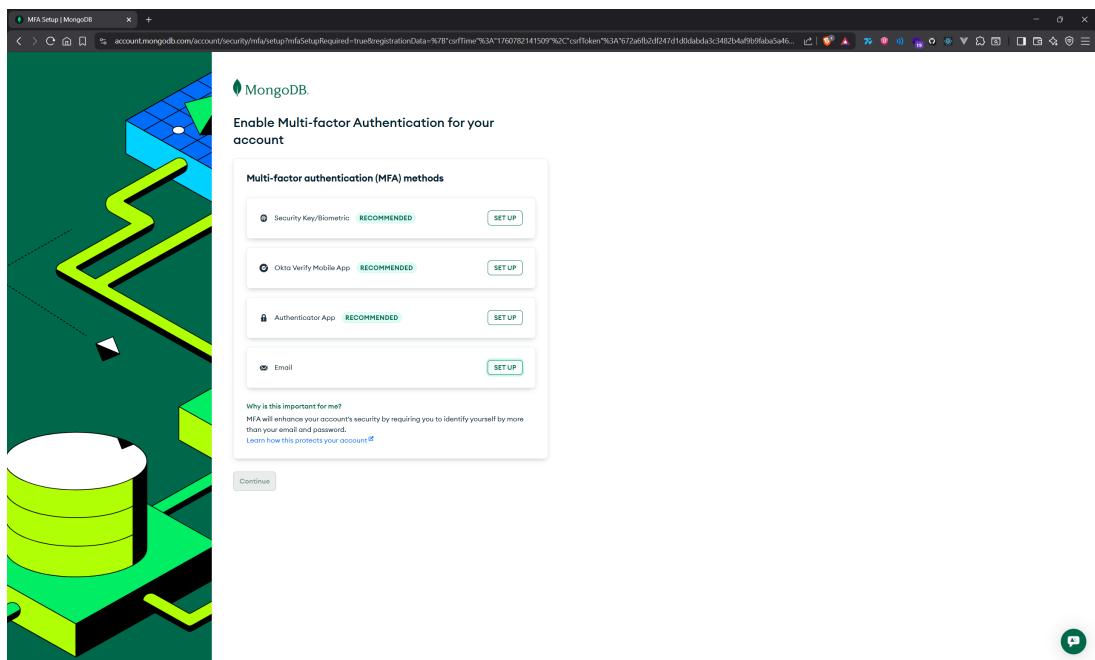


Figura B.7: Configuración opcional de Multi-Factor Authentication (MFA).

SmartParking Flow — Monitorización Inteligente

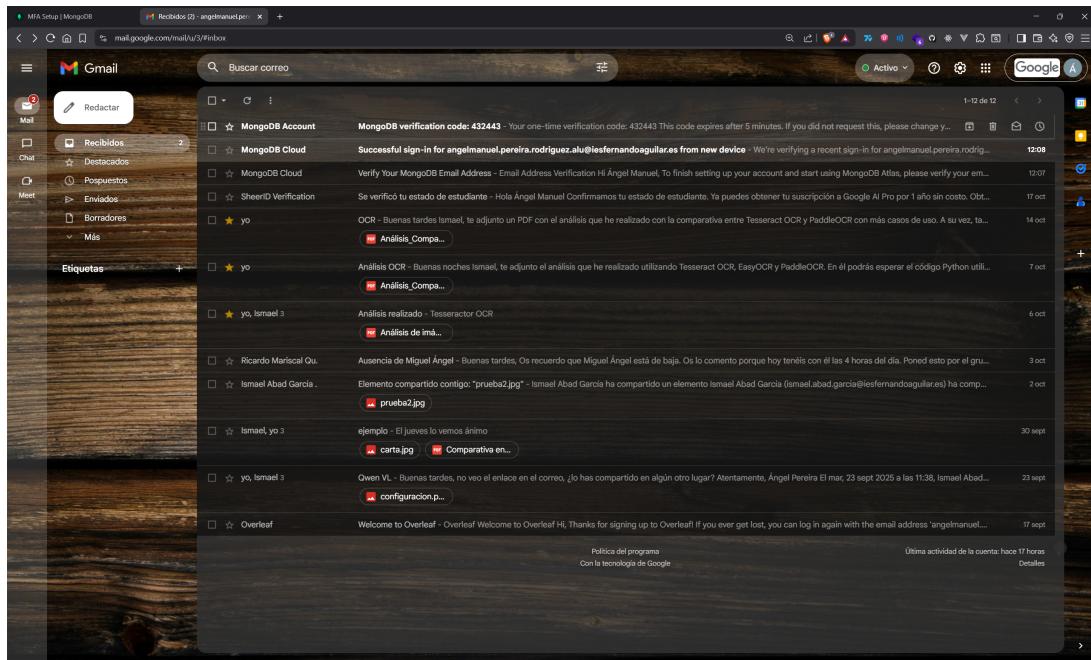


Figura B.8: Recepción del código de verificación de MongoDB.

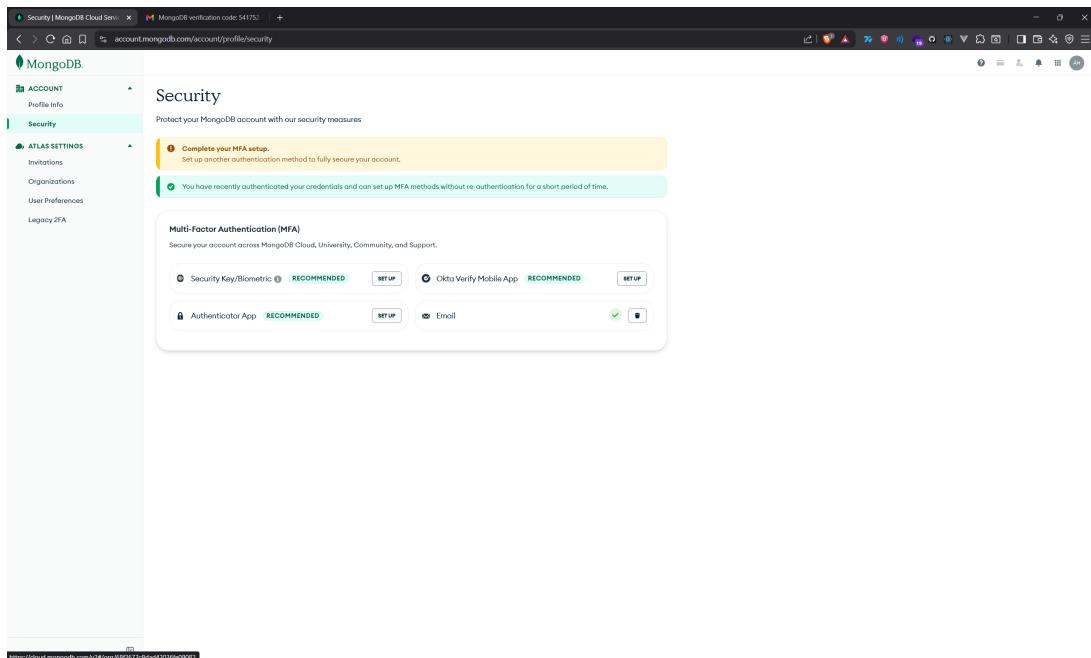


Figura B.9: Pantalla de configuración de seguridad de la cuenta.

SmartParking Flow — Monitorización Inteligente

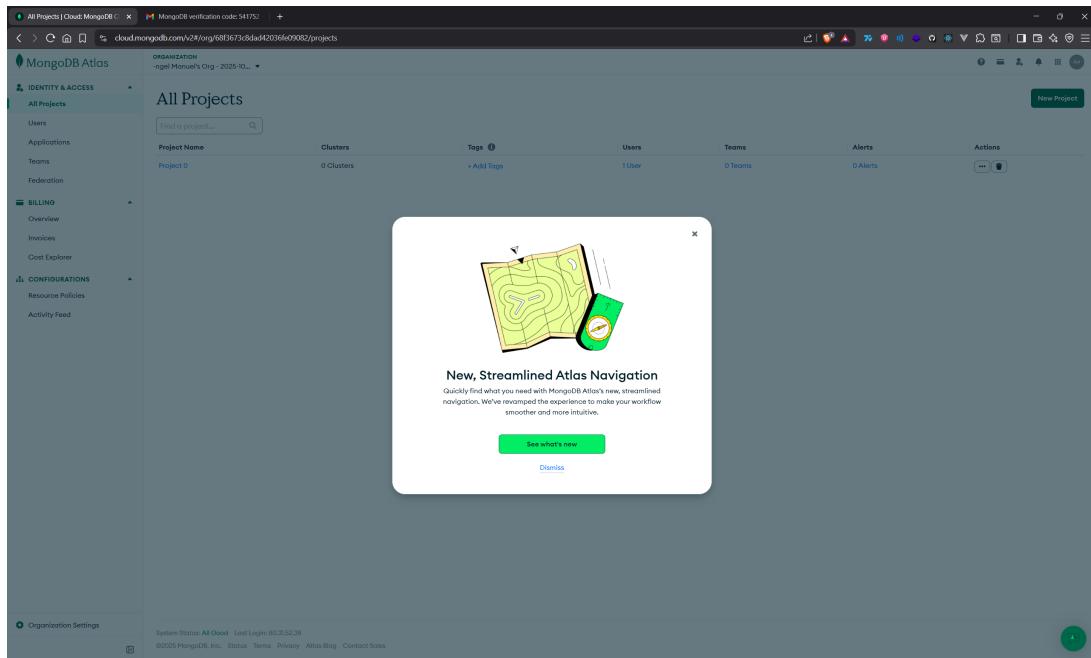


Figura B.10: Pantalla de bienvenida a la nueva navegación de Atlas.

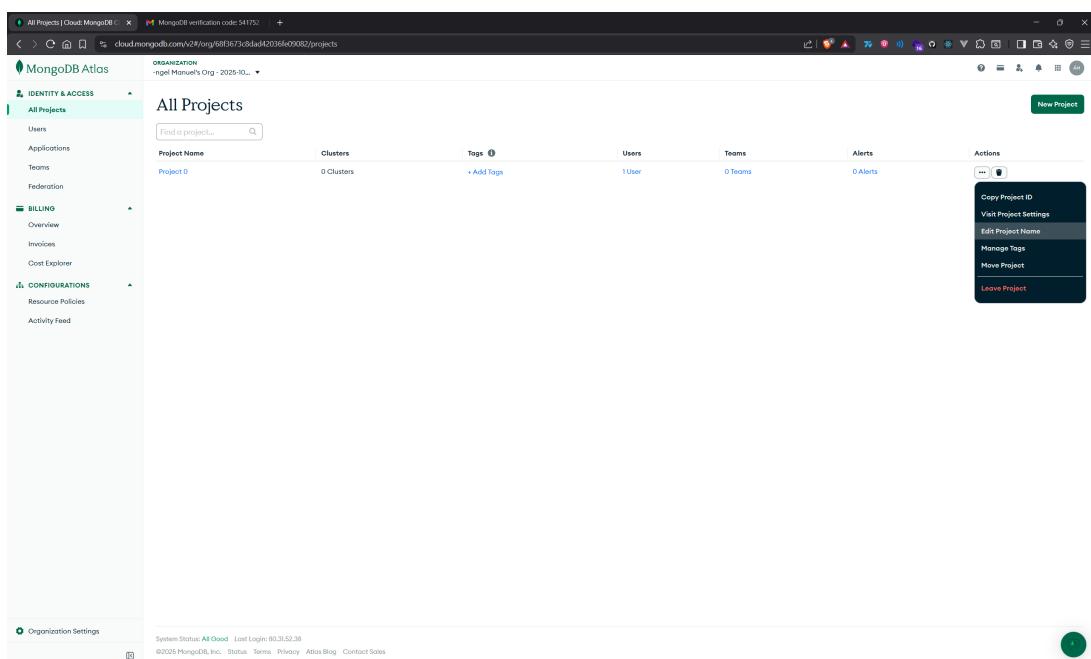


Figura B.11: Panel de 'All Projects' inicial.

SmartParking Flow — Monitorización Inteligente

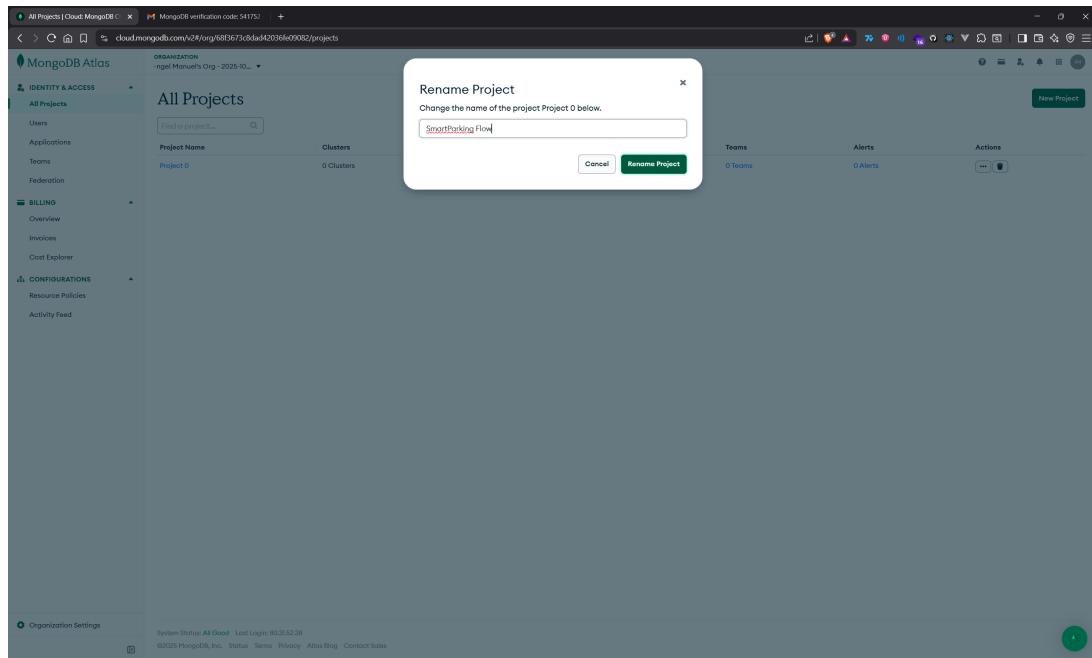


Figura B.12: Renombrando el proyecto a 'SmartParking Flow'.

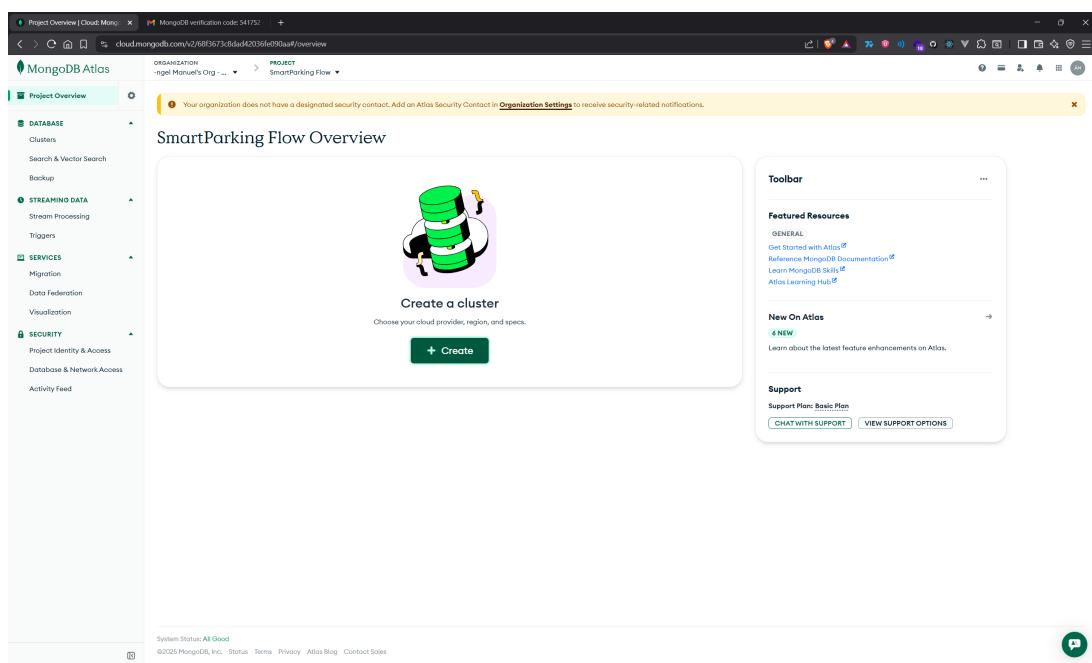


Figura B.13: Panel principal del proyecto 'SmartParking Flow'.

SmartParking Flow — Monitorización Inteligente

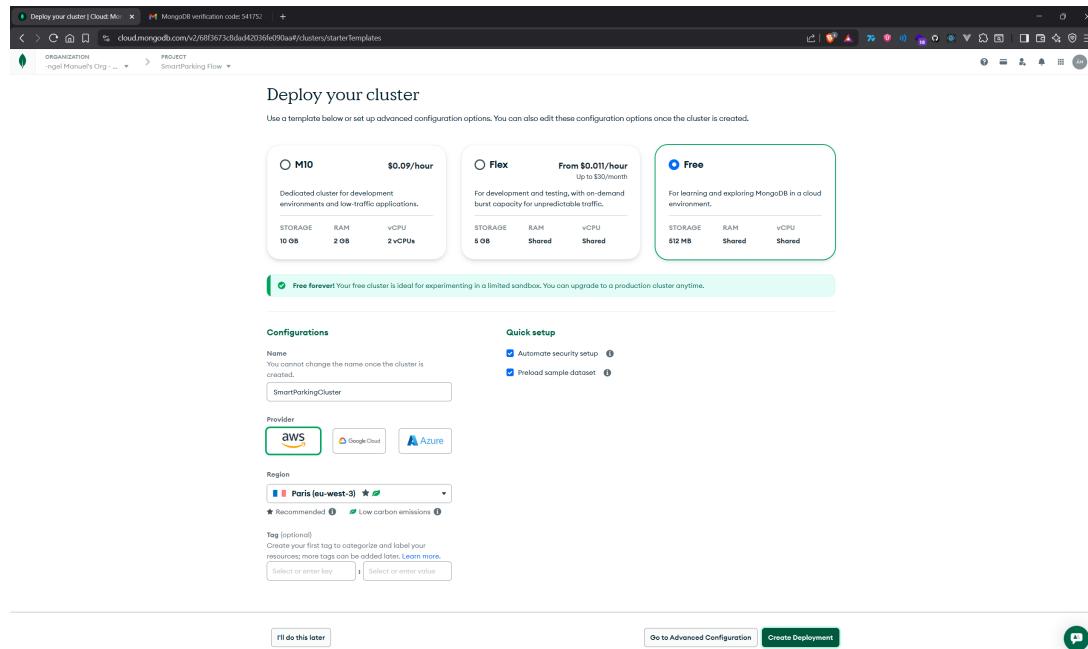


Figura B.14: Configuración del cluster gratuito (M0).

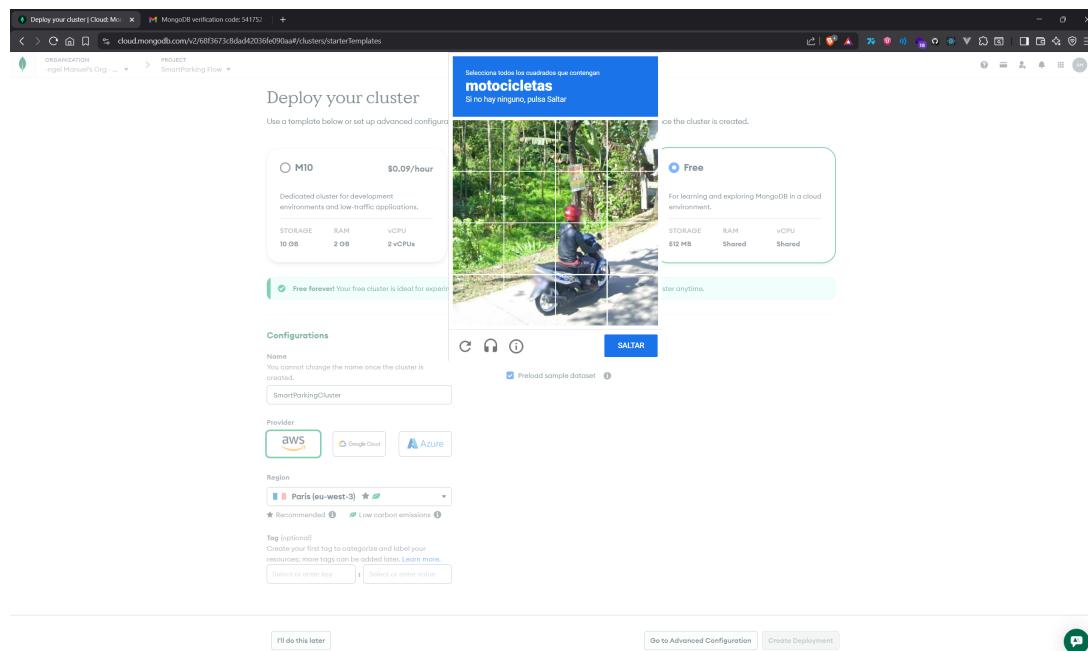


Figura B.15: Verificación Captcha para la creación del cluster.

SmartParking Flow — Monitorización Inteligente

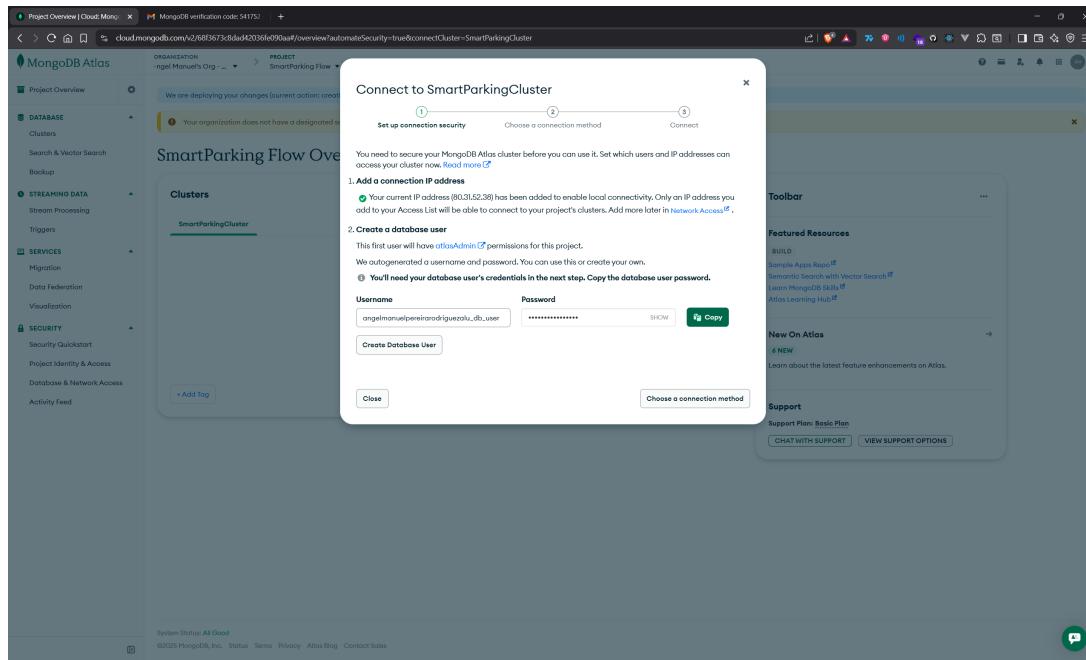


Figura B.16: Creación del usuario de la base de datos.

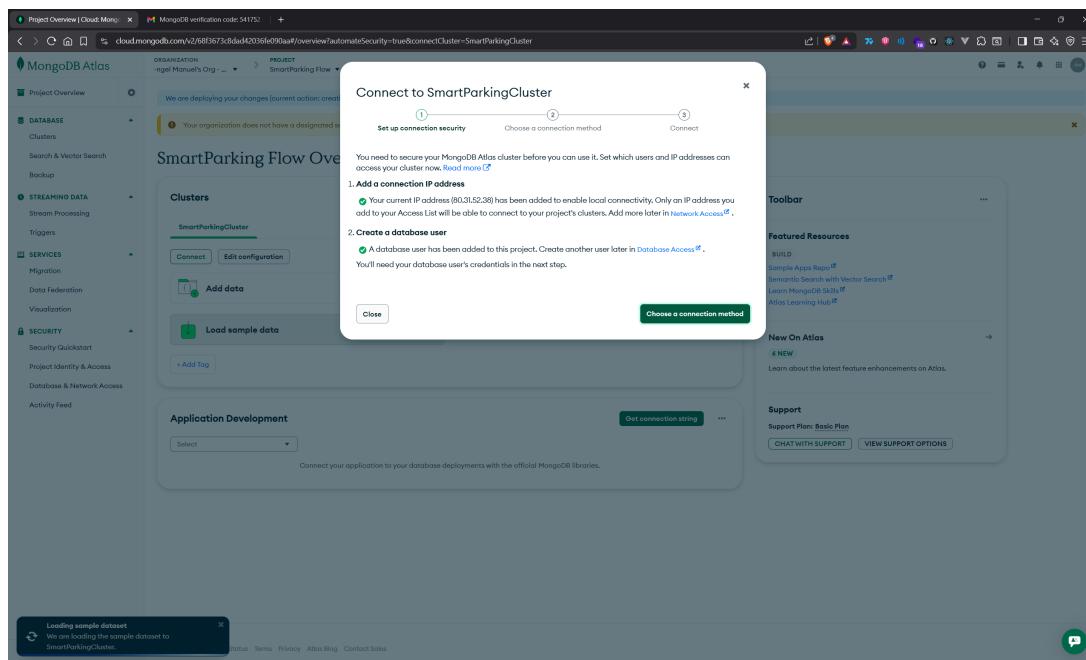


Figura B.17: Confirmación de creación de usuario.

SmartParking Flow — Monitorización Inteligente

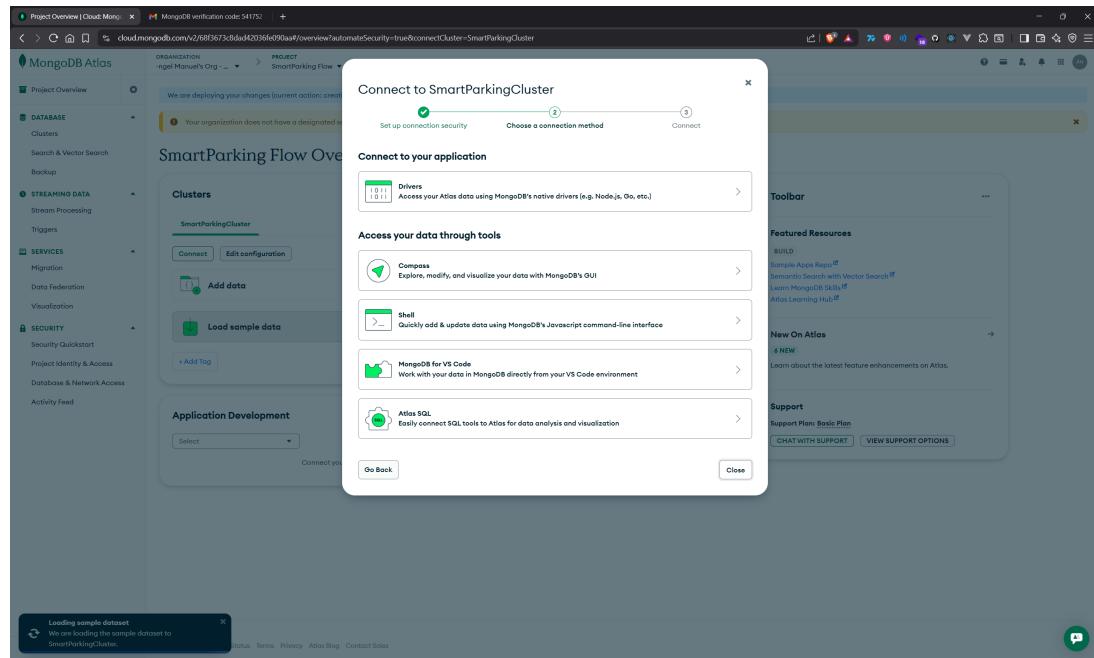


Figura B.18: Selección del método de conexión al cluster.

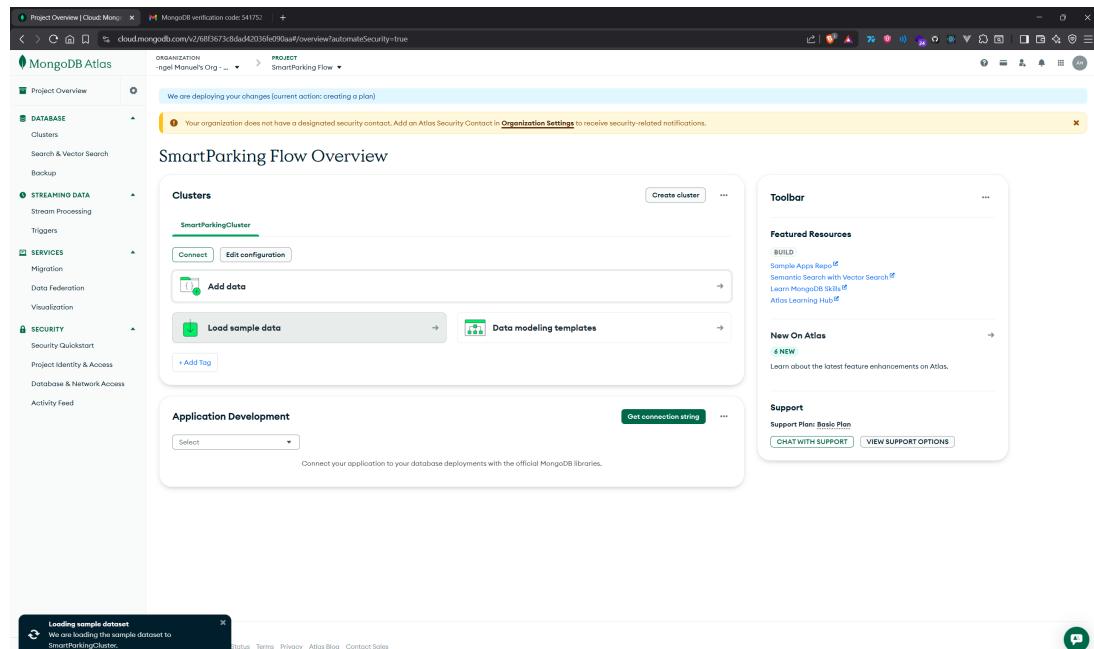


Figura B.19: Cargando datos de ejemplo (sample data).

SmartParking Flow — Monitorización Inteligente

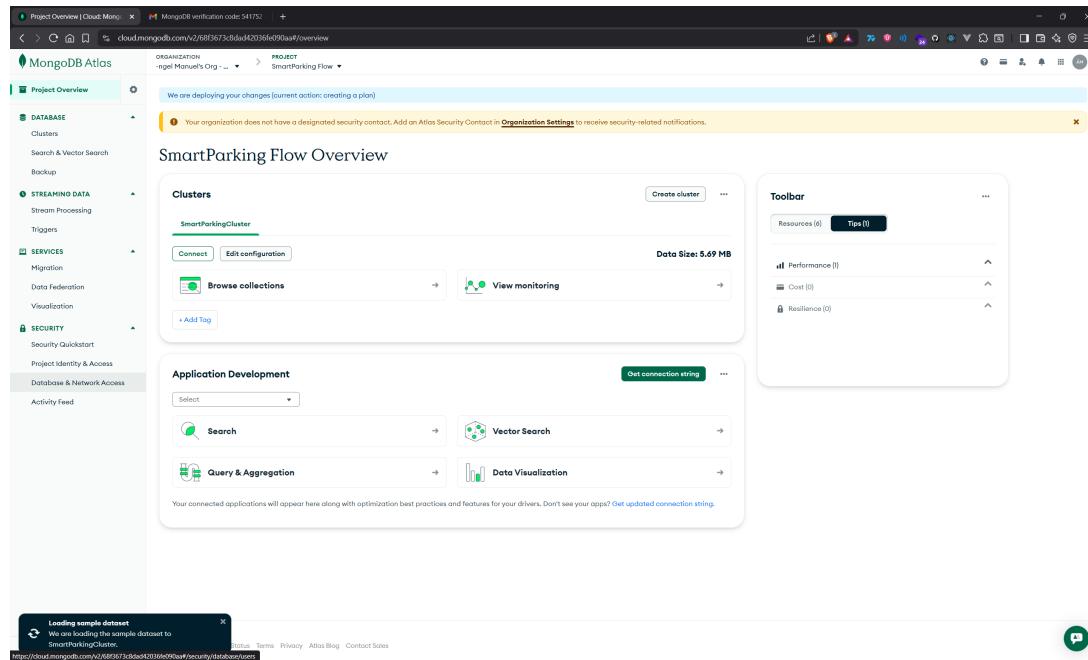


Figura B.20: Vista del cluster tras finalizar la carga de datos de ejemplo.

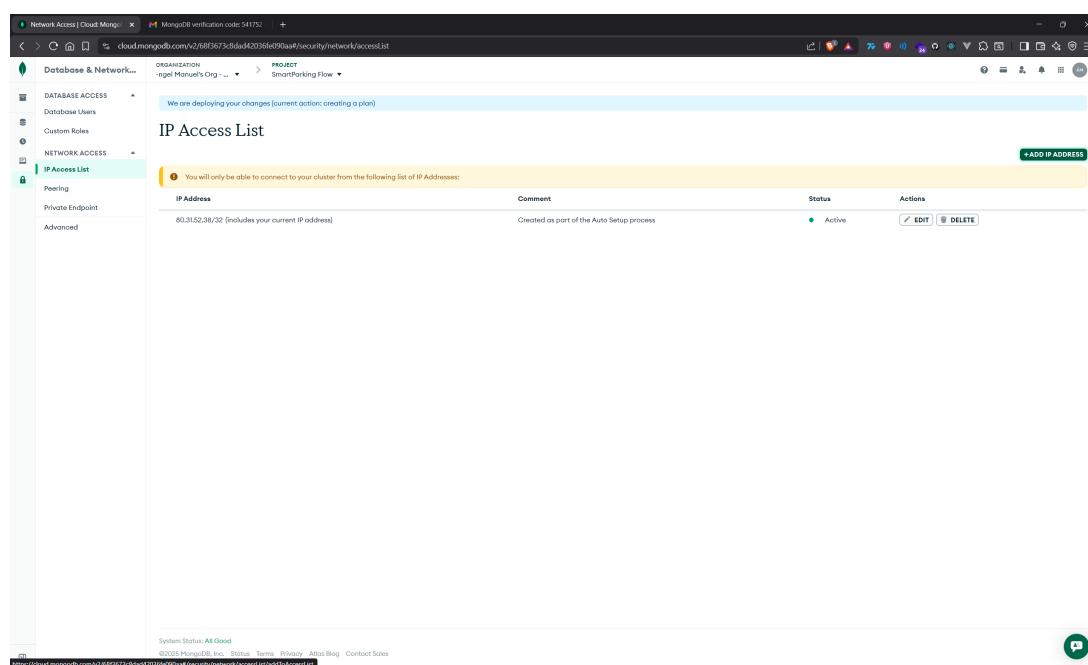


Figura B.21: Configuración de 'IP Access List' (IP actual).

SmartParking Flow — Monitorización Inteligente

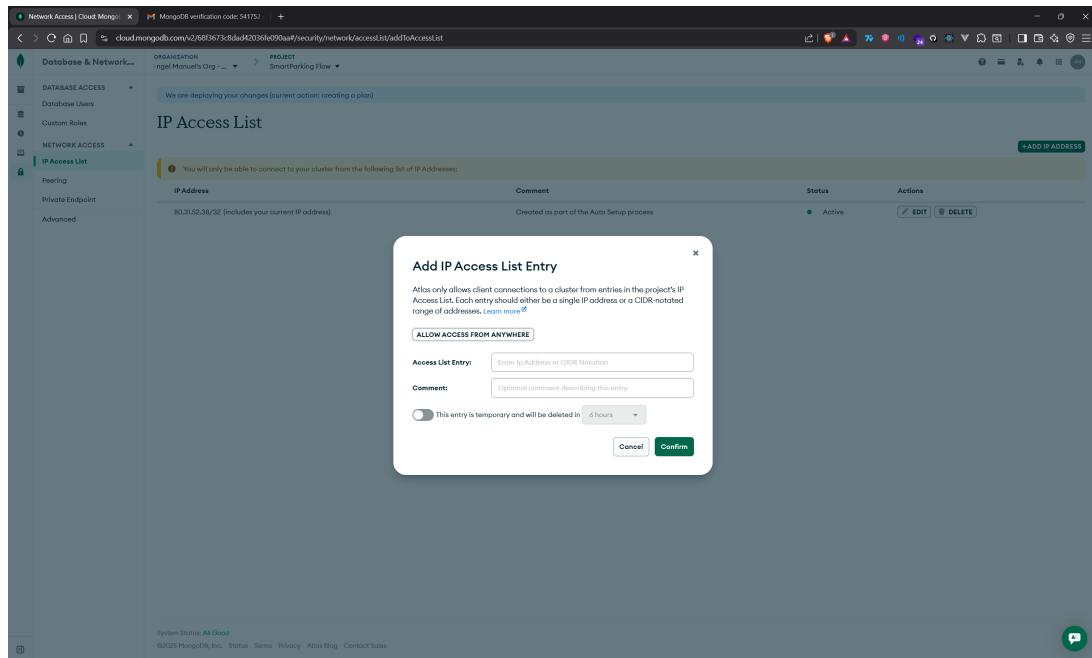


Figura B.22: Añadiendo una nueva entrada a 'IP Access List'.

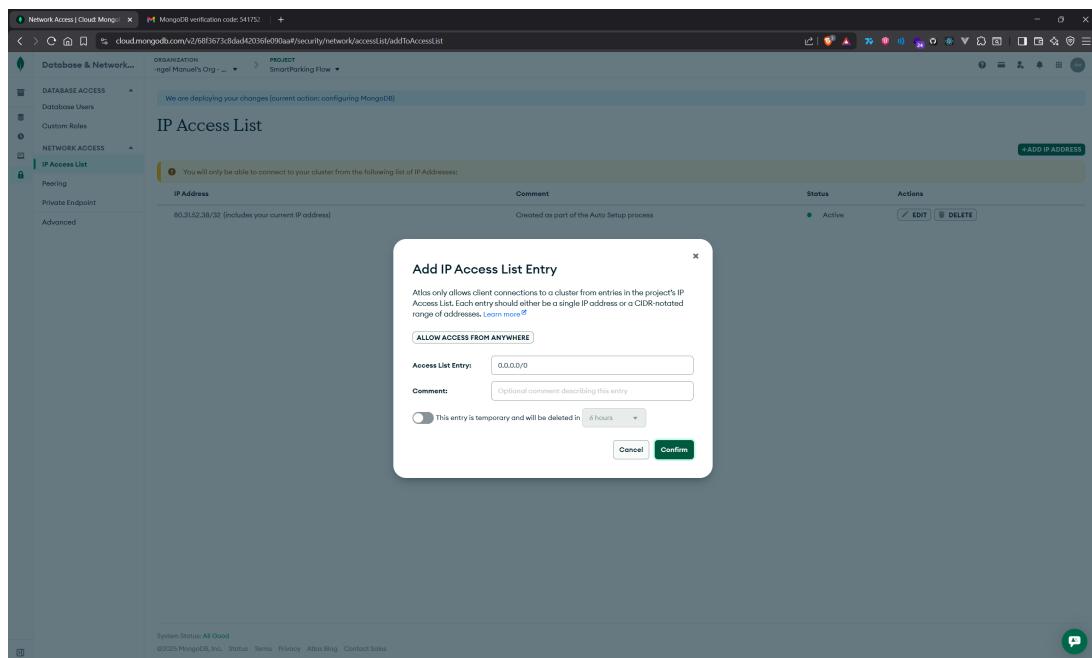


Figura B.23: Añadiendo '0.0.0.0/0' (Allow Access From Anywhere).

SmartParking Flow — Monitorización Inteligente

The screenshot shows the MongoDB Cloud interface for the 'SmartParking Flow' project. On the left, a sidebar lists 'Database & Network...' sections: DATABASE ACCESS, Database Users, Custom Roles, NETWORK ACCESS, and IP Access List (which is selected). The main content area is titled 'IP Access List'. It displays a table with two rows of IP addresses: '0.0.0.0/0 [includes your current IP address]' and '60.31.52.36/32 [includes your current IP address]'. Both entries have a status of 'Active' and 'Created as part of the Auto Setup process'. A yellow banner at the top states: 'We are deploying your changes [current action: creating a plan]'. A green button '+ADD IP ADDRESS' is located in the top right corner. At the bottom, there's a footer with 'System Status: All Good' and links to '©2025 MongoDB, Inc.', 'Status', 'Terms', 'Privacy', 'Atlas Blog', and 'Contact Sales'.

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

The screenshot shows the 'Project Overview' page for the 'SmartParking Flow' project in MongoDB Atlas. The left sidebar includes sections for Project Overview, DATABASE (Clusters, Search & Vector Search, Backup), STREAMING DATA (Stream Processing, Triggers), SERVICES (Migration, Data Federation, Visualization), SECURITY (Security Quickstart, Project Identity & Access, Database & Network Access), and Activity Feed. The main content area is titled 'SmartParking Flow Overview'. It features a 'Clusters' section with a 'SmartParkingCluster' entry, showing 'Sample data loaded successfully.' and a 'Data Size: 74.12 MB'. Below this is a 'Browse collections' button and a 'View monitoring' button. To the right is a 'Toolbar' with 'Resources (0)', 'Tips (0)', 'Performance (0)', 'Cost (0)', and 'Resilience (0)'. At the bottom, there's a footer with 'System Status: All Good' and links to '©2025 MongoDB, Inc.', 'Status', 'Terms', 'Privacy', 'Atlas Blog', and 'Contact Sales'.

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

SmartParking Flow — Monitorización Inteligente

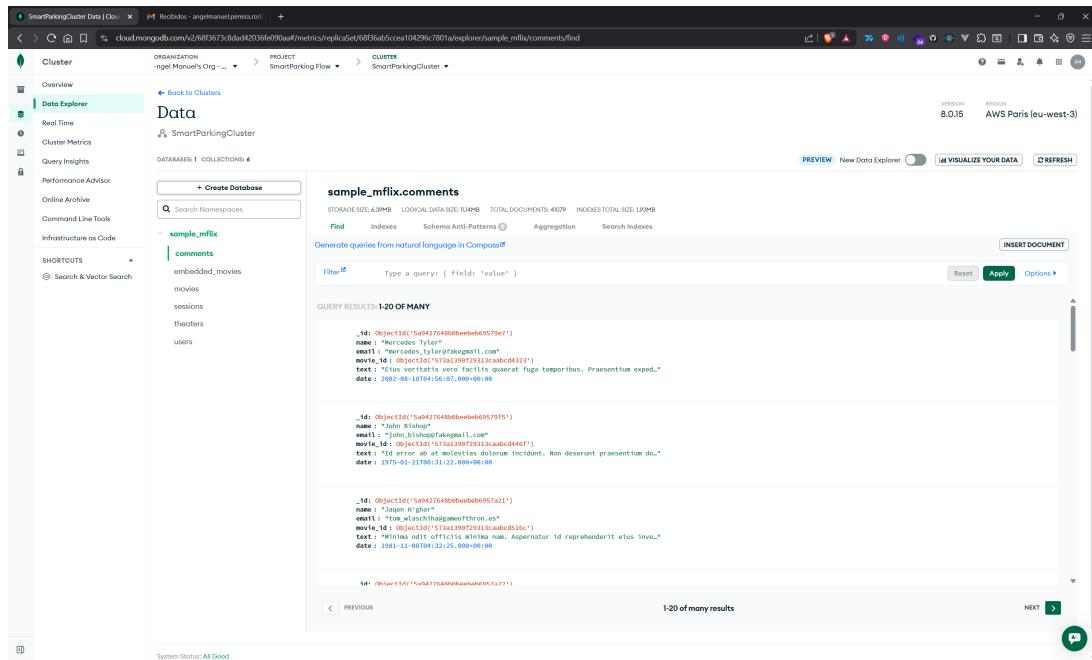


Figura B.26: Explorador de datos (Data Explorer) viendo 'sample_mflix.comments'.

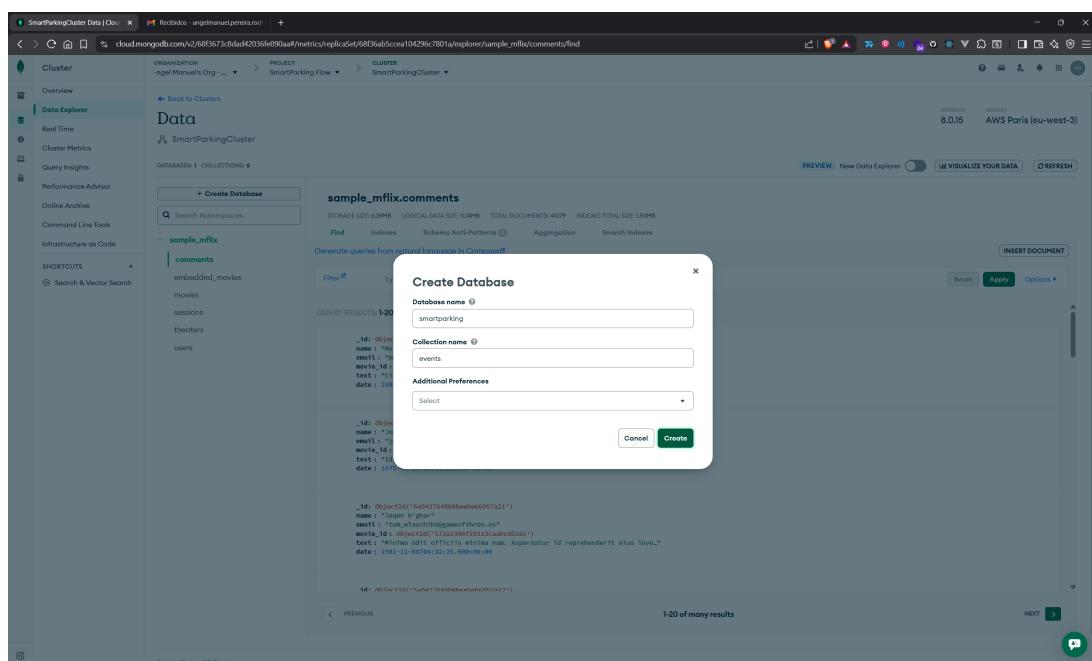


Figura B.27: Creación de la base de datos 'smartparking' y la colección 'events'.

SmartParking Flow — Monitorización Inteligente

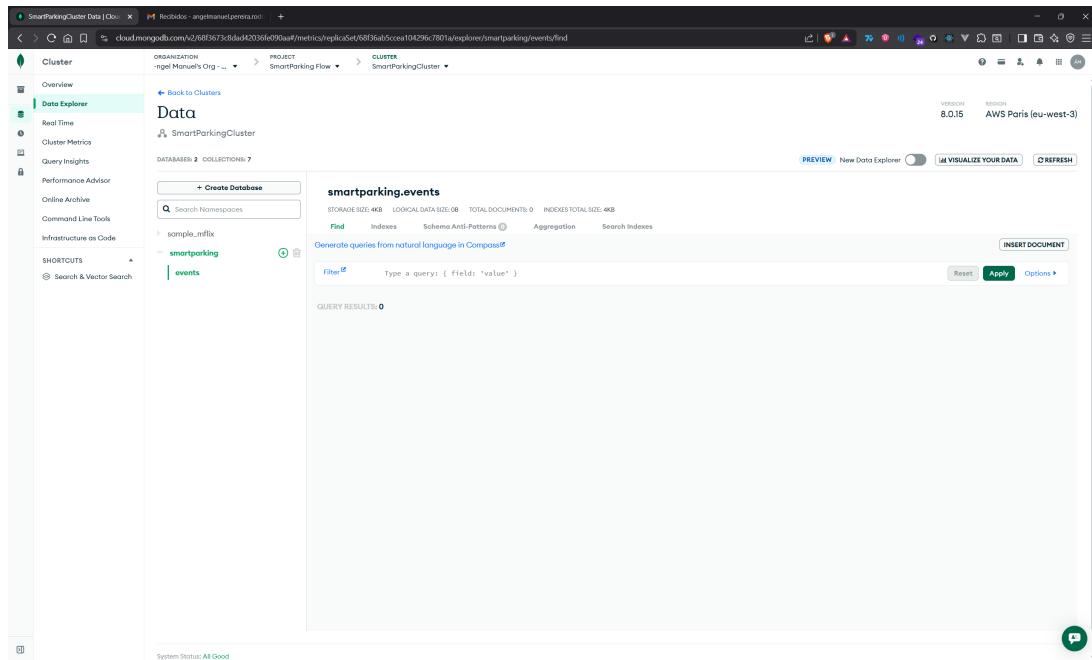


Figura B.28: Vista de la colección 'events' vacía.

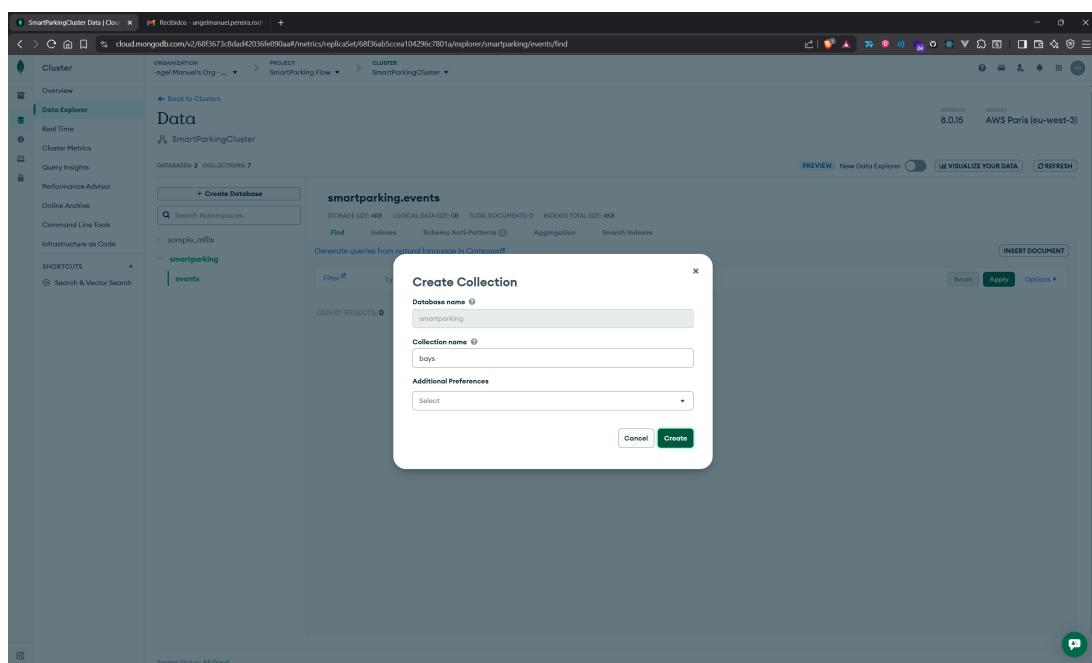
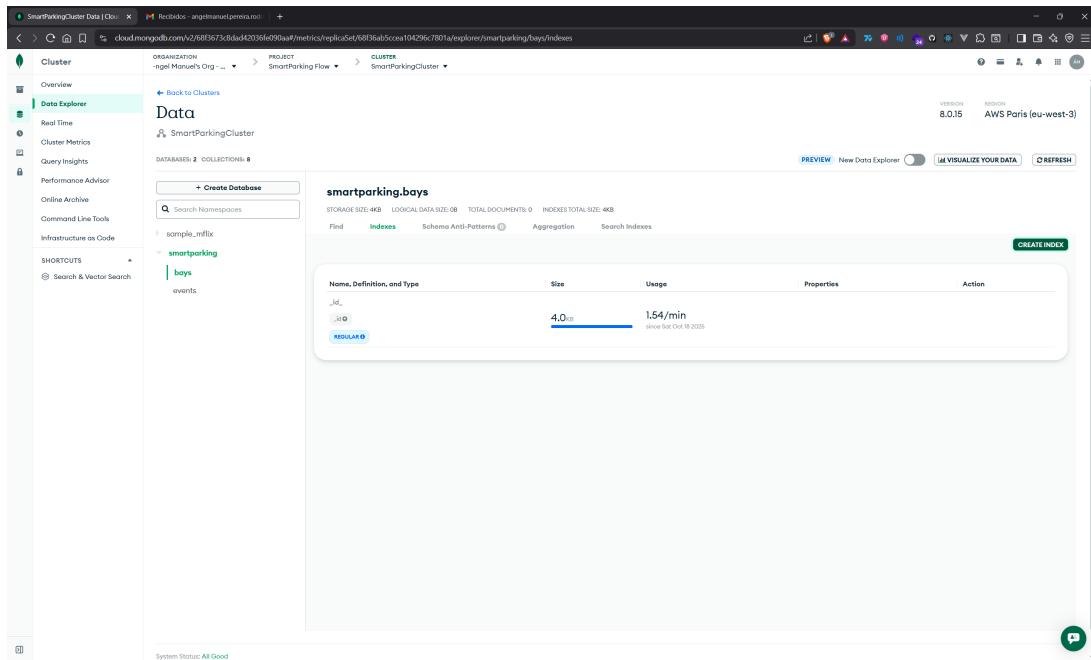


Figura B.29: Creación de la colección 'bays'.

SmartParking Flow — Monitorización Inteligente

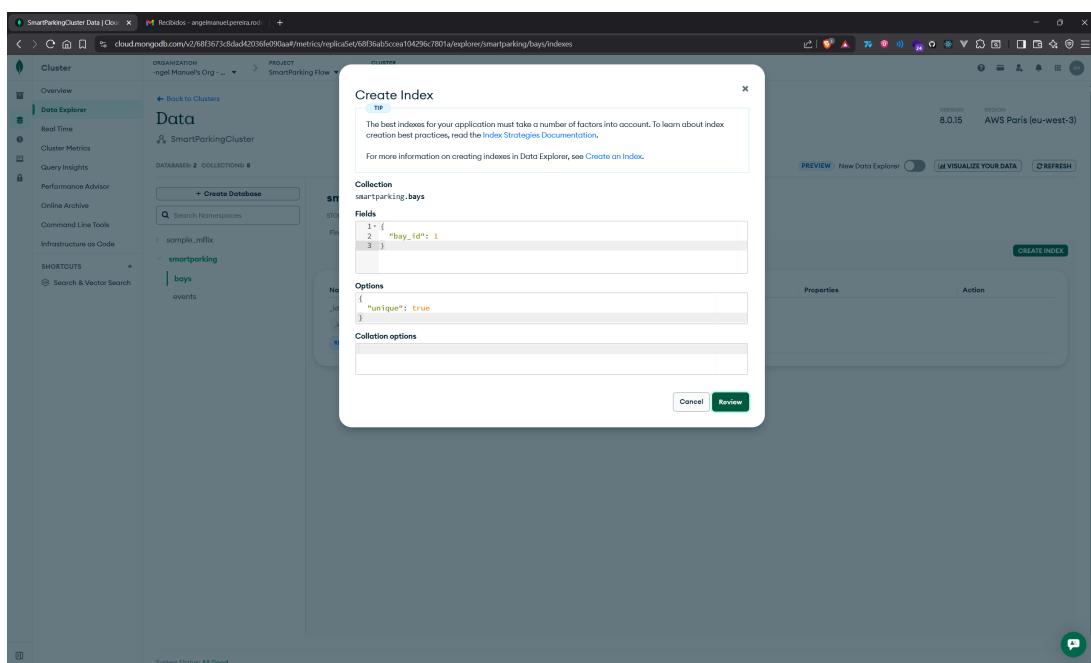


The screenshot shows the MongoDB Data Explorer interface. On the left sidebar, under the 'Data Explorer' section, the 'bays' collection is selected. The main panel displays the 'smartparking.bays' collection with one document listed:

Name, Definition, and Type	Size	Usage	Properties	Action
_id _id	4.0B	1.54/min since Oct 18 2020		

At the bottom right of the main panel, there is a green 'CREATE INDEX' button.

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



The screenshot shows the 'Create Index' dialog box overlaid on the MongoDB Data Explorer interface. The dialog has the following fields:

- Collection:** smartparking.bays
- Fields:** An array containing a single field: `1: { "bay_id": 1 }`
- Options:** An array containing an option: `["unique": true]`
- Collection options:** An empty input field.

At the bottom right of the dialog, there are 'Cancel' and 'Review' buttons.

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

SmartParking Flow — Monitorización Inteligente

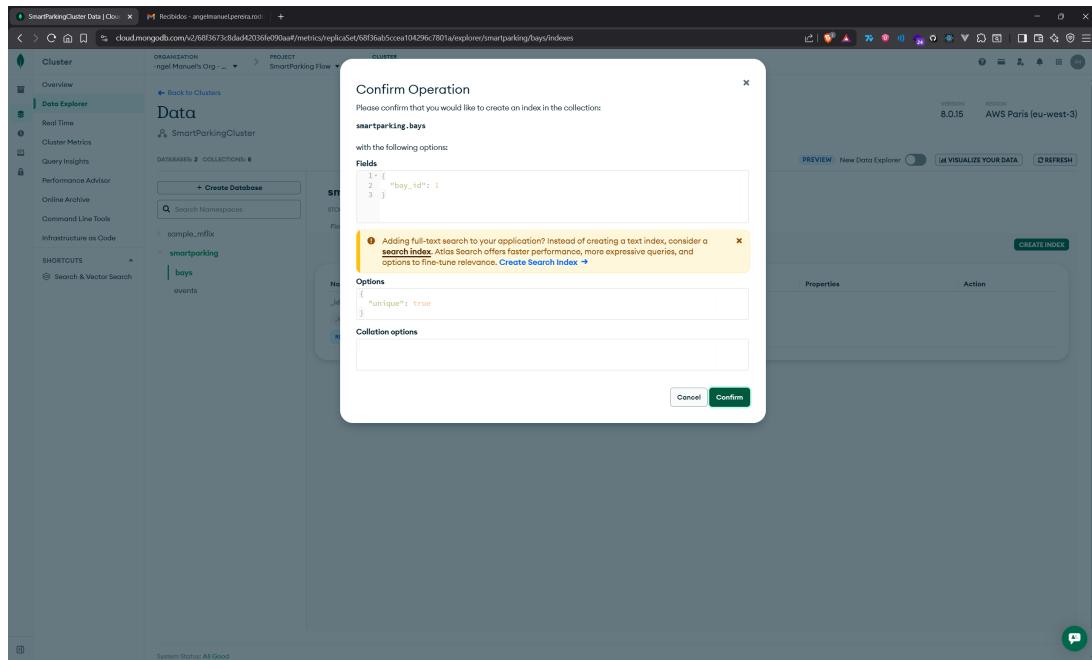


Figura B.32: Confirmación de creación de índice en 'bays' sobre 'bay_id'.

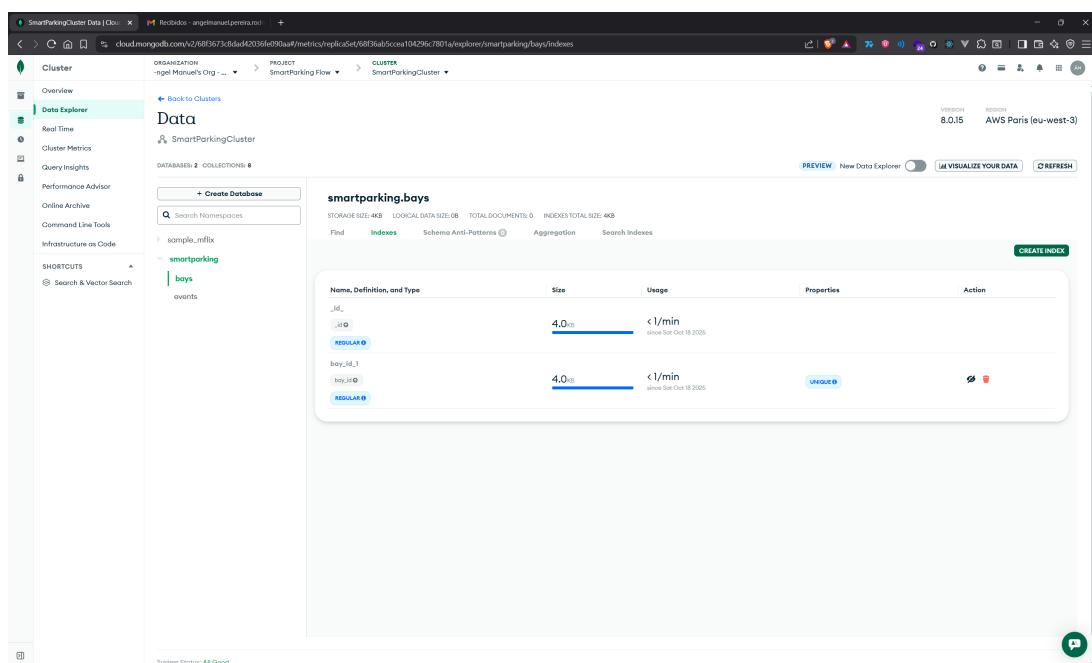


Figura B.33: Índice 'bay_id_1' creado y listado en la colección 'bays'.

SmartParking Flow — Monitorización Inteligente

The screenshot shows the MongoDB Data Explorer interface. The left sidebar is titled 'Data Explorer' and includes sections for Overview, Real Time, Cluster Metrics, Query Insights, Performance Advisor, Online Archive, Command Line Tools, Infrastructure as Code, and SHORTCUTS. The main area is titled 'smartparking.events' and shows the 'Indexes' tab selected. It displays a table with one index entry:

Name, Definition, and Type	Size	Usage	Properties	Action
<code>_id_</code>	4.0 kB	< 1/min		REGULAR

At the bottom right of the main area, there is a green 'CREATE INDEX' button.

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

The screenshot shows the 'Create Index' dialog box overlaid on the MongoDB Data Explorer interface. The dialog has a title 'Create Index' and a tip section: 'The best indexes for your application must take a number of factors into account. To learn about index creation best practices, read the [Index Strategies Documentation](#).'. Below this, it says 'For more information on creating indexes in Data Explorer, see [Create an Index](#)'. The 'Collection' dropdown is set to 'smartparking.events'. The 'Fields' section contains the following code:

```
1: {  
2:   "bay_id": 1,  
3:   "last_event_ts": -1  
4: }
```

The 'Options' section contains:

```
[  
  "name": "bay_event_ts_desc_idx"  
]
```

The 'Collection options' section is empty. At the bottom right of the dialog are 'Cancel' and 'Review' buttons.

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

SmartParking Flow — Monitorización Inteligente

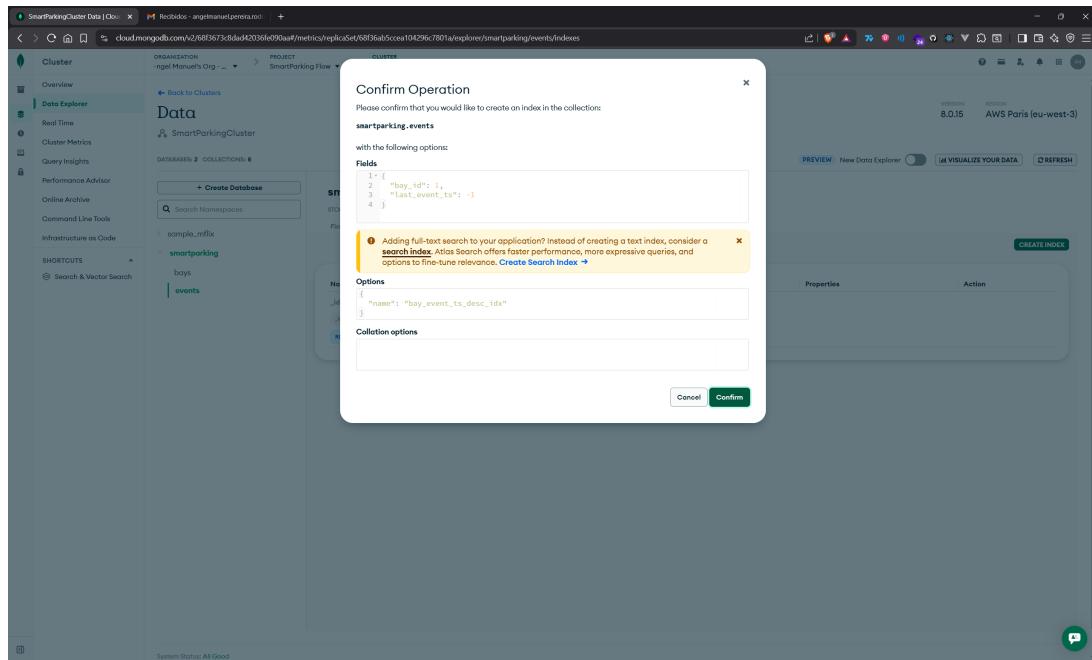


Figura B.36: Confirmación de creación de índice en 'events'.

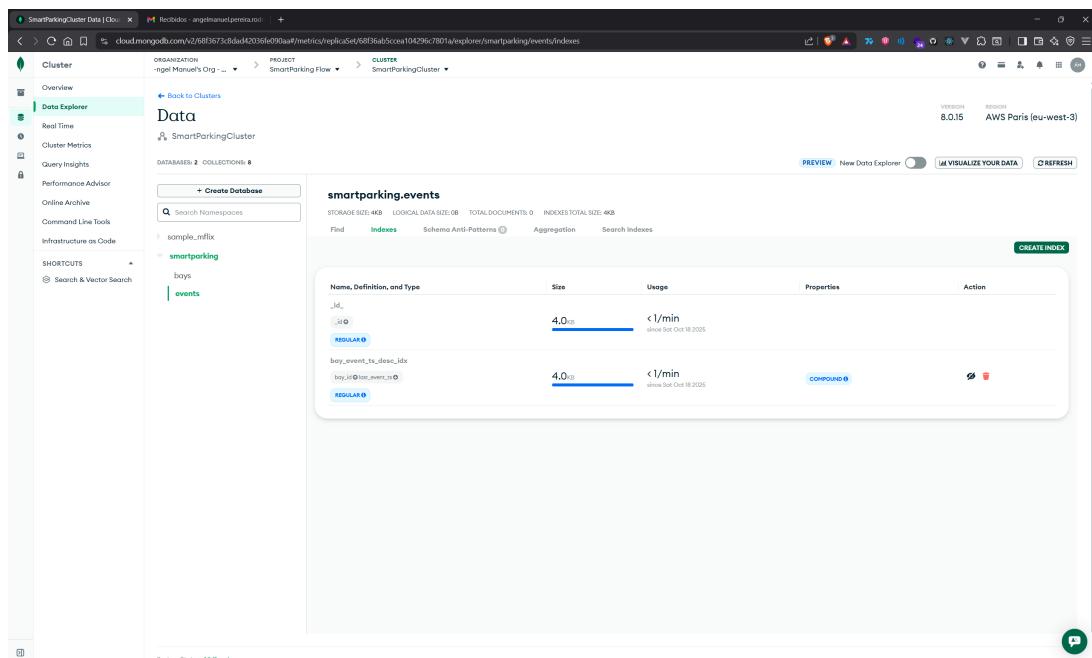


Figura B.37: Índice 'bay_event_ts_desc_idx' creado en la colección 'events'.

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 the application via Drivers (MongoDB native drivers), Compose (MongoDB's GUI), Shell (MongoDB command-line interface), MongoDB for VS Code (MongoDB extension for VS Code), and Atlas SQL (MongoDB's SQL interface). 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

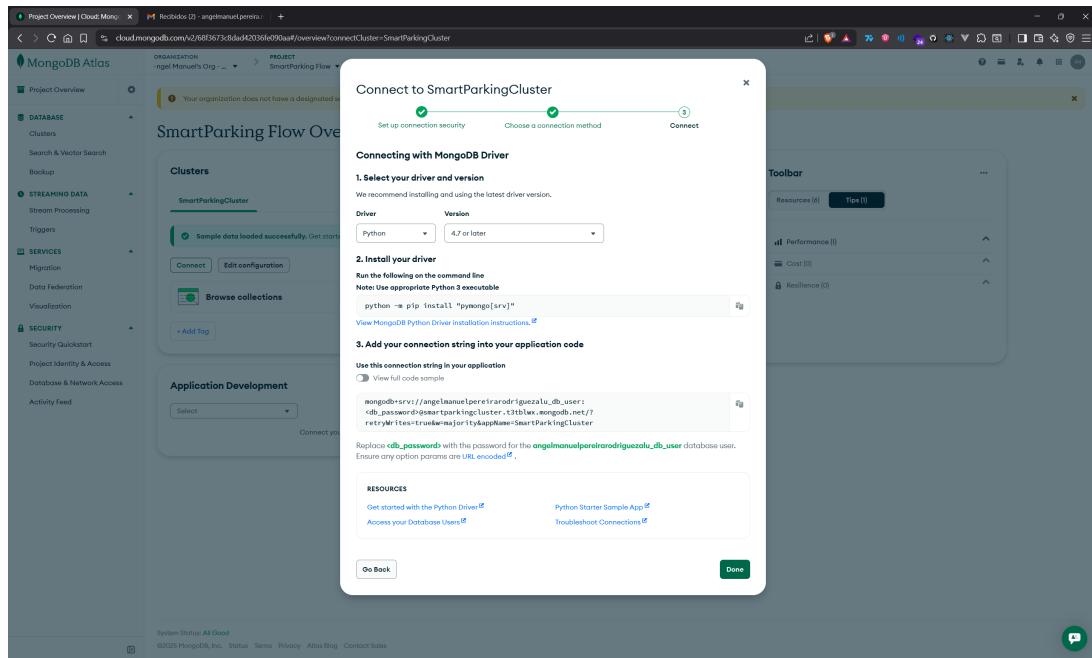


Figura B.40: Obteniendo la cadena de conexión (Connection String) para Python.

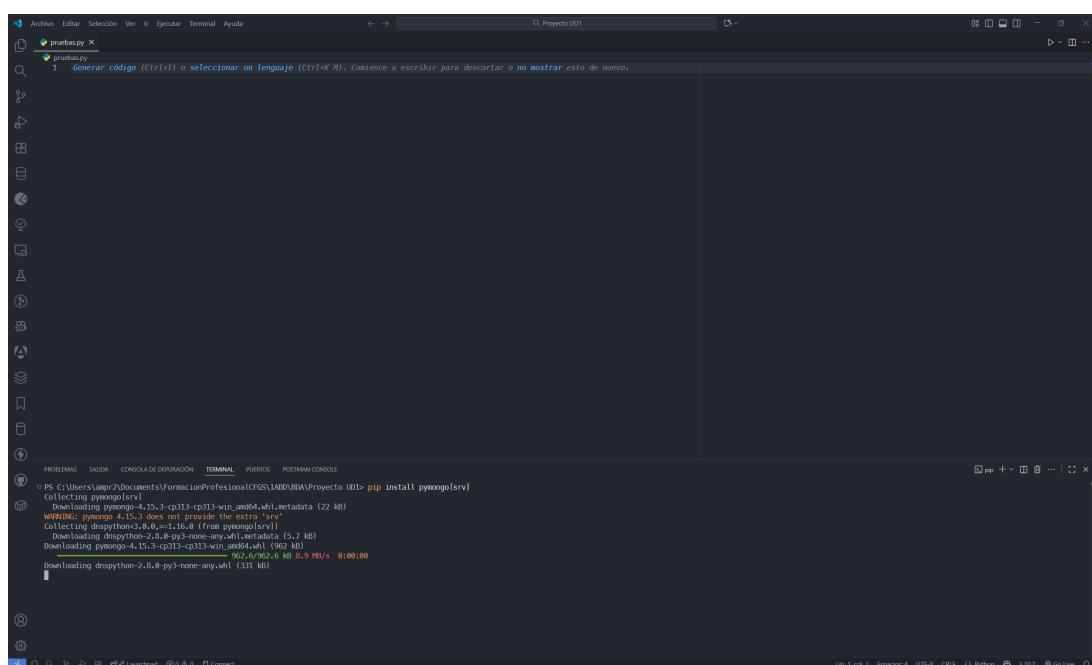


Figura B.41: Instalando 'pymongo[srv]' en el anfitrión (Windows).

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 the terminal window showing the command 'pip install pymongo[srv]' being run, followed by the output of the package installation.

```
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 = {
    "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.metadata (22 kB)
    WARNING: pymongo-4.15.3 does not provide the extra 'srv'
  Collecting dnspython==2.8.0->1.16.4 (from pymongo[srv])
    Downloading dnspython-2.8.0-py3-none-any.whl.metadata (5.7 kB)
  Downloading pymongo-4.15.3-py3-none-any.whl.metadata (26.9 kB)
  Downloading dnspython-2.8.0-py3-none-any.whl (331 kB)
  Installing dnspython-2.8.0
Successfully installed dnspython-2.8.0 pymongo-4.15.3
PS C:\Users\apr2\Documents\FormacionProfesional\CFGS\IABD\BDA\Proyecto UD1> [REDACTED]
```

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 shows the command 'python pruebas.py' being run, followed by the output of the script execution. The output indicates successful connection to MongoDB Atlas.

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

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

SmartParking Flow — Monitorización Inteligente

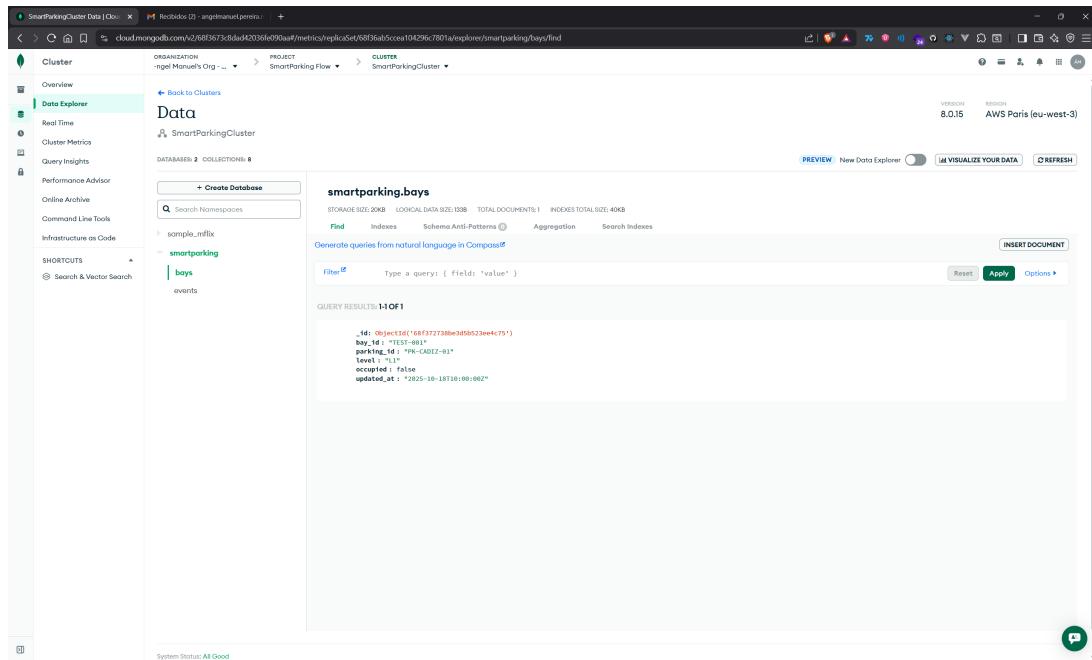


Figura B.44: Documento de prueba 'TEST-001' insertado en 'smartparking.bays'.

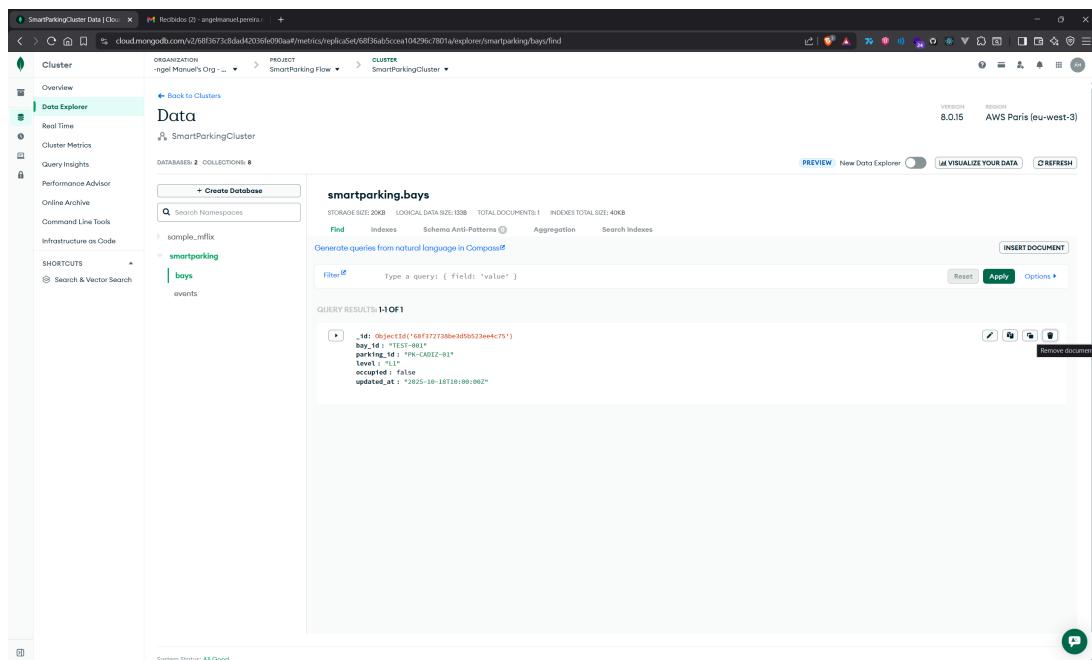


Figura B.45: Documento 'TEST-001' marcado para eliminación parte 1.

SmartParking Flow — Monitorización Inteligente

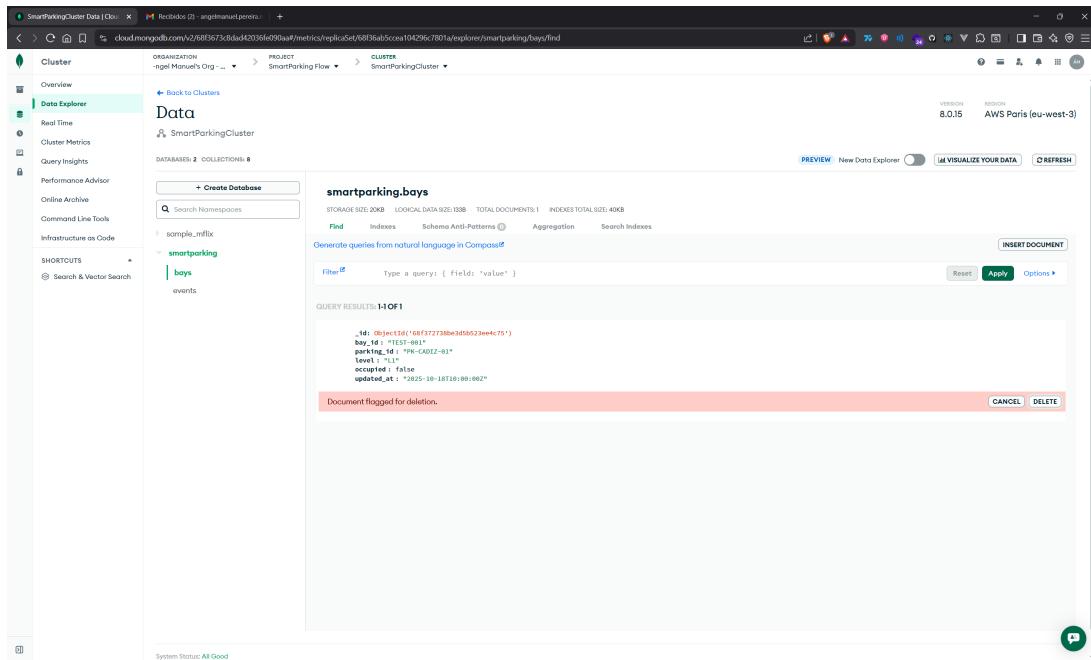


Figura B.46: Documento 'TEST-001' marcado para eliminación parte 2.

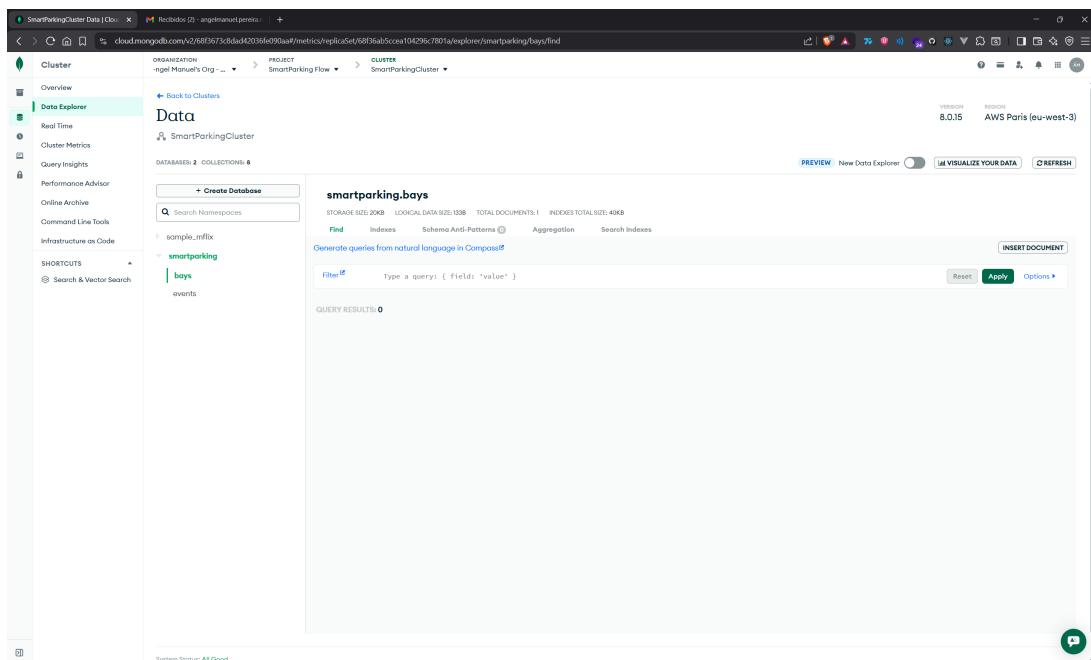
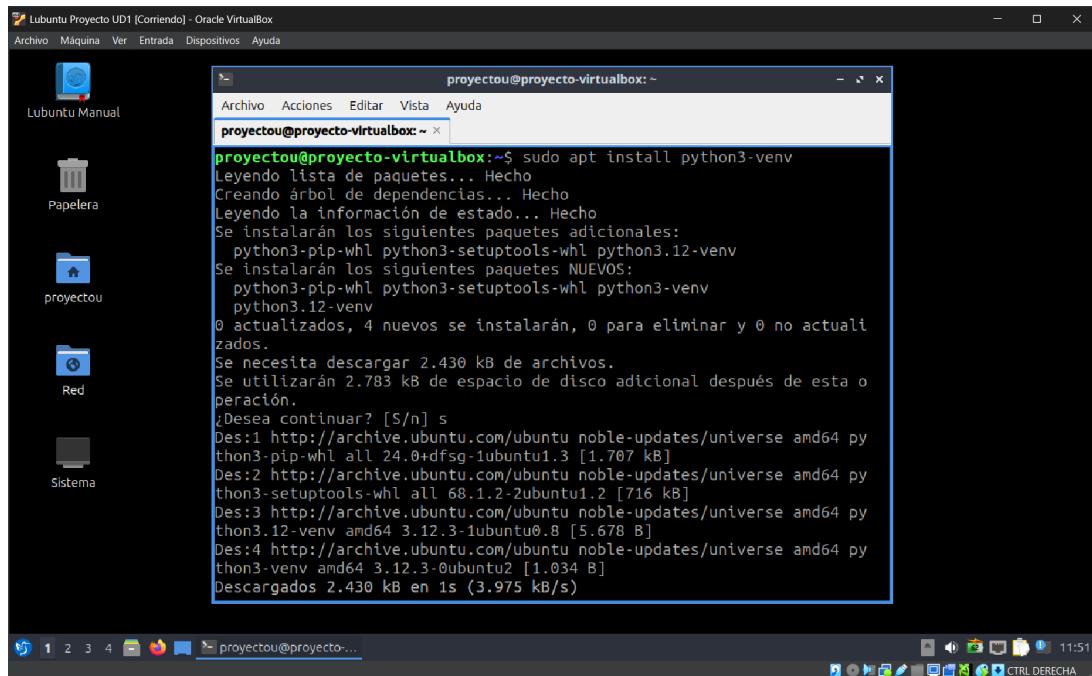


Figura B.47: Colección 'smartparking.bays' vacía tras borrado.

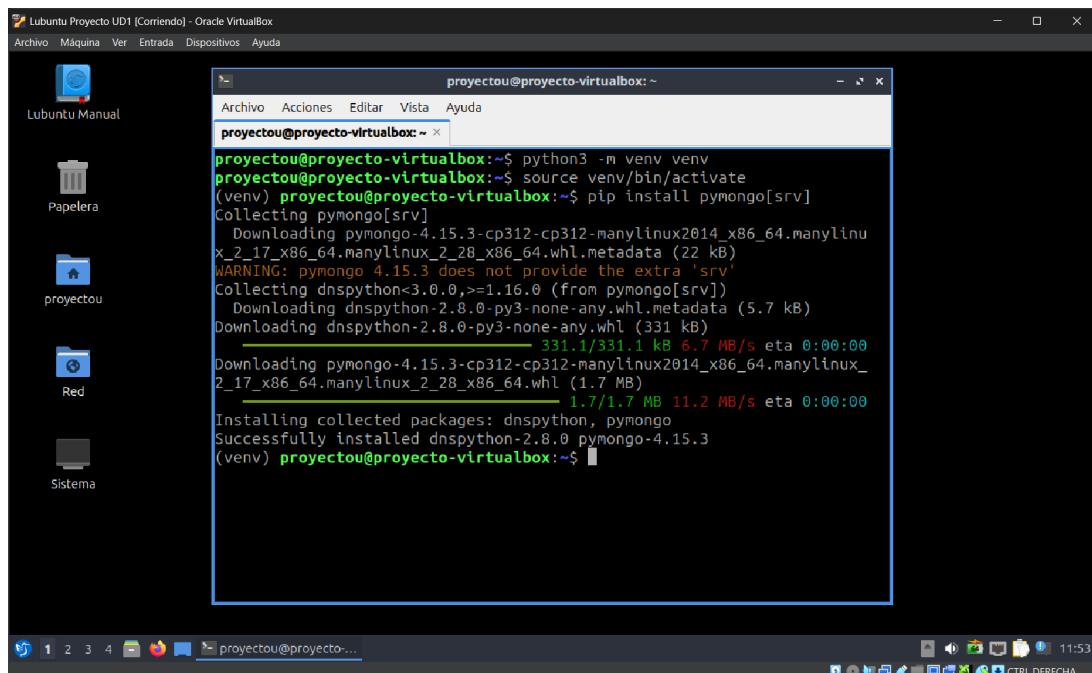


```

proyecto@projeto-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.



```

proyecto@projeto-virtualbox:~$ python3 -m venv venv
proyecto@projeto-virtualbox:~$ source venv/bin/activate
(venv) proyecto@projeto-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@projeto-virtualbox:~$ 

```

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

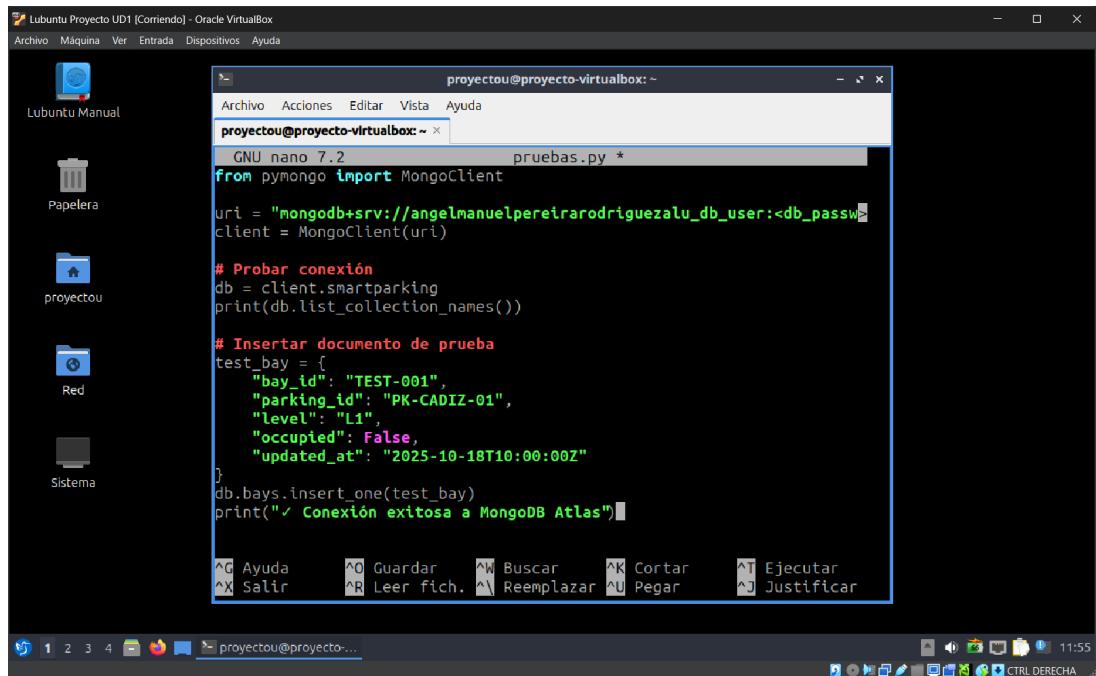


Figura B.50: Editando el script de prueba `pruebas.py` en Lubuntu (nano).

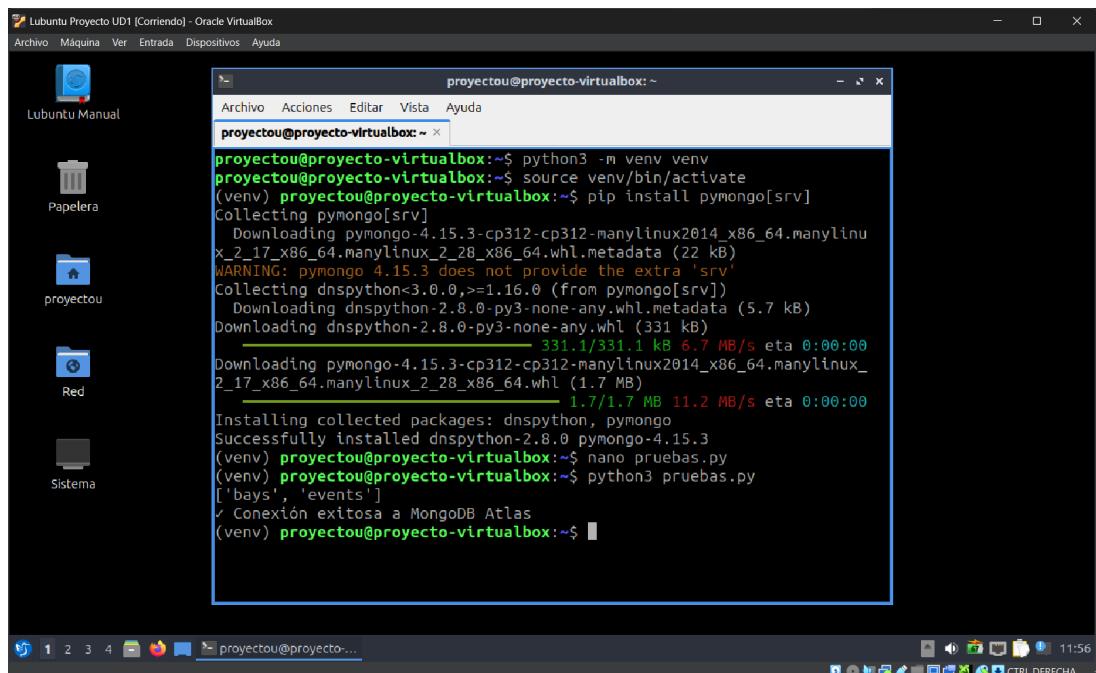


Figura B.51: Resultado de la ejecución del script de prueba en Lubuntu.

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.

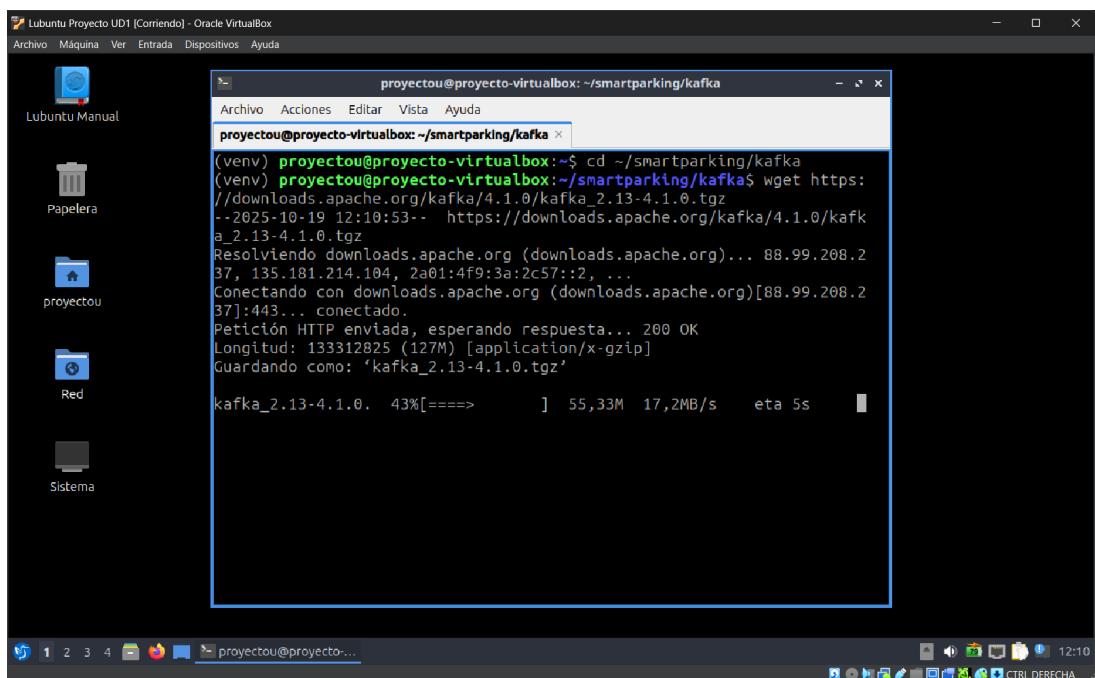
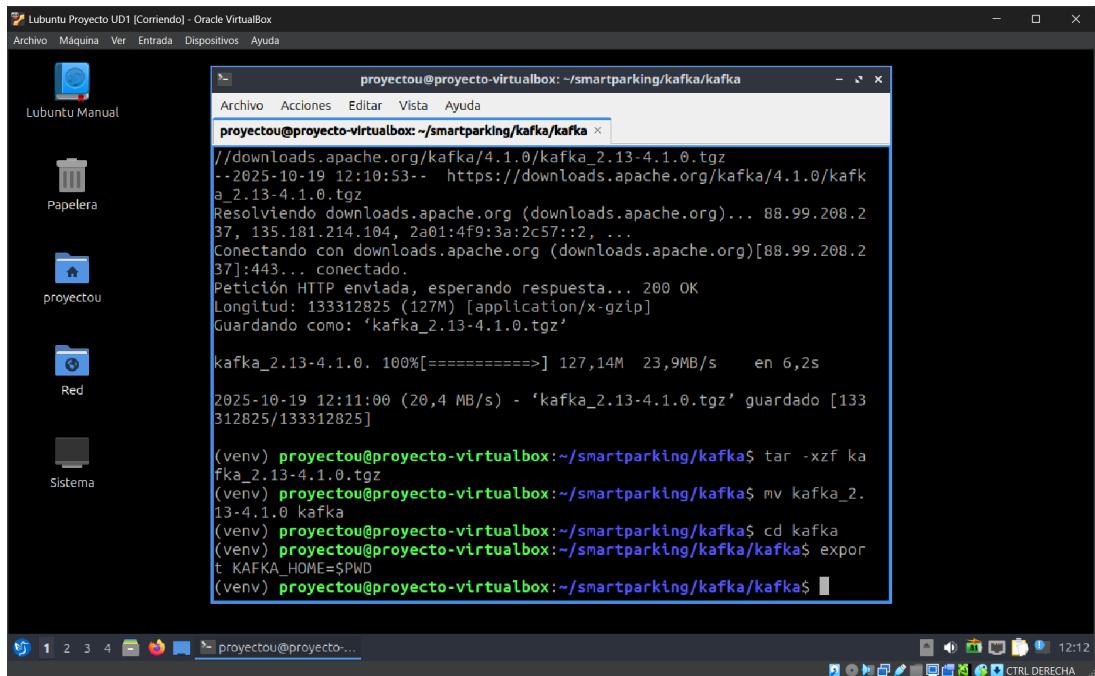


Figura C.1: Descarga de Apache Kafka ('wget').



```

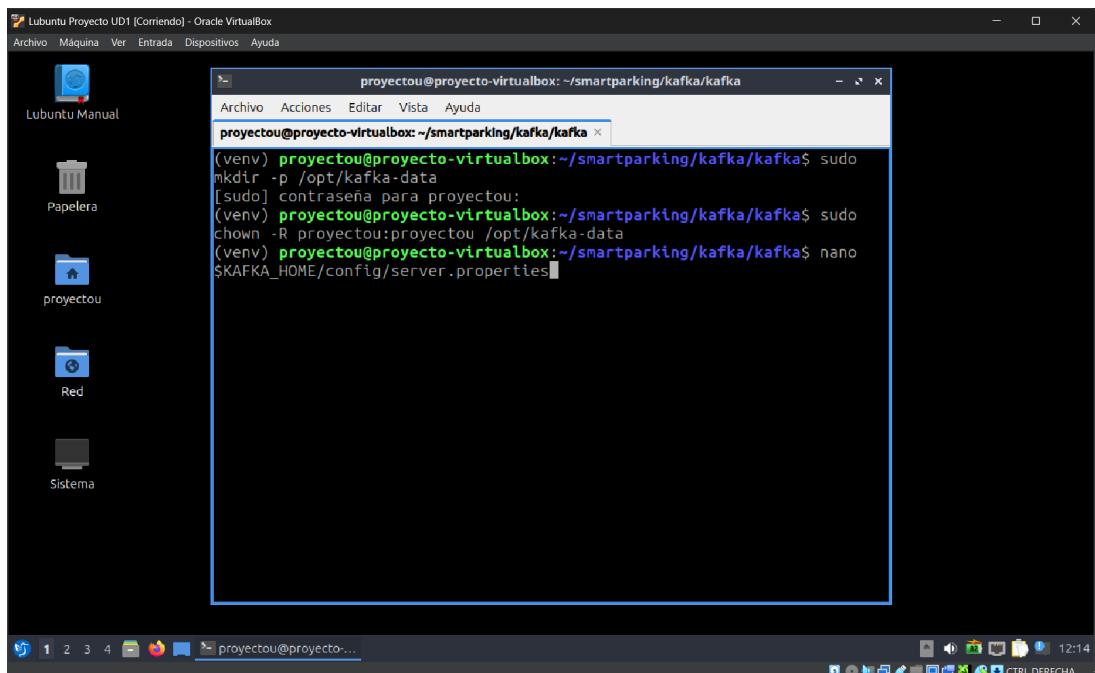
Lubuntu Proyecto UD1 [Corriendo] - Oracle VirtualBox
Archivo Máquina Ver Entrada Dispositivos Ayuda
proyectou@projeto-virtualbox:~/smartparking/kafka/kafka ~
proyectou@projeto-virtualbox:~/smartparking/kafka/kafka ~
//downloads.apache.org/kafka/4.1.0/kafka_2.13-4.1.0.tgz
--2025-10-19 12:10:53-- https://downloads.apache.org/kafka/4.1.0/kafka_2.13-4.1.0.tgz
Resolviendo downloads.apache.org (downloads.apache.org)... 88.99.208.2
37, 135.181.214.104, 2a01:4f9:3a:2c57::2, ...
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) proyectou@projeto-virtualbox:~/smartparking/kafka$ tar -xzf ka
fka_2.13-4.1.0.tgz
(venv) proyectou@projeto-virtualbox:~/smartparking/kafka$ mv kafka_2.
13-4.1.0 kafka
(venv) proyectou@projeto-virtualbox:~/smartparking/kafka$ cd kafka
(venv) proyectou@projeto-virtualbox:~/smartparking/kafka/kafka$ expor
t KAFKA_HOME=$PWD
(venv) proyectou@projeto-virtualbox:~/smartparking/kafka/kafka$ █

```

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

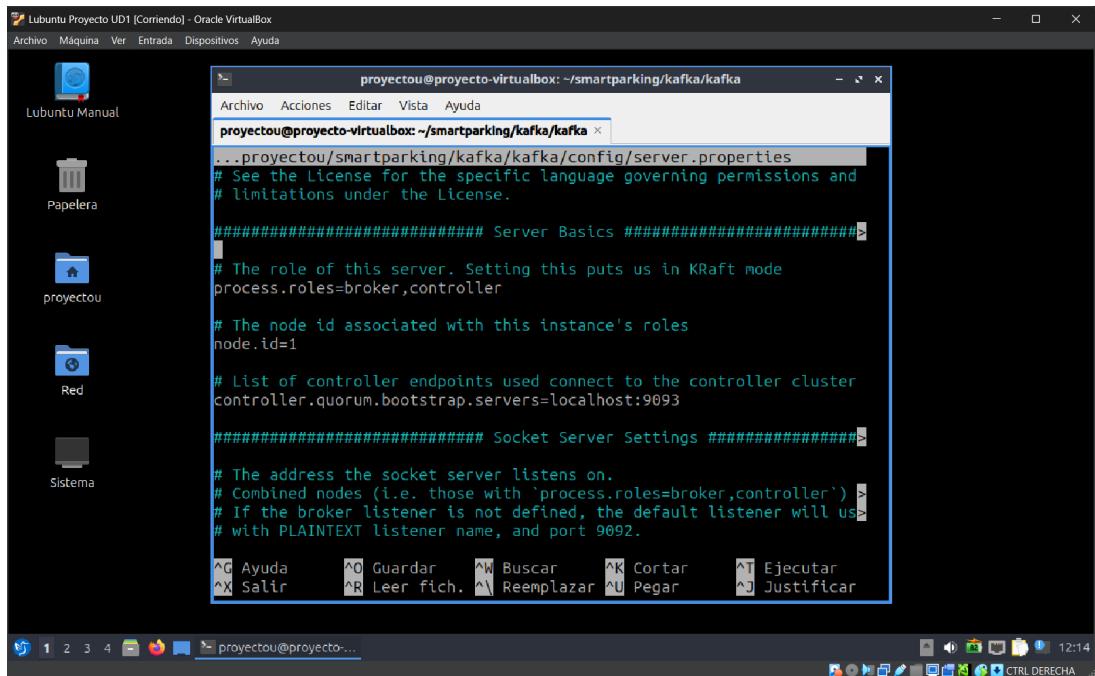


```

Lubuntu Proyecto UD1 [Corriendo] - Oracle VirtualBox
Archivo Máquina Ver Entrada Dispositivos Ayuda
proyectou@projeto-virtualbox:~/smartparking/kafka/kafka ~
proyectou@projeto-virtualbox:~/smartparking/kafka/kafka ~
(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█

```

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



```
proyectou@projeto-virtualbox:~/smartparking/kafka/kafka$ cat 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

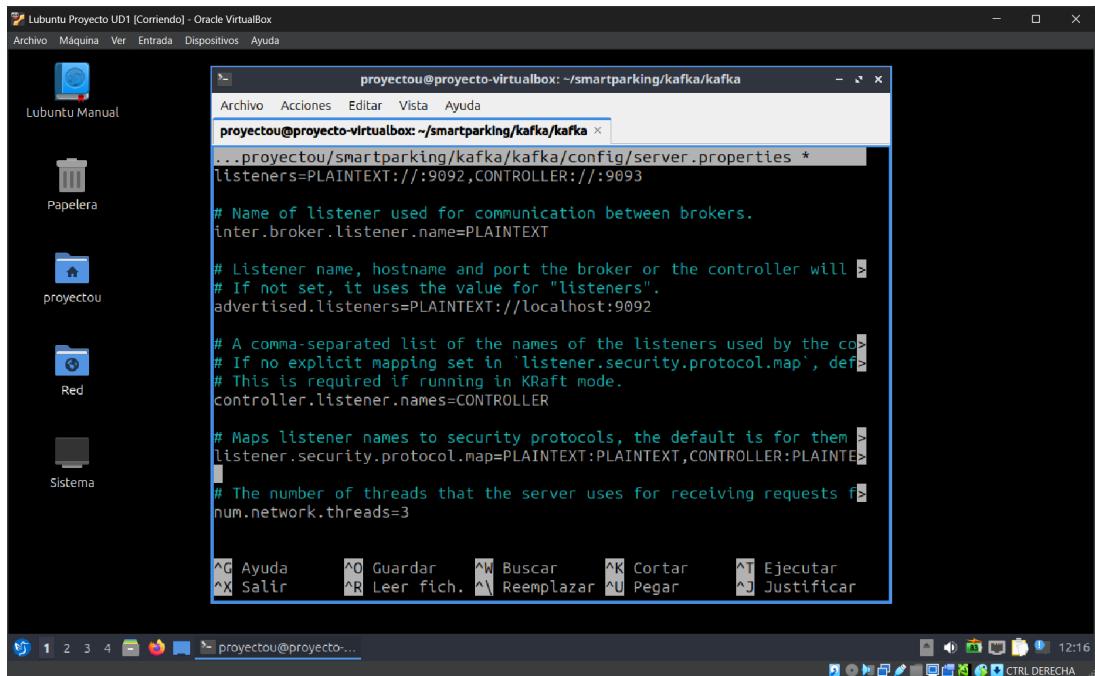
# The node id associated with this instance's roles
node.id=1

# 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 use
# 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 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 use
# If not set, it uses the value for "listeners".
advertised.listeners=PLAINTEXT://localhost:9092

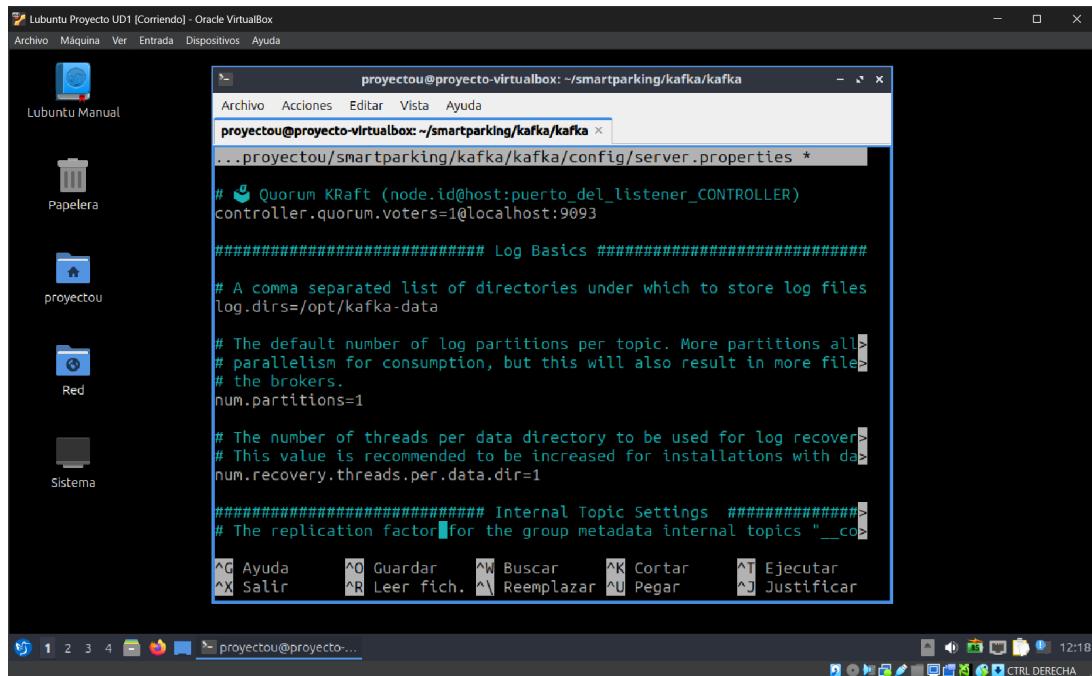
# A comma-separated list of the names of the listeners used by the controller
# If no explicit mapping set in 'listener.security.protocol.map', default
# This is required if running in KRaft mode.
controller.listener.names=CONTROLLER

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

# The number of threads that the server uses for receiving requests from clients
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

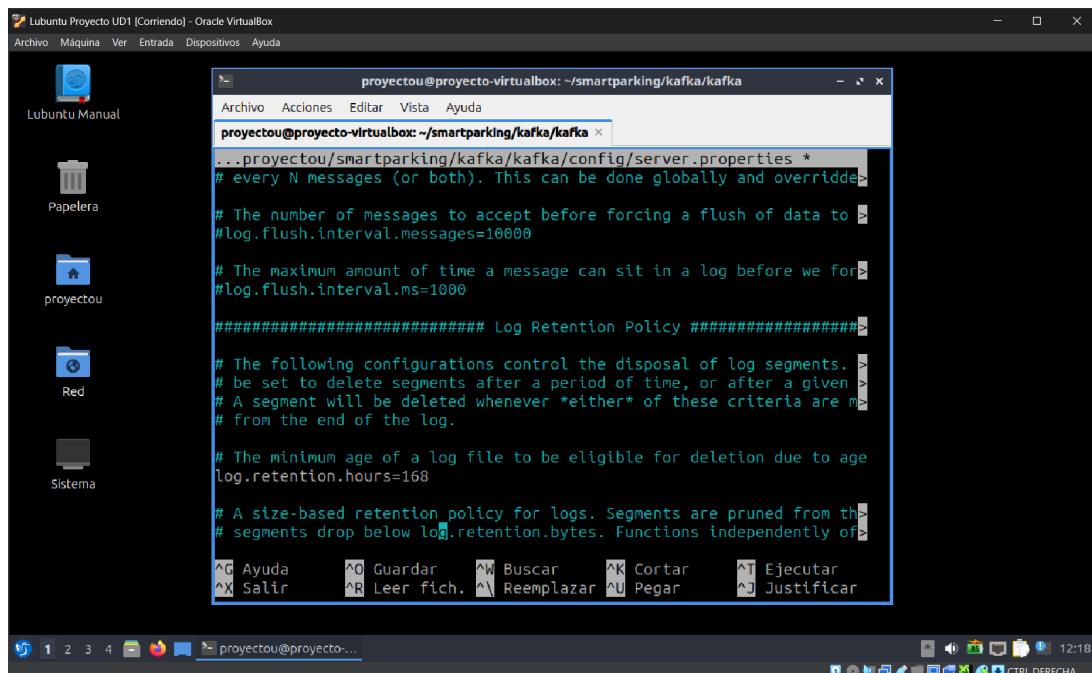
#####
# Internal Topic Settings
# The replication factor for the group metadata internal topics "_co"
# and _schemas
replication.factor=1

# 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

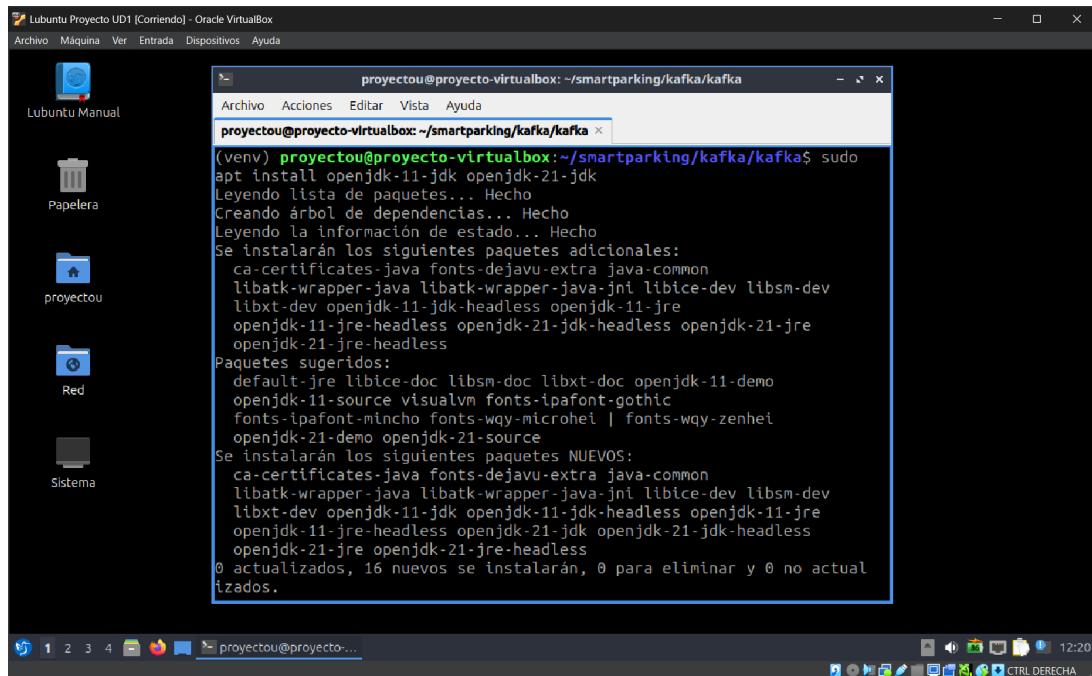
# The maximum amount of time a message can sit in a log before we force a
#log.flush.interval.ms=1000

#####
# 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

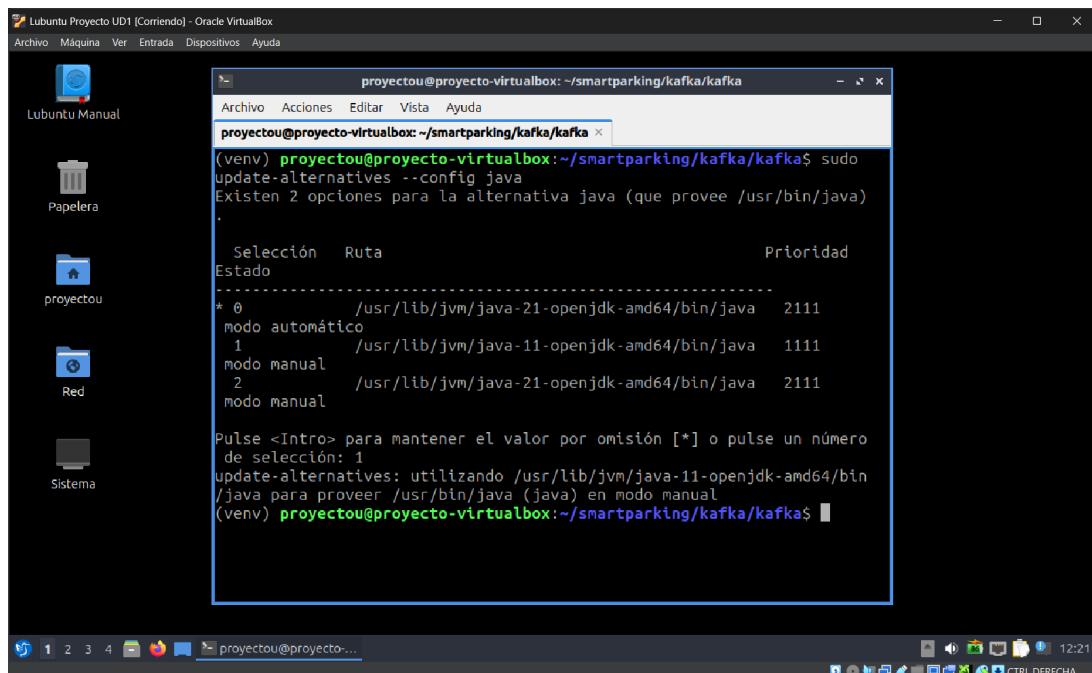
# A size-based retention policy for logs. Segments are pruned from the log
# segments drop below log.retention.bytes. Functions independently of the
# log retention policy
log.retention.bytes=104857600
```

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'.

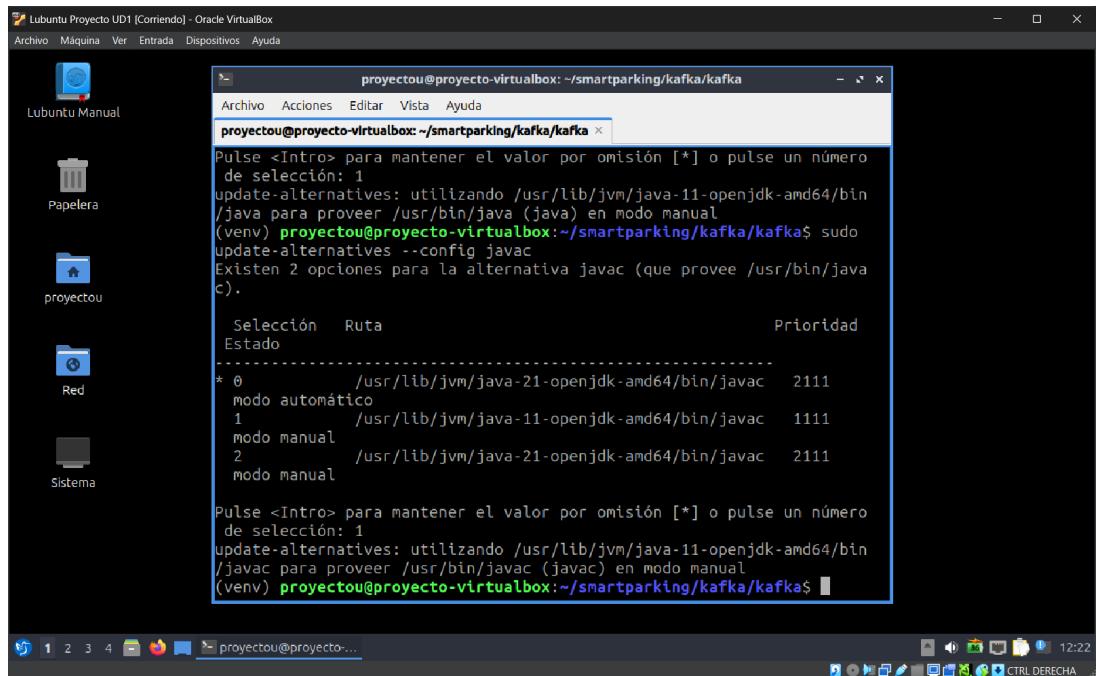


Figura C.10: Selección de la versión de 'javac' (JDK 11).

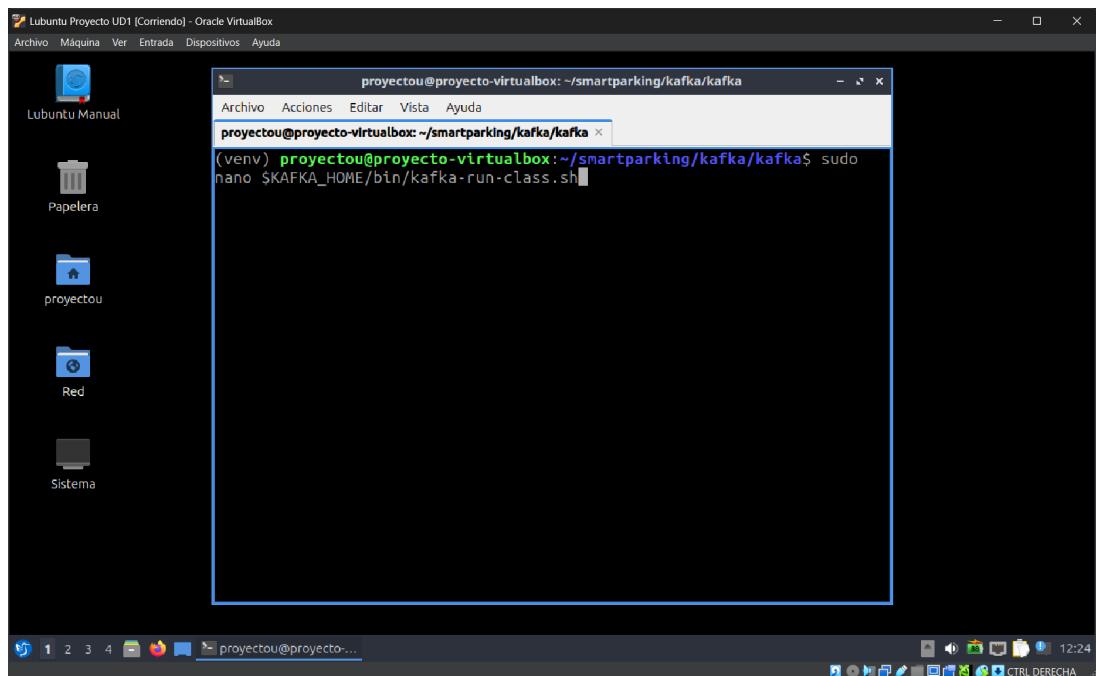
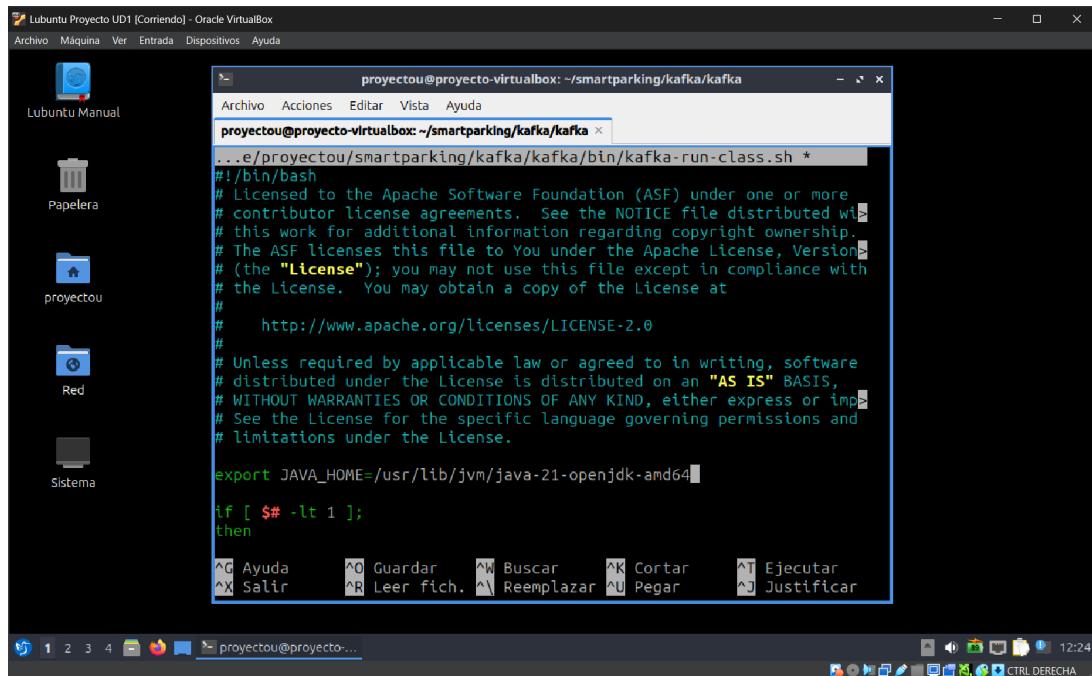


Figura C.11: Editando el script 'kafka-run-class.sh'.



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.

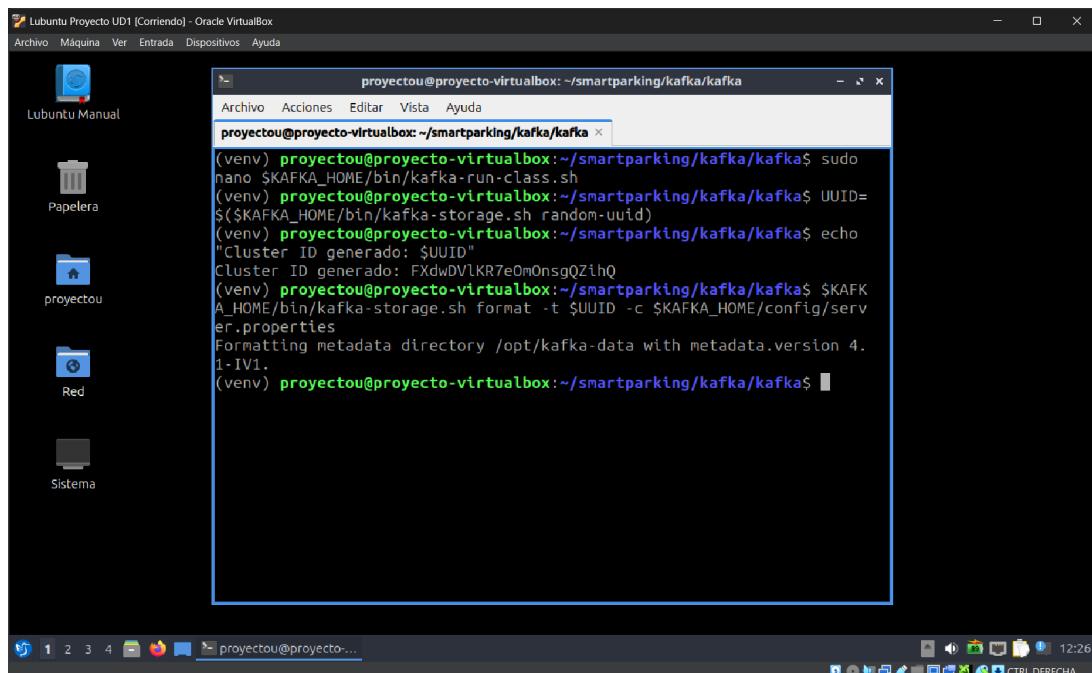
export JAVA_HOME=/usr/lib/jvm/java-21-openjdk-amd64

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: $UUID"
Cluster ID generado: FXdwDVlKR7e0mOnsgQZihQ
(venv) proyectou@projeto-virtualbox:~/smartparking/kafka/kafka$ $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').

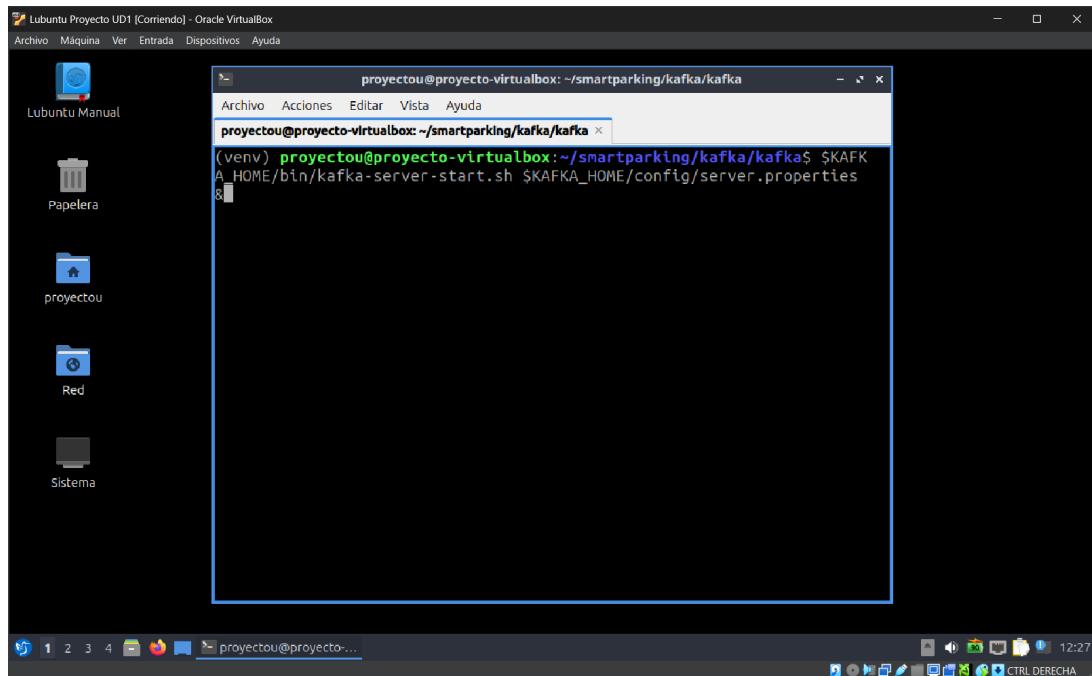


Figura C.14: Inicio del servidor Kafka ('kafka-server-start.sh').

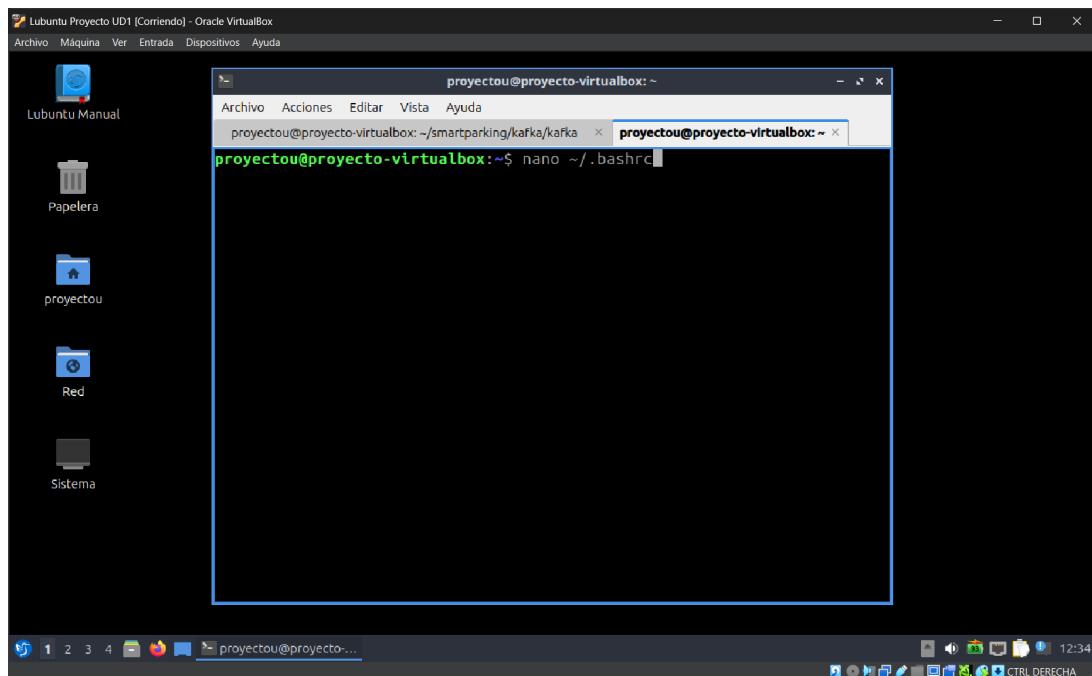


Figura C.15: Editando el archivo ' /.bashrc' para añadir variables de entorno.

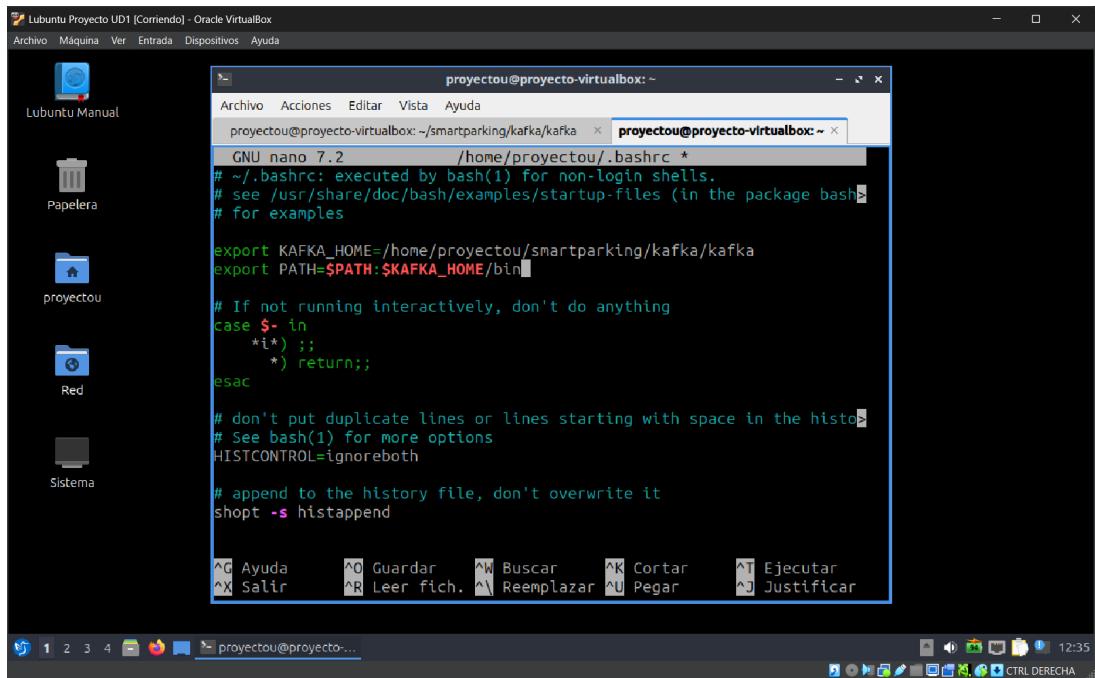


Figura C.16: Añadiendo 'KAFKA_HOME' y actualizando el 'PATH' en ' /.bashrc'.

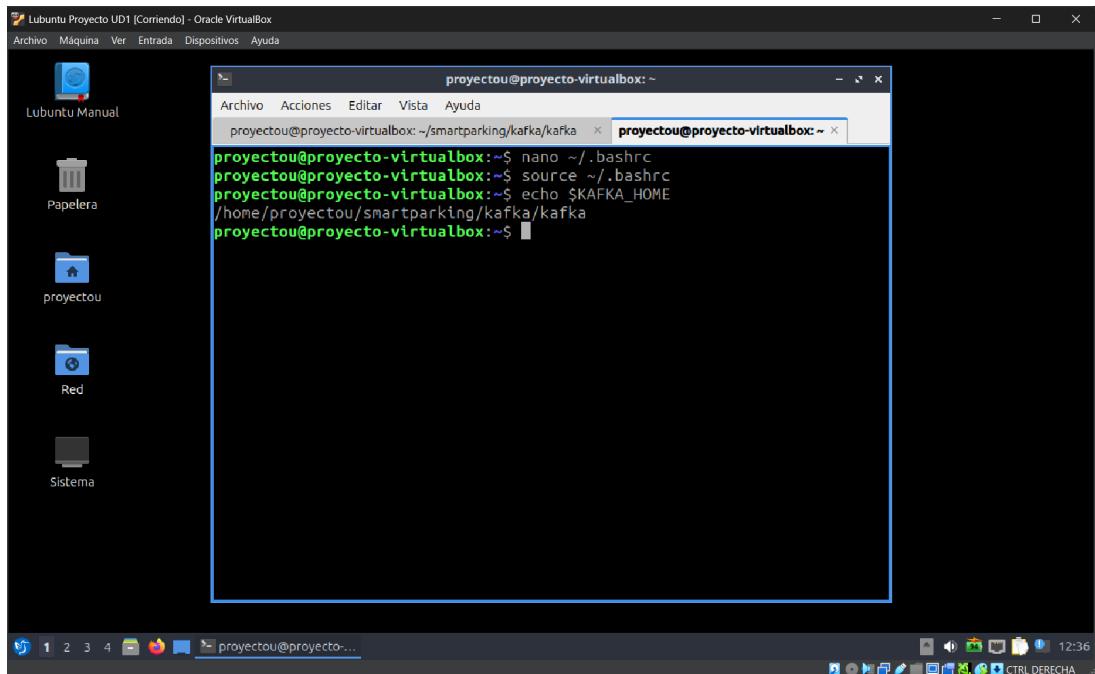


Figura C.17: Recargando la configuración de la terminal ('source ~./.bashrc').

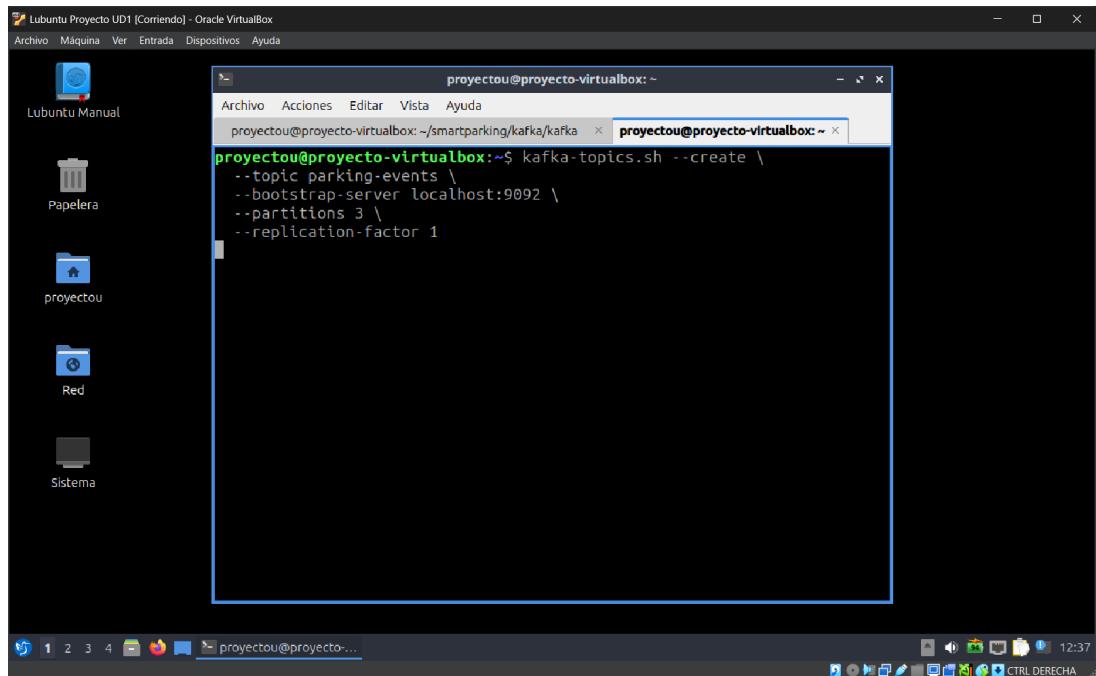


Figura C.18: Comando para crear un nuevo tópico de Kafka.

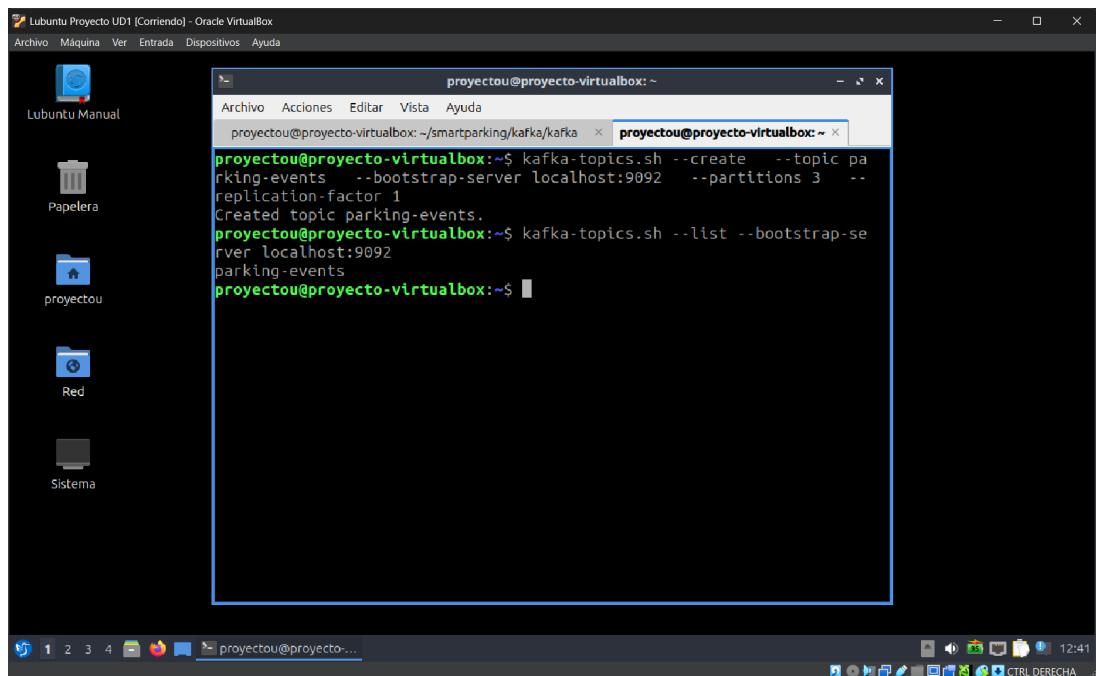


Figura C.19: Creación del tópico `parking-events` y listado de tópicos.

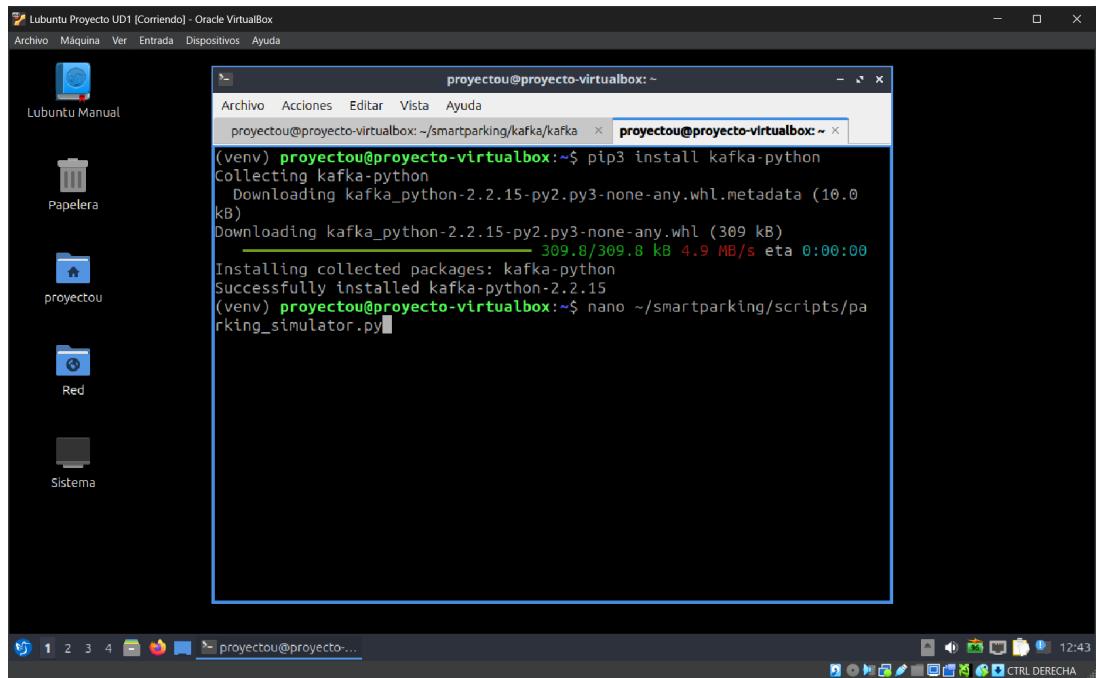


Figura C.20: Instalación de la librería 'kafka-python' con 'pip3'.

```

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()

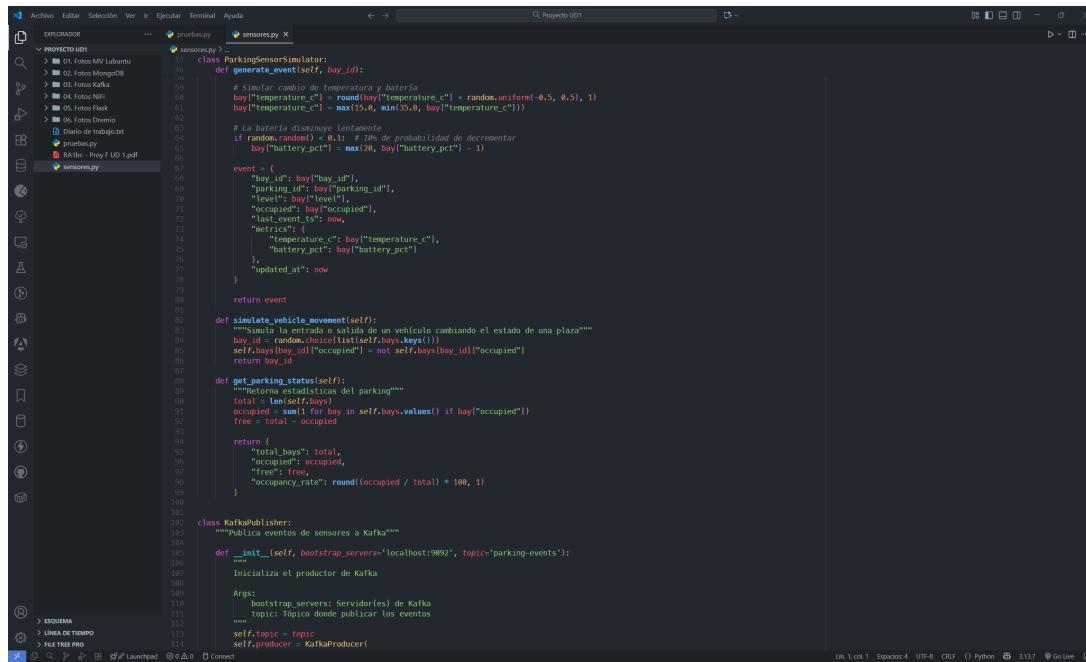
    def _initialize_bays(self):
        """Inicializa el estado de todas las plazas"""
        bays = []
        for level in range(1, self.levels + 1):
            for bay in range(1, self.bays_per_level + 1):
                bay_id = f"({level})-{bay}:{03d}"
                bay_data = {
                    "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)
                }
                bays.append(bay_data)
        return bays

    def generate_event(self, bay_id):
        """Genera un evento de sensor para una plaza específica"""
        Args:
            bay_id: Identificador de la plaza
        Returns:
            dict: Documento JSON del evento
        bay = self.bays[bay_id]
        now = datetime.utcnow().strftime("%Y-%m-%dT%H:%M:%S%z")
        # Simular cambio de temperatura y batería

```

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

SmartParking Flow — Monitorización Inteligente



```

    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"]))
            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, 1)
            }

        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
                )

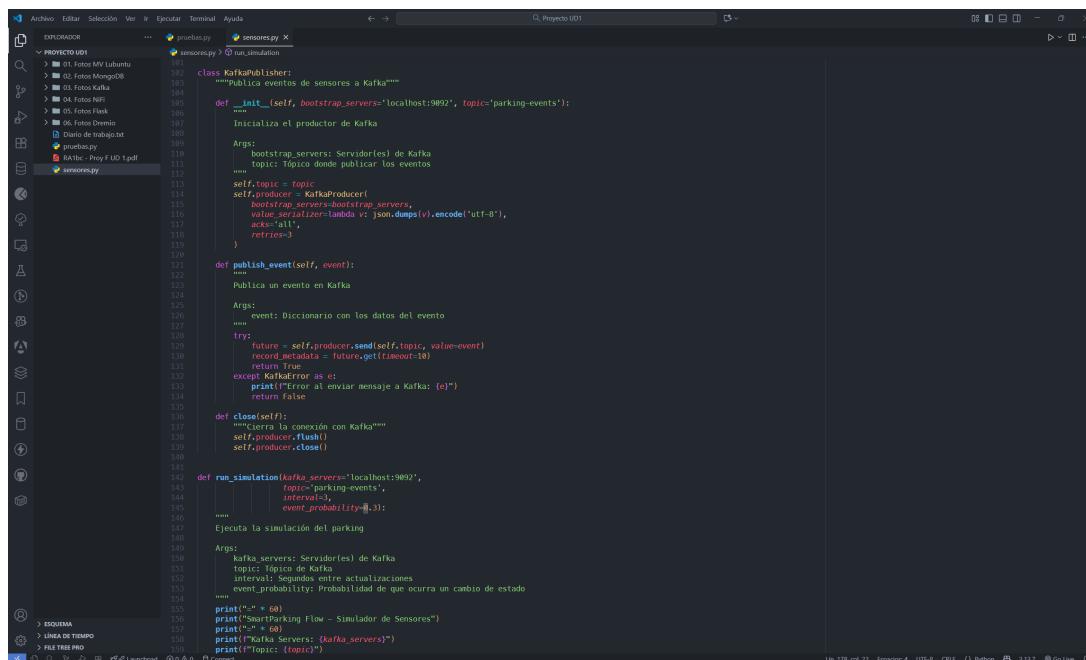
            def publish_event(self, event):
                """Publica un evento en Kafka
                Args:
                    event: Diccionario con los datos del evento
                """
                try:
                    future = self.producer.send(self.topic, value=event)
                    record_metadata = 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'.



```

    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}")

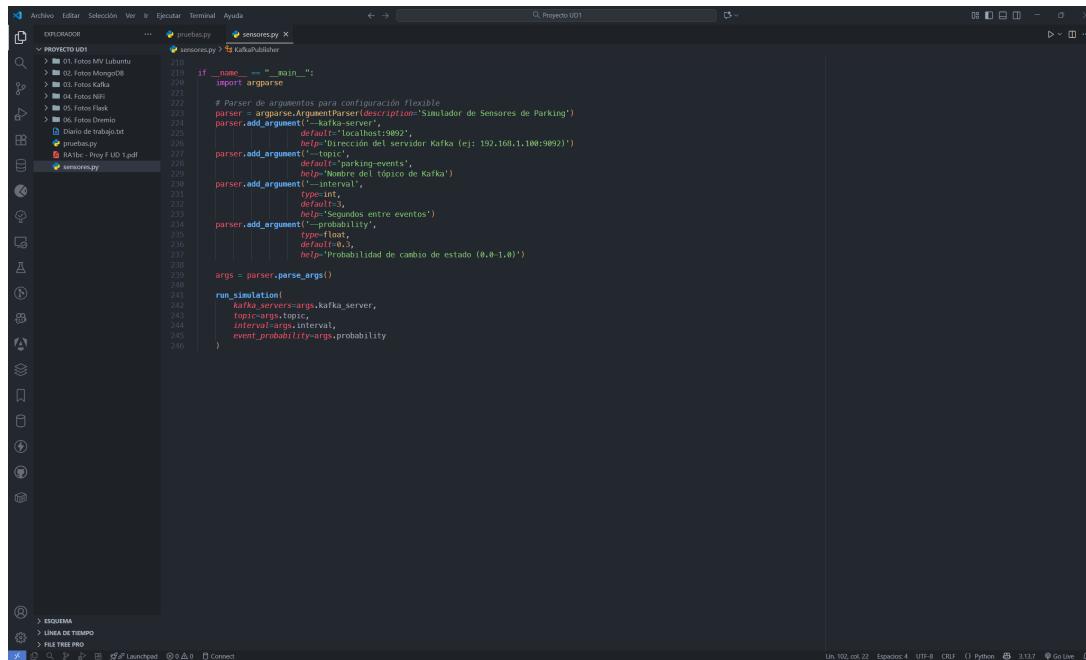
```

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).

SmartParking Flow — Monitorización Inteligente

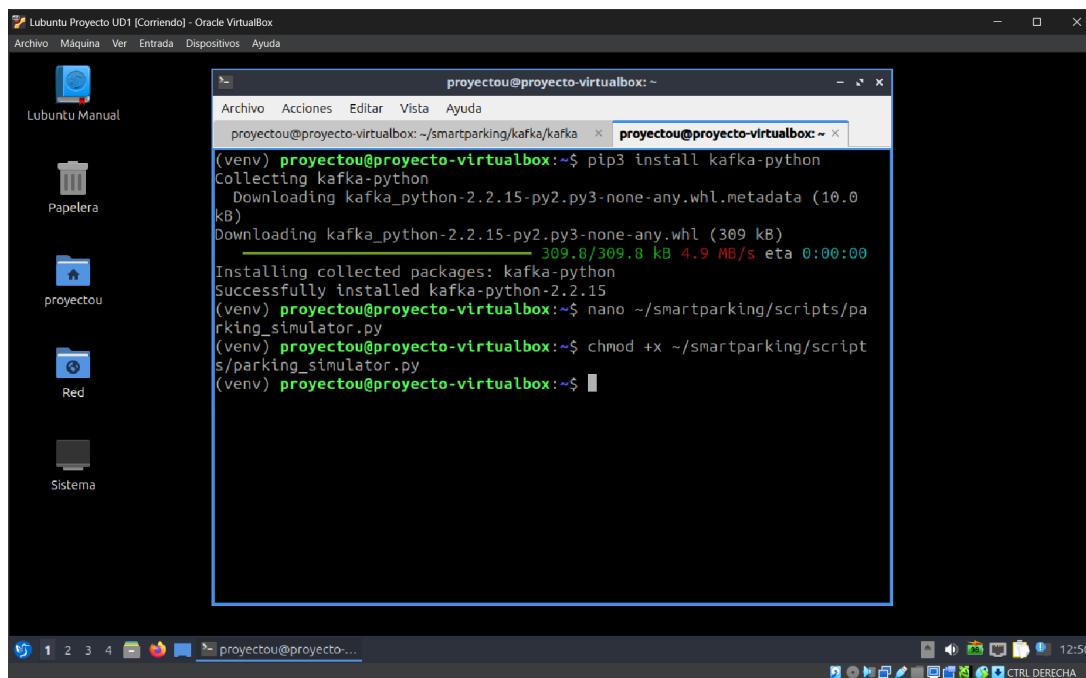


```
if __name__ == "__main__":
    parser = argparse.ArgumentParser(description='Simulador de Sensores de Parking')
    parser.add_argument('--kafka-server',
                        default='localhost:9092',
                        help='Dirección del servidor Kafka (ej: 192.168.1.100:9092)')
    parser.add_argument('--topic',
                        default='parking-events',
                        help='Nombre del tema de Kafka')
    parser.add_argument('--interval',
                        type=int,
                        default=3,
                        help='Número de segundos 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.



```
proyecto@proyecto-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
(venv) proyecto@proyecto-virtualbox:~$ nano ~/smartparking/scripts/parking_simulator.py
(venv) proyecto@proyecto-virtualbox:~$ chmod +x ~/smartparking/scripts/parking_simulator.py
(venv) proyecto@proyecto-virtualbox:~$
```

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

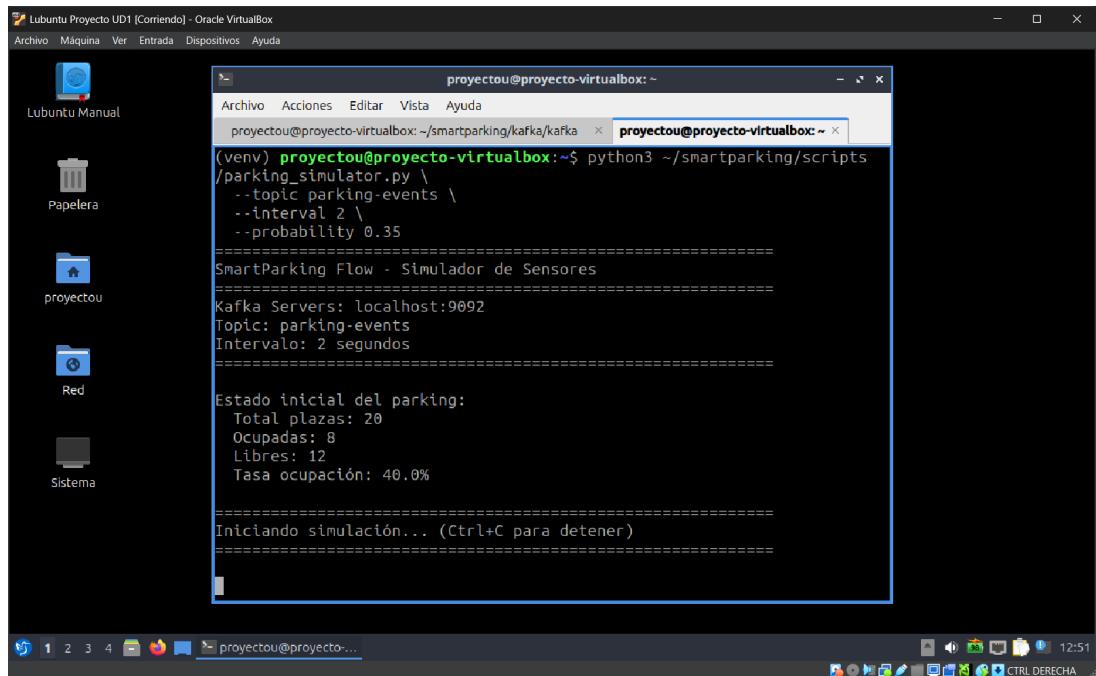


Figura C.27: Ejecución del script simulador de sensores ('parking_simulator.py').

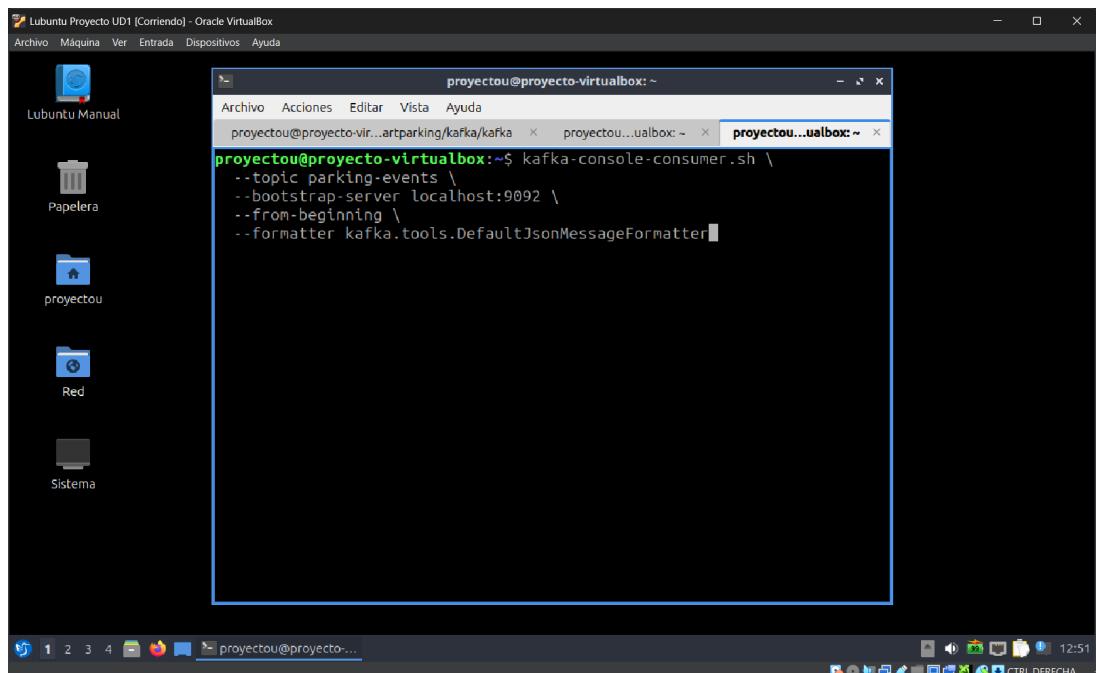


Figura C.28: Comando para iniciar un consumidor de consola en formato JSON.

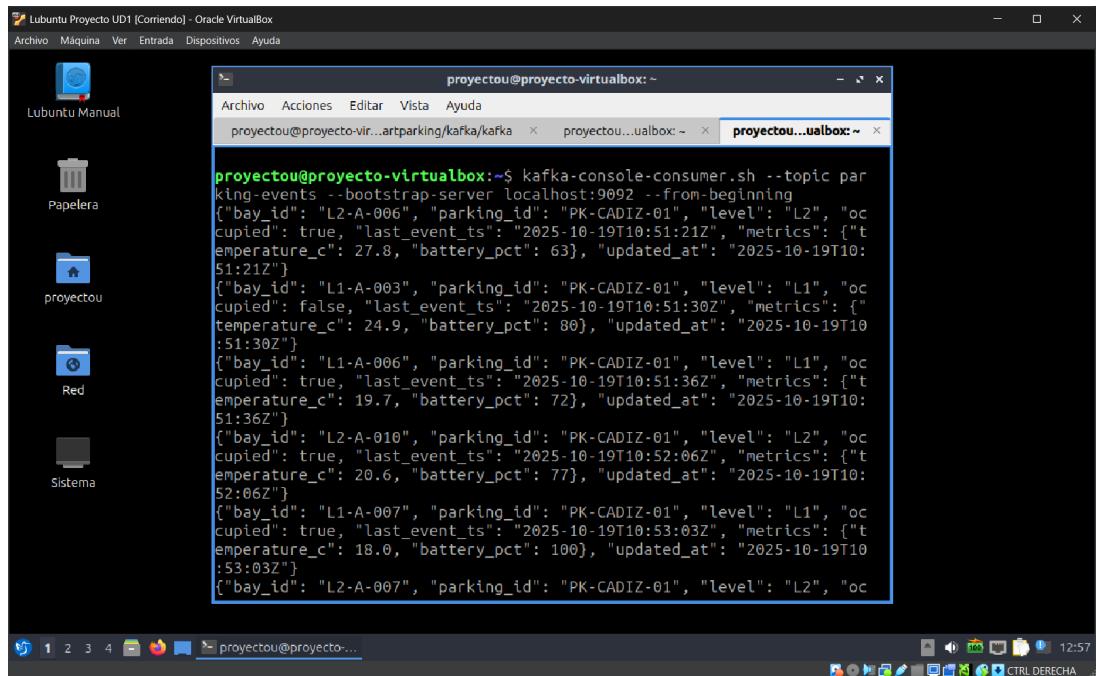
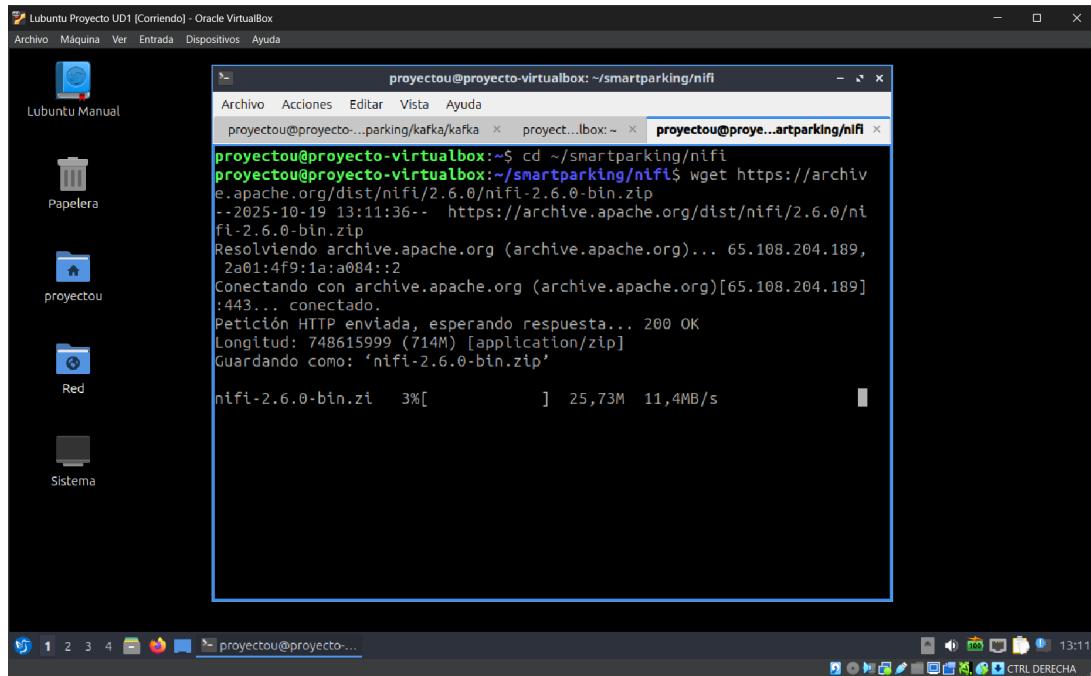


Figura C.29: Consumidor de consola de Kafka mostrando los mensajes JSON entrantes.

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.

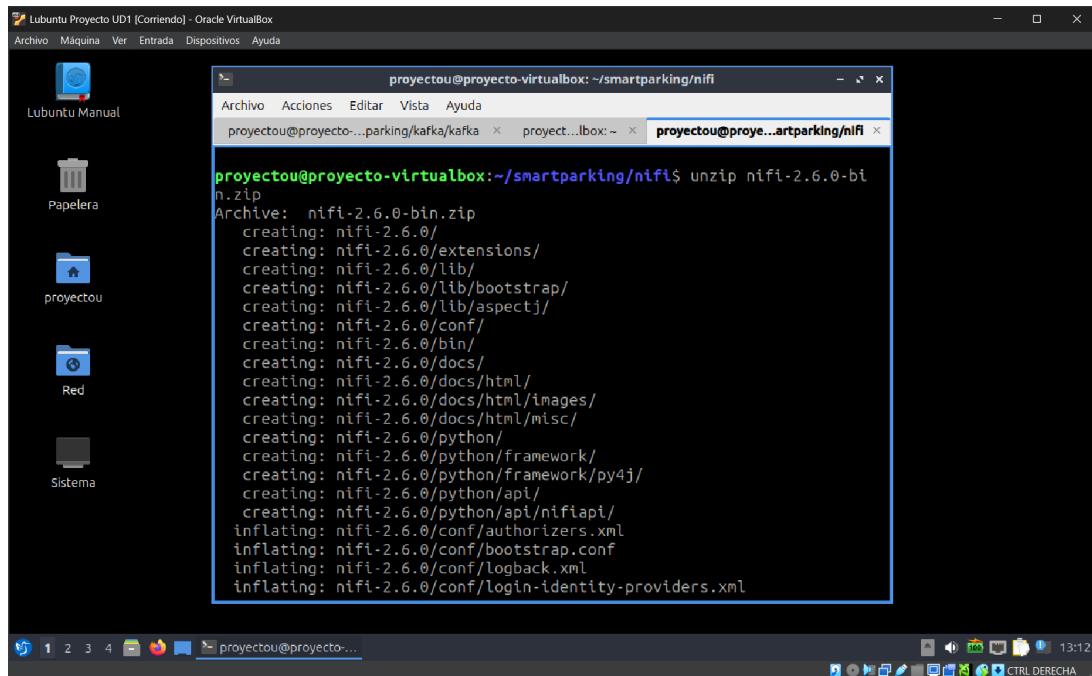


The screenshot shows a terminal window titled "proyecto@projeto-virtualbox: ~/smartparking/nifi". The window is part of a desktop environment with a sidebar containing icons for "Lubuntu Manual", "Papelera", "proyecto", "Red", and "Sistema". The terminal content is as follows:

```
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'

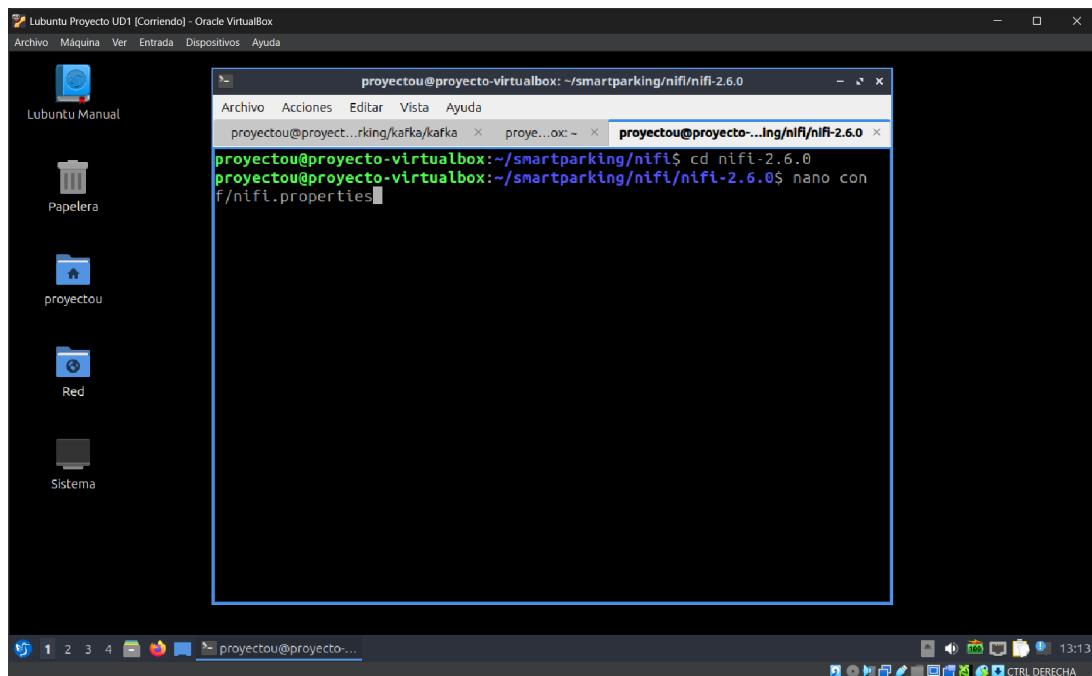
nifi-2.6.0-bin.zip 3%[          ] 25,73M 11,4MB/s
```

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



A screenshot of a terminal window titled "proyecto@proj...rtparking/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 sub-directories and configuration files such as "bin", "conf", "docs", "extensions", "lib", and "logback.xml". The terminal is running on a Linux desktop environment with a dark theme, showing icons for "Lubuntu Manual", "Papelera", "proyecto", "Red", and "Sistema" on the left.

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



A screenshot of a terminal window titled "proyecto@proj...rtparking/nifi/nifi-2.6.0". The user has navigated to the "nifi-2.6.0" directory using the command "cd nifi-2.6.0". They are now in the "conf" directory, as indicated by the command "nano conf/nifi.properties" which is shown in the terminal. The terminal window is part of a desktop environment with a dark theme, showing icons for "Lubuntu Manual", "Papelera", "proyecto", "Red", and "Sistema" on the left.

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

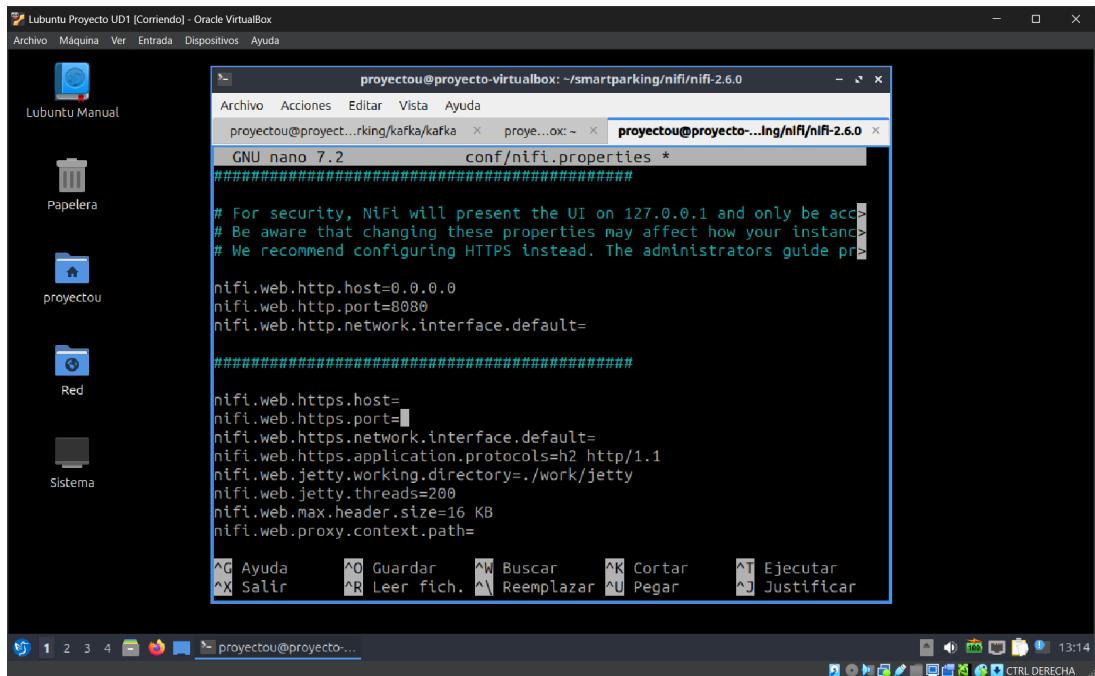


Figura D.4: Editando 'nifi.properties' parte 1.

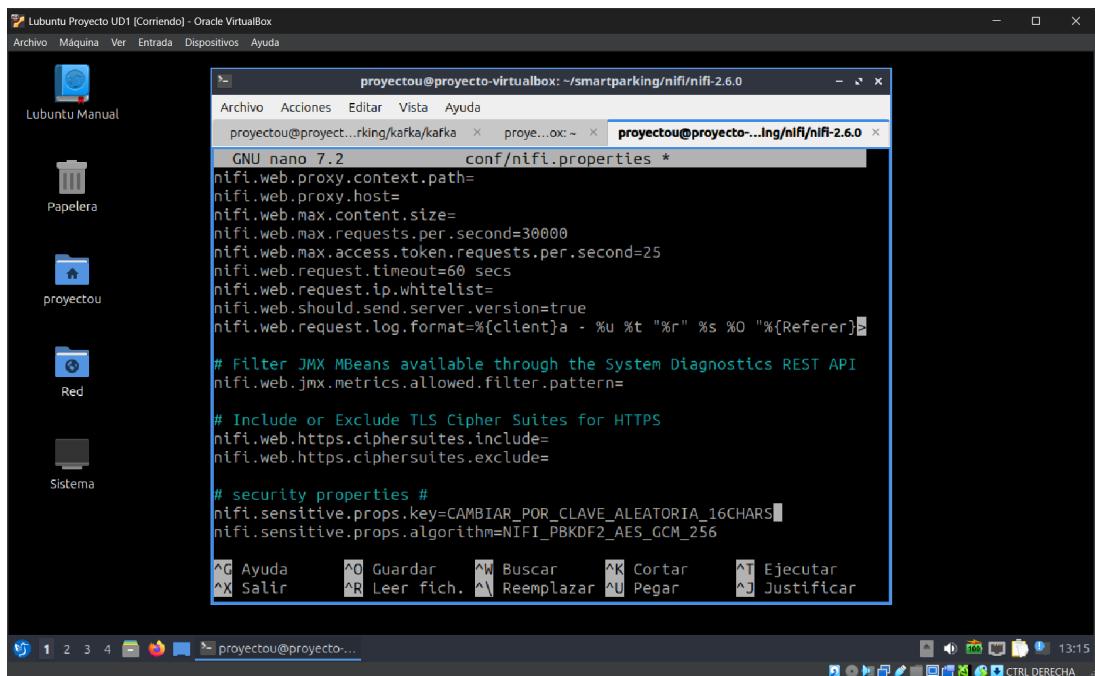
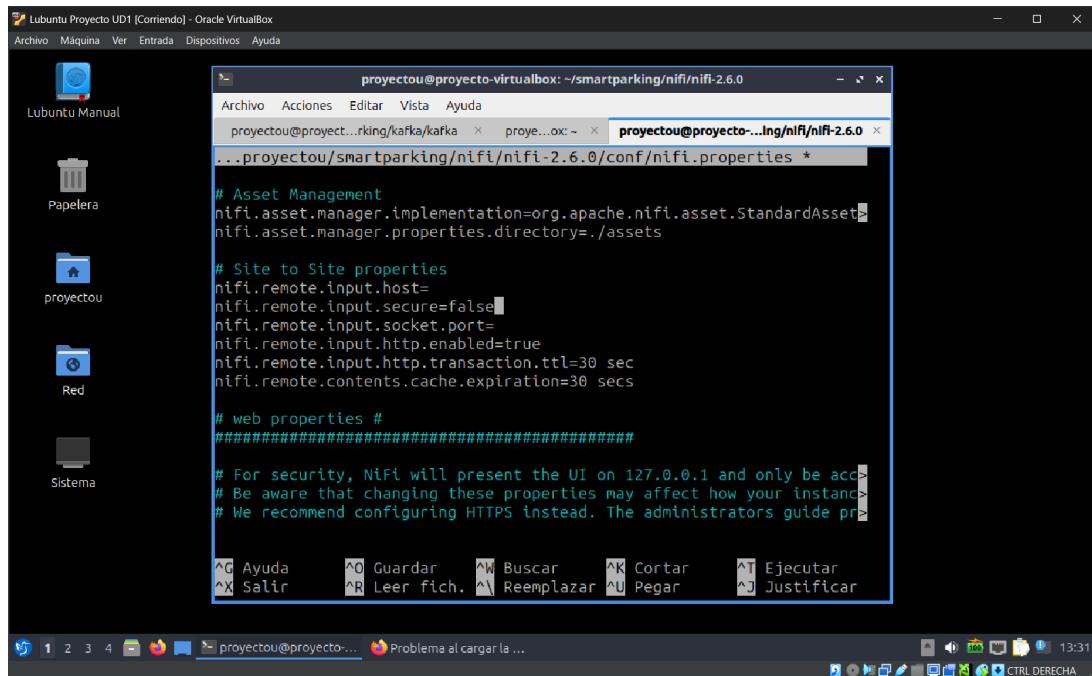


Figura D.5: Editando 'nifi.properties' parte 2.



```

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

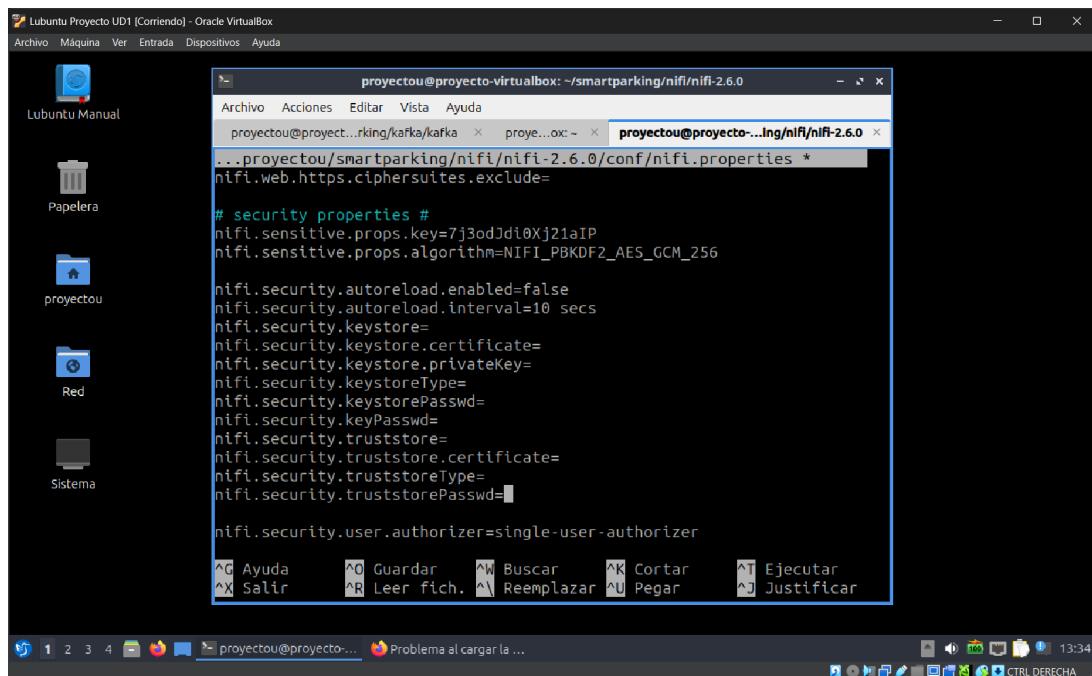
# Site to Site properties
nifi.remote.input.host=
nifi.remote.input.secure=false
nifi.remote.input.socket.port=
nifi.remote.input.http.enabled=true
nifi.remote.input.http.transaction.ttl=30 sec
nifi.remote.contents.cache.expiration=30 secs

# 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' parte 3.



```

nifi.security.user.authorizer=single-user-authorizer

nifi.security.keyPasswd=
nifi.security.truststore=
nifi.security.truststore.certificate=
nifi.security.truststore.type=
nifi.security.truststorePasswd=
```

Figura D.7: Editando 'nifi.properties' parte 4.

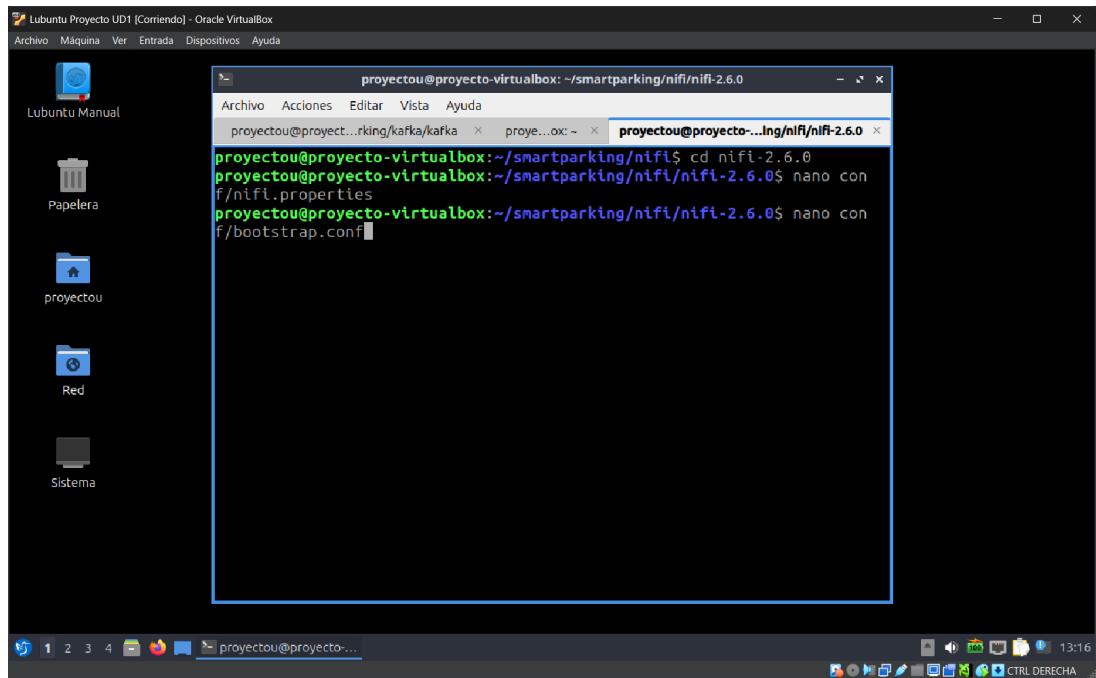


Figura D.8: Accediendo al archivo 'bootstrap.conf'.

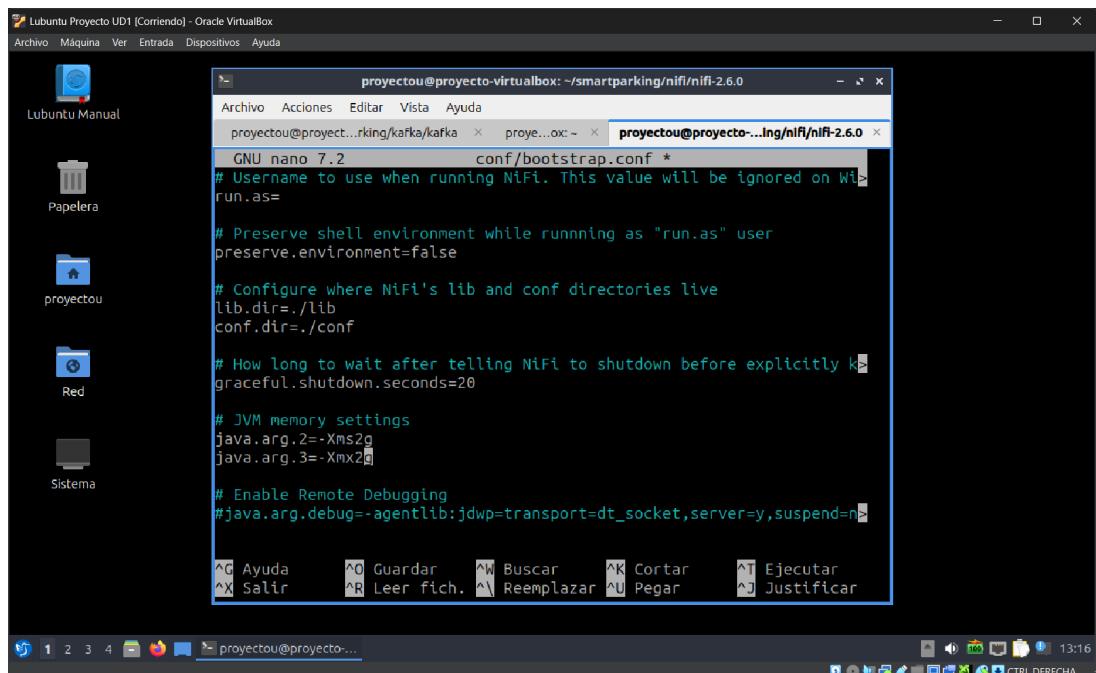


Figura D.9: Editando 'bootstrap.conf': Configuración de memoria JVM (Xms2g, Xmx2g).

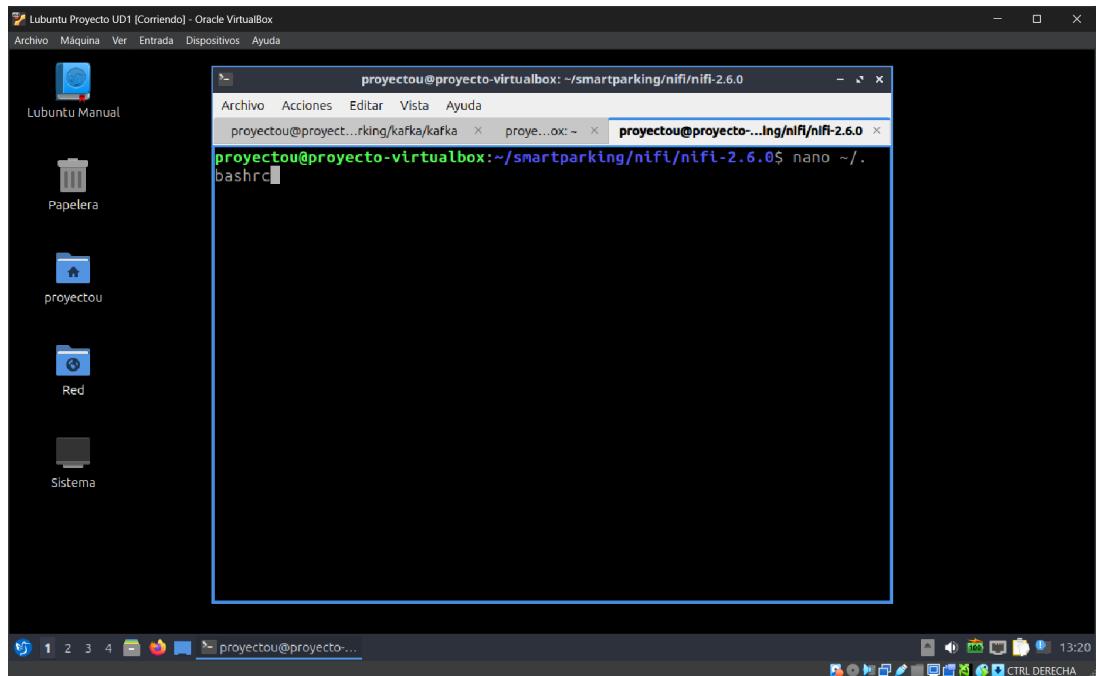


Figura D.10: Editando ' /.bashrc' para añadir 'NIFI_HOME'.

A screenshot of a Linux desktop environment, specifically Lubuntu, running in Oracle VirtualBox. The desktop has a dark theme with icons for 'Lubuntu Manual', 'Papelera' (Recycle Bin), 'proyecto', 'Red', and 'Sistema'. A terminal window titled 'proyectou@proj...virtualbox: ~\$' is open, showing the contents of the .bashrc file. The file includes environment variable definitions for KAFKA_HOME, PATH, and NIFI_HOME, along with other bash startup logic. The terminal shows the command 'GNU nano 7.2 /home/proyectou/.bashrc *' at the top.

```
GNU nano 7.2 /home/proyectou/.bashrc *
#!/bin/sh -e
# ~/.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=$KAFKA_HOME/bin:$PATH
export NIFI_HOME=/home/proyectou/smartparking/nifi/nifi-2.6.0
export PATH=$NIFI_HOME/bin:$PATH

# 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'.

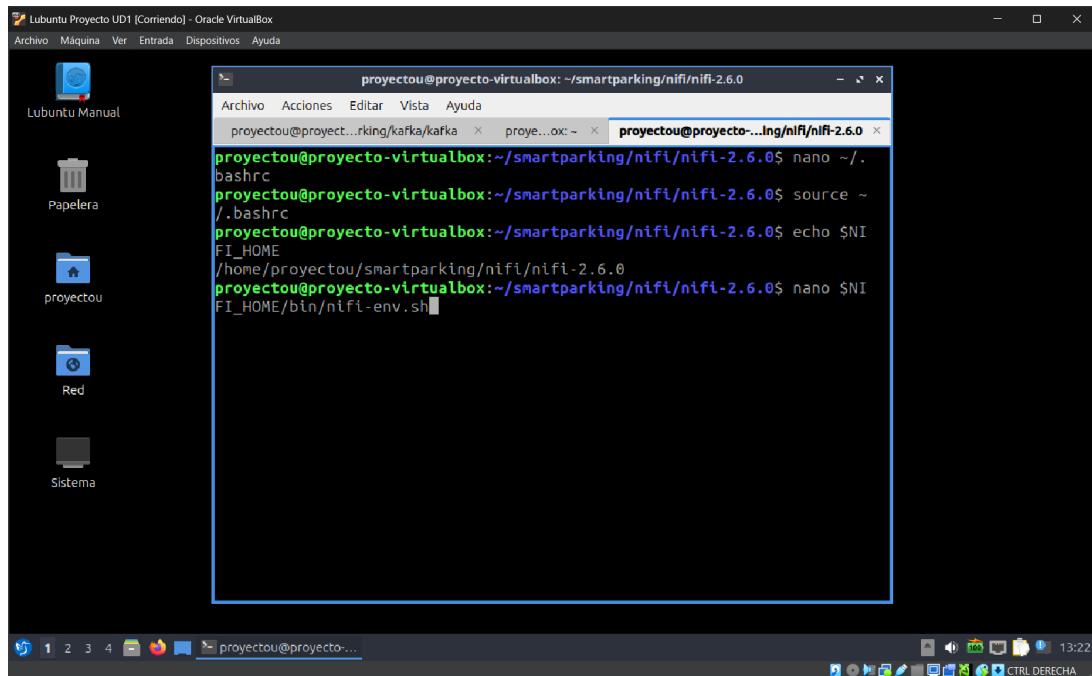


Figura D.12: Accediendo al script de entorno 'nifi-env.sh'.

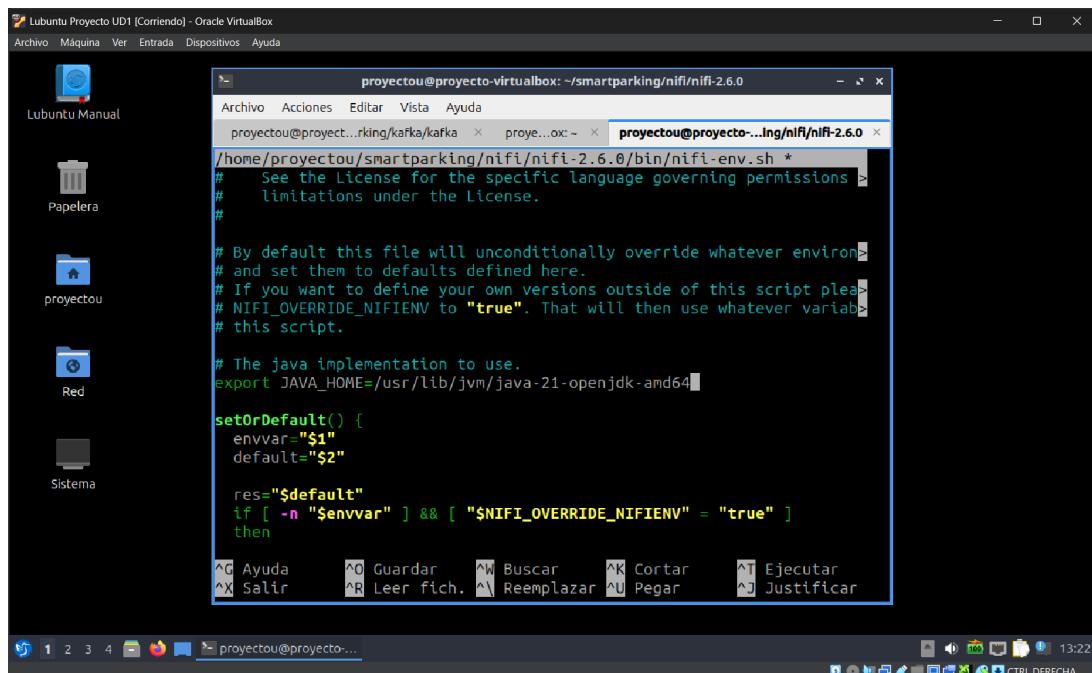


Figura D.13: Estableciendo 'JAVA_HOME' en 'nifi-env.sh'.

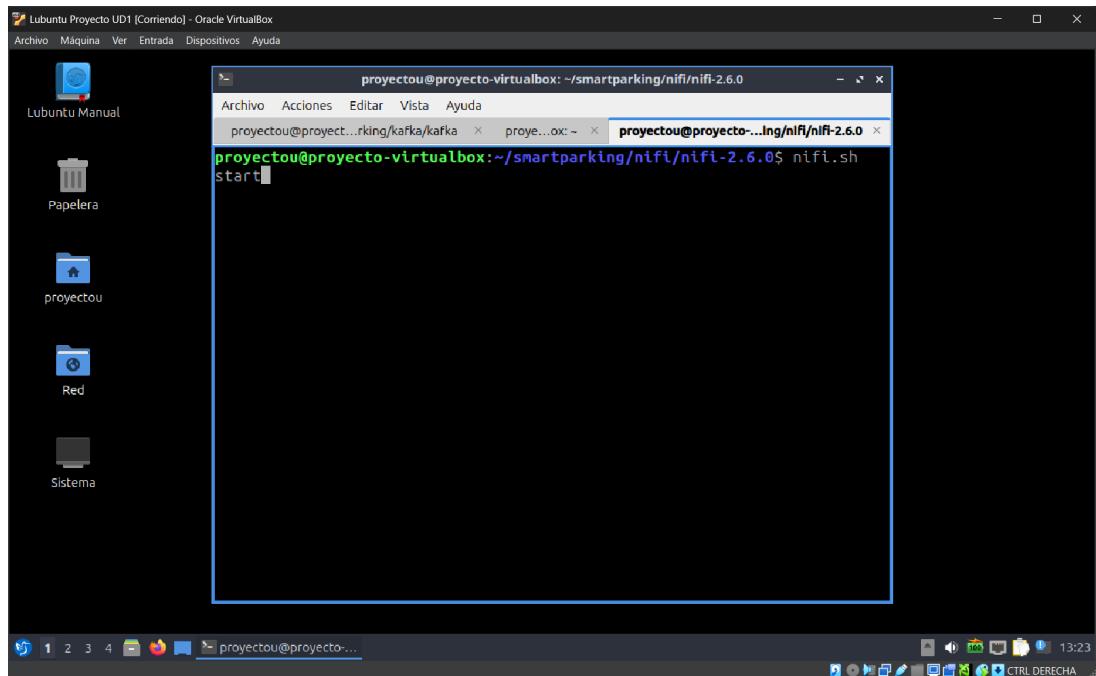


Figura D.14: Iniciando el servicio de NiFi ('nifi.sh start').

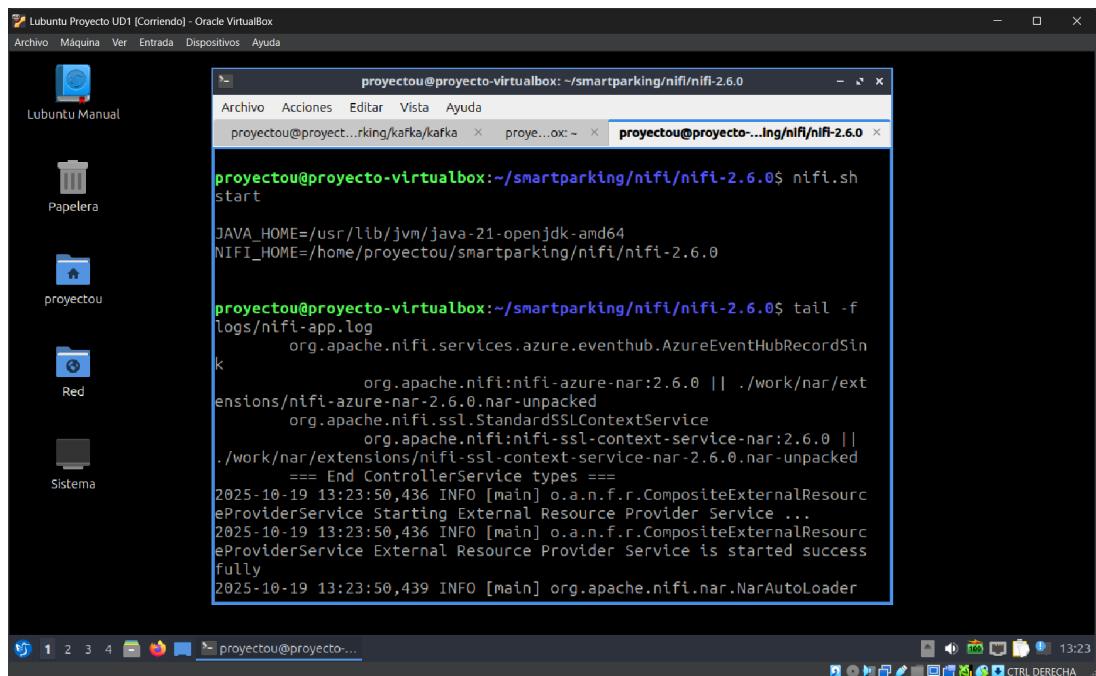


Figura D.15: Comprobando los logs de inicio de NiFi ('tail -f nifi-app.log').

SmartParking Flow — Monitorización Inteligente

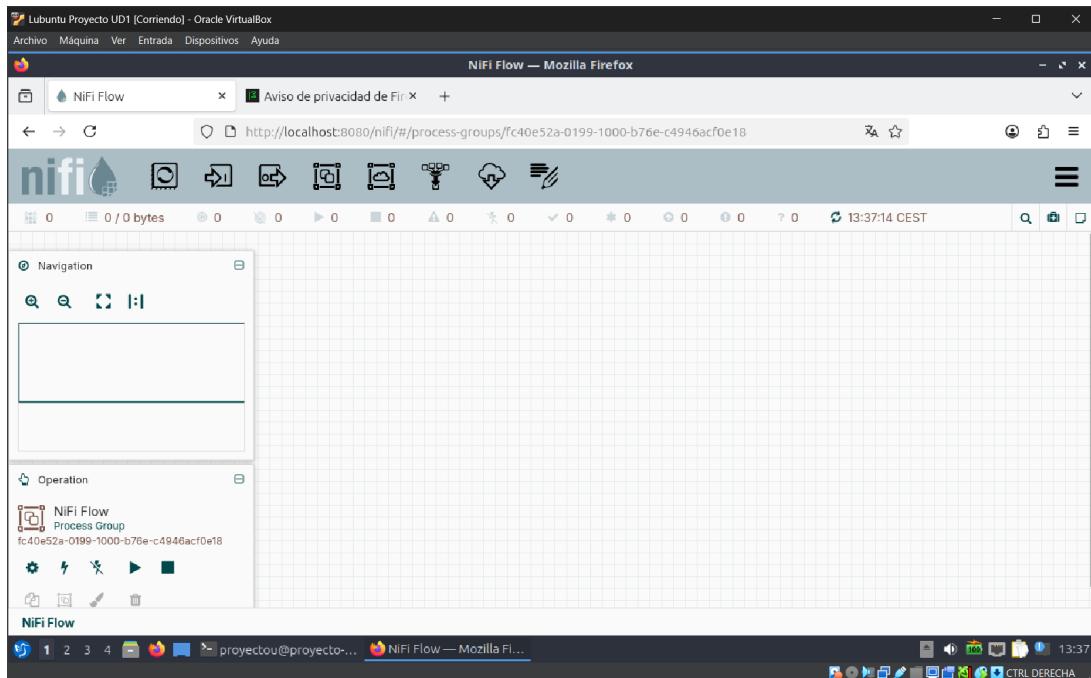


Figura D.16: Accediendo a la interfaz web de NiFi (localhost:8080/nifi).

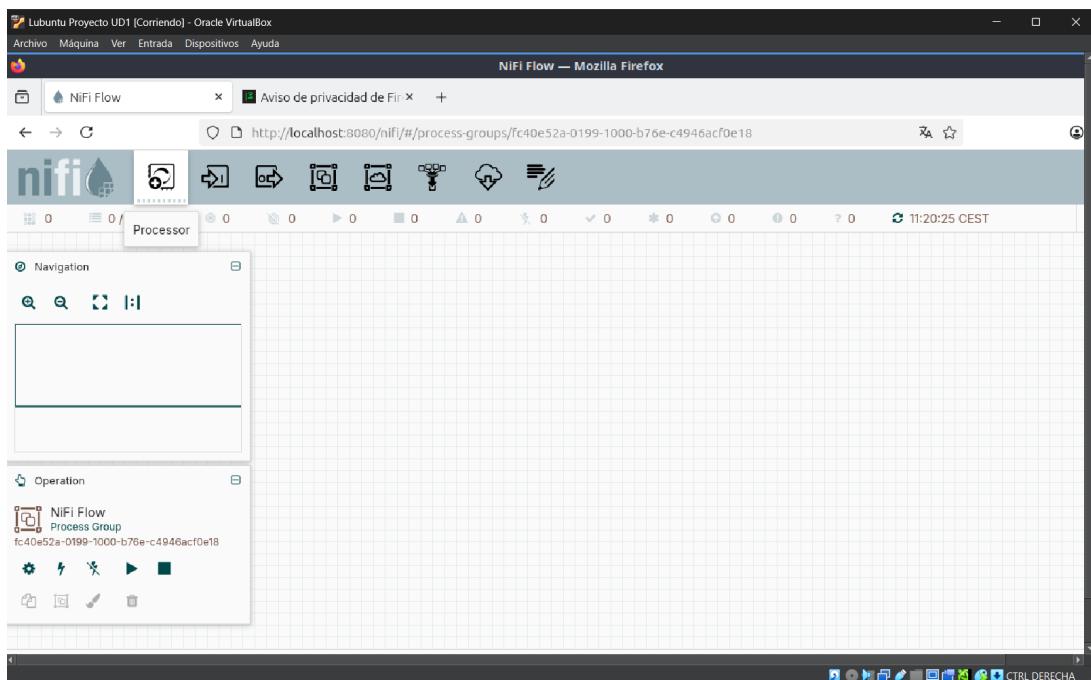


Figura D.17: Arrastrando un nuevo procesador al canvas.

SmartParking Flow — Monitorización Inteligente

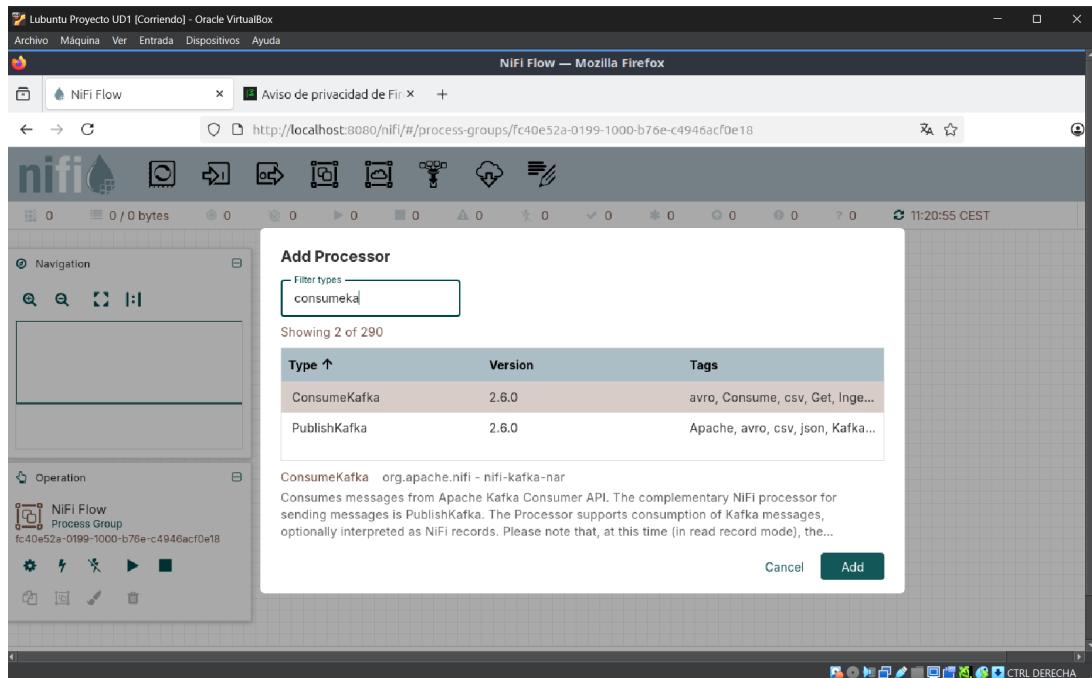


Figura D.18: Buscando el procesador 'ConsumeKafka'.

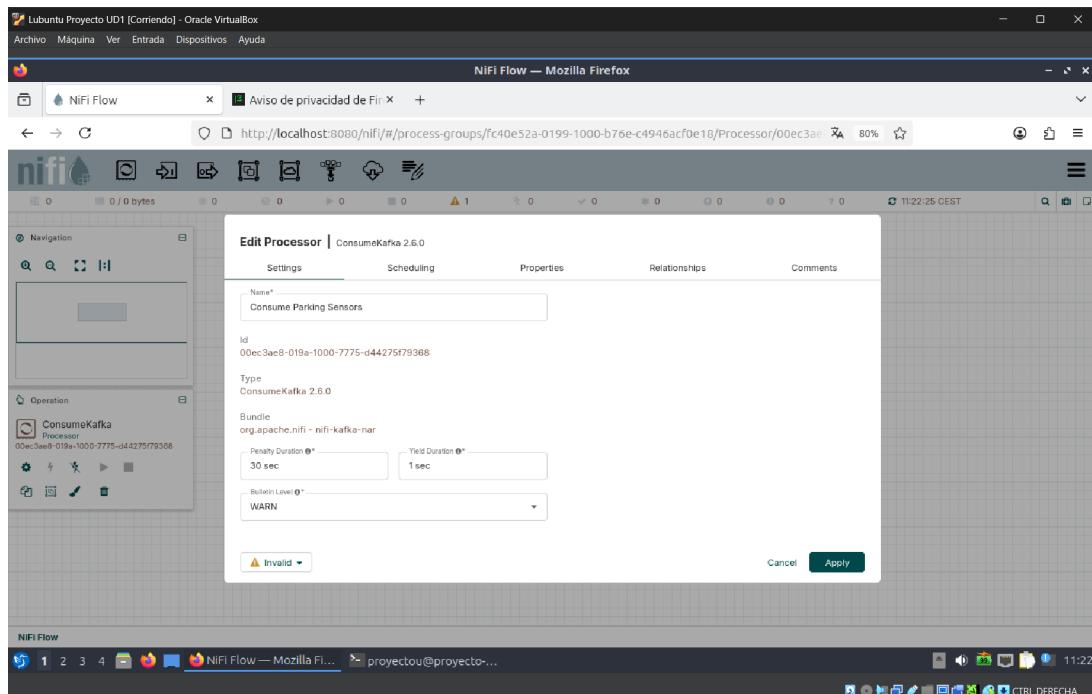


Figura D.19: Configuración (Settings) del procesador 'ConsumeKafka'.

SmartParking Flow — Monitorización Inteligente

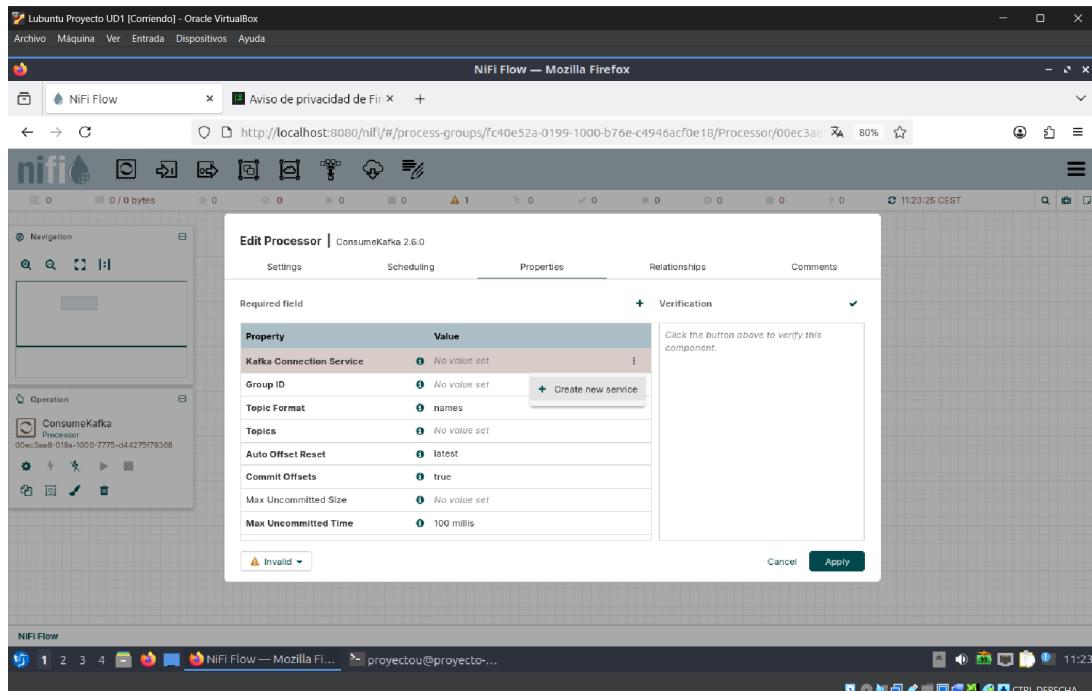


Figura D.20: Configuración (Properties) del procesador 'ConsumeKafka'.

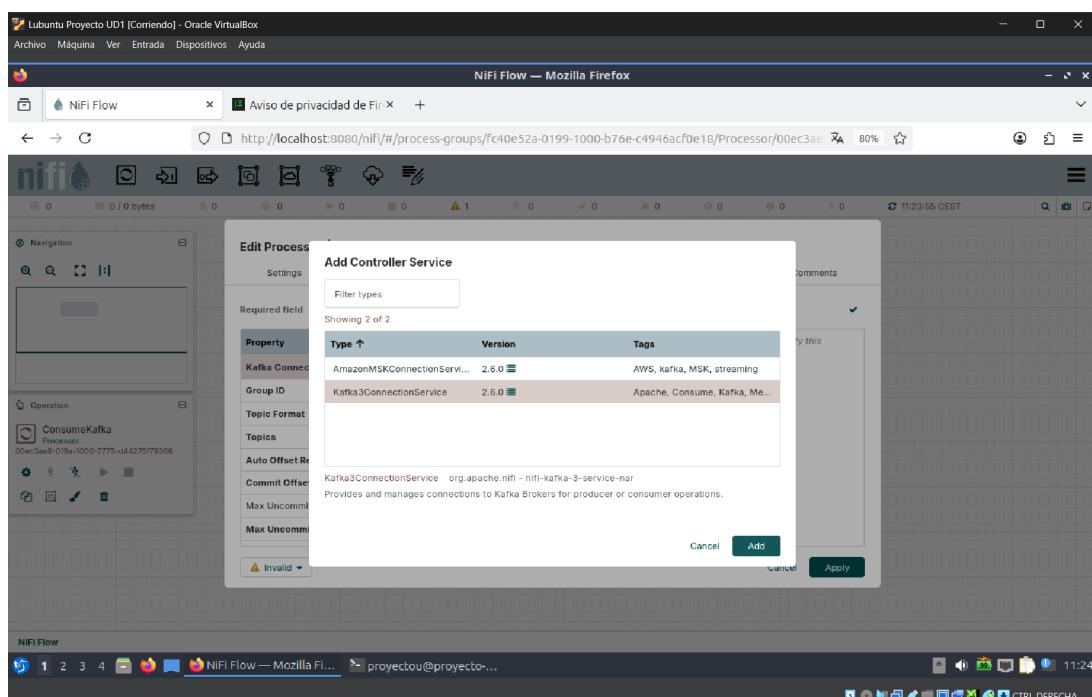


Figura D.21: Creación de un nuevo Controller Service: 'Kafka3ConnectionService'.

SmartParking Flow — Monitorización Inteligente

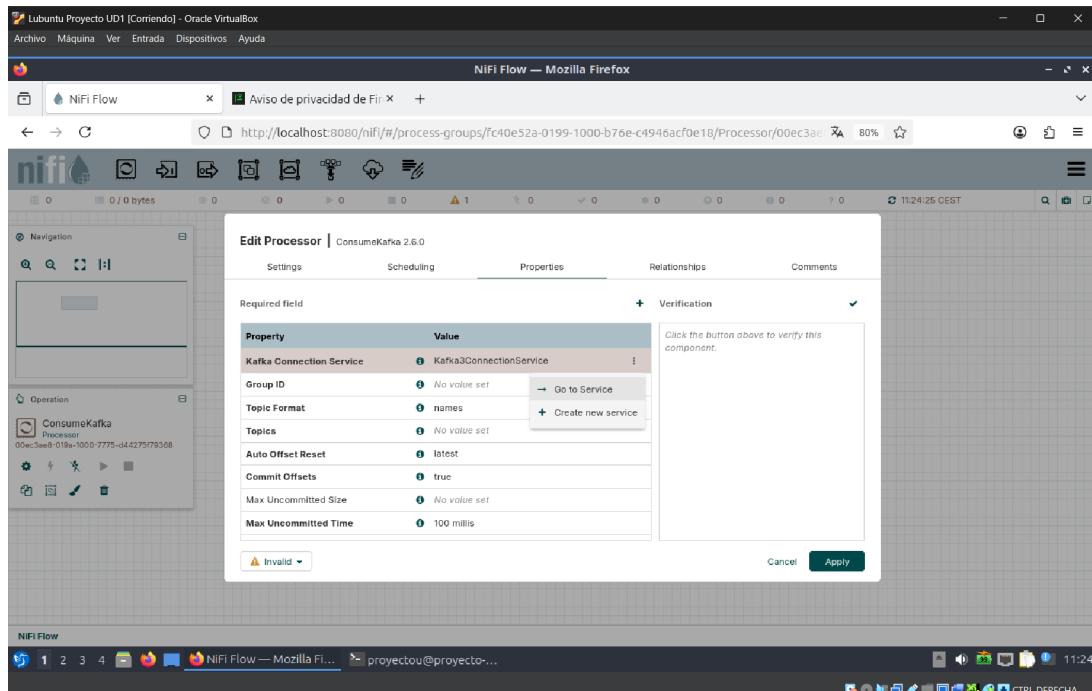


Figura D.22: Asignando el 'Kafka3ConnectionService' al procesador.

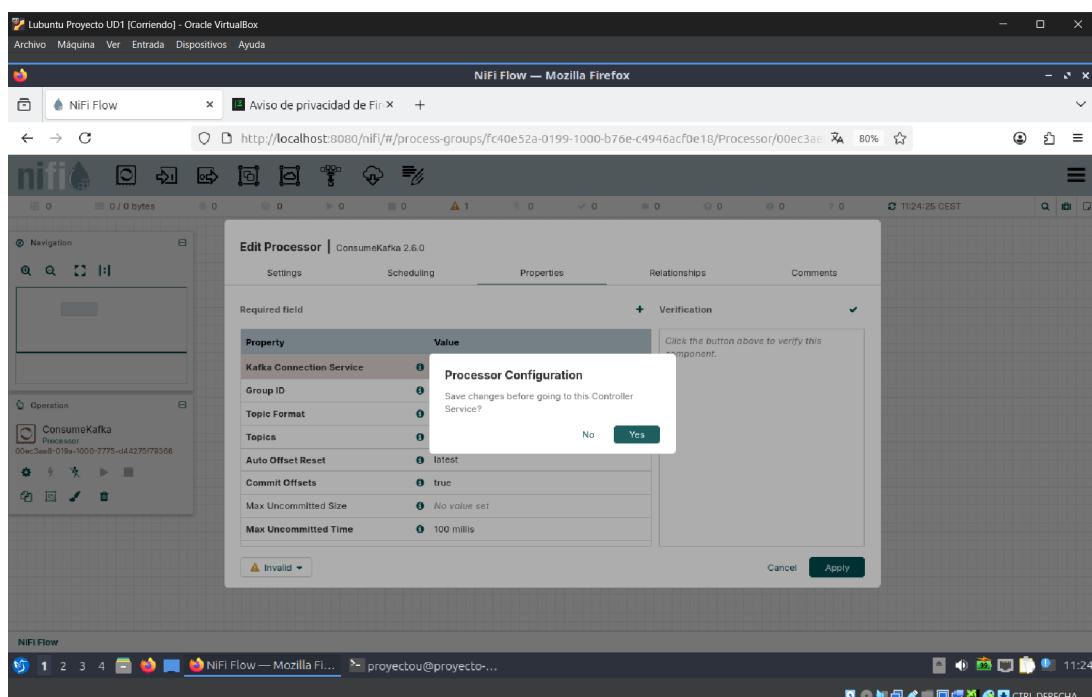


Figura D.23: Guardando cambios en el procesador.

SmartParking Flow — Monitorización Inteligente

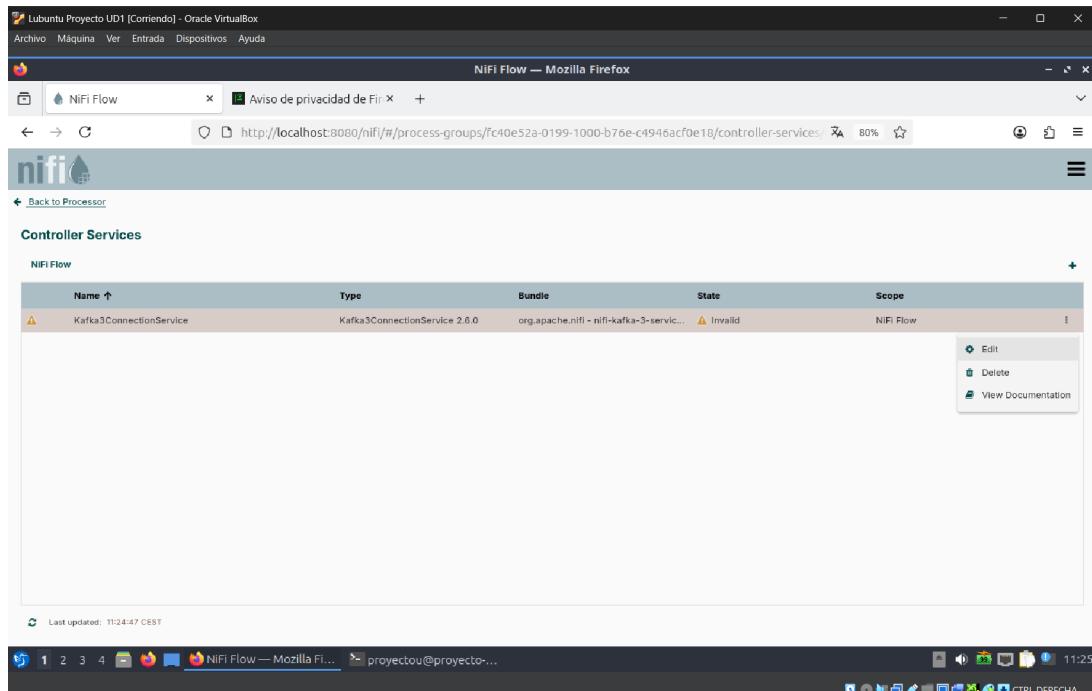


Figura D.24: Accediendo a la configuración del Controller Service ('Kafka3ConnectionService').

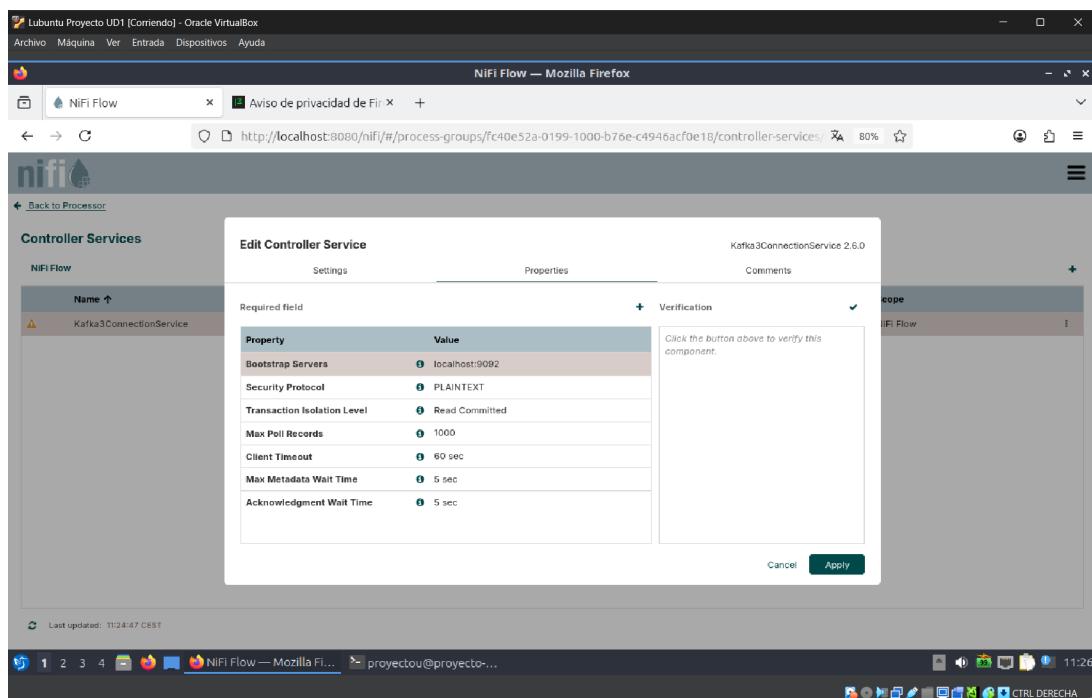


Figura D.25: Configurando el 'Kafka3ConnectionService' (Bootstrap Servers: localhost:9092).

SmartParking Flow — Monitorización Inteligente

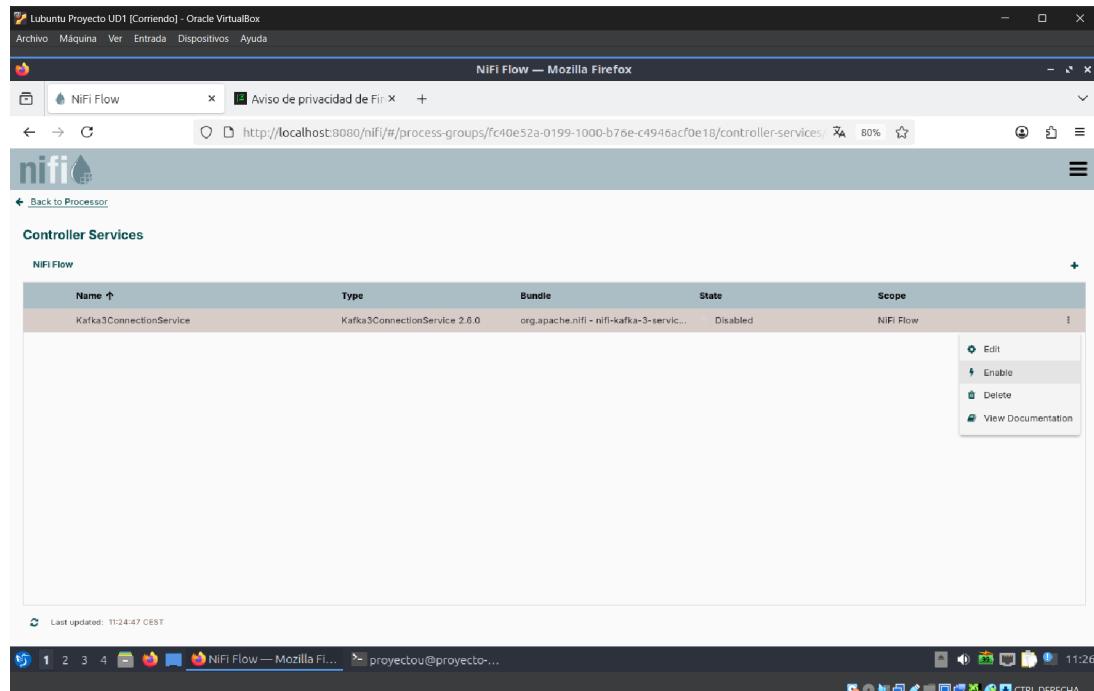


Figura D.26: Controller Service 'Kafka3ConnectionService' en estado 'Disabled'.

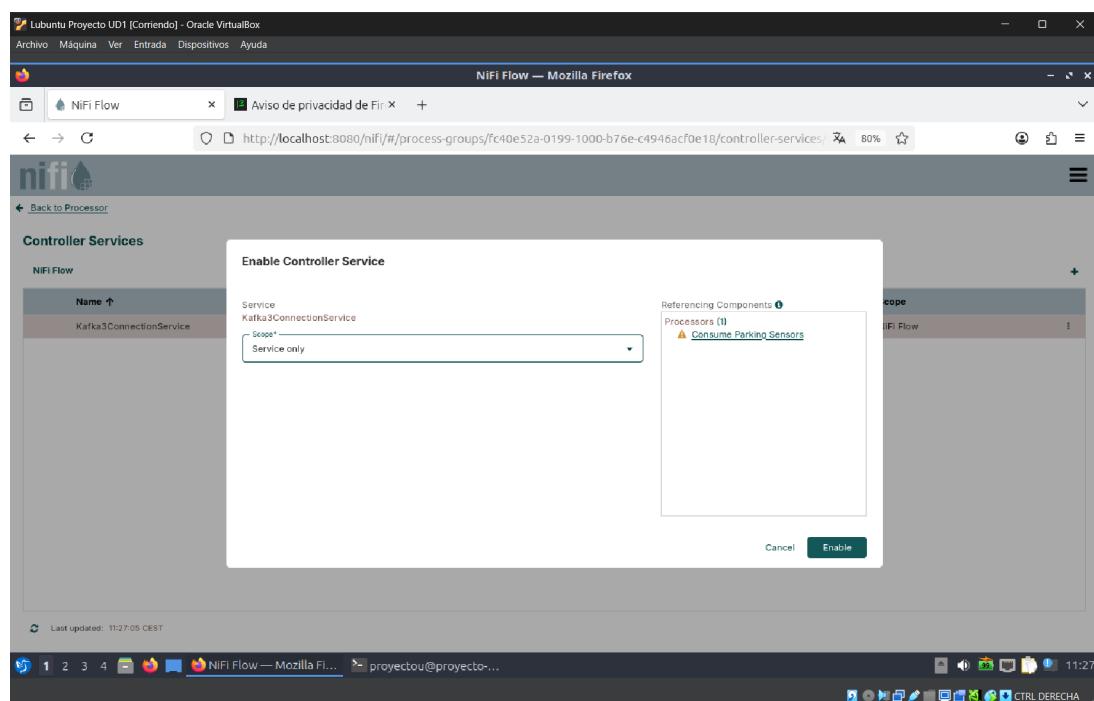


Figura D.27: Habilitando el 'Kafka3ConnectionService'.

SmartParking Flow — Monitorización Inteligente

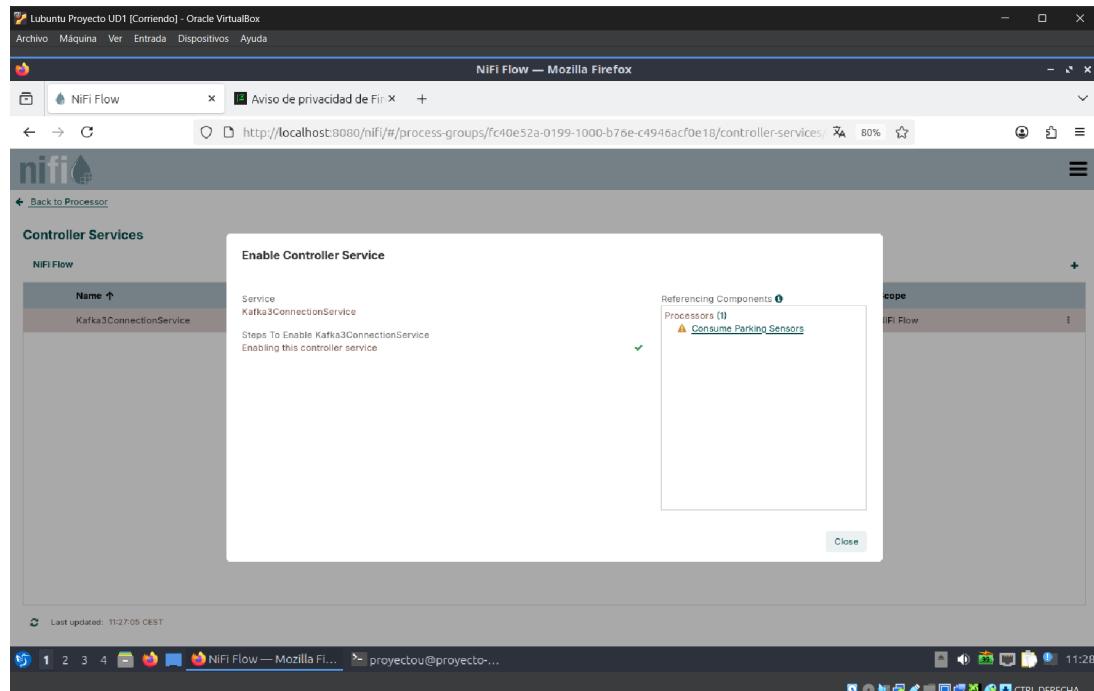


Figura D.28: Controller Service 'Kafka3ConnectionService' siendo habilitado.

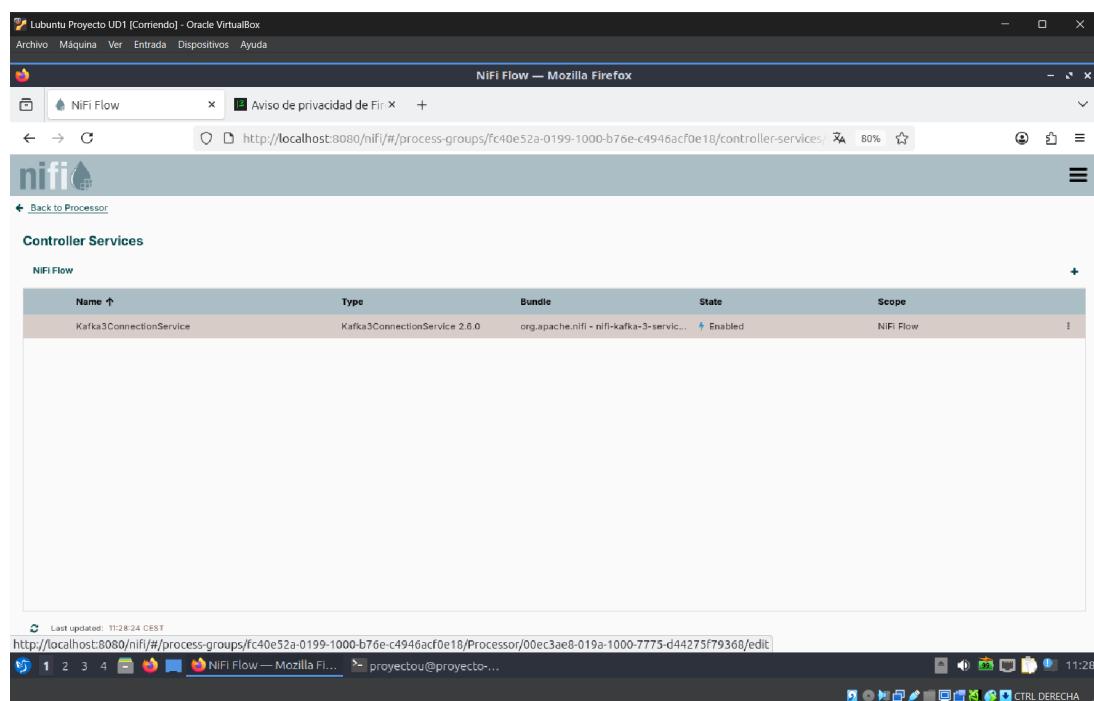


Figura D.29: Controller Service 'Kafka3ConnectionService' en estado 'Enabled'.

SmartParking Flow — Monitorización Inteligente

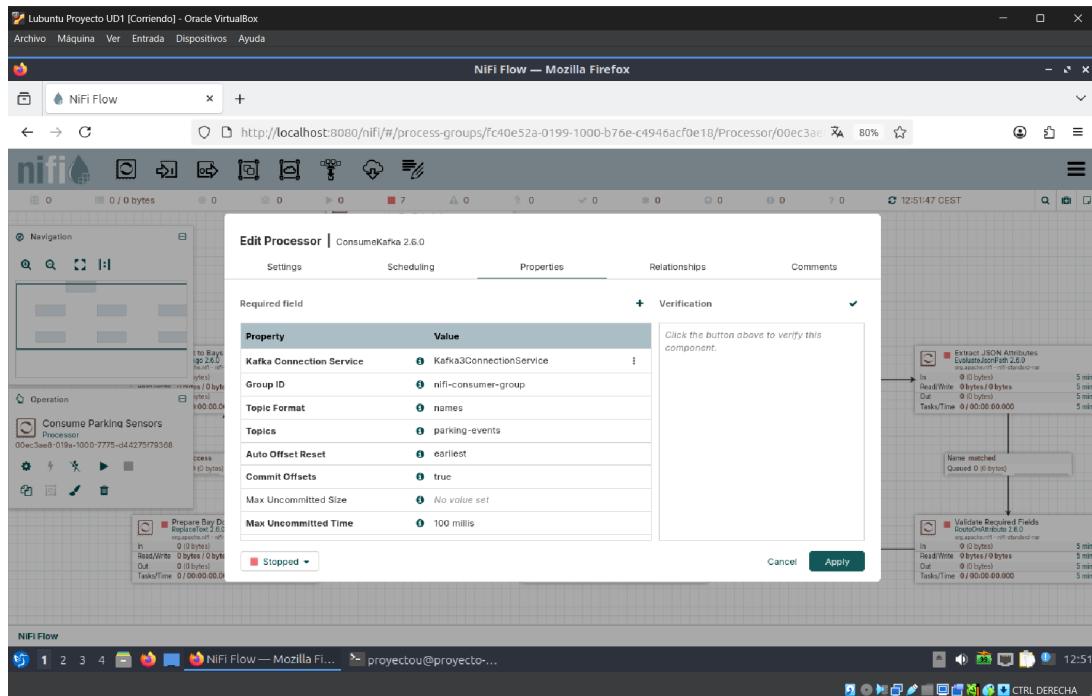


Figura D.30: Configuración final del procesador 'ConsumeKafka' (Tópico: parking-events).

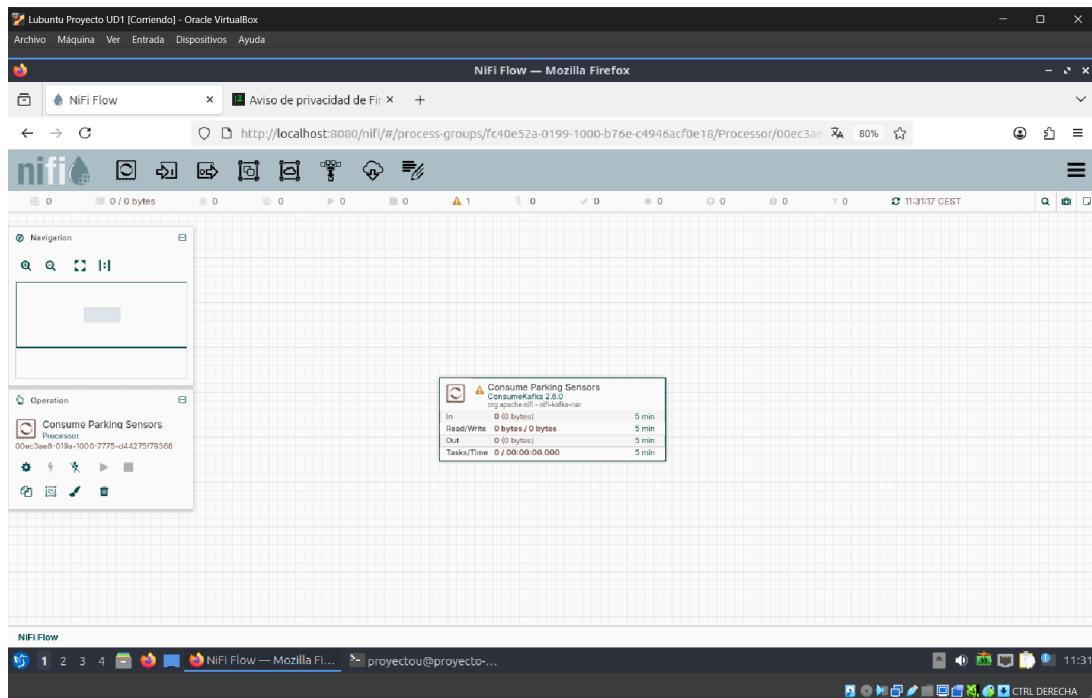


Figura D.31: Procesador 'Consume Parking Sensors' en el canvas.

SmartParking Flow — Monitorización Inteligente

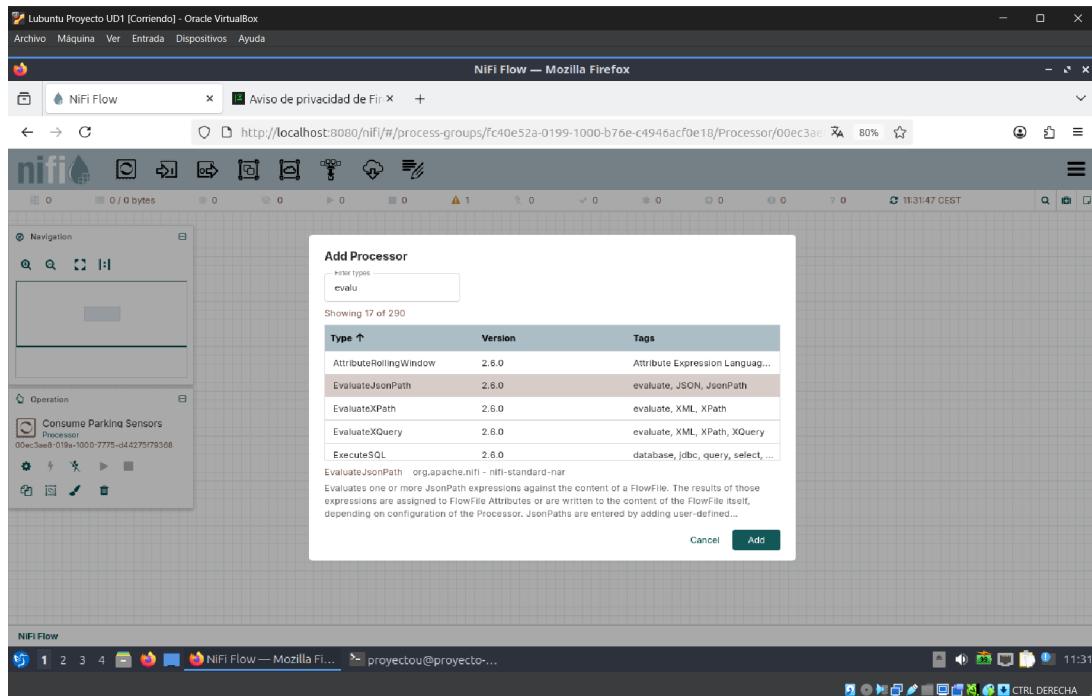


Figura D.32: Añadiendo el procesador 'EvaluateJsonPath'.

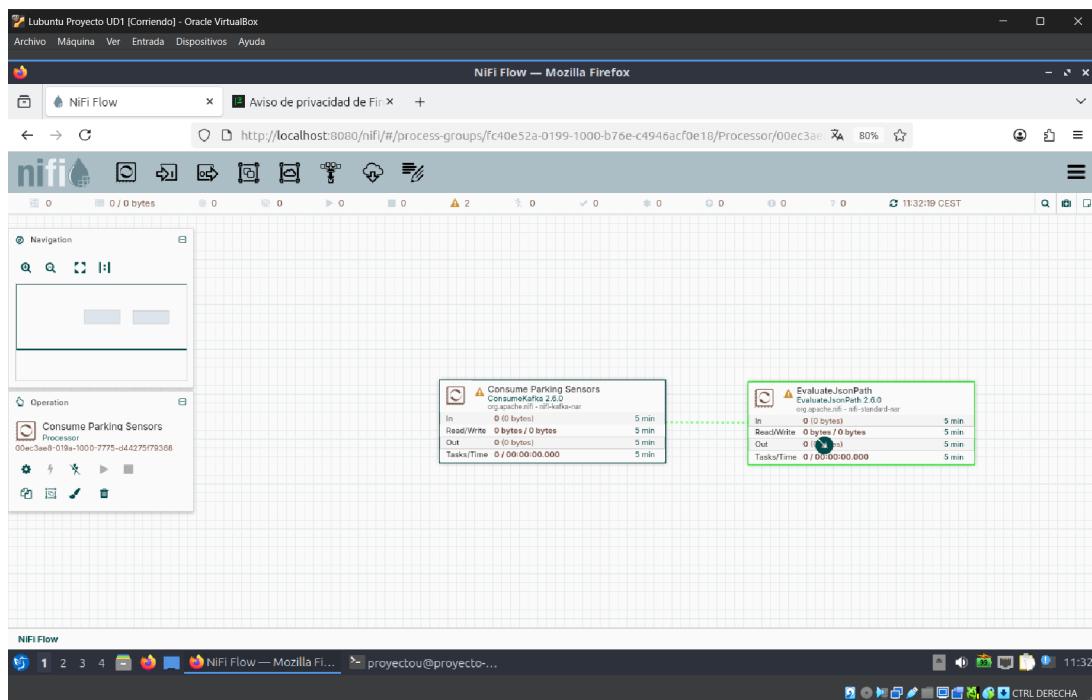


Figura D.33: Procesadores 'Consume Parking Sensors' y 'EvaluateJsonPath' en el canvas.

SmartParking Flow — Monitorización Inteligente

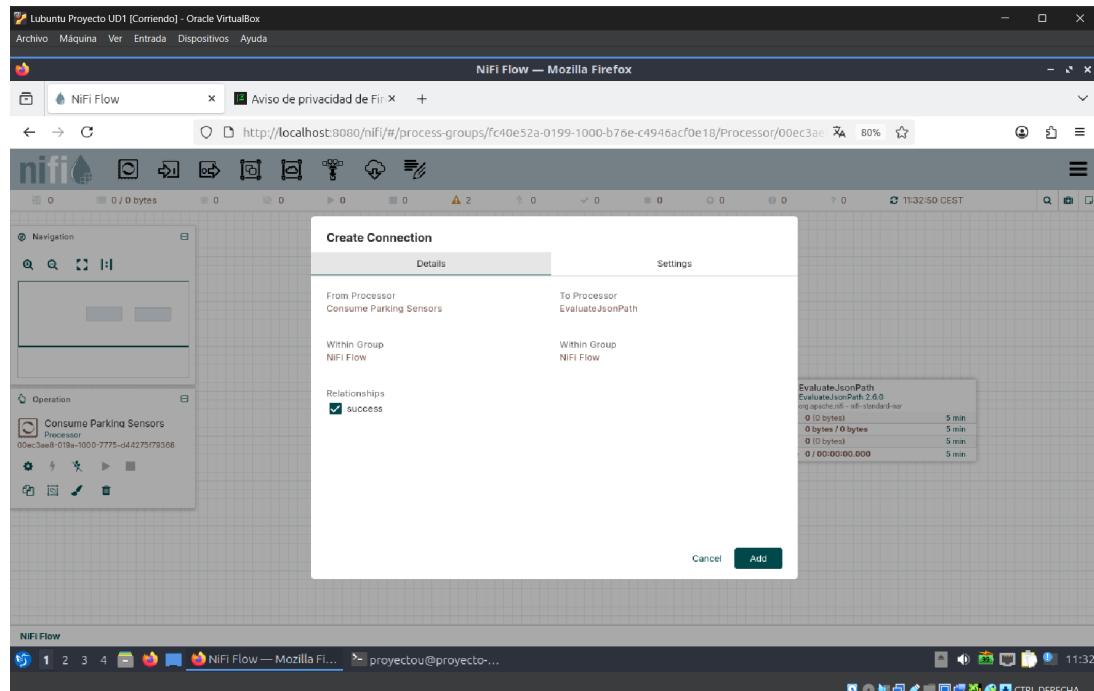


Figura D.34: Creando conexión 'success' entre 'ConsumeKafka' y 'EvaluateJsonPath'.

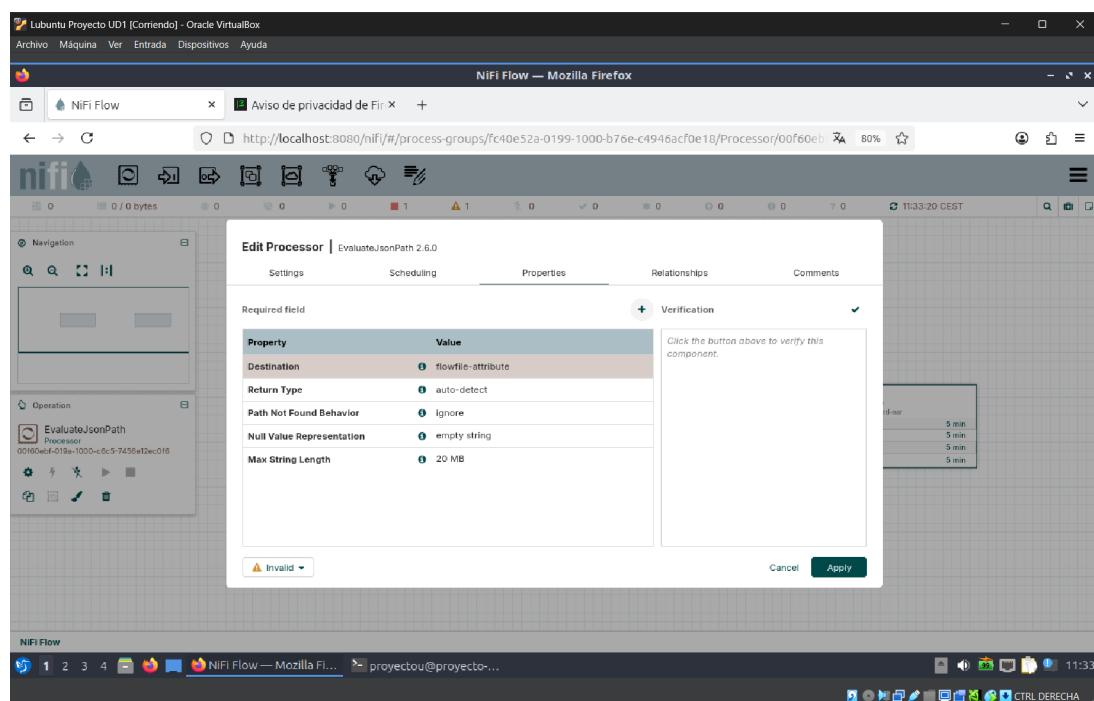


Figura D.35: Configuración (Properties) de 'EvaluateJsonPath': 'Destination: flowfile-attribute'.

SmartParking Flow — Monitorización Inteligente

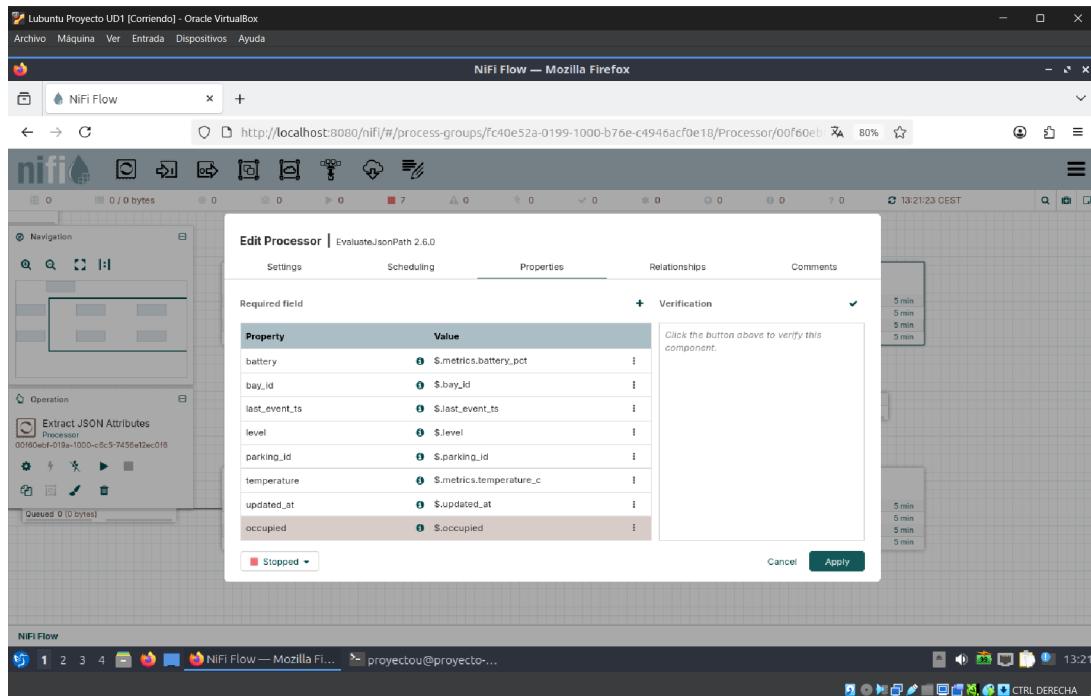


Figura D.36: Configuración (Properties) de 'EvaluateJsonPath': Extracción de atributos (battery, bay_id, ...).

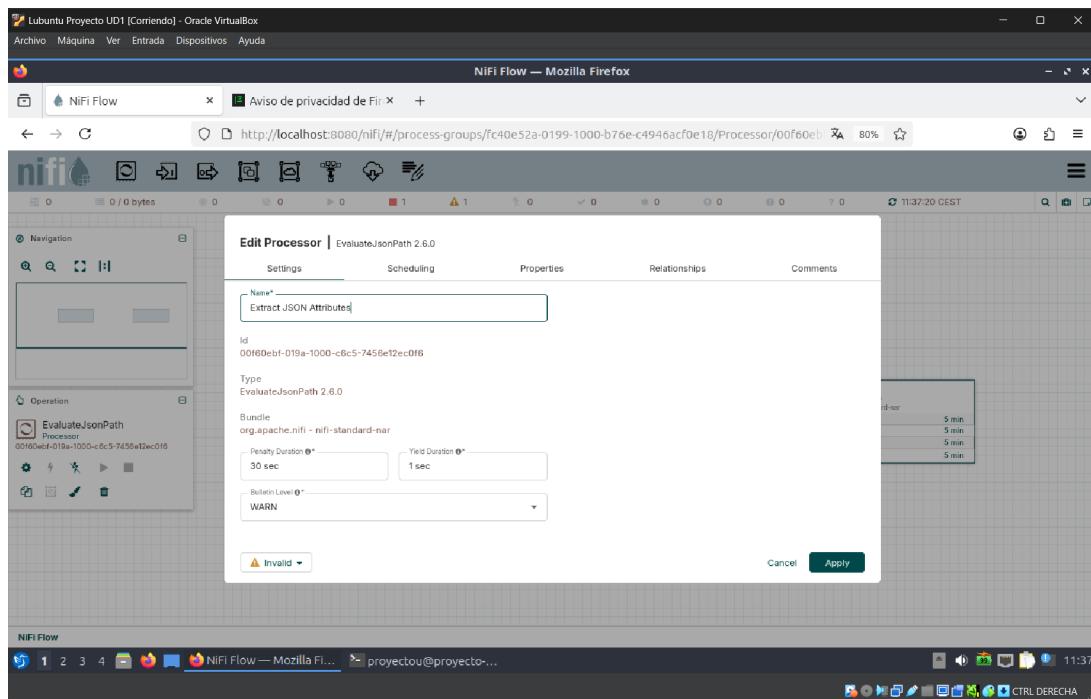


Figura D.37: Configuración (Settings) de 'EvaluateJsonPath': 'Name: Extract JSON Attributes'.

SmartParking Flow — Monitorización Inteligente

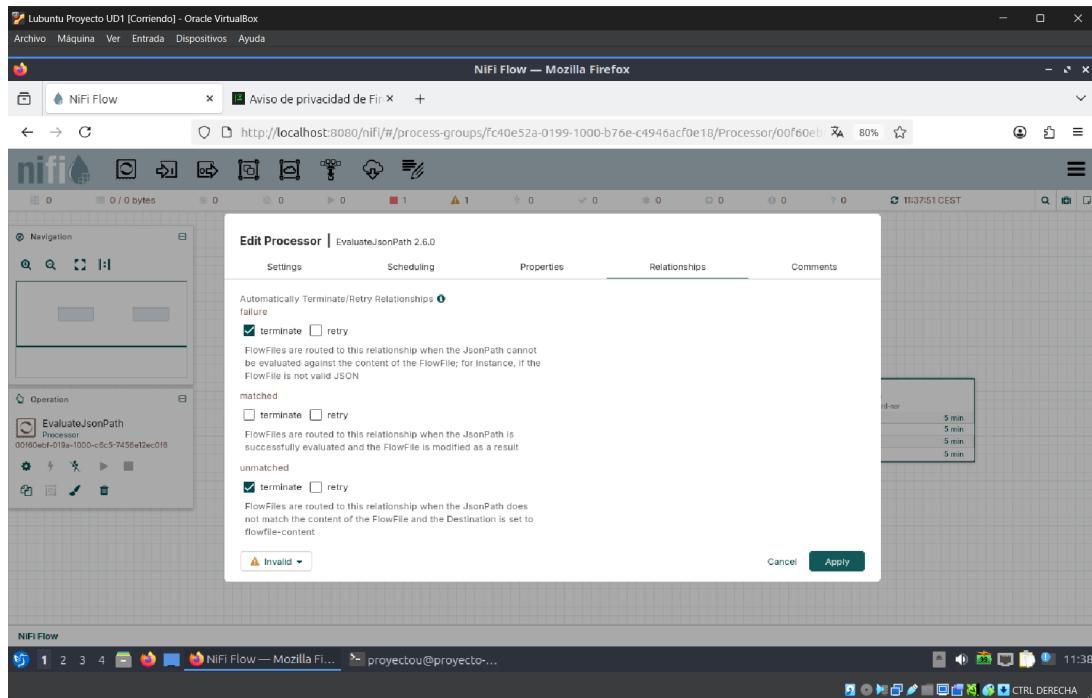


Figura D.38: Configuración (Relationships) de 'EvaluateJsonPath': Terminar 'unmatched'.

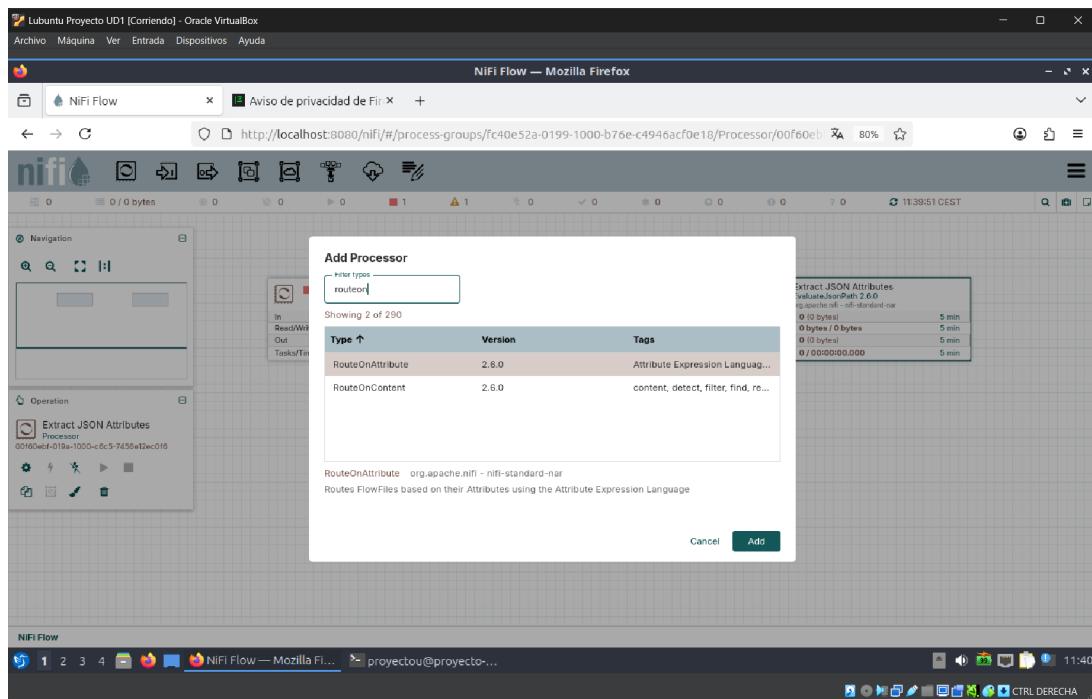


Figura D.39: Añadiendo el procesador 'RouteOnAttribute'.

SmartParking Flow — Monitorización Inteligente

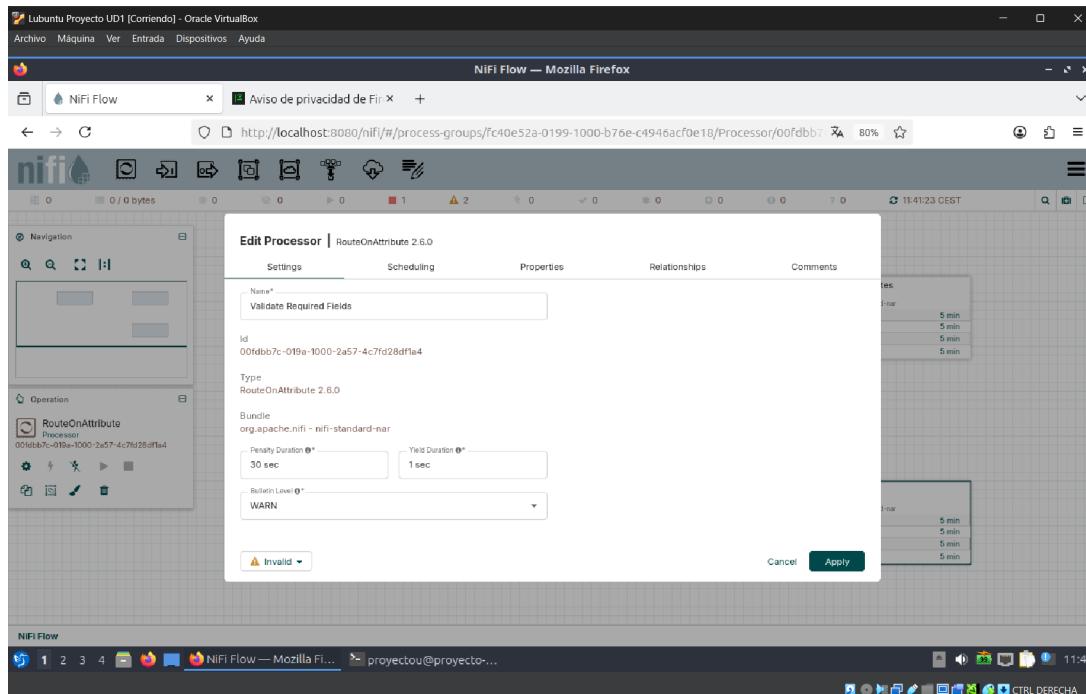


Figura D.40: Configuración (Settings) de 'RouteOnAttribute': 'Name: Validate Required Fields'.

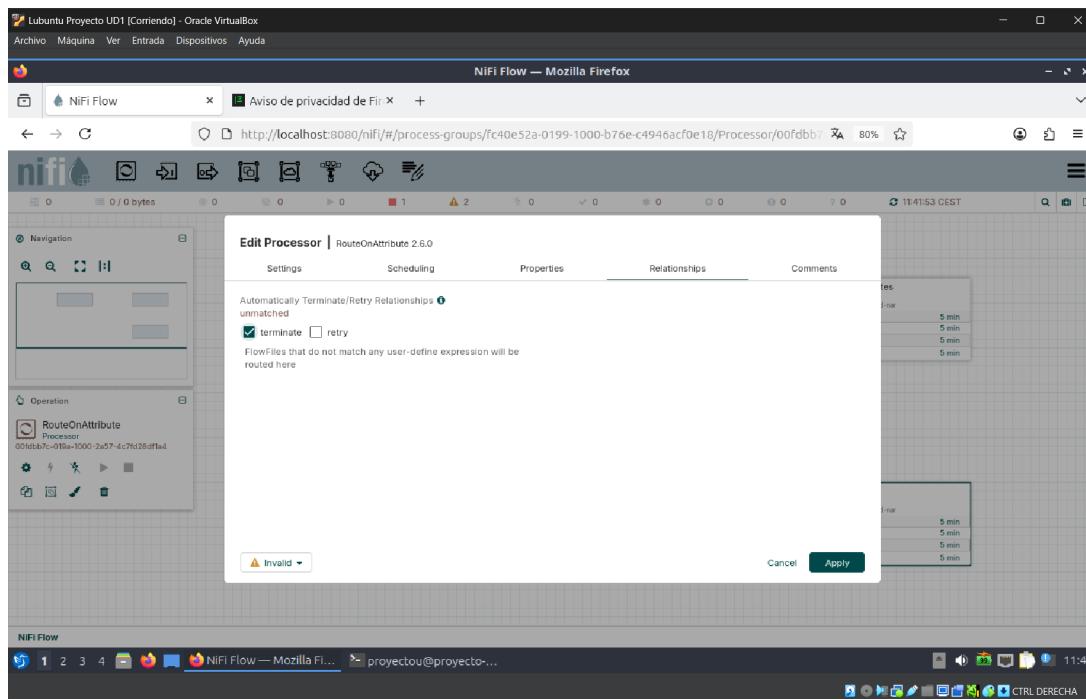


Figura D.41: Configuración (Relationships) de 'RouteOnAttribute': Terminar 'unmatched'.

SmartParking Flow — Monitorización Inteligente

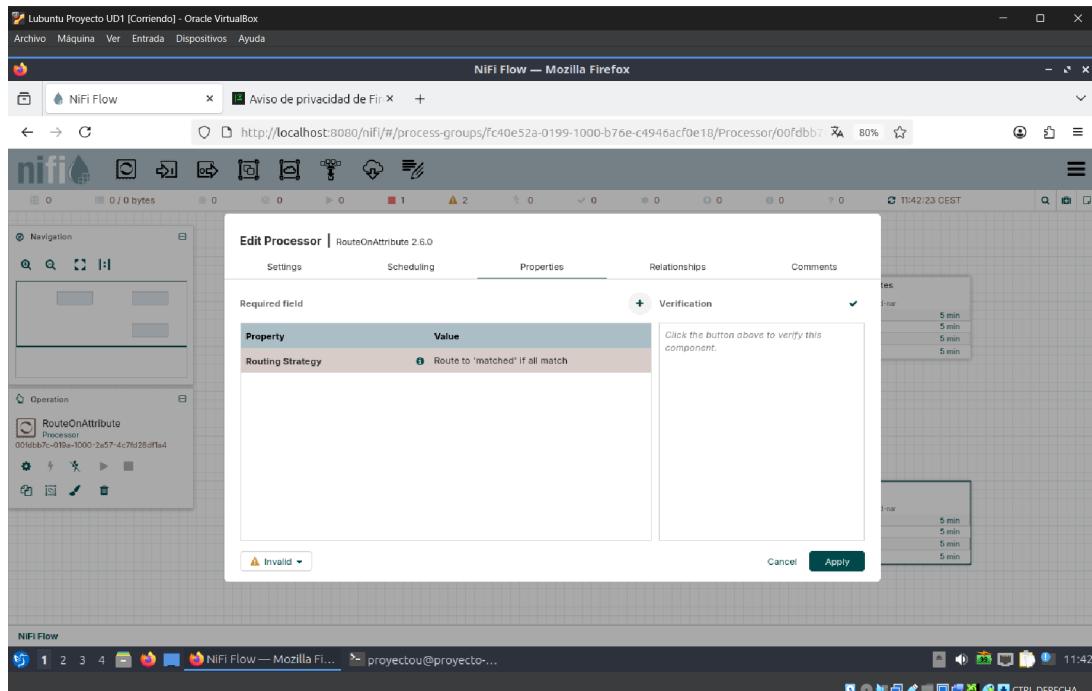


Figura D.42: Configuración (Properties) de 'RouteOnAttribute': 'Routing Strategy'.

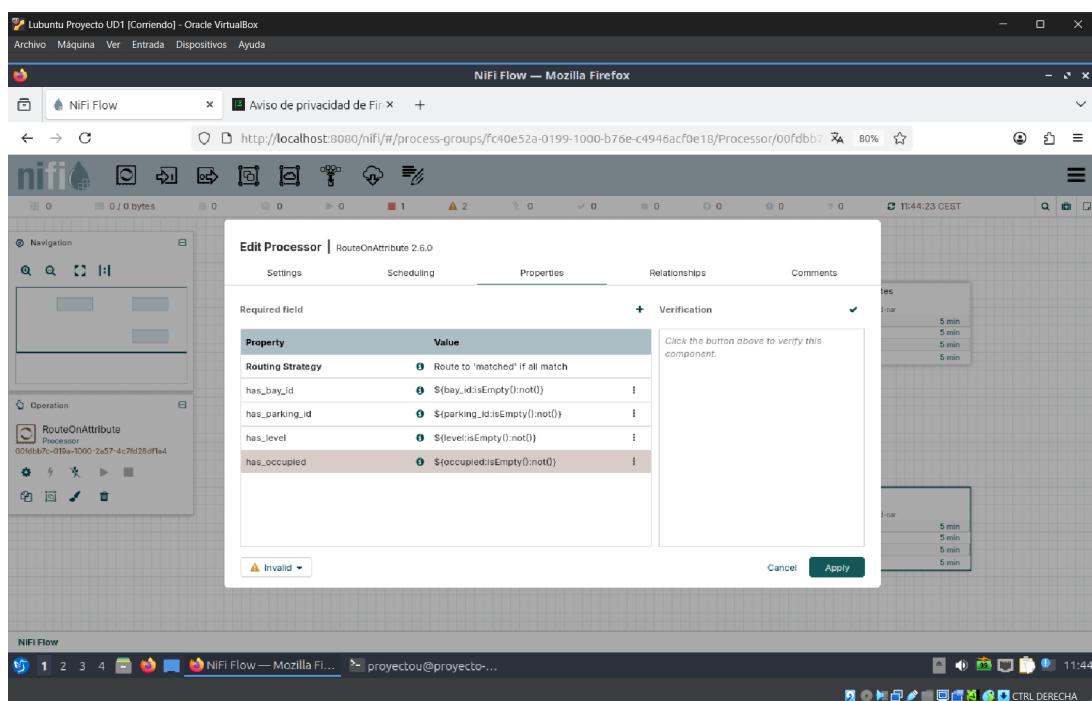


Figura D.43: Configuración (Properties) de 'RouteOnAttribute': Añadiendo propiedades de validación.

SmartParking Flow — Monitorización Inteligente

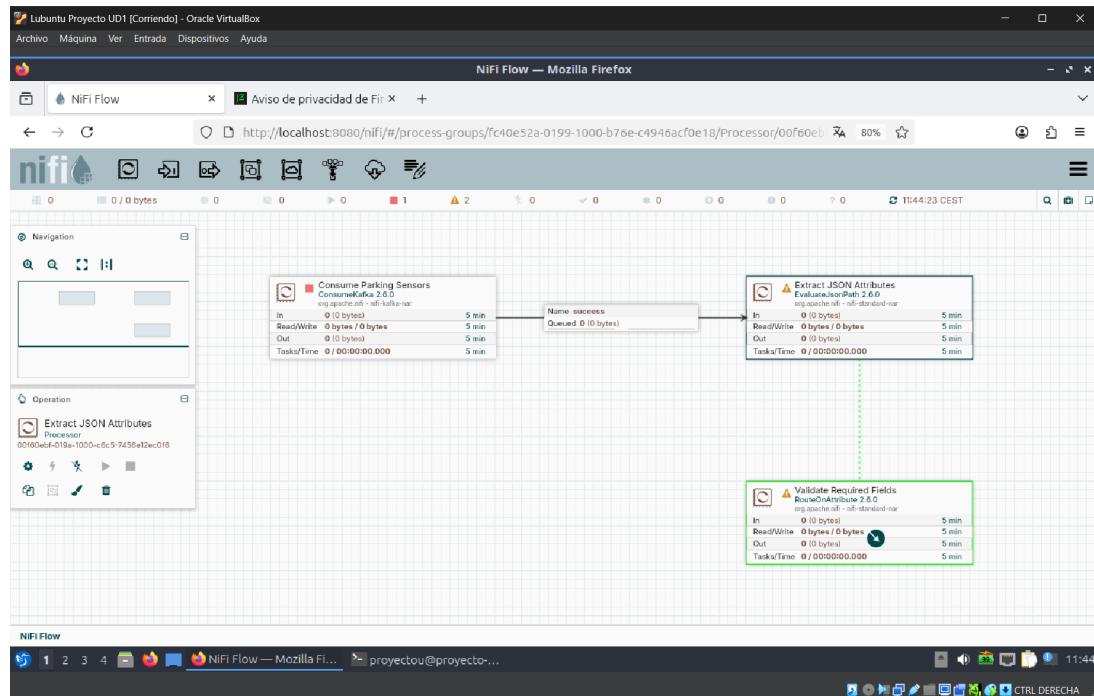


Figura D.44: Flujo con los tres procesadores iniciales.

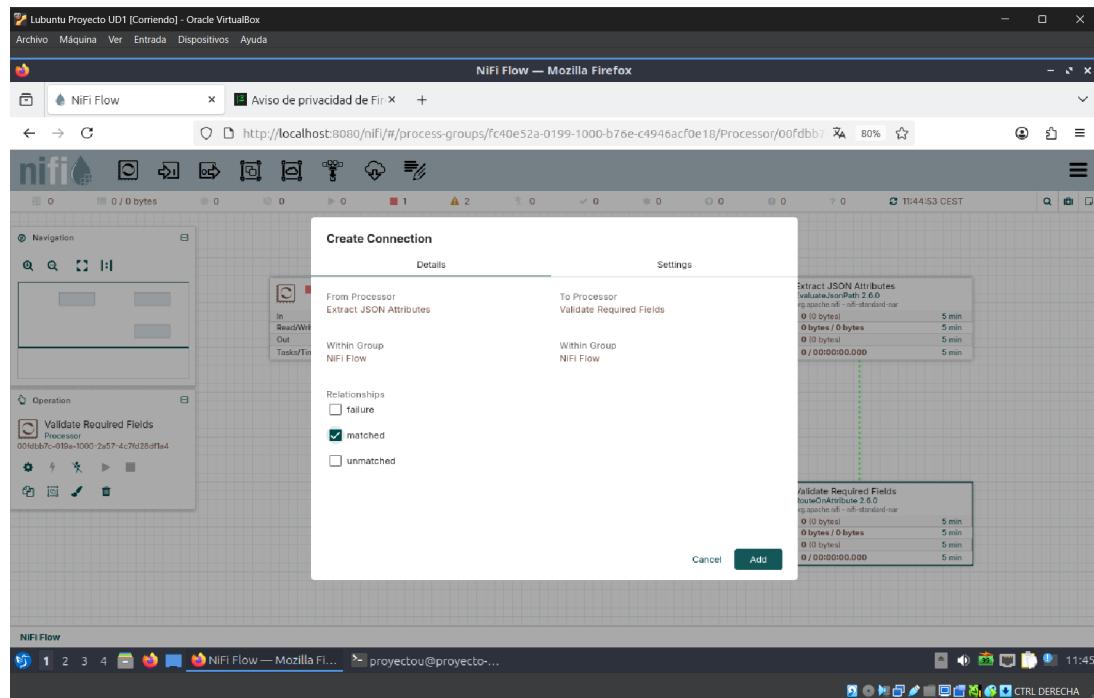


Figura D.45: Creando conexión 'matched' entre 'EvaluateJsonPath' y 'RouteOnAttribute'.

SmartParking Flow — Monitorización Inteligente

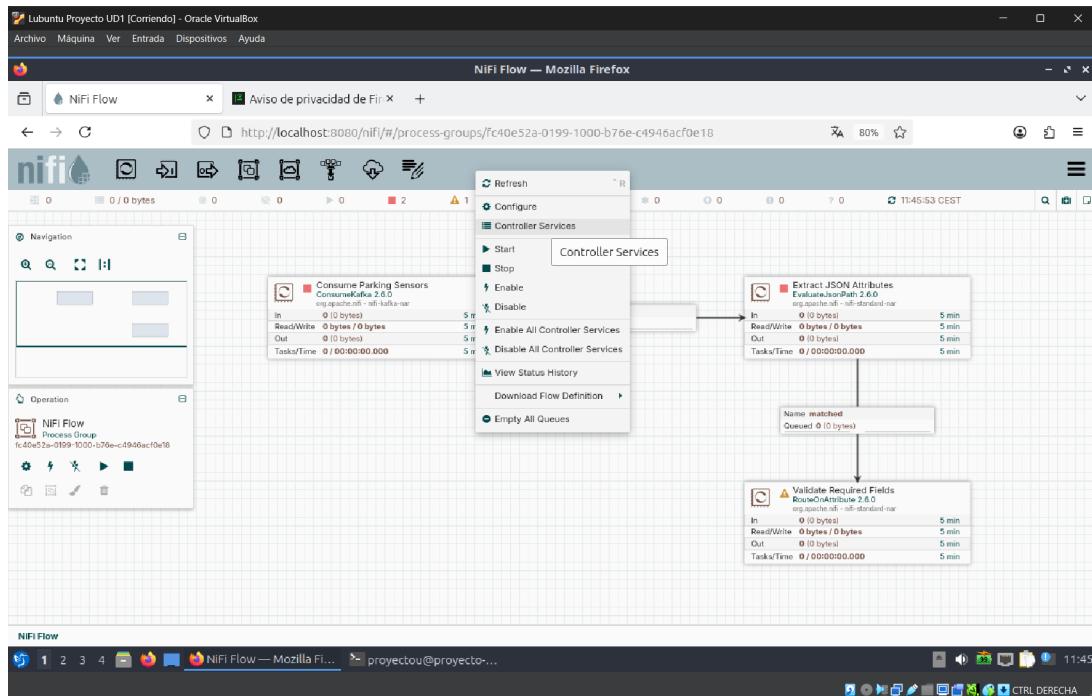


Figura D.46: Accediendo a la configuración de Controller Services del Process Group.

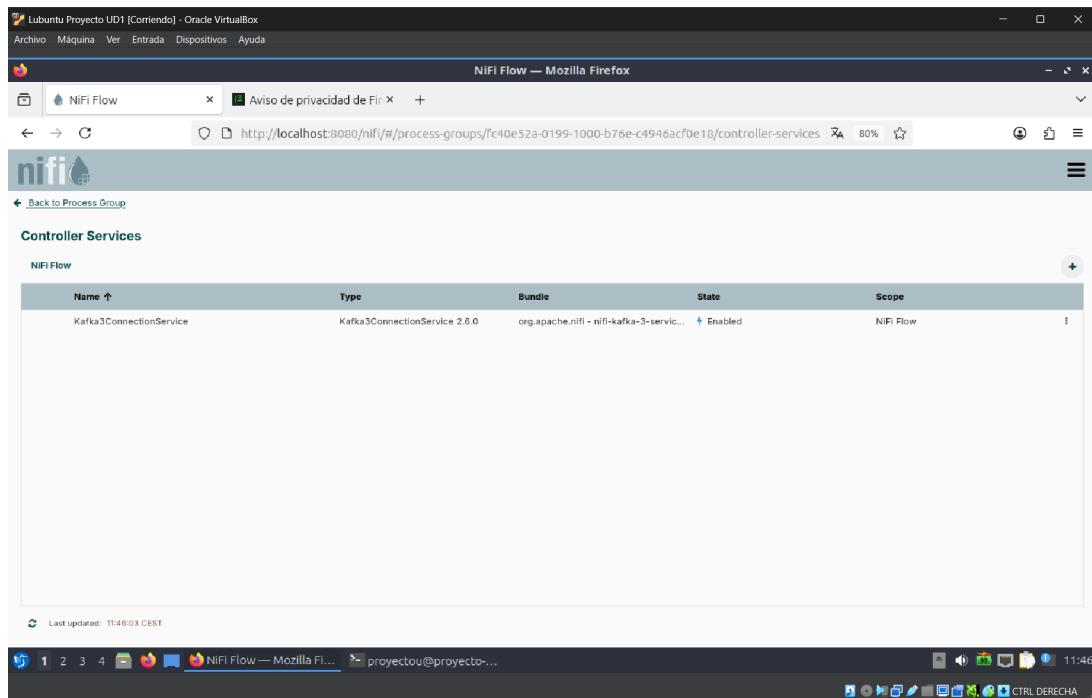


Figura D.47: Vista de 'Kafka3ConnectionService' habilitado.

SmartParking Flow — Monitorización Inteligente

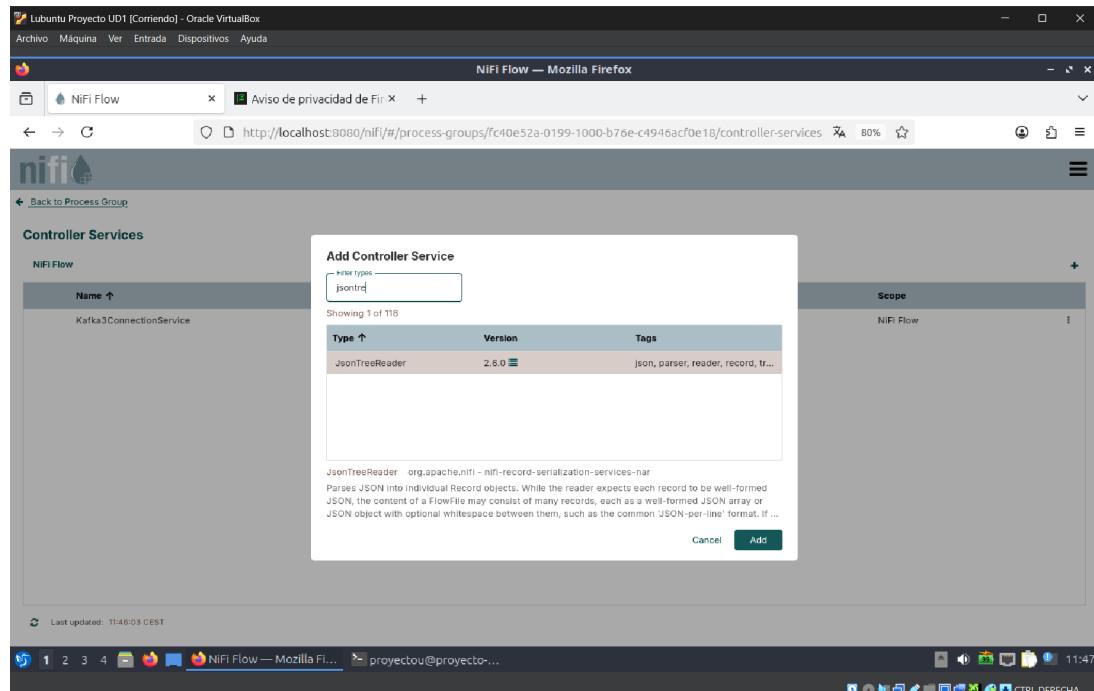


Figura D.48: Añadiendo un nuevo Controller Service: 'JsonTreeReader'.

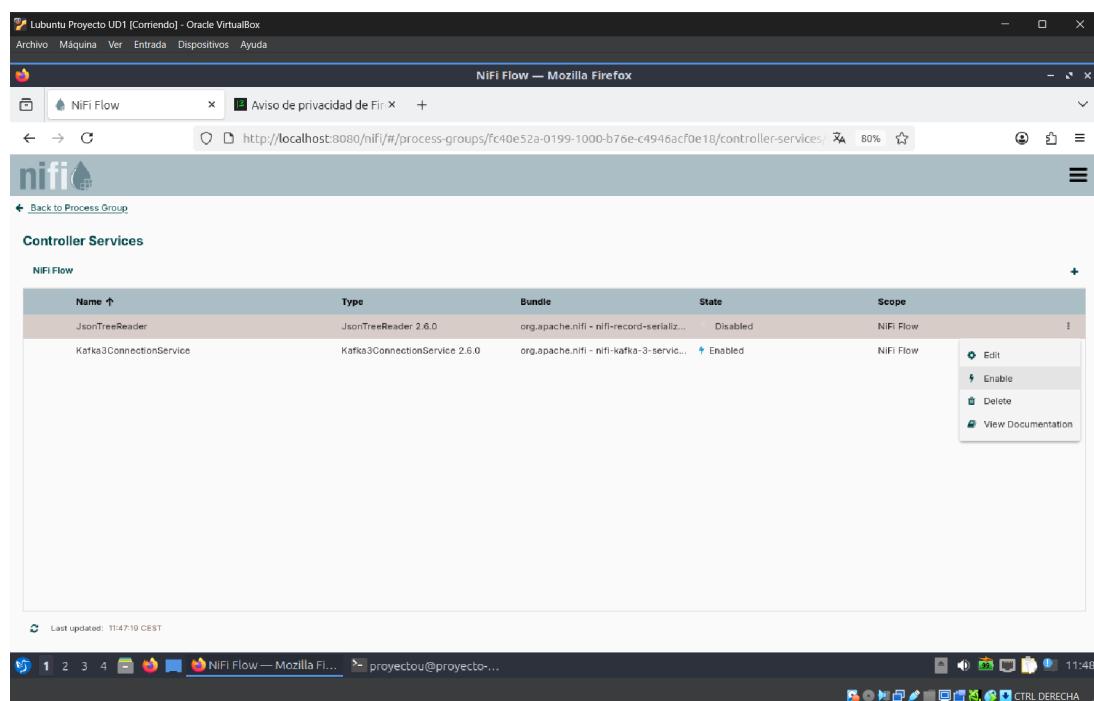


Figura D.49: Controller Service 'JsonTreeReader' en estado 'Disabled'.

SmartParking Flow — Monitorización Inteligente

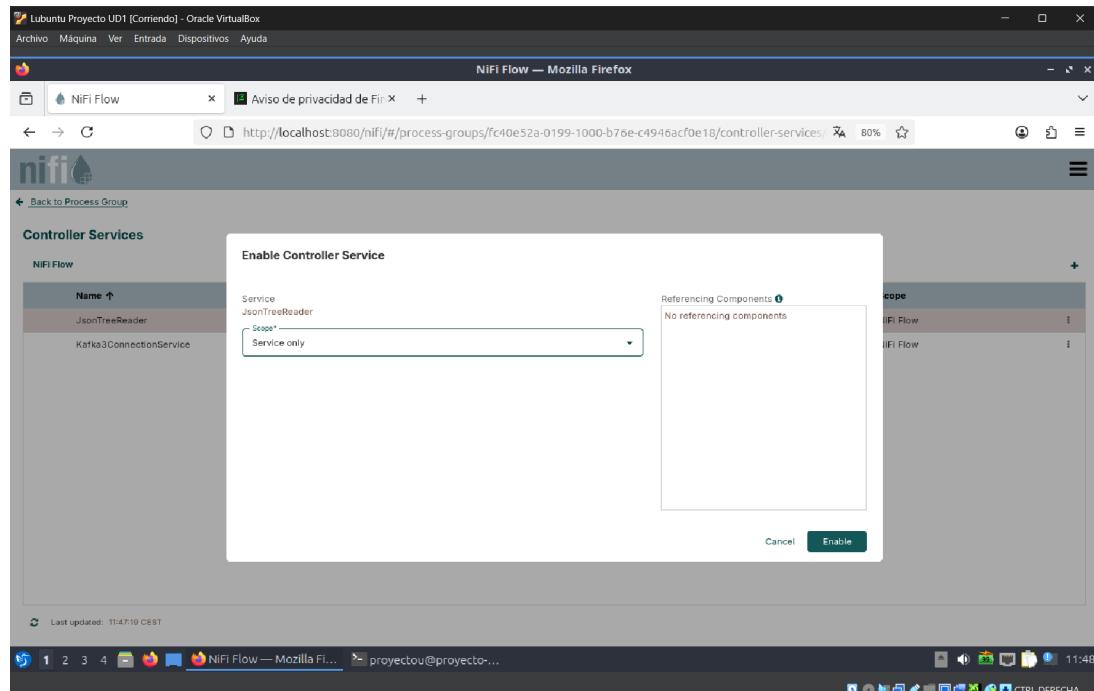


Figura D.50: Habilitando el 'JsonTreeReader'.

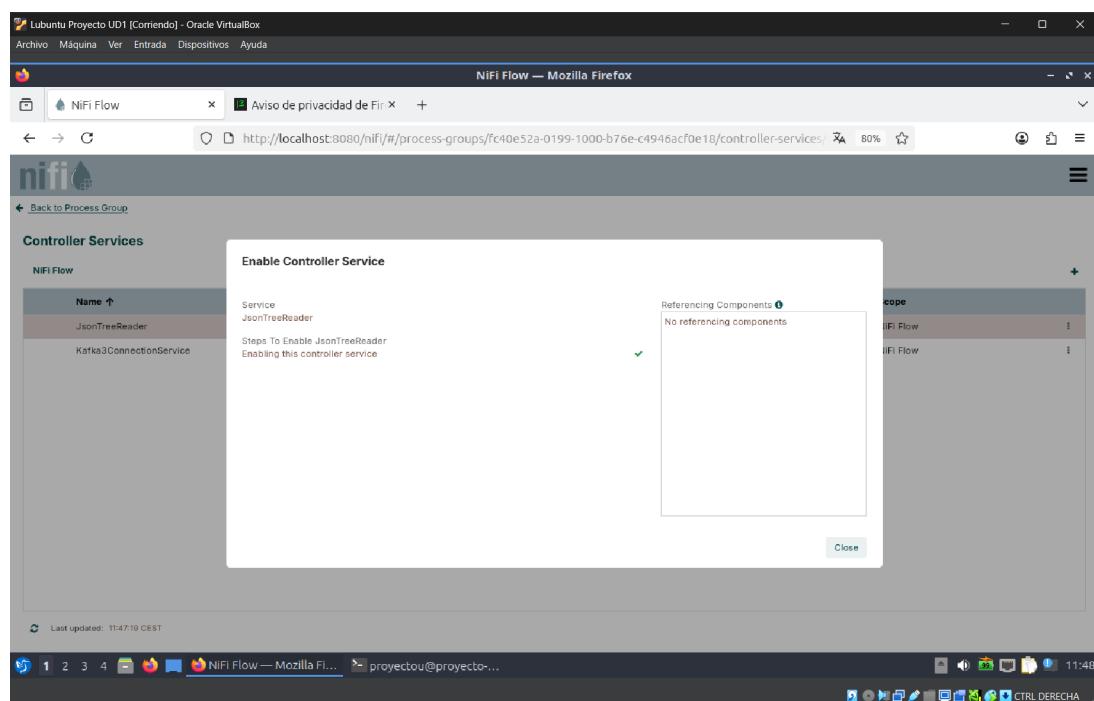


Figura D.51: Confirmación de habilitación del 'JsonTreeReader'.

SmartParking Flow — Monitorización Inteligente

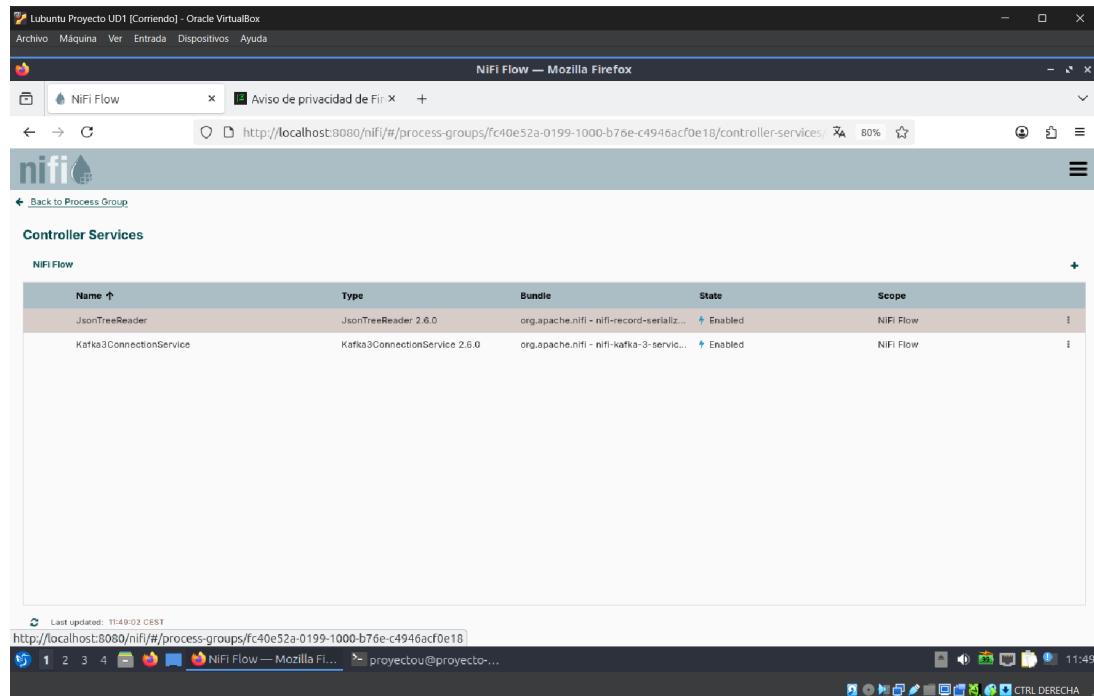


Figura D.52: Ambos Controller Services ('JsonTreeReader' y 'Kafka3ConnectionService') habilitados.

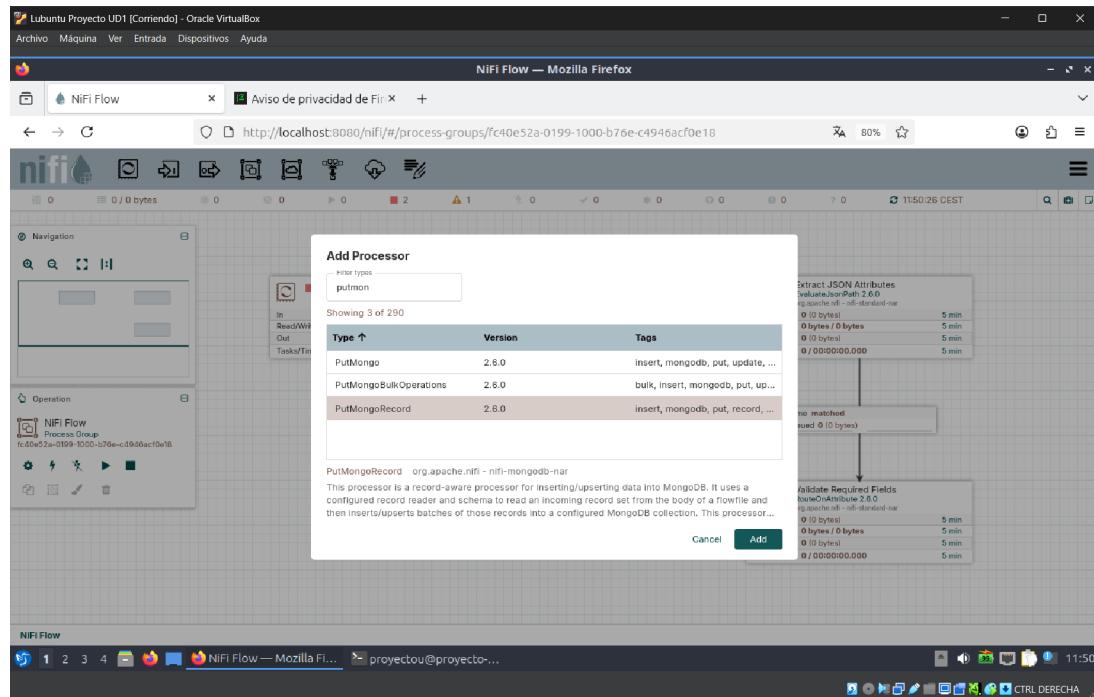


Figura D.53: Añadiendo el procesador 'PutMongoRecord'.

SmartParking Flow — Monitorización Inteligente

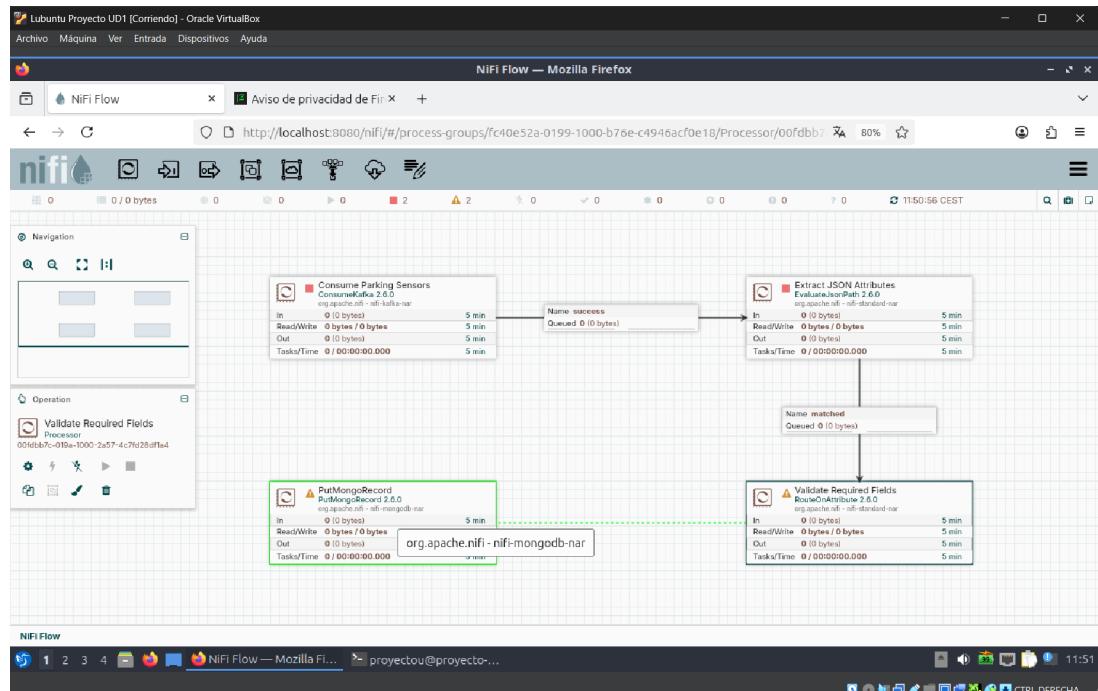


Figura D.54: Flujo con el procesador 'PutMongoRecord' añadido.

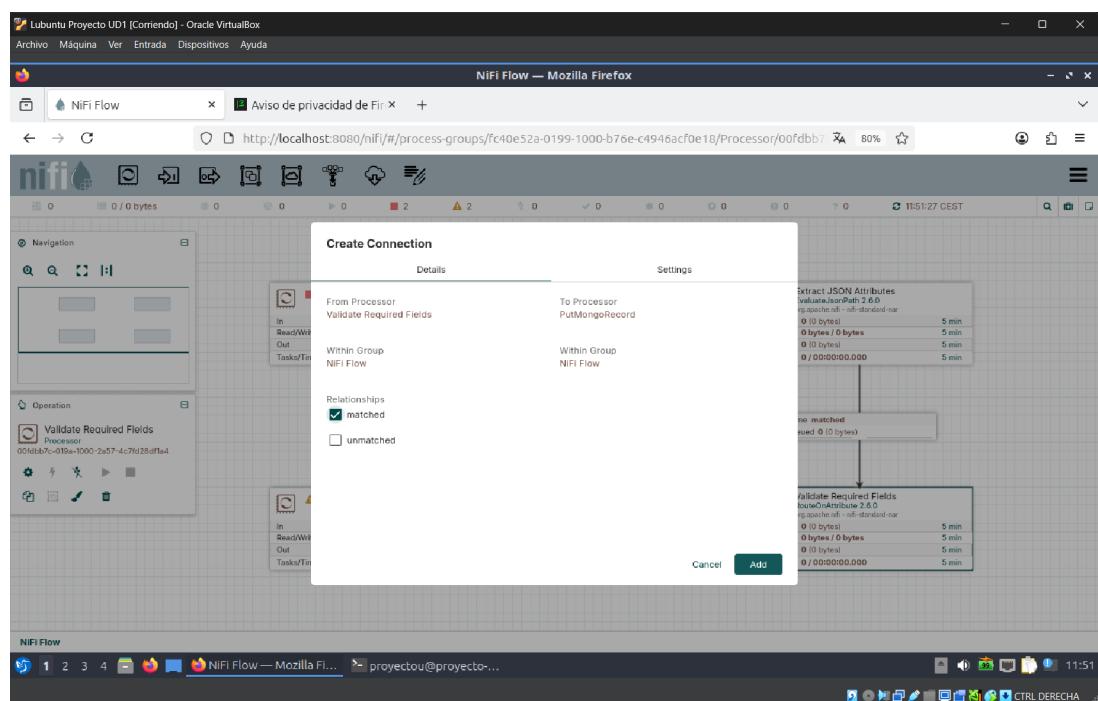


Figura D.55: Creando conexión 'matched' entre 'RouteOnAttribute' y 'PutMongoRecord'.

SmartParking Flow — Monitorización Inteligente

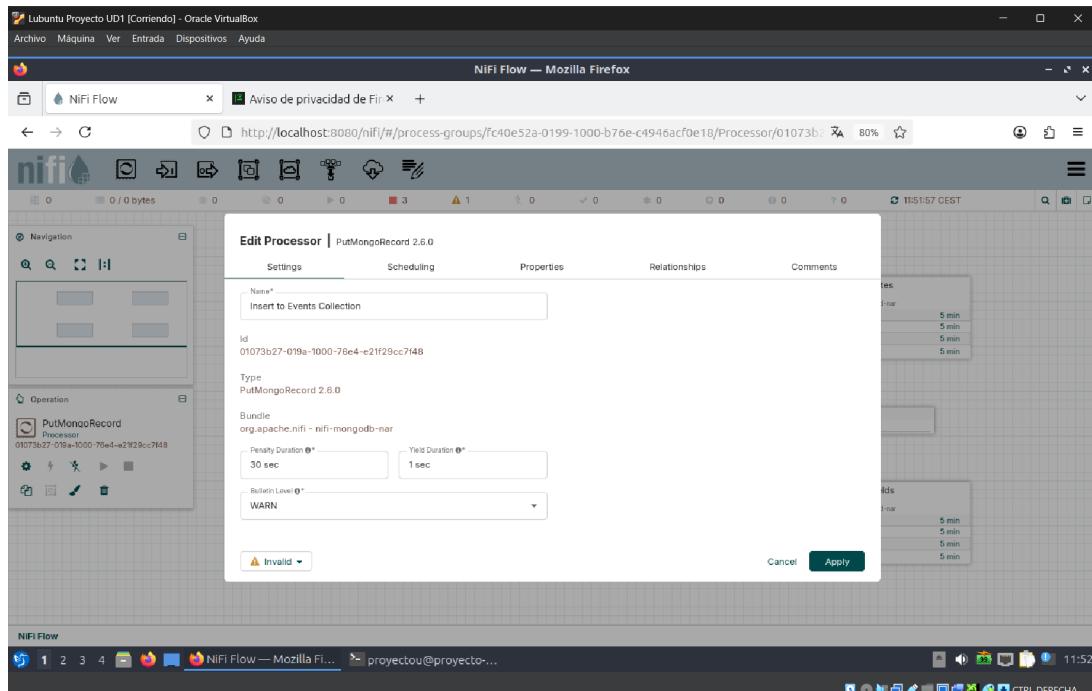


Figura D.56: Configuración (Settings) de 'PutMongoRecord': 'Name: Insert to Events Collection'.

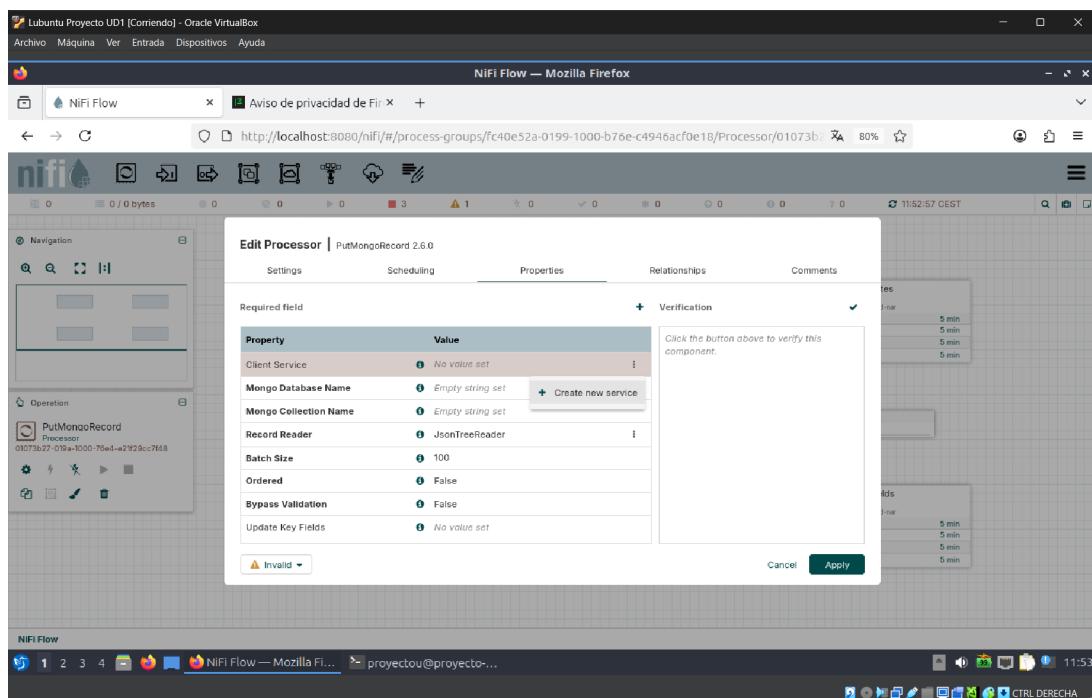


Figura D.57: Configuración (Properties) de 'PutMongoRecord' (vacía).

SmartParking Flow — Monitorización Inteligente

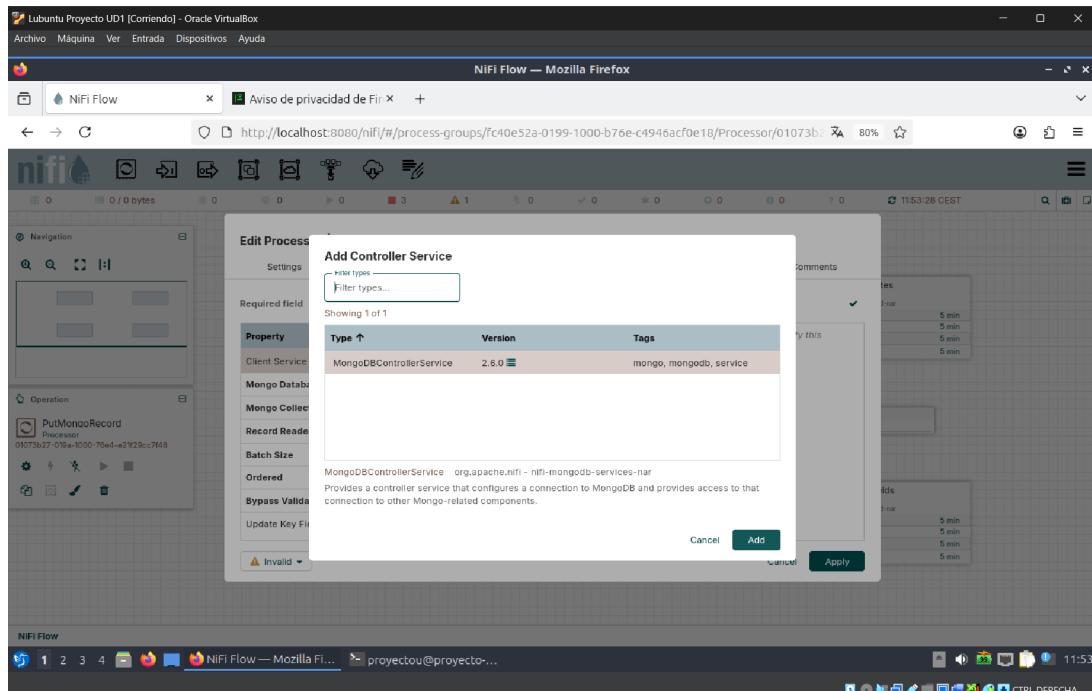


Figura D.58: Creando un nuevo Controller Service: 'MongoDBControllerService'.

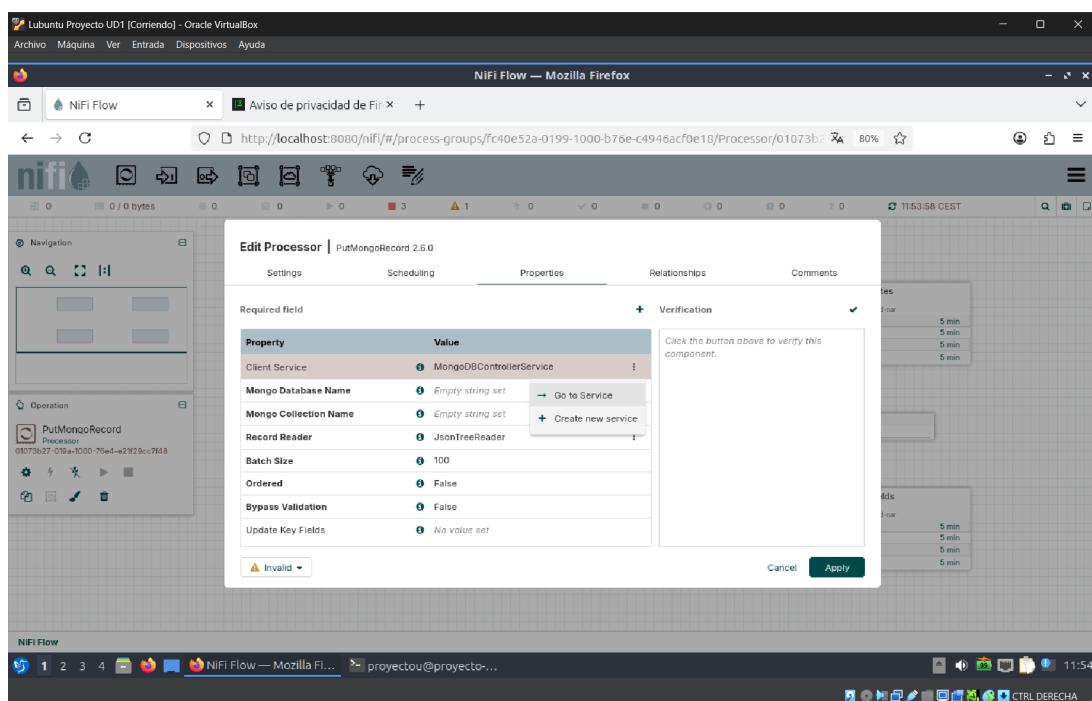


Figura D.59: Configuración de 'PutMongoRecord' (Properties) - se usará 'MongoDBControllerService'.

SmartParking Flow — Monitorización Inteligente

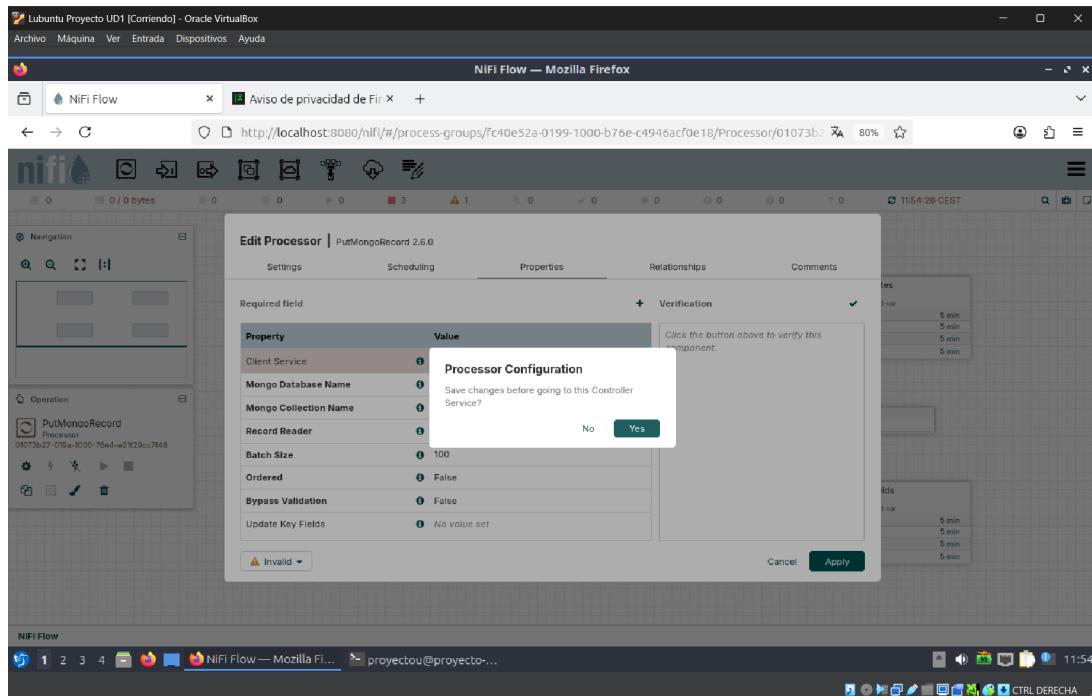


Figura D.60: Aviso de guardado de Controller Service.

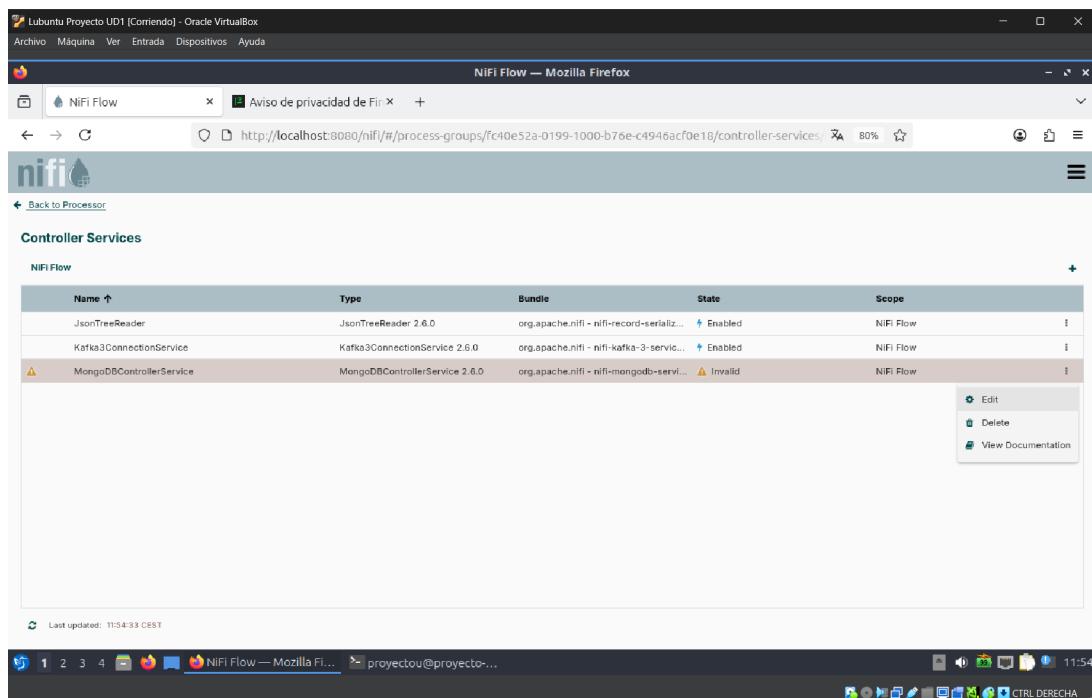


Figura D.61: Accediendo a la configuración del 'MongoDBControllerService'.

SmartParking Flow — Monitorización Inteligente

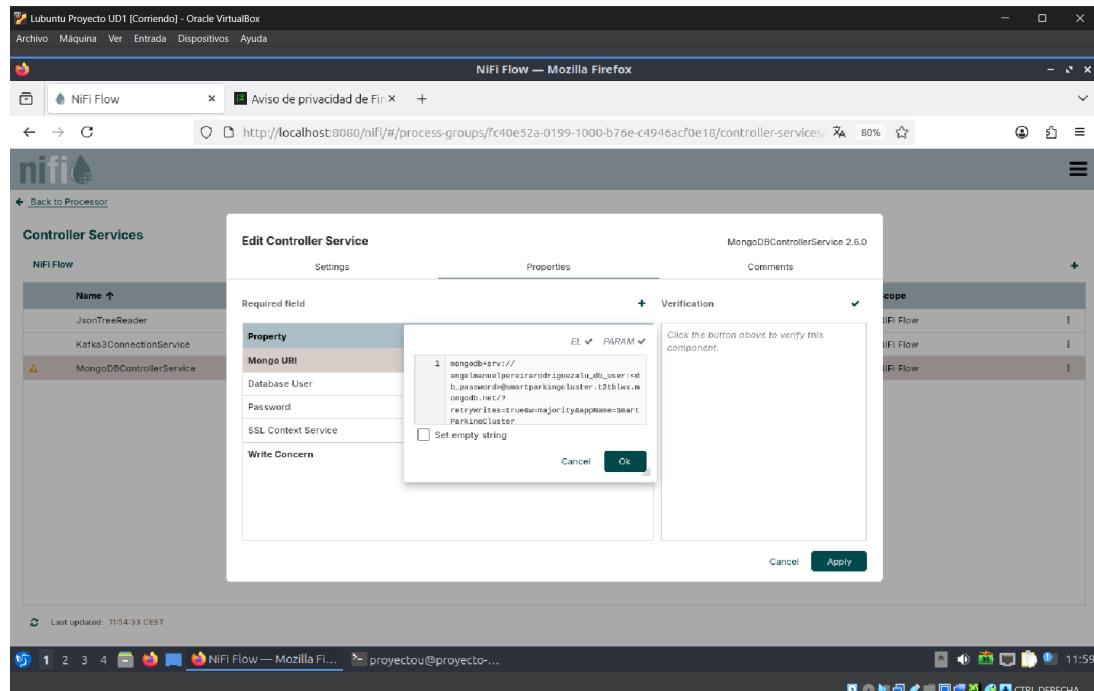


Figura D.62: Editando 'MongoDBControllerService': Pegando la URI de conexión de Atlas.

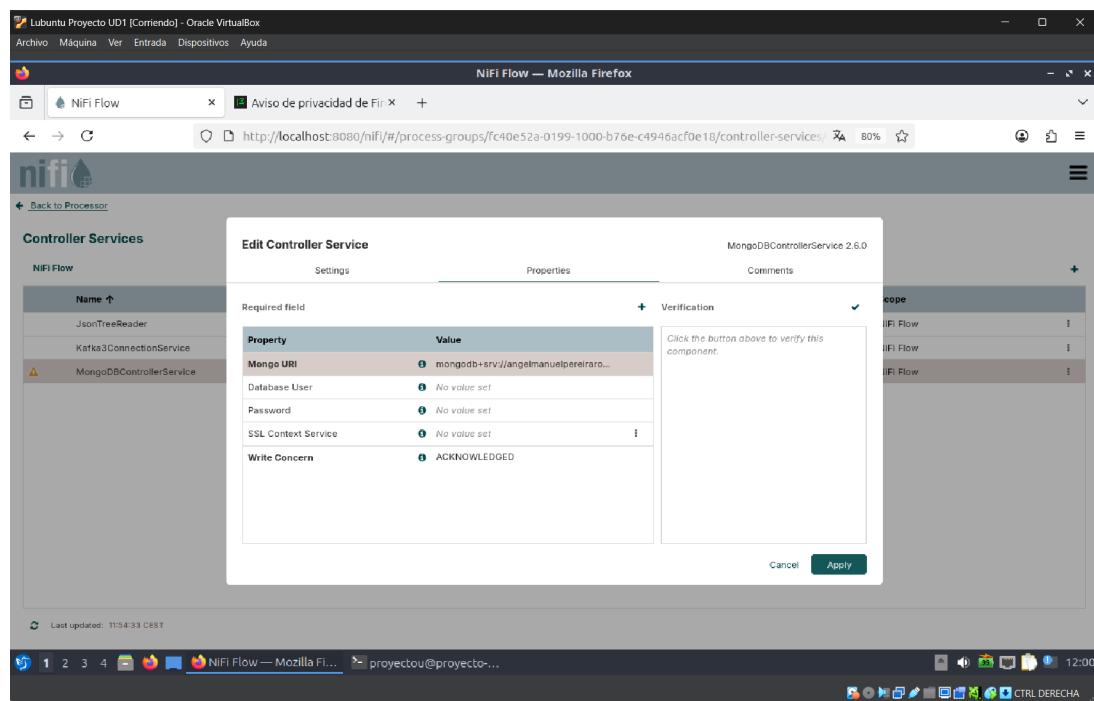


Figura D.63: Configuración (Properties) de 'MongoDBControllerService' con la URI.

SmartParking Flow — Monitorización Inteligente

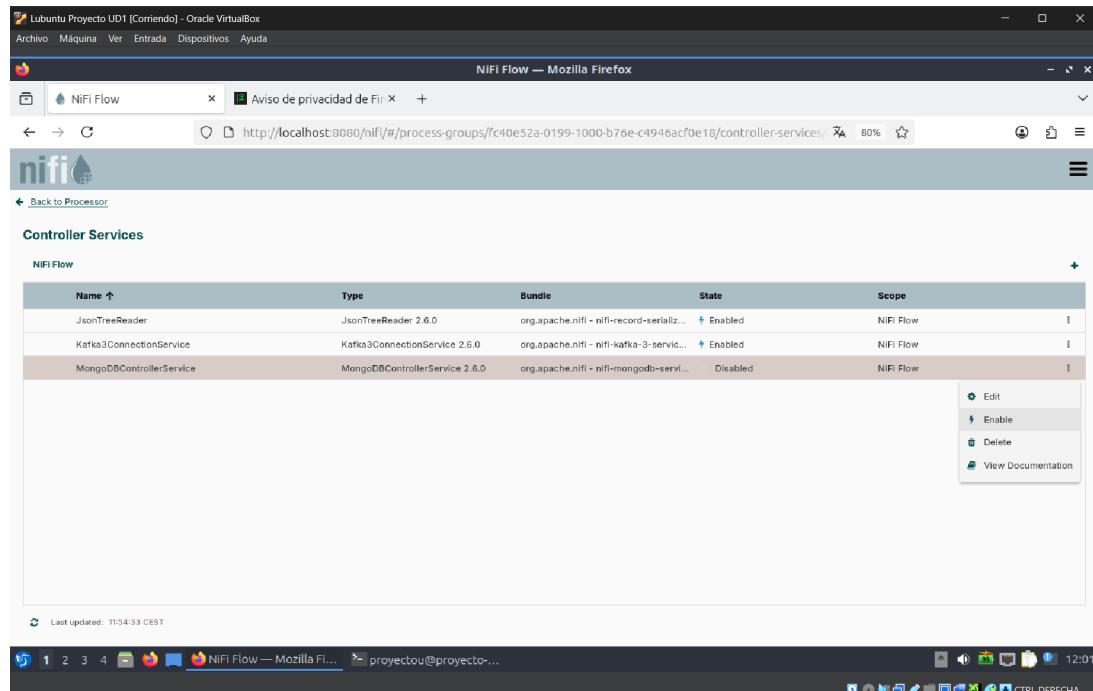


Figura D.64: Habilitando el 'MongoDBControllerService'.

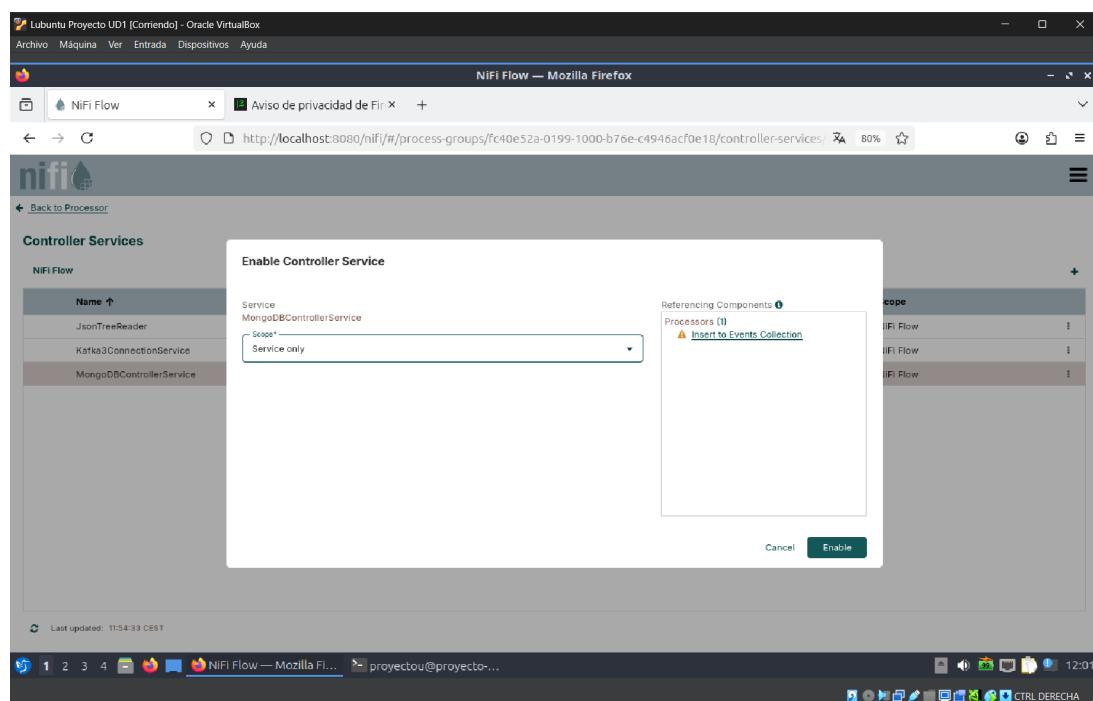


Figura D.65: Confirmación de habilitación del 'MongoDBControllerService'.

SmartParking Flow — Monitorización Inteligente

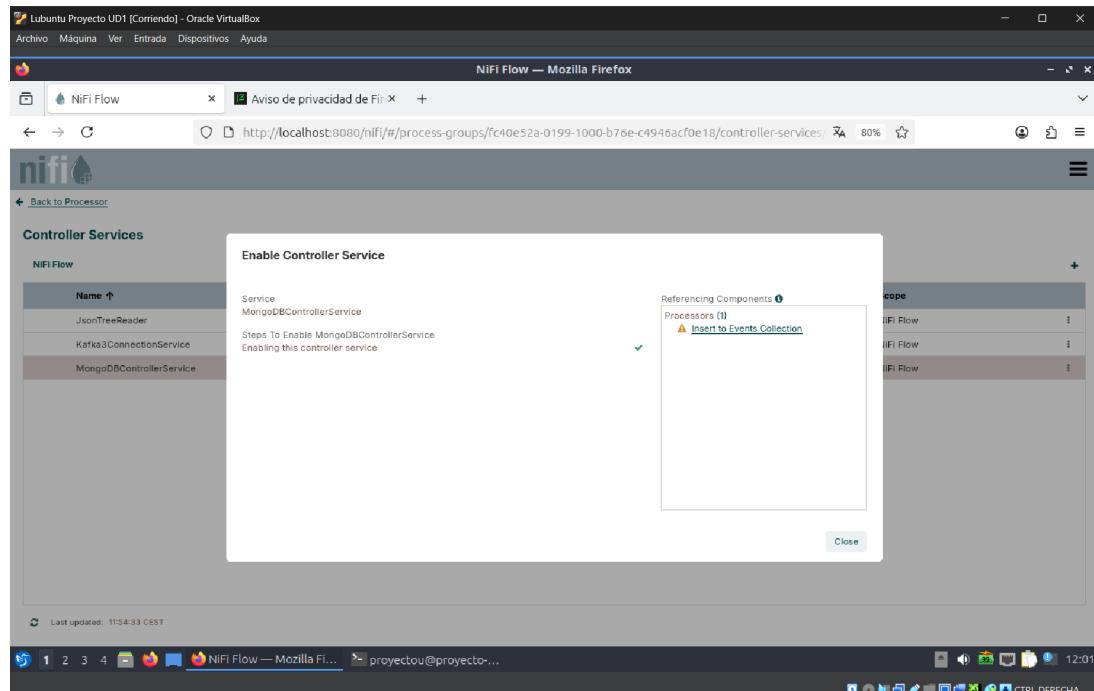


Figura D.66: Habilitando el 'MongoDBControllerService'.

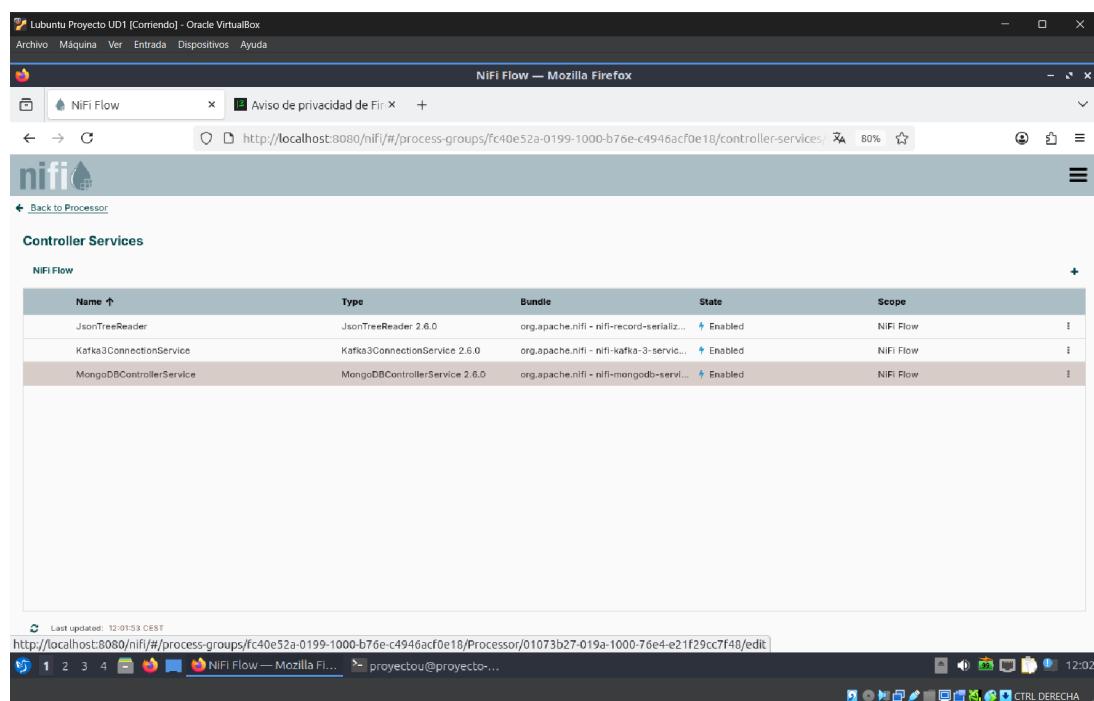


Figura D.67: Todos los Controller Services habilitados.

SmartParking Flow — Monitorización Inteligente

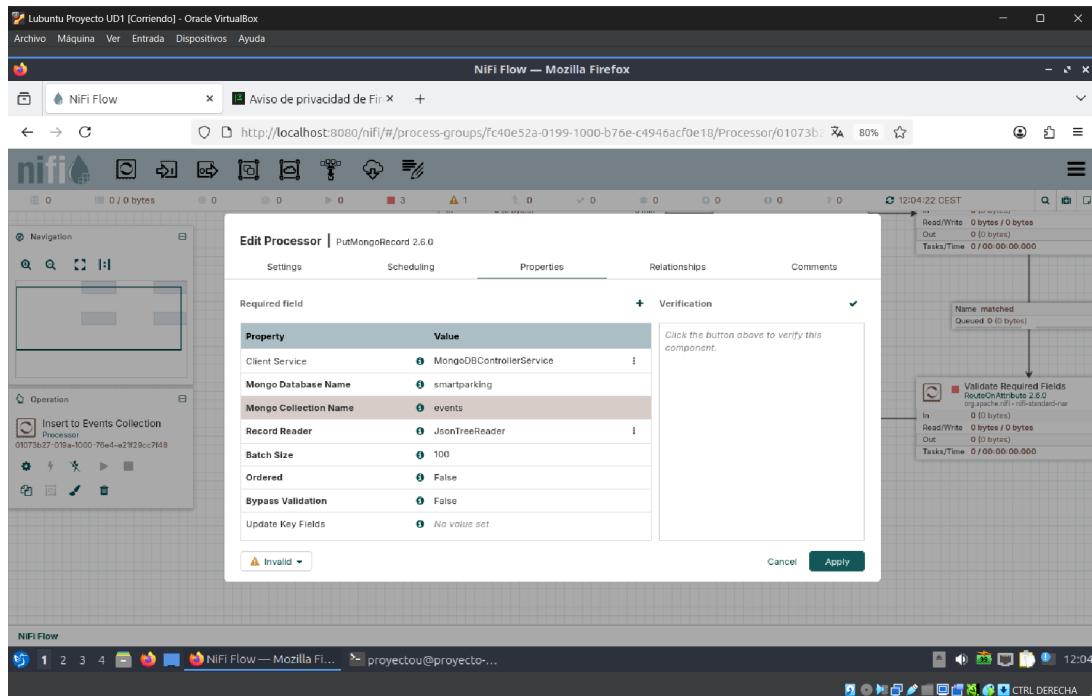


Figura D.68: Configuración de 'PutMongoRecord': 'Mongo Database Name: smartparking', 'Collection: events'.

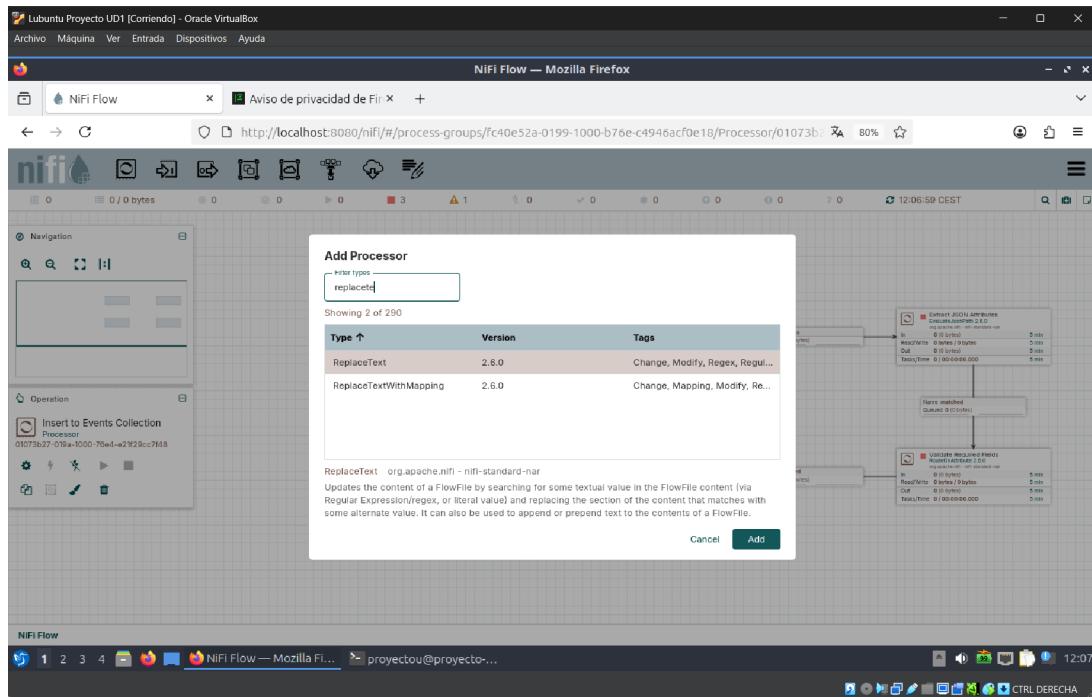


Figura D.69: Añadiendo el procesador 'ReplaceText'.

SmartParking Flow — Monitorización Inteligente

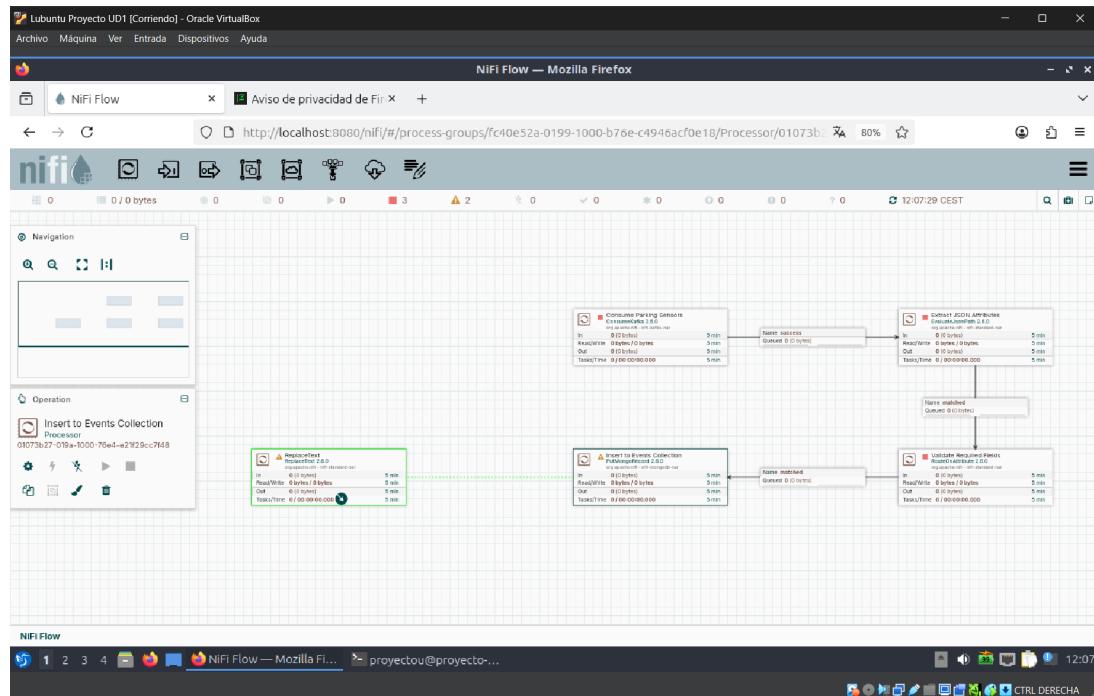


Figura D.70: Flujo con el procesador 'ReplaceText' añadido.

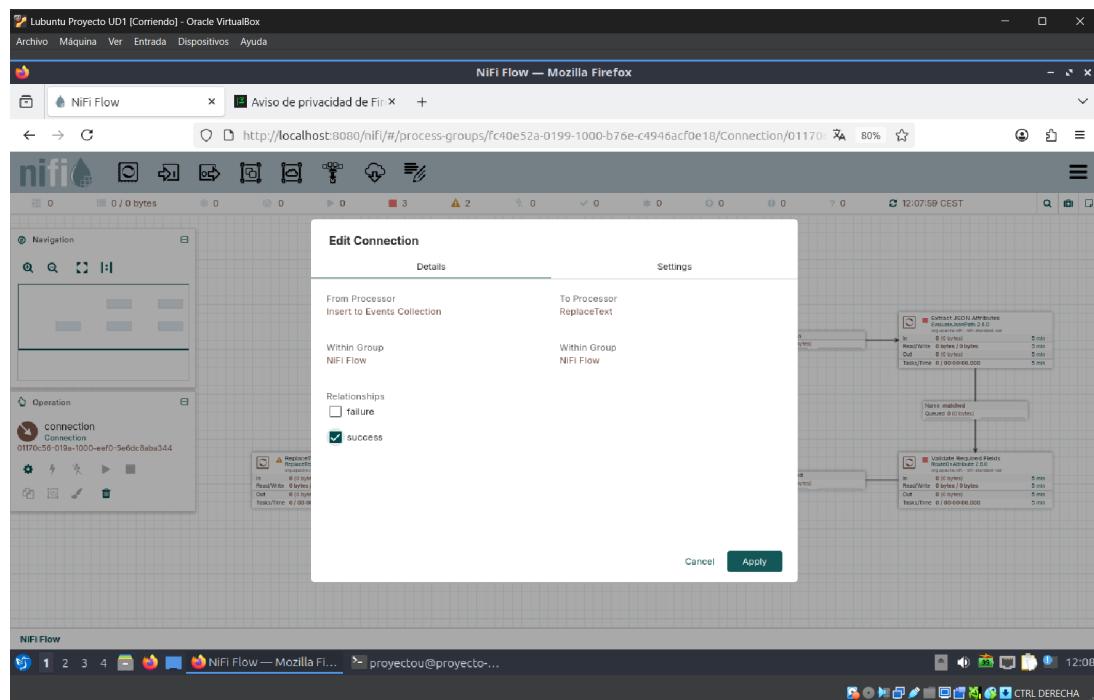


Figura D.71: Creando conexión 'success' entre 'Insert to Events Collection' y 'ReplaceText'.

SmartParking Flow — Monitorización Inteligente

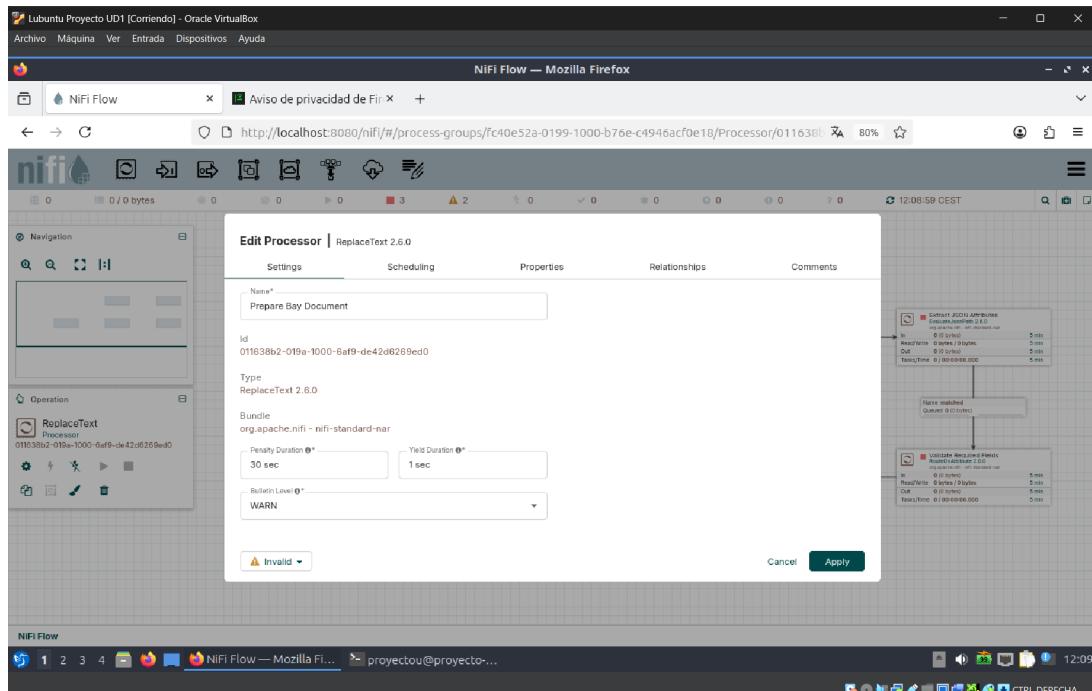


Figura D.72: Configuración (Settings) de 'ReplaceText': 'Name: Prepare Bay Document'.

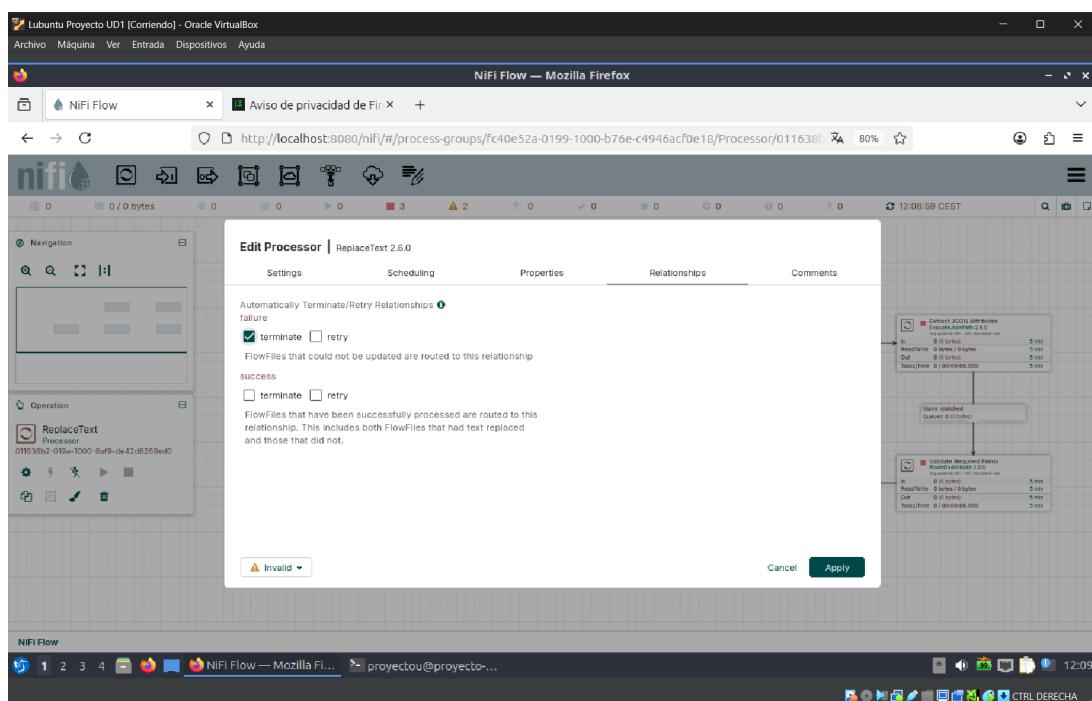


Figura D.73: Configuración (Relationships) de 'ReplaceText': Terminar 'failure'.

SmartParking Flow — Monitorización Inteligente

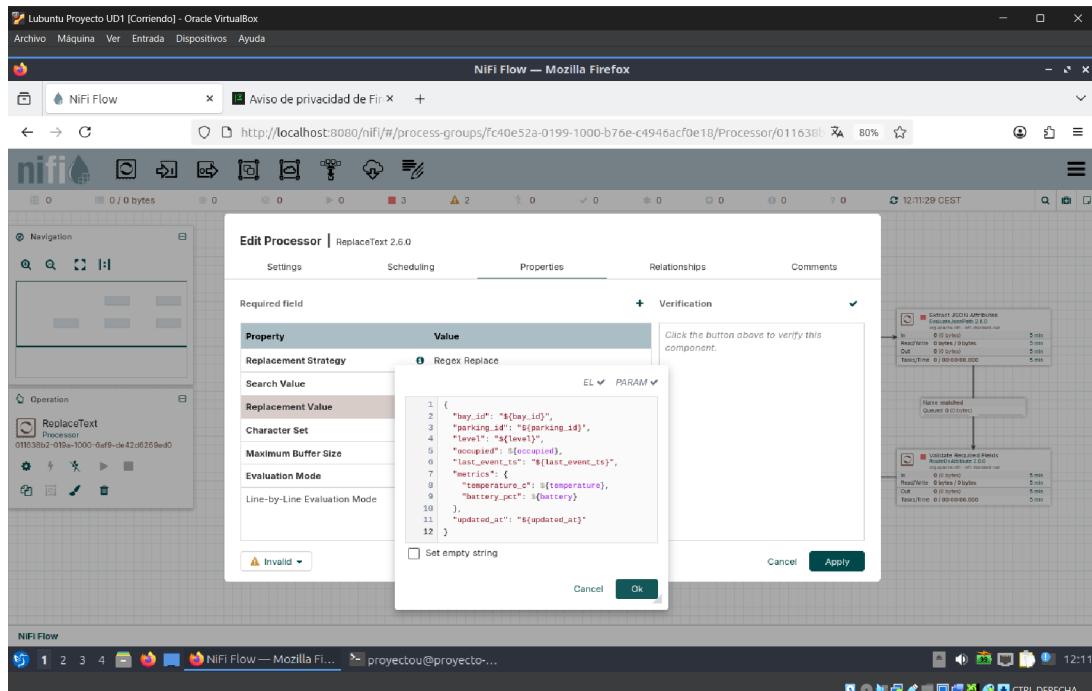


Figura D.74: Configuración (Properties) de ReplaceText: Replacement Value (JSON del documento bays).

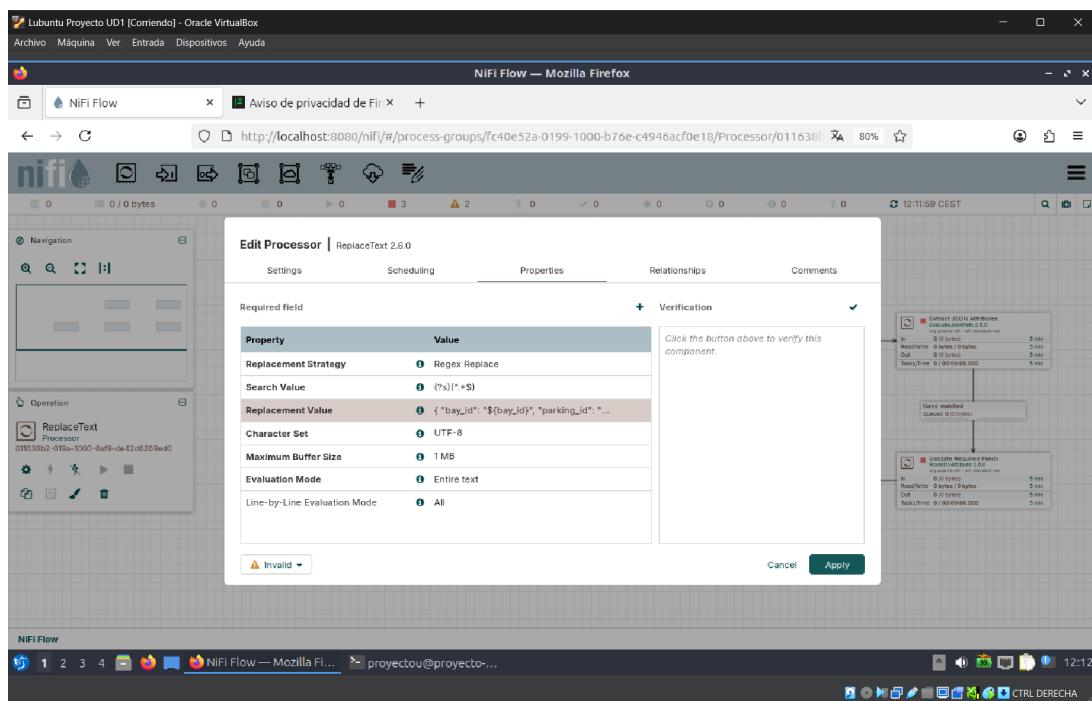


Figura D.75: Configuración (Properties) de ReplaceText: Search Value: (?s)(^.*\$).

SmartParking Flow — Monitorización Inteligente

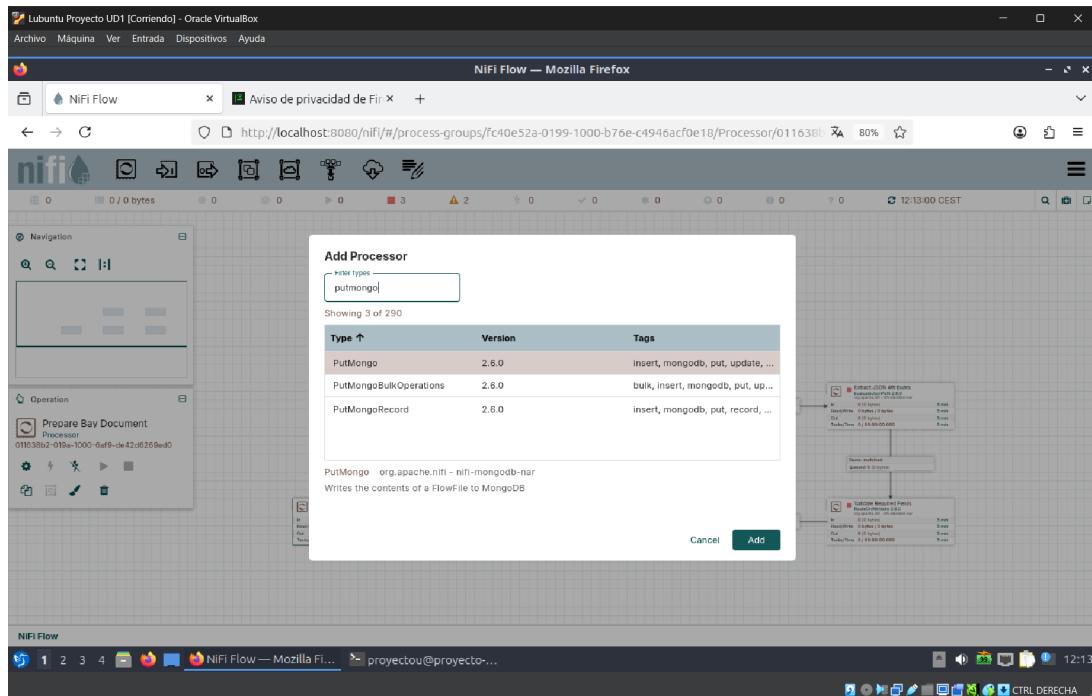


Figura D.76: Añadiendo el procesador 'PutMongo' (para el upsert).

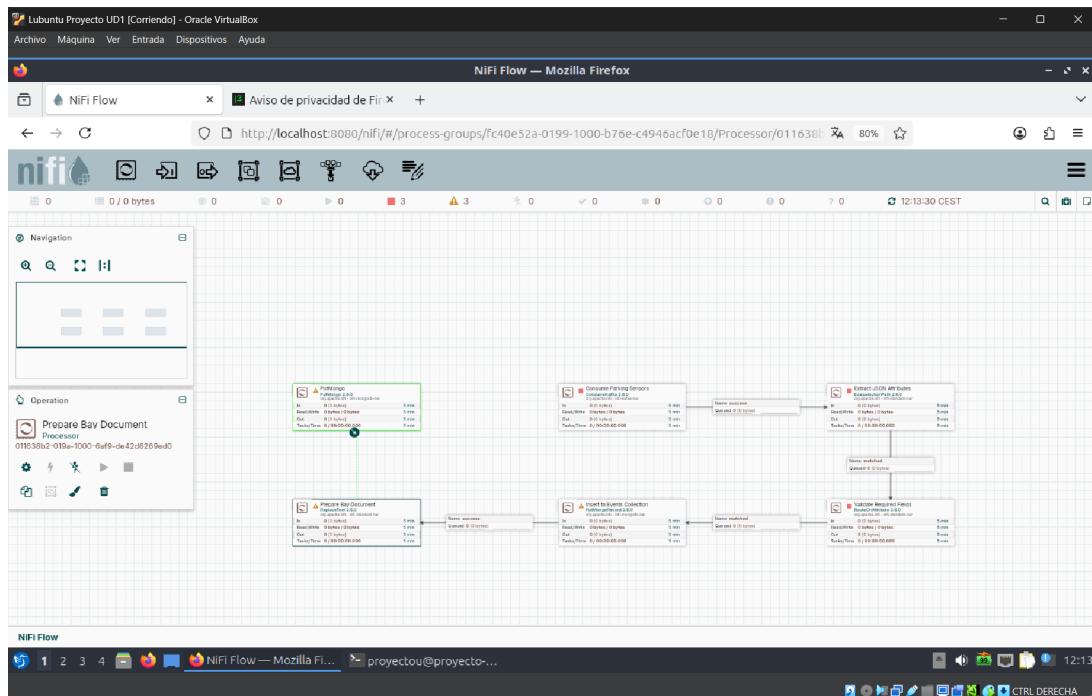


Figura D.77: Flujo con el procesador 'PutMongo' añadido.

SmartParking Flow — Monitorización Inteligente

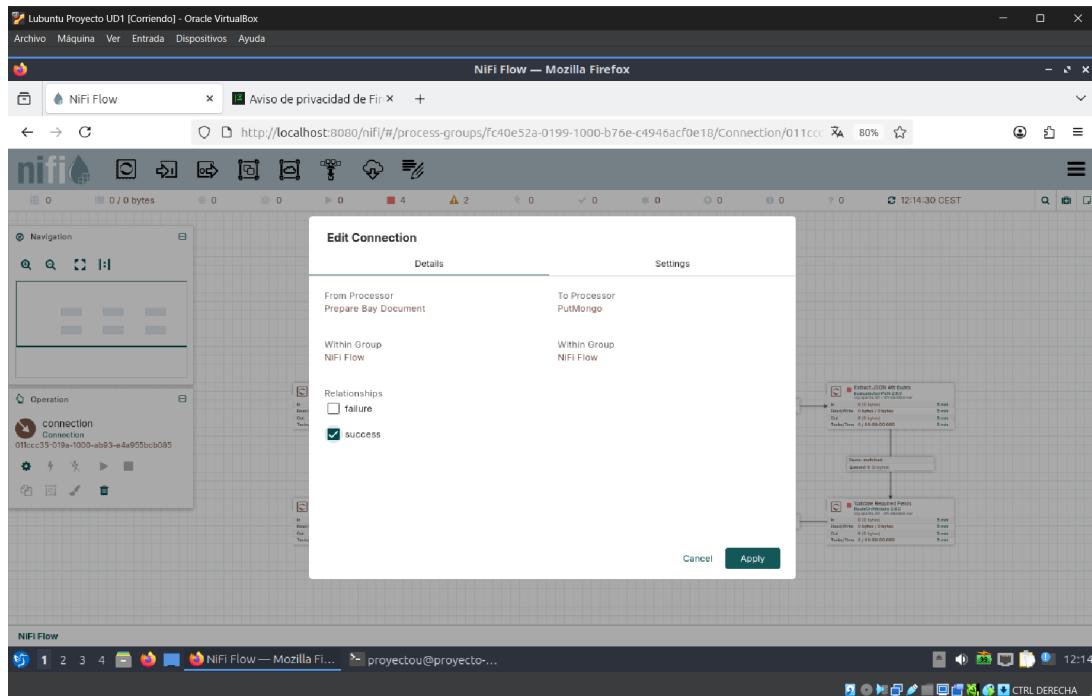


Figura D.78: Creando conexión 'success' entre 'ReplaceText' y 'PutMongo'.

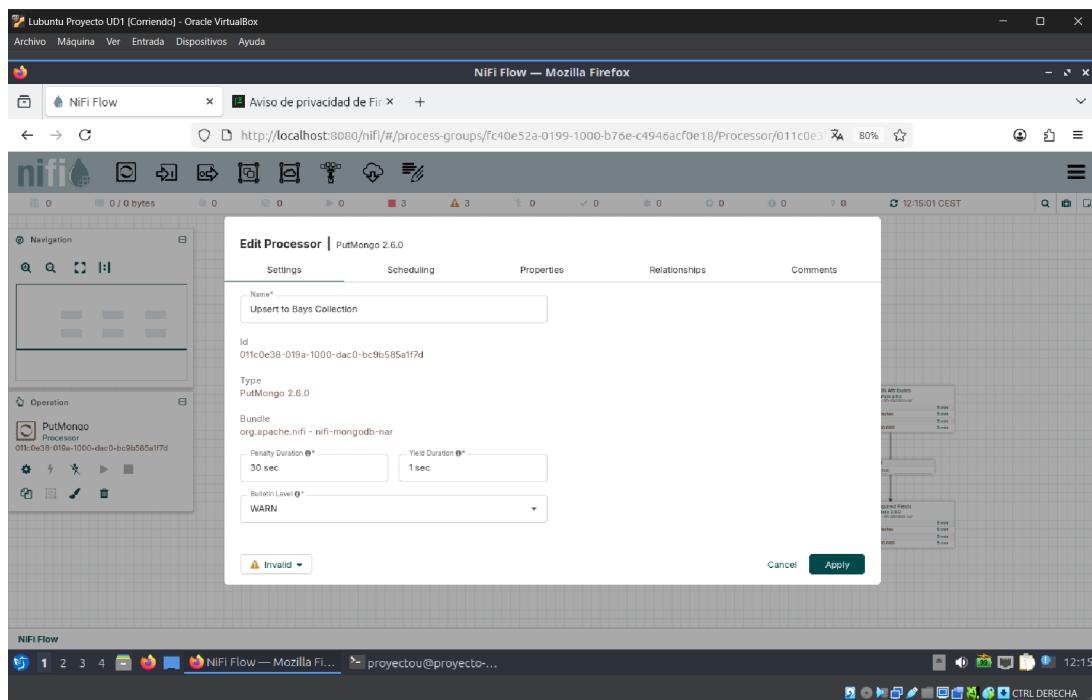


Figura D.79: Configuración (Settings) de 'PutMongo': 'Name: Upsert to Bays Collection'.

SmartParking Flow — Monitorización Inteligente

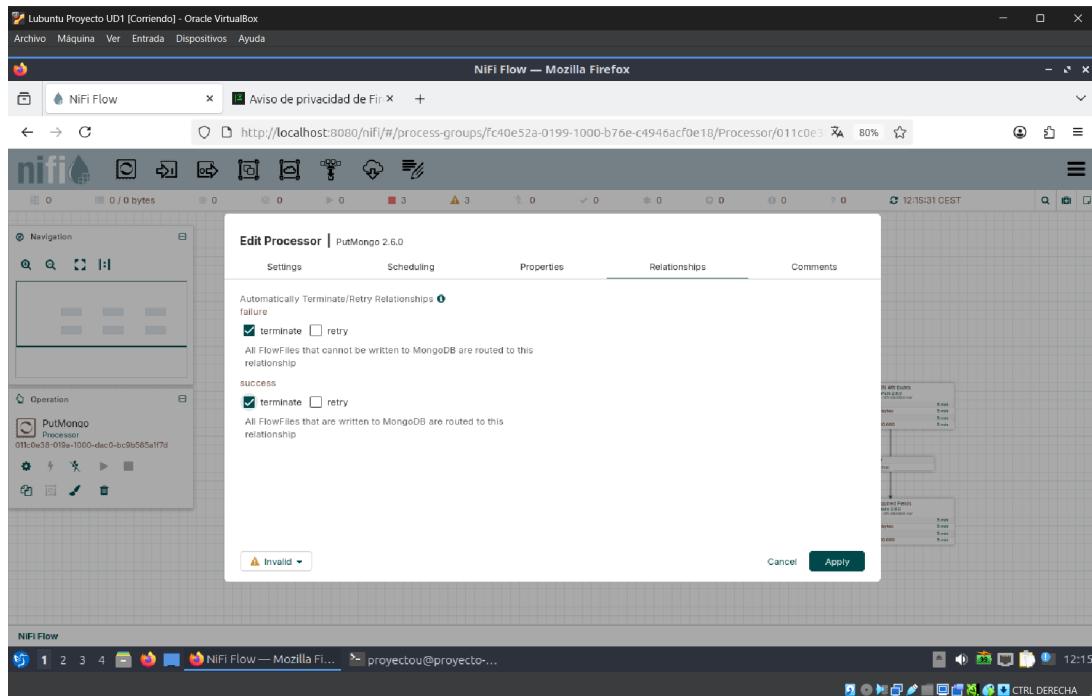


Figura D.80: Configuración (Relationships) de 'PutMongo': Terminar 'failure' y 'success'.

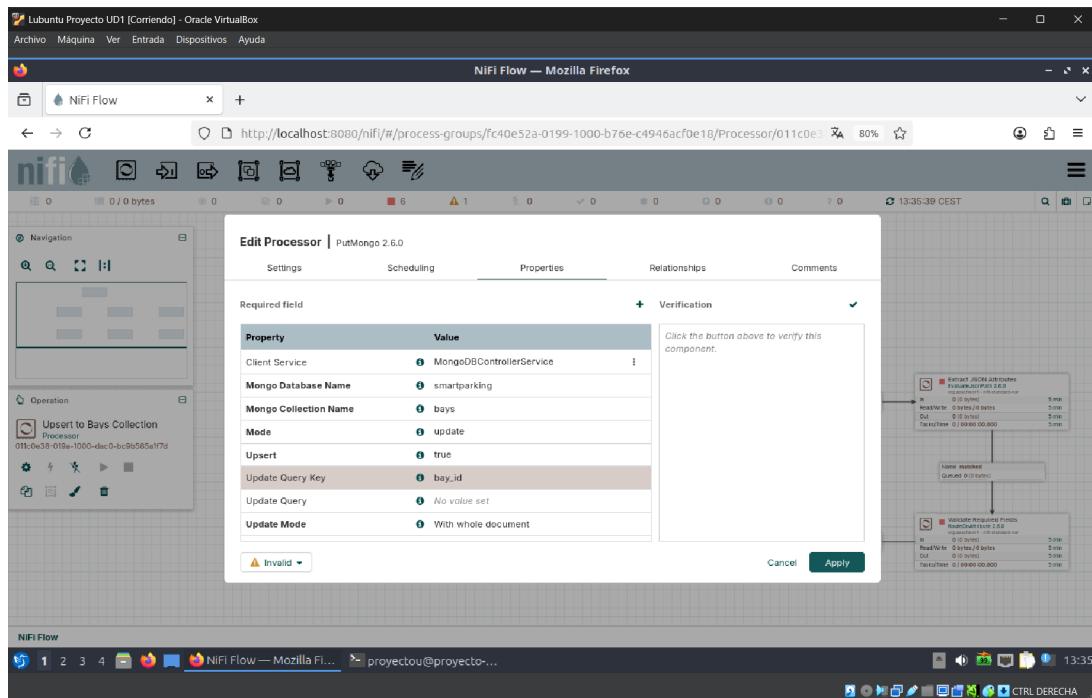


Figura D.81: Configuración (Properties) de 'PutMongo': 'Collection: bays', 'Mode: upsert', 'Key: bay_id'.

SmartParking Flow — Monitorización Inteligente

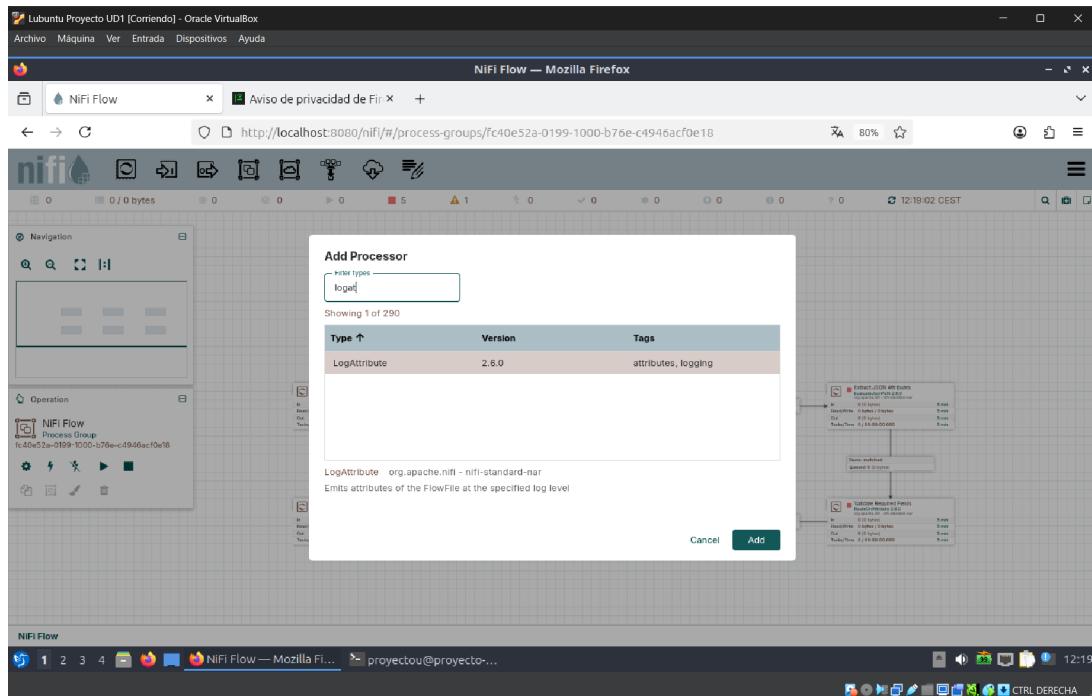


Figura D.82: Añadiendo el procesador 'LogAttribute' (para errores).

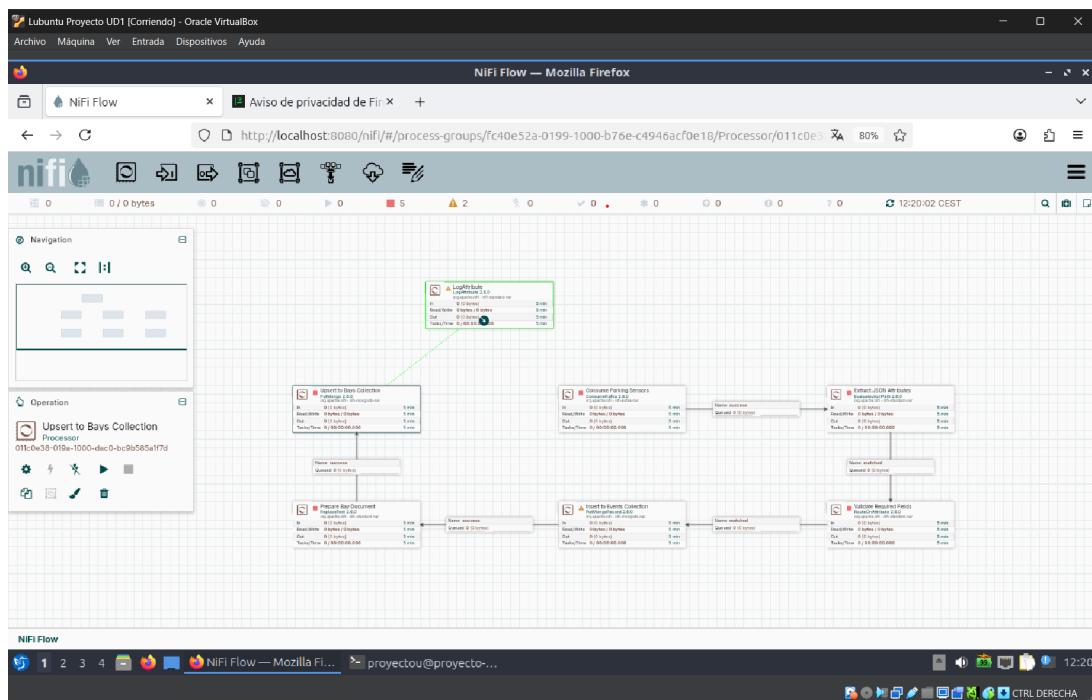


Figura D.83: Flujo con el procesador 'LogAttribute' añadido.

SmartParking Flow — Monitorización Inteligente

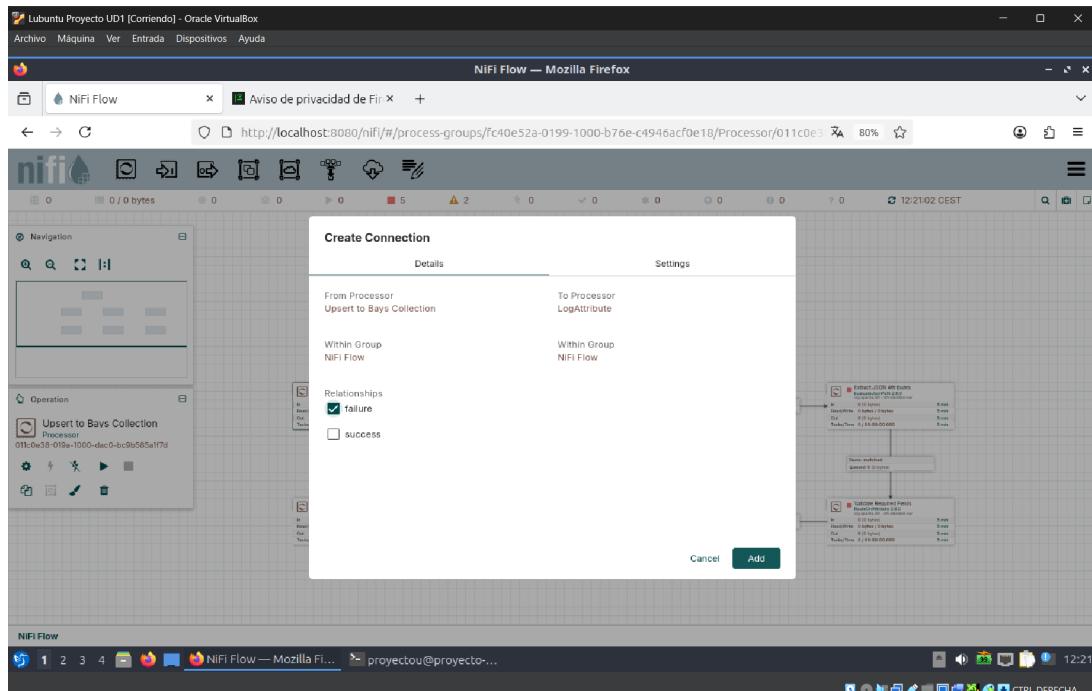


Figura D.84: Creando conexión 'failure' entre 'Upsert to Bays Collection' y 'Log Errors'.

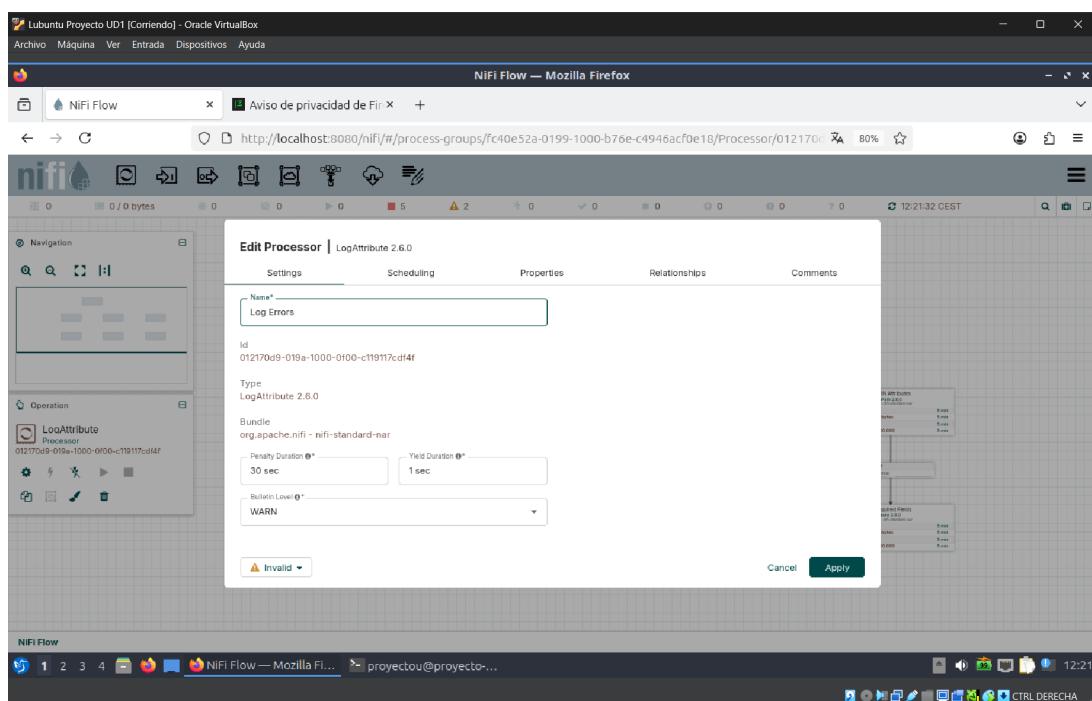


Figura D.85: Configuración (Settings) de 'LogAttribute': 'Name: Log Errors'.

SmartParking Flow — Monitorización Inteligente

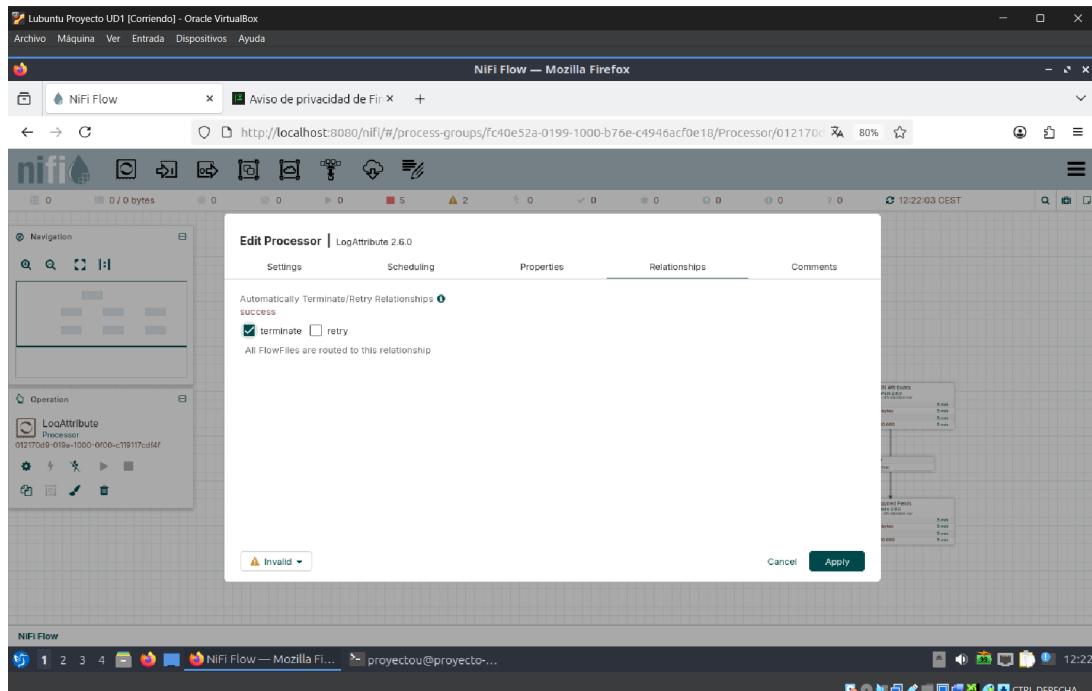


Figura D.86: Configuración (Relationships) de 'LogAttribute': Terminar 'success'.

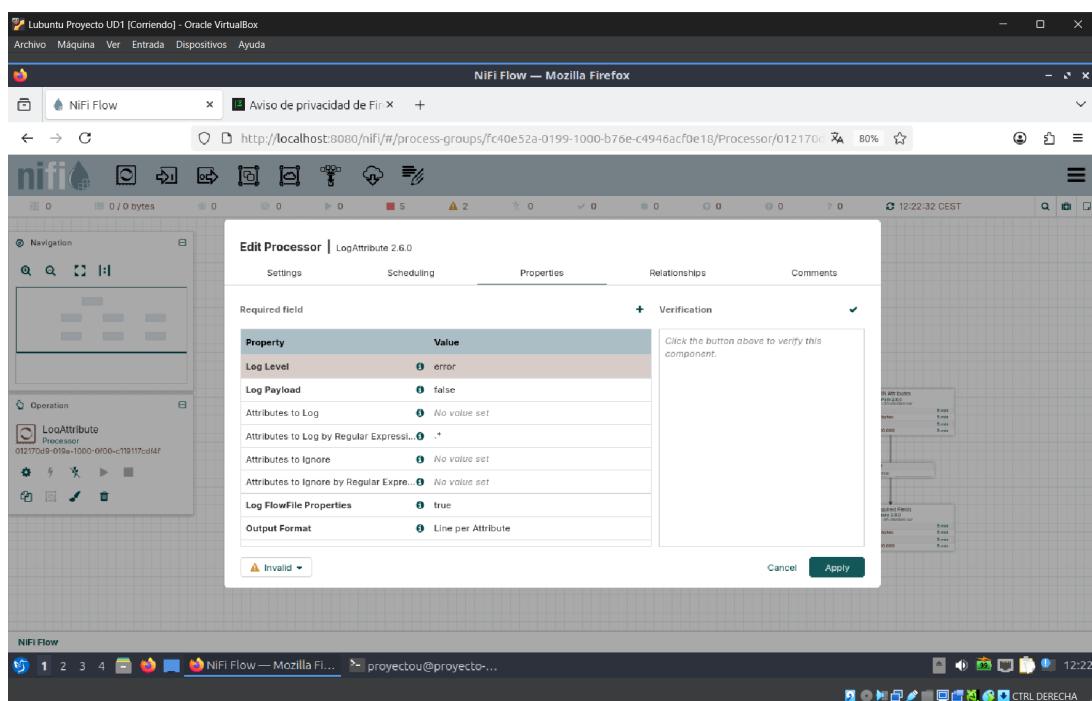


Figura D.87: Configuración (Properties) de 'LogAttribute': 'Log Level: error'.

SmartParking Flow — Monitorización Inteligente

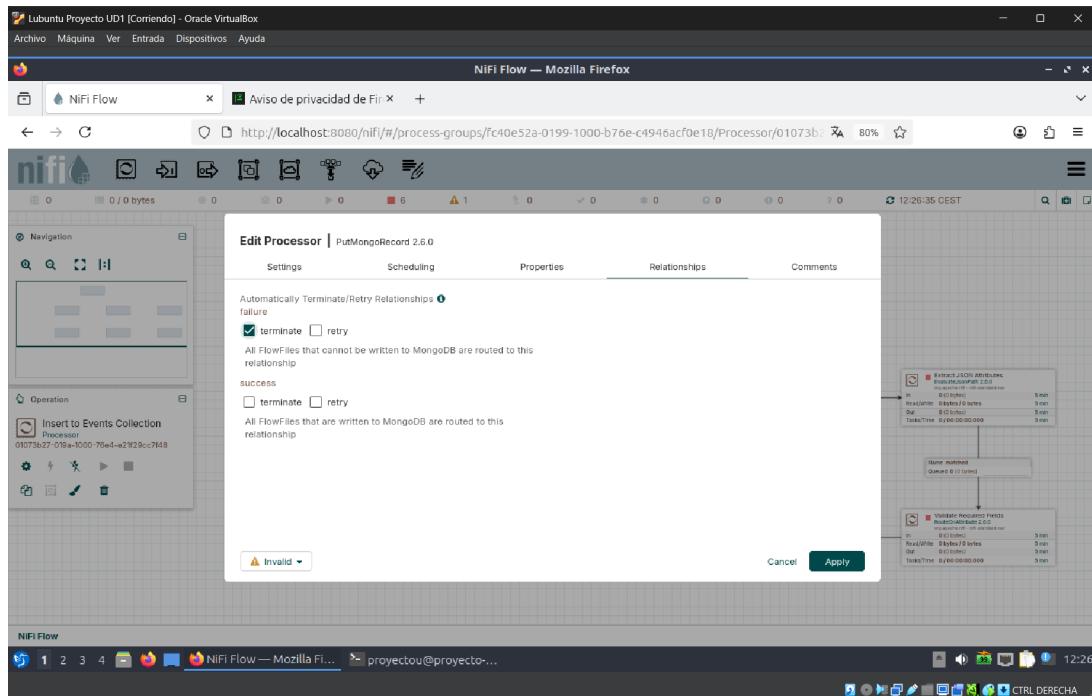


Figura D.88: Configuración (Relationships) de 'Insert to Events Collection': Terminar 'failure' y 'success'.

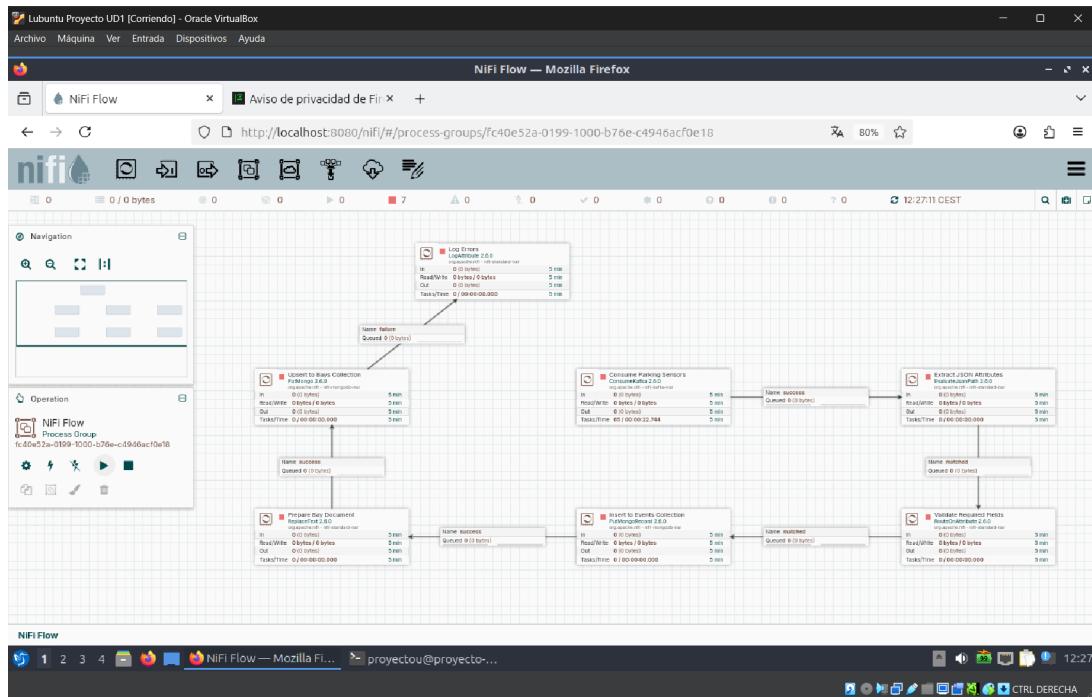


Figura D.89: Vista general del flujo de NiFi (incompleta).

SmartParking Flow — Monitorización Inteligente

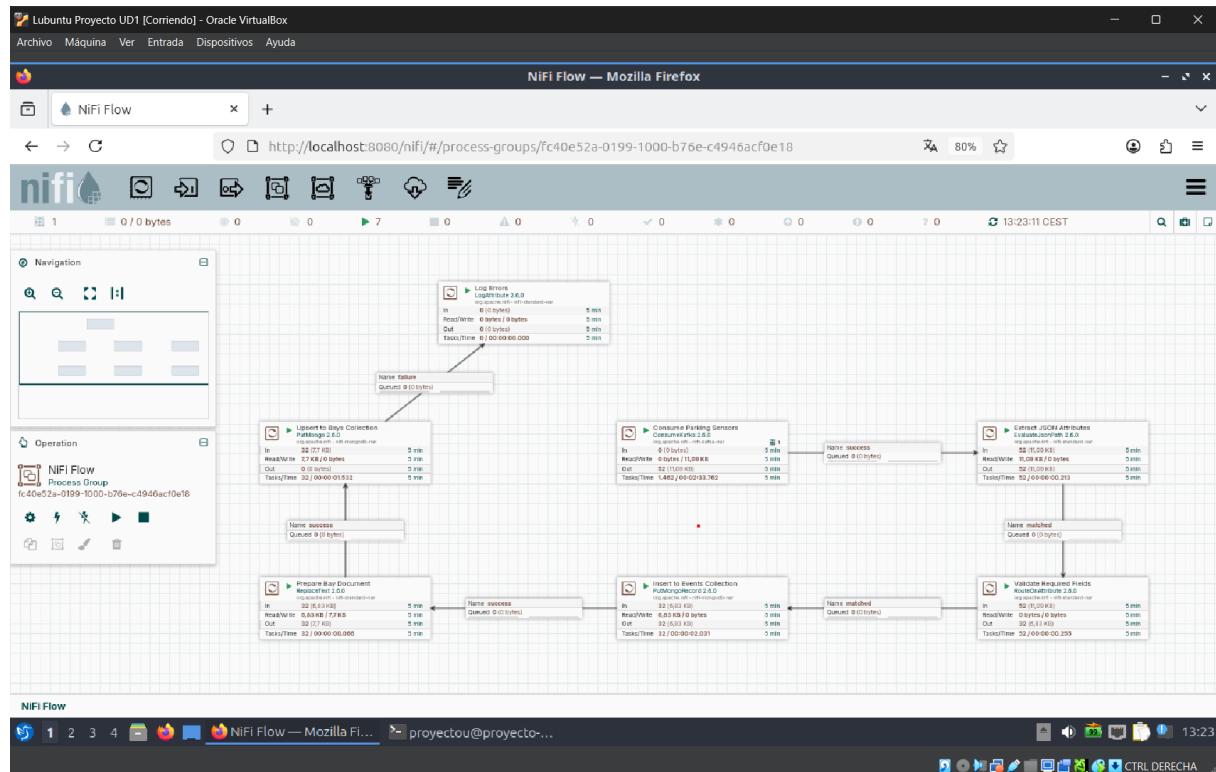


Figura D.90: Vista general del flujo de datos completo en Apache NiFi.

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 a database named 'SmartParkingCluster' with two collections: 'sample_mflix' and 'bays'. The 'bays' collection is selected. It displays a summary: STORAGE SIZE: 36KB, LOGICAL DATA SIZE: 4.4KB, TOTAL DOCUMENTS: 20, INDEXES TOTAL SIZE: 72KB. Below this is a search bar with placeholder 'Search namespaces' and a 'Find' button. To the right are tabs for Indexes, Schema Anti-Patterns, Aggregation, and Search Indexes. A query builder is present with a 'Filter' dropdown and a text input 'Type a query: { field: 'value' }'. At the bottom, there's a 'QUERY RESULTS: 1-20 OF 20' section showing 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 system status at the bottom is '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
34 MongoDB
35 MONGO_URI = os.getenv('MONGO_URI')
36 MONGO_DB = os.getenv('MONGO_DB', 'smartparking')
37
38 # Conectar a MongoDB Atlas
39 db = None
40
41 try:
42     client = MongoClient(
43         MONGO_URI,
44         serverSelectionTimeoutMS=5000,
45         connectTimeoutMS=10000,
46         socketTimeoutMS=10000
47     )
48     # Verificar conexion
49     client.admin.command('ping')
50     db = client[MONGO_DB]
51     logger.info(f'V Conectado exitosamente a MongoDB Atlas - Base de
      datos: {MONGO_DB}')
52 except ConnectionFailure as e:
53     logger.error(f'X Error de conexion a MongoDB Atlas: {e}')
54
55 db = None
56 except Exception as e:
57     logger.error(f'X Error inesperado conectando a MongoDB: {e}')
58 db = None
59
60
61 # ===== RUTAS DE LA APLICACION =====
62
63 @app.route('/')
64 def index():
65     ,
66     Pagina principal - Mapa visual del parking
67     ,
68     return render_template('index.html')
69
70
71 @app.route('/api/health')
72 def health_check():
73     ,
74     Endpoint de health check para verificar estado del servicio
75
76 Returns:
77     JSON con estado del servicio y conexion a BD
78     ,
79
80 if db is None:
81     return jsonify({
```

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

```
137         'metrics': 1,
138         'updated_at': 1
139     }
140 ).sort('bay_id', 1))

141 logger.info(f'API /api/bays: Retornadas {len(bays)} plazas')

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

```
191         },
192         'free': {
193             '$sum': {$cond: ['$occupied', 0, 1]}
194         },
195         'avg_temperature': {$avg: '$metrics.temperature_c'},
196     },
197     'avg_battery': {$avg: '$metrics.battery_pct'},
198     'low_battery_count': {
199
200         '$sum': {
201             '$cond': [
202                 {$lt: ['$metrics.battery_pct', 20]}, 1,
203                 0
204             ]
205         }
206     }
207 }
208
209 }
210
211 },
212 {'$sort': {'_id': 1}}
213 ]
214
215 by_level = list(db.bays.aggregate(pipeline))
216
217 # Formatear resultados por nivel
218 levels_stats =
219 []
220 for level in by_level:
221     levels_stats.append({
222         'level': level['_id'],
223         'total': level['total'],
224         'occupied': level['occupied'],
225         'free': level['free'],
226
227         'occupancy_rate': round((level['occupied'] / level['total']) * 100), 2) if level['total'] > 0 else 0,
228         'avg_temperature': round(level['avg_temperature'], 1) if level['avg_temperature'] else None,
229         'avg_battery': round(level['avg_battery'], 1) if level['avg_battery'] else None,
230         'low_battery_sensors': level['low_battery_count']
231     })
232
233
234     stats = {
235         'success': True,
236         'total': total,
237         'occupied': occupied,
238         'free': free,
239         'occupancy_rate': occupancy_rate,
240         'levels': levels_stats,
241
242 'timestamp': datetime.now().isoformat()
```

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

```
297         'level': level_upper,
298
299     'count': len(bays),
300         'data': bays
301     }, 200
302
303 except Exception as e:
304     logger.error(f'Error en /api/bays/level/{level}: {e}')
305     return jsonify({
306         'success': False,
307         'error': str(e)
308     }), 500
309
310
311 @app.route('/api/maintenance/low-battery')
312 def get_low_battery_bays():
313     ,
314
315     Obtener plazas con bateria baja que requieren mantenimiento
316
317     Returns:
318         JSON con plazas que tienen bateria < 20%
319     ,
320
321     if db is None:
322         return jsonify({'error': 'Database connection not available'}), 503
323
324     try:
325         low_battery_threshold = 20
326
327         low_battery_bays = list(db.bays.find(
328             {'metrics.battery_pct': {'$lt': low_battery_threshold}},
329             {'_id': 0}
330         ).sort('metrics.battery_pct', 1))
331
332         # Clasificar por prioridad
333         urgent = [b for b in low_battery_bays if b['metrics'][
334             'battery_pct'] < 10]
335         high = [b for b in low_battery_bays if 10 <= b['metrics'][
336             'battery_pct'] < 20]
337
338
339         logger.warning(f'Mantenimiento: {len(low_battery_bays)} sensores con
340             bateria baja')
341
342         return jsonify({
343             'success': True,
344             'total_count': len(low_battery_bays),
345             'urgent_count': len(urgent),
346             'high_priority_count': len(high),
347             'urgent': urgent,
348
349             'high_priority': high,
350             'threshold': low_battery_threshold
351         }), 200
```

```
349
350     except Exception as e:
351         logger.error(f'Error en /api/maintenance/low-battery: {e}')
352         return jsonify({
353             'success': False,
354             'error': str(e)
355         }), 500
356
357
358 # ====== MANEJADORES DE ERRORES ======
359 ======
360
361 @app.errorhandler(404)
362 def not_found(error):
363     'Manejador para errores 404'
364     return jsonify({
365         'success': False,
366         'error': 'Endpoint no encontrado',
367         'code': 404
368     }), 404
369
370
371 @app.errorhandler(500)
372 def internal_error(error):
373     'Manejador para errores 500'
374     logger.error(f'Error interno del servidor: {error}')
375     return jsonify({
376         'success': False,
377         'error': 'Error interno del servidor',
378         'code': 500
379
380     }), 500
381
382
383 @app.errorhandler(503)
384 def service_unavailable(error):
385     'Manejador para errores 503'
386     return jsonify({
387         'success': False,
388         'error': 'Servicio no disponible',
389         'code': 503
390     }), 503
391
392
393 # ====== PUNTO DE ENTRADA ======
394
395 if __name__ == '__main__':
396     logger.info('=' * 60)
397     logger.info('Iniciando SmartParking Flask Application')
398     logger.info('=' * 60)
399     logger.info(f'Base de datos: {MONGO_DB}')
400     logger.info(f'Servidor: http://localhost:5000')
401     logger.info('=' * 60)
402
403
404 # Ejecutar aplicacion
```

```
405     app.run(  
406         host='localhost',  
407         port=5000,  
408         debug=True  
409     )
```

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

E.2 Plantilla de la Interfaz (index.html)

A continuación, se muestra el código completo de la plantilla `index.html`, que define la estructura, estilos (CSS) y lógica de cliente (JavaScript) para la visualización en tiempo real.

```
1 <!DOCTYPE html>
2 <html lang='es'>
3   <head>
4     <meta charset='UTF-8' />
5     <meta name='viewport' content='width=device-width, initial-scale
6       =1.0' />
7     <title>SmartParking Cadiz - Monitorizacion en Tiempo Real</title>
8     <style>
9       /* ====== VARIABLES Y RESET ====== */
10      * {
11        margin: 0;
12        padding: 0;
13        box-sizing: border-box;
14      }
15
16      :root {
17        --primary-color: #2c3e50;
18        --success-color: #27ae60;
19        --danger-color: #e74c3c;
20        --warning-color: #f39c12;
21        --info-color: #3498db;
22        --bg-gradient-start: #667eea;
23        --bg-gradient-end: #764ba2;
24        --card-bg: #ffffff;
25        --text-color: #2c3e50;
26        --text-muted: #7f8c8d;
27        --border-radius: 12px;
28        --shadow-sm: 0 2px 4px rgba(0, 0, 0, 0.1);
29        --shadow-md: 0 4px 8px rgba(0, 0, 0, 0.15);
30        --shadow-lg: 0 8px 16px rgba(0, 0, 0, 0.2);
31        --transition: all 0.3s cubic-bezier(0.4, 0, 0.2, 1);
32      }
33
34      body {
35        font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;
36        background: linear-gradient(
37          135deg,
38          var(--bg-gradient-start) 0%,
39          var(--bg-gradient-end) 100%
40        );
41        min-height: 100vh;
42        padding: 20px;
43        color: var(--text-color);
44      }
45
46      /* ====== CONTENEDOR PRINCIPAL ====== */
47    </>
48    .container {
49      max-width: 1600px;
```

```
48     margin: 0 auto;
49 }
50
51 /* ===== HEADER ===== */
52 .header {
53   background: var(--card-bg);
54   padding: 30px;
55   border-radius: var(--border-radius);
56   margin-bottom: 25px;
57   box-shadow: var(--shadow-md);
58   text-align: center;
59 }
60
61 .header h1 {
62   font-size: 2.5rem;
63   color: var(--primary-color);
64   margin-bottom: 8px;
65   font-weight: 700;
66 }
67
68 .subtitle {
69   color: var(--text-muted);
70   font-size: 1.1rem;
71   font-weight: 500;
72 }
73
74 /* ===== PANEL DE ESTADISTICAS ===== */
75 */
76 .stats-panel {
77   display: grid;
78   grid-template-columns: repeat(auto-fit, minmax(220px, 1fr));
79   gap: 20px;
80   margin-bottom: 25px;
81 }
82
83 .stat-card {
84   background: var(--card-bg);
85   padding: 25px;
86   border-radius: var(--border-radius);
87   text-align: center;
88   box-shadow: var(--shadow-md);
89   border-left: 5px solid var(--primary-color);
90   transition: var(--transition);
91   position: relative;
92   overflow: hidden;
93 }
94
95 .stat-card::before {
96   content: '';
97   position: absolute;
98   top: 0;
99   left: 0;
100  width: 100%;
101  height: 100%;
102  background: linear-gradient(
    135deg,
```

```
103         rgba(255, 255, 255, 0) 0%,
104         rgba(255, 255, 255, 0.1) 100%
105     );
106     pointer-events: none;
107 }
108
109 .stat-card:hover {
110     transform: translateY(-5px);
111     box-shadow: var(--shadow-lg);
112 }
113
114 .stat-card.success {
115     border-left-color: var(--success-color);
116 }
117
118 .stat-card.danger {
119     border-left-color: var(--danger-color);
120 }
121
122 .stat-card.info {
123     border-left-color: var(--info-color);
124 }
125
126 .stat-number {
127     font-size: 3rem;
128     font-weight: 700;
129     color: var(--primary-color);
130     margin-bottom: 8px;
131     line-height: 1;
132 }
133
134 .stat-label {
135     color: var(--text-muted);
136     font-size: 0.95rem;
137     text-transform: uppercase;
138     letter-spacing: 1.5px;
139     font-weight: 600;
140 }
141
142 /* ===== FILTROS ===== */
143 .filters {
144     display: flex;
145     gap: 12px;
146     margin-bottom: 25px;
147     flex-wrap: wrap;
148     justify-content: center;
149 }
150
151 .filter-btn {
152     padding: 14px 28px;
153     border: 2px solid var(--card-bg);
154     background: var(--card-bg);
155     color: var(--primary-color);
156     border-radius: var(--border-radius);
157     cursor: pointer;
158     font-size: 1rem;
```

```
159     font-weight: 600;
160     transition: var(--transition);
161     box-shadow: var(--shadow-sm);
162   }
163
164   .filter-btn:hover {
165     background: var(--primary-color);
166     color: white;
167     transform: translateY(-2px);
168     box-shadow: var(--shadow-md);
169   }
170
171   .filter-btn.active {
172     background: var(--primary-color);
173     color: white;
174     box-shadow: var(--shadow-md);
175   }
176
177   /* ===== CONTENEDOR DE PARKING ===== */
178
179   #parking-container {
180     display: flex;
181     flex-direction: column;
182     gap: 25px;
183   }
184
185   .level-section {
186     background: var(--card-bg);
187     padding: 30px;
188     border-radius: var(--border-radius);
189     box-shadow: var(--shadow-md);
190     animation: fadeIn 0.5s ease-in-out;
191   }
192
193   @keyframes fadeIn {
194     from {
195       opacity: 0;
196       transform: translateY(20px);
197     }
198     to {
199       opacity: 1;
200       transform: translateY(0);
201     }
202   }
203
204   .level-header {
205     display: flex;
206     justify-content: space-between;
207     align-items: center;
208     margin-bottom: 25px;
209     padding-bottom: 15px;
210     border-bottom: 3px solid #ecf0f1;
211   }
212
213   .level-title {
214     font-size: 1.8rem;
```

```
214     color: var(--primary-color);
215     font-weight: 700;
216     display: flex;
217     align-items: center;
218     gap: 12px;
219   }
220
221   .level-title::before {
222     content: 'P';
223     font-size: 2rem;
224   }
225
226   .level-stats {
227     display: flex;
228     gap: 25px;
229     font-size: 1rem;
230     font-weight: 600;
231   }
232
233   .level-stat {
234     display: flex;
235     align-items: center;
236     gap: 8px;
237     padding: 8px 16px;
238     border-radius: 8px;
239     background: #f8f9fa;
240   }
241
242   .level-stat.free {
243     color: var(--success-color);
244   }
245
246   .level-stat.occupied {
247     color: var(--danger-color);
248   }
249
250   /* ===== GRID DE PLAZAS ===== */
251   .parking-grid {
252     display: grid;
253     grid-template-columns: repeat(auto-fill, minmax(90px, 1fr));
254     gap: 15px;
255   }
256
257   .bay {
258     aspect-ratio: 1;
259     border-radius: var(--border-radius);
260     display: flex;
261     flex-direction: column;
262     align-items: center;
263     justify-content: center;
264     cursor: pointer;
265     transition: var(--transition);
266     position: relative;
267     box-shadow: var(--shadow-sm);
268     font-weight: 600;
269   }
```

```
270
271     .bay:hover {
272         transform: scale(1.08);
273         box-shadow: var(--shadow-lg);
274         z-index: 10;
275     }
276
277     .bay.free {
278         background: linear-gradient(135deg, #27ae60 0%, #2ecc71 100%);
279         color: white;
280     }
281
282     .bay.occupied {
283         background: linear-gradient(135deg, #e74c3c 0%, #c0392b 100%);
284         color: white;
285     }
286
287     .bay.free:hover {
288         background: linear-gradient(135deg, #2ecc71 0%, #27ae60 100%);
289     }
290
291     .bay.occupied:hover {
292         background: linear-gradient(135deg, #c0392b 0%, #e74c3c 100%);
293     }
294
295     .bay-id {
296         font-size: 0.8rem;
297         font-weight: 700;
298         margin-bottom: 6px;
299         text-shadow: 0 1px 2px rgba(0, 0, 0, 0.2);
300     }
301
302     .bay-status-icon {
303         font-size: 1.5rem;
304     }
305
306     .bay-status-text {
307         font-size: 0.7rem;
308         opacity: 0.95;
309         margin-top: 4px;
310         text-transform: uppercase;
311         letter-spacing: 0.5px;
312     }
313
314     /* ===== TOOLTIP ===== */
315     .bay-tooltip {
316         position: absolute;
317         bottom: 110%;
318         left: 50%;
319         transform: translateX(-50%) scale(0.95);
320         background: rgba(0, 0, 0, 0.95);
321         color: white;
322         padding: 15px;
323         border-radius: 10px;
324         font-size: 0.85rem;
325         white-space: nowrap;
```

```
326     opacity: 0;
327     pointer-events: none;
328     transition: all 0.3s ease;
329     z-index: 1000;
330     box-shadow: 0 8px 16px rgba(0, 0, 0, 0.3);
331     backdrop-filter: blur(10px);
332   }
333
334   .bay:hover .bay-tooltip {
335     opacity: 1;
336     transform: translateX(-50%) scale(1);
337   }
338
339   .bay-tooltip::after {
340     content: '';
341     position: absolute;
342     top: 100%;
343     left: 50%;
344     transform: translateX(-50%);
345     border: 8px solid transparent;
346     border-top-color: rgba(0, 0, 0, 0.95);
347   }
348
349   .tooltip-line {
350     margin: 5px 0;
351     display: flex;
352     align-items: center;
353     gap: 8px;
354   }
355
356   .tooltip-line strong {
357     color: #3498db;
358   }
359
360   /* ===== INDICADORES ===== */
361   .update-indicator {
362     background: var(--card-bg);
363     padding: 18px 25px;
364     border-radius: var(--border-radius);
365     margin-top: 25px;
366     display: flex;
367     justify-content: space-between;
368     align-items: center;
369     font-size: 0.95rem;
370     color: var(--text-muted);
371     box-shadow: var(--shadow-md);
372   }
373
374   #connection-status {
375     font-weight: 700;
376     display: flex;
377     align-items: center;
378     gap: 8px;
379   }
380
381   .status-dot {
```

```
382     width: 12px;
383     height: 12px;
384     border-radius: 50%;
385     animation: pulse 2s ease-in-out infinite;
386   }
387
388   @keyframes pulse {
389     0%,
390     100% {
391       opacity: 1;
392     }
393     50% {
394       opacity: 0.5;
395     }
396   }
397
398   .status-connected .status-dot {
399     background: var(--success-color);
400   }
401
402   .status-disconnected .status-dot {
403     background: var(--danger-color);
404   }
405
406   .status-connected {
407     color: var(--success-color);
408   }
409
410   .status-disconnected {
411     color: var(--danger-color);
412   }
413
414   /* ====== LOADING ===== */
415   .loading-overlay {
416     position: fixed;
417     top: 0;
418     left: 0;
419     width: 100%;
420     height: 100%;
421     background: rgba(0, 0, 0, 0.7);
422     display: flex;
423     align-items: center;
424     justify-content: center;
425     z-index: 9999;
426     backdrop-filter: blur(5px);
427   }
428
429   .loading-spinner {
430     width: 60px;
431     height: 60px;
432     border: 5px solid rgba(255, 255, 255, 0.3);
433     border-top-color: white;
434     border-radius: 50%;
435     animation: spin 1s linear infinite;
436   }
437
```

```
438     @keyframes spin {
439         to {
440             transform: rotate(360deg);
441         }
442     }
443
444     .hidden {
445         display: none !important;
446     }
447
448     /* ===== RESPONSIVE ===== */
449     @media (max-width: 768px) {
450         .header h1 {
451             font-size: 1.8rem;
452         }
453
454         .stats-panel {
455             grid-template-columns: repeat(2, 1fr);
456         }
457
458         .parking-grid {
459             grid-template-columns: repeat(auto-fill, minmax(70px, 1fr));
460             gap: 10px;
461         }
462
463         .level-header {
464             flex-direction: column;
465             align-items: flex-start;
466             gap: 15px;
467         }
468
469         .level-stats {
470             flex-direction: column;
471             gap: 10px;
472             width: 100%;
473         }
474
475         .bay-id {
476             font-size: 0.7rem;
477         }
478
479         .bay-status-icon {
480             font-size: 1.2rem;
481         }
482     }
483     </style>
484 </head>
485 <body>
486     <div id='loading-overlay' class='loading-overlay hidden'>
487         <div class='loading-spinner'></div>
488     </div>
489
490     <div class='container'>
491         <header class='header'>
492             <h1>SmartParking Cadiz</h1>
493             <p class='subtitle'>Monitorizacion en Tiempo Real - PK-CADIZ-01<
```

```
494     /p>
495   </header>
496
497   <div class='stats-panel'>
498     <div class='stat-card'>
499       <div class='stat-number' id='total-bays'>-</div>
500       <div class='stat-label'>Total Plazas</div>
501     </div>
502     <div class='stat-card success'>
503       <div class='stat-number' id='free-bays'>-</div>
504       <div class='stat-label'>Libres</div>
505     </div>
506     <div class='stat-card danger'>
507       <div class='stat-number' id='occupied-bays'>-</div>
508       <div class='stat-label'>Ocupadas</div>
509     </div>
510     <div class='stat-card info'>
511       <div class='stat-number' id='occupancy-rate'>- %</div>
512       <div class='stat-label'>Ocupacion</div>
513     </div>
514   </div>
515
516   <div class='filters'>
517     <button class='filter-btn active' data-level='all'>
518       Todos los niveles
519     </button>
520     <button class='filter-btn' data-level='L1'>Nivel L1</button>
521     <button class='filter-btn' data-level='L2'>Nivel L2</button>
522   </div>
523
524   <div id='parking-container'></div>
525
526   <div class='update-indicator'>
527     <span id='connection-status' class='status-connected'>
528       <span class='status-dot'></span>
529       Conectado
530     </span>
531     <span id='last-update'>Ultima actualizacion: -</span>
532   </div>
533 </div>
534
535 <script>
536   // ===== CONFIGURACION =====
537   const API_BASE_URL = window.location.origin;
538   const UPDATE_INTERVAL = 20000; // 20 segundos
539   let updateTimer = null;
540   let currentFilter = 'all';
541
542   // ===== INICIALIZACION =====
543   document.addEventListener('DOMContentLoaded', function () {
544     console.log('SmartParking UI inicializado');
545     setupFilters();
546     loadParkingData();
547     startAutoUpdate();
548   });
549
```

```
549 // ===== CONFIGURAR FILTROS =====
550 function setupFilters() {
551     const filterButtons = document.querySelectorAll('.filter-btn');
552
553     filterButtons.forEach((btn) => {
554         btn.addEventListener('click', function () {
555             // Remover clase active de todos
556             filterButtons.forEach((b) => b.classList.remove('active'));
557             // Anadir a boton clickeado
558             this.classList.add('active');
559
560             // Aplicar filtro
561             currentFilter = this.getAttribute('data-level');
562             filterParkingLevels(currentFilter);
563         });
564     });
565 }
566
567 // ===== CARGAR DATOS DEL PARKING
568 =====
569 async function loadParkingData() {
570     try {
571         showLoading(true);
572
573         // Obtener estadisticas y plazas en paralelo
574         const [statsResponse, baysResponse] = await Promise.all([
575             fetch(`${API_BASE_URL}/api/stats`),
576             fetch(`${API_BASE_URL}/api/bays`),
577         ]);
578
579         if (!statsResponse.ok || !baysResponse.ok) {
580             throw new Error('Error al cargar datos del servidor');
581         }
582
583         const statsData = await statsResponse.json();
584         const baysData = await baysResponse.json();
585
586         // Actualizar UI
587         updateStatsPanel(statsData);
588         renderParkingGrid(baysData.data || baysData);
589         updateLastUpdateTime();
590         updateConnectionStatus(true);
591
592         console.log(
593             `Datos cargados: ${
594                 baysData.data ? baysData.data.length : baysData.length
595             } plazas`;
596         );
597     } catch (error) {
598         console.error('Error cargando datos:', error);
599         updateConnectionStatus(false);
600         showError('Error al conectar con el servidor. Reintentando
...');
601     } finally {
602         showLoading(false);
603     }
604 }
```

```
603     }
604
605     // ===== ACTUALIZAR PANEL DE ESTADISTICAS
606     =====
607     function updateStatsPanel(stats) {
608         document.getElementById('total-bays').textContent = stats.total || 0;
609         document.getElementById('free-bays').textContent = stats.free || 0;
610         document.getElementById('occupied-bays').textContent =
611             stats.occupied || 0;
612         document.getElementById('occupancy-rate').textContent = `${stats.occupancy_rate || 0}%`;
613     }
614
615     // ===== RENDERIZAR GRID DEL PARKING
616     =====
617     function renderParkingGrid(bays) {
618         const container = document.getElementById('parking-container');
619
620         if (!bays || bays.length === 0) {
621             container.innerHTML =
622                 '<div style="text-align:center;padding:40px;color:white;">No
623 hay datos de plazas disponibles</div>';
624             return;
625         }
626
627         // Agrupar plazas por nivel
628         const baysByLevel = {};
629         bays.forEach((bay) => {
630             if (!baysByLevel[bay.level]) {
631                 baysByLevel[bay.level] = [];
632             }
633             baysByLevel[bay.level].push(bay);
634         });
635
636         // Ordenar niveles (L1, L2, L3...)
637         const sortedLevels = Object.keys(baysByLevel).sort();
638
639         // Generar HTML
640         container.innerHTML = '';
641         sortedLevels.forEach((level) => {
642             const levelBays = baysByLevel[level];
643             const occupied = levelBays.filter((b) => b.occupied).length;
644             const free = levelBays.length - occupied;
645
646             const levelSection = document.createElement('div');
647             levelSection.className = 'level-section';
648             levelSection.setAttribute('data-level', level);
649
650             levelSection.innerHTML =
651                 '<div class="level-header">
652                     <h2 class="level-title">Nivel ${level}</h2>
653                     <div class="level-stats">
654                         <span class="level-stat free">
```

```
654                     ${free} libres
655             </span>
656             <span class='level-stat occupied'>
657                 ${occupied} ocupadas
658             </span>
659         </div>
660     </div>
661     <div class='parking-grid'>
662         ${levelBays.map((bay) => createBayHTML(bay)).join('')}
663     </div>
664     ;
665
666     container.appendChild(levelSection);
667 }
668
669 // Aplicar filtro actual
670 filterParkingLevels(currentFilter);
671
672 // ====== CREAR HTML DE UNA PLAZA
673 ======
674 function createBayHTML(bay) {
675     const statusClass = bay.occupied ? 'occupied' : 'free';
676     const statusText = bay.occupied ? 'OCUPADA' : 'LIBRE';
677
678     return `
679         <div class='bay ${statusClass}', data-bay-id='${bay.
680 bay_id}'>
681             <div class='bay-id'>${bay.bay_id}</div>
682             <div class='bay-status-text'>${statusText}</div>
683             <div class='bay-tooltip'>
684                 <div class='tooltip-line'><strong>${
685                     bay.bay_id
686                 }</strong></div>
687                 <div class='tooltip-line'>Estado: ${statusText}<
688 /div>
689                 <div class='tooltip-line'>Parking: ${
690                     bay.parking_id
691                 }</div>
692                 <div class='tooltip-line'>Nivel: ${bay.level}</
693 div>
694             ${
695                 bay.metrics
696                 ?
697                     <div class='tooltip-line'>Temperatura: ${bay.
698 metrics.temperature_c}C</div>
699                     <div class='tooltip-line'>Bateria: ${bay.
700 metrics.battery_pct}%</div>
701                 '
702                     :
703                 }
704                 <div class='tooltip-line'>${formatTimestamp(
705                     bay.updated_at
706                 )}</div>
707             </div>
708         
```

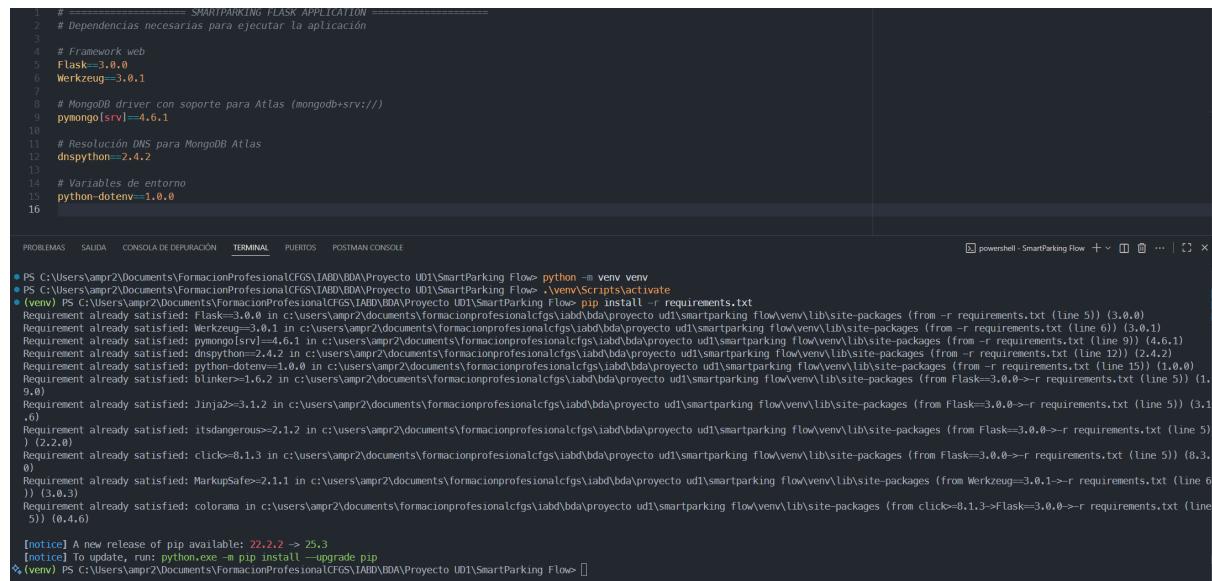
```
703             </div>
704         ‘;
705     }
706
707     // ====== FILTRAR NIVELES VISIBLES ======
708     ======
709     function filterParkingLevels(filter) {
710         const levelSections = document.querySelectorAll('.level-section
711 ');
712
713         levelSections.forEach((section) => {
714             if (filter === 'all') {
715                 section.style.display = 'block';
716             } else {
717                 const level = section.getAttribute('data-level');
718                 section.style.display = level === filter ? 'block' : 'none';
719             }
720         });
721
722     // ====== FORMATEAR TIMESTAMP ======
723     function formatTimestamp(timestamp) {
724         if (!timestamp) return 'N/A';
725
726         try {
727             const date = new Date(timestamp);
728             return date.toLocaleString('es-ES', {
729                 day: '2-digit',
730                 month: '2-digit',
731                 hour: '2-digit',
732                 minute: '2-digit',
733                 second: '2-digit',
734             });
735         } catch (e) {
736             return timestamp;
737         }
738     }
739
740     // ====== ACTUALIZAR TIEMPO DE ULTIMA ACTUALIZACION ======
741     ======
742     function updateLastUpdateTime() {
743         const now = new Date();
744         const timeString = now.toLocaleString('es-ES', {
745             day: '2-digit',
746             month: '2-digit',
747             hour: '2-digit',
748             minute: '2-digit',
749             second: '2-digit',
750         });
751         document.getElementById(
752             'last-update'
753         ).textContent = 'Ultima actualizacion: ${timeString}';
754     }
755
756     // ====== ACTUALIZAR ESTADO DE CONEXION ======
757     ======
```

```
755     function updateConnectionStatus(connected) {
756         const statusElement = document.getElementById('connection-status');
757     );
758
759     if (connected) {
760         statusElement.innerHTML =
761             '<span class='status-dot'></span> Conectado';
762         statusElement.className = 'status-connected';
763     } else {
764         statusElement.innerHTML =
765             '<span class='status-dot'></span> Desconectado';
766         statusElement.className = 'status-disconnected';
767     }
768
769 // ===== MOSTRAR/OCULTAR LOADING =====
770
771     function showLoading(show) {
772         const overlay = document.getElementById('loading-overlay');
773         if (show) {
774             overlay.classList.remove('hidden');
775         } else {
776             overlay.classList.add('hidden');
777         }
778
779 // ===== MOSTRAR ERROR =====
780     function showError(message) {
781         console.error(message);
782         // Podrias anadir un toast o notificacion aqui
783     }
784
785 // ===== INICIAR ACTUALIZACION AUTOMATICA =====
786
787     function startAutoUpdate() {
788         // Limpiar timer existente si hay
789         if (updateTimer) {
790             clearInterval(updateTimer);
791         }
792
793         // Configurar nuevo timer
794         updateTimer = setInterval(() => {
795             console.log('Auto-actualizacion... ');
796             loadParkingData();
797         }, UPDATE_INTERVAL);
798
799         console.log(
800             'Auto-actualizacion configurada cada $f
801             UPDATE_INTERVAL / 1000
802             } segundos'
803         );
804
805 // ===== DETENER ACTUALIZACION AUTOMATICA =====
806
807     function stopAutoUpdate() {
```

```
807     if (updateTimer) {
808         clearInterval(updateTimer);
809         updateTimer = null;
810         console.log('Auto-actualización pausada');
811     }
812 }
813
814 // ===== GESTIÓN DE VISIBILIDAD DE PESTANA
815 =====
816 document.addEventListener('visibilitychange', function () {
817     if (document.hidden) {
818         console.log('Pestaña oculta - pausando actualizaciones');
819         stopAutoUpdate();
820     } else {
821         console.log('Pestaña visible - reanudando actualizaciones');
822         loadParkingData();
823         startAutoUpdate();
824     }
825 });
826
827 // ===== MANEJO DE ERRORES GLOBALES
828 =====
829 window.addEventListener('error', function (e) {
830     console.error('Error global:', e.error);
831 });
832
833 window.addEventListener('unhandledrejection', function (e) {
834     console.error('Promise rechazada:', e.reason);
835 });
836 </script>
837 </body>
838 </html>
```

Listing E.2: Código fuente de templates/index.html.

E.3 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

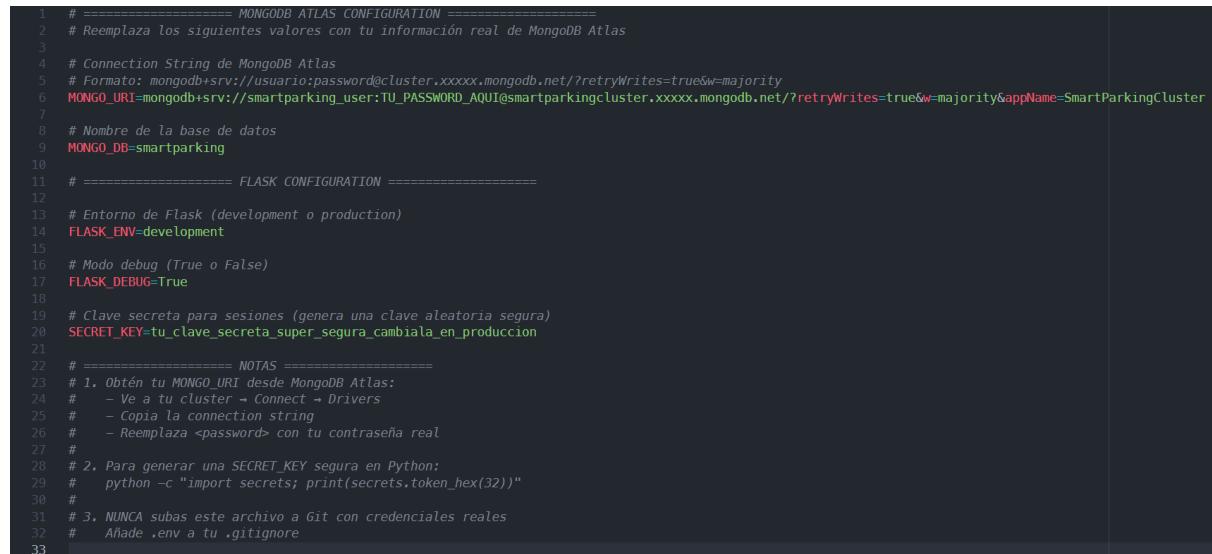
```

PS C:\Users\ampr2\Documents\FormacionProfesional\CFGS\IABD\BDA\Proyecto UD1\SmartParking Flow> python -m venv venv
PS C:\Users\ampr2\Documents\FormacionProfesional\CFGS\IABD\BDA\Proyecto UD1\SmartParking Flow> \venv\Scripts\activate
(venv) PS C:\Users\ampr2\Documents\FormacionProfesional\CFGS\IABD\BDA\Proyecto UD1\SmartParking Flow> pip install -r requirements.txt
Requirement already satisfied: Flask==3.0.0 in c:\users\ampr2\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\ampr2\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\ampr2\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\ampr2\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\ampr2\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\ampr2\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\ampr2\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\ampr2\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\ampr2\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\ampr2\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

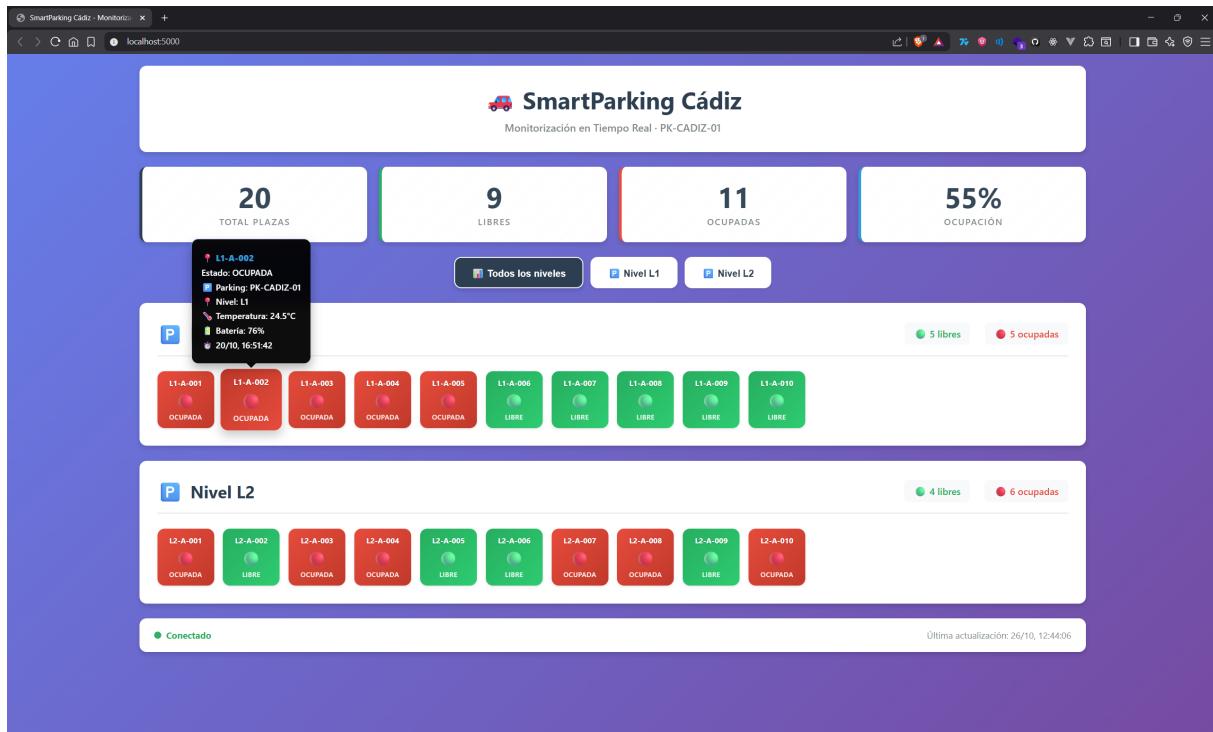


Figura E.4: Interfaz web de SmartParking mostrando el estado en tiempo real.

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.

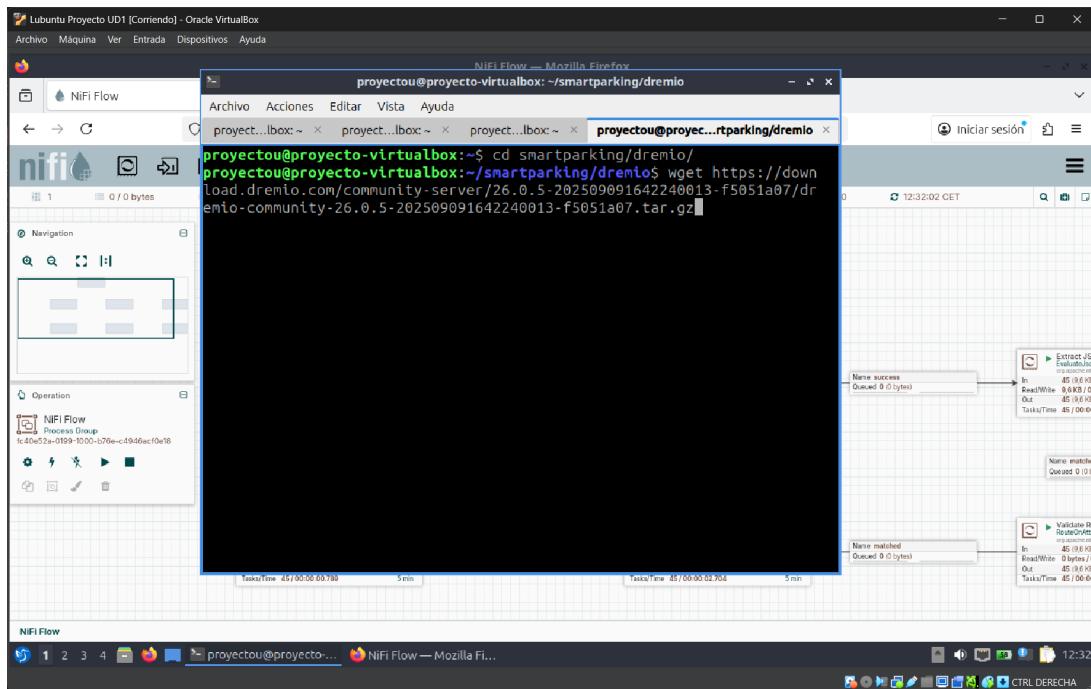


Figura F.1: Descarga de Dremio Community ('wget').

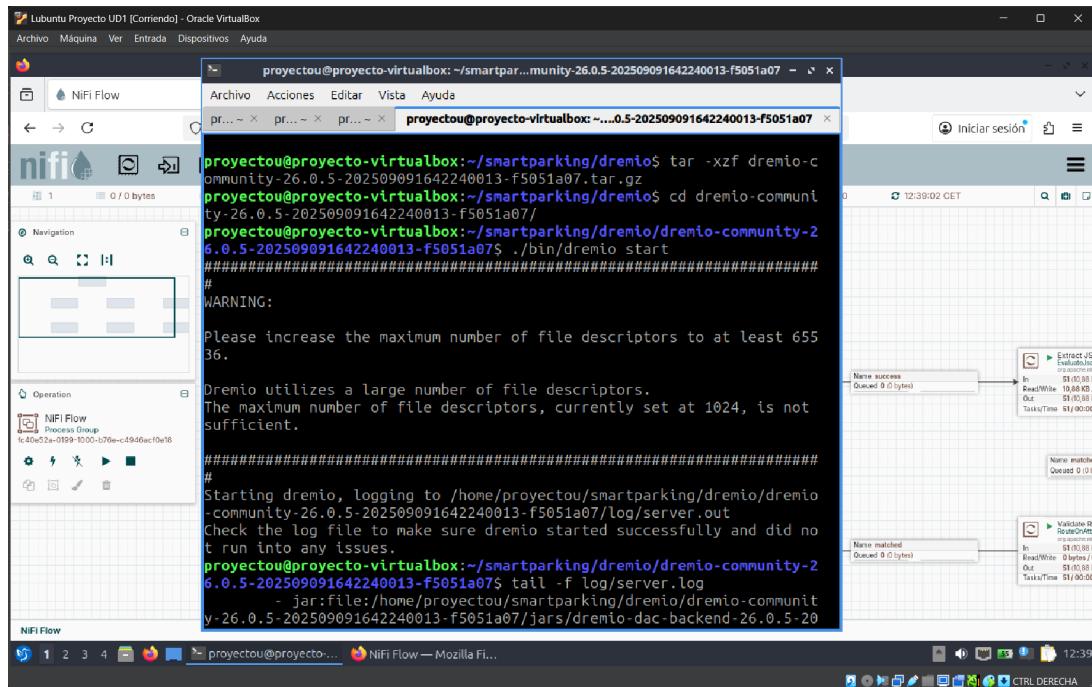


Figura F.2: Descompresión, inicio del servicio ('dremio start') y comprobación de logs.

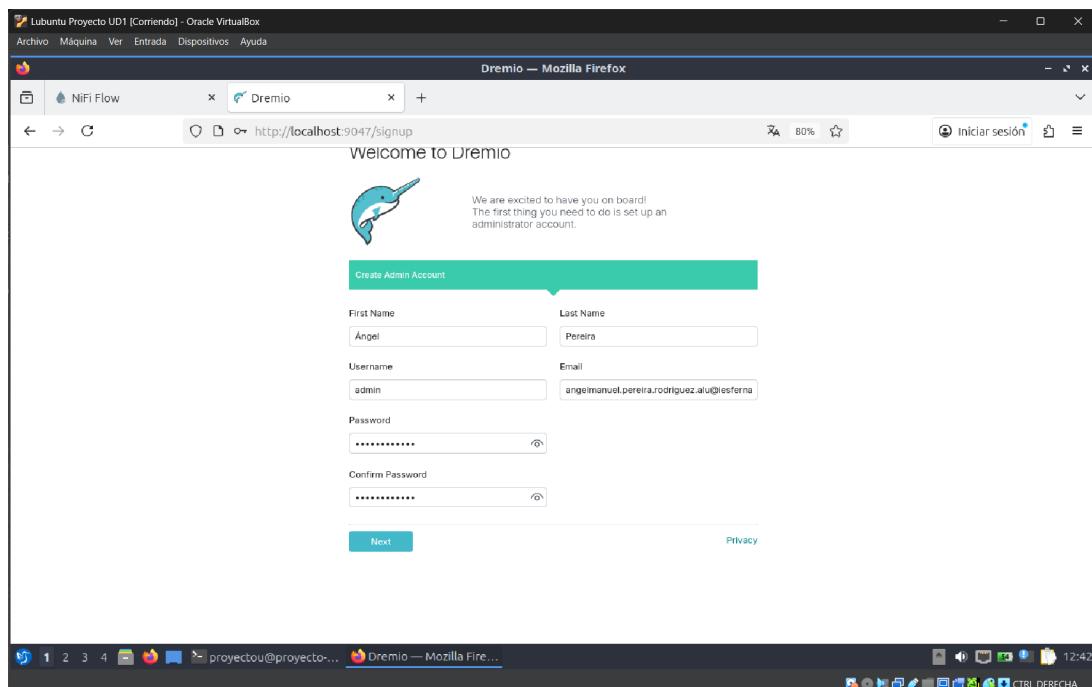


Figura F.3: Creación de la cuenta de administrador en la interfaz web de Dremio.

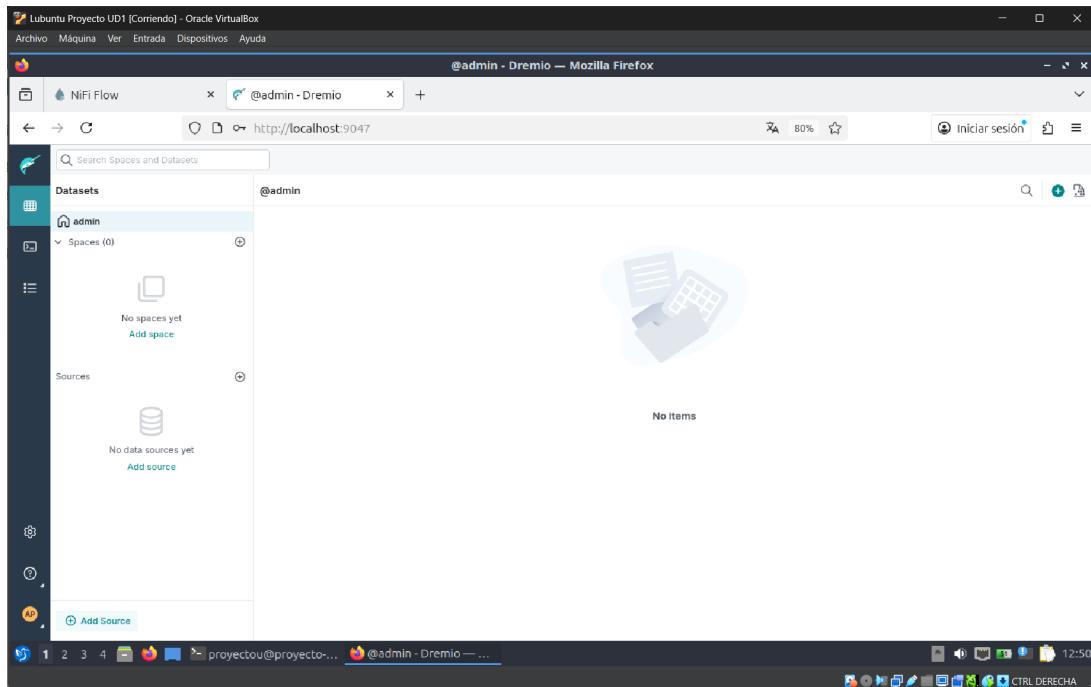


Figura F.4: Panel principal de Dremio (Datasets).

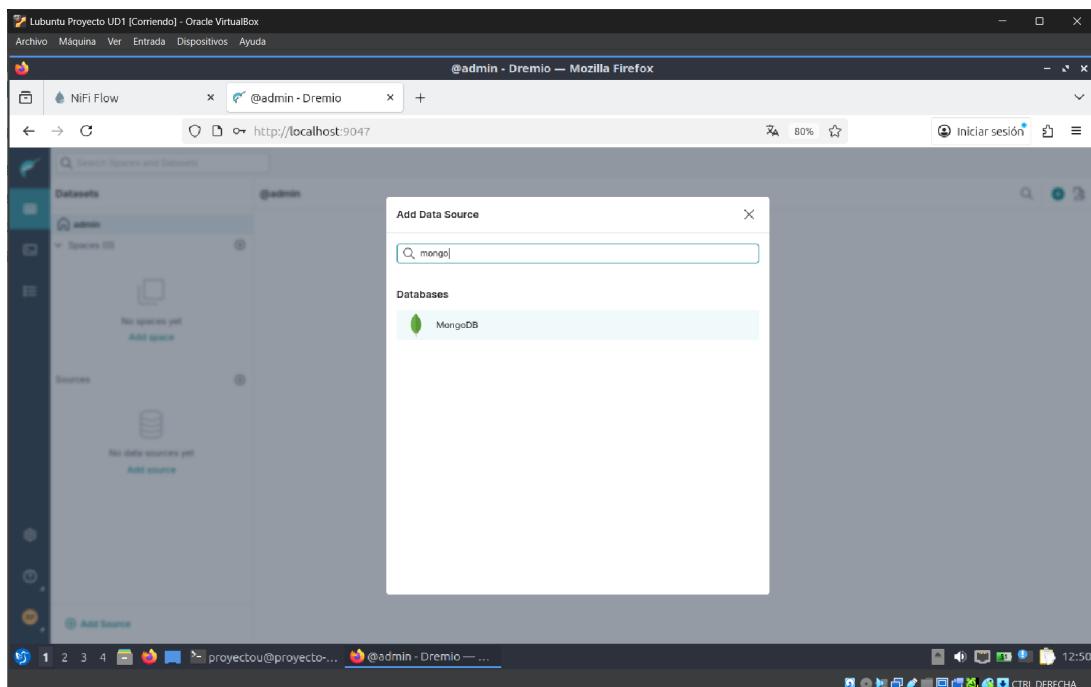


Figura F.5: Añadiendo una nueva fuente de datos (Add Data Source): MongoDB.

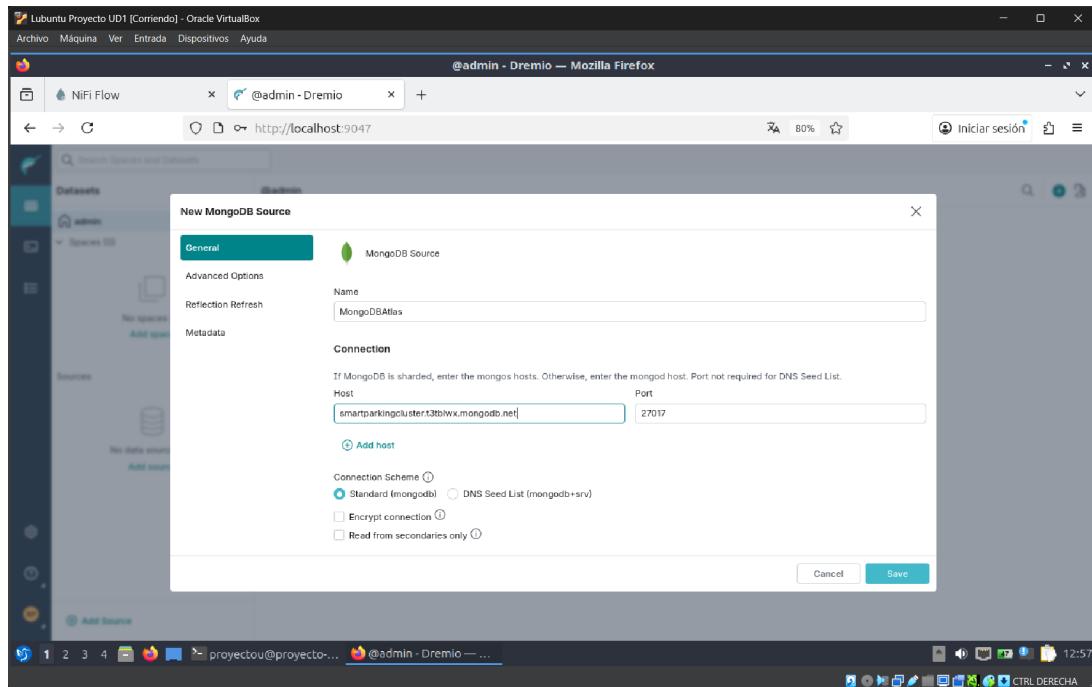


Figura F.6: Configuración de la fuente 'MongoDB Source': Host y Puerto.

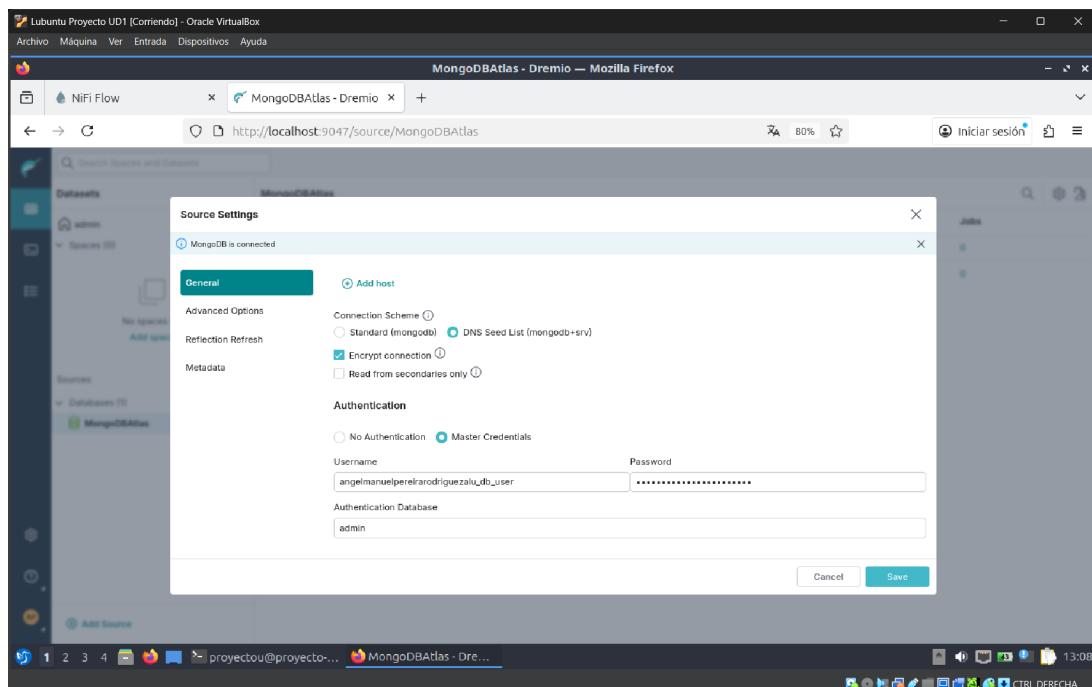


Figura F.7: Configuración de la autenticación de la fuente 'MongoDB Source'.

SmartParking Flow — Monitorización Inteligente

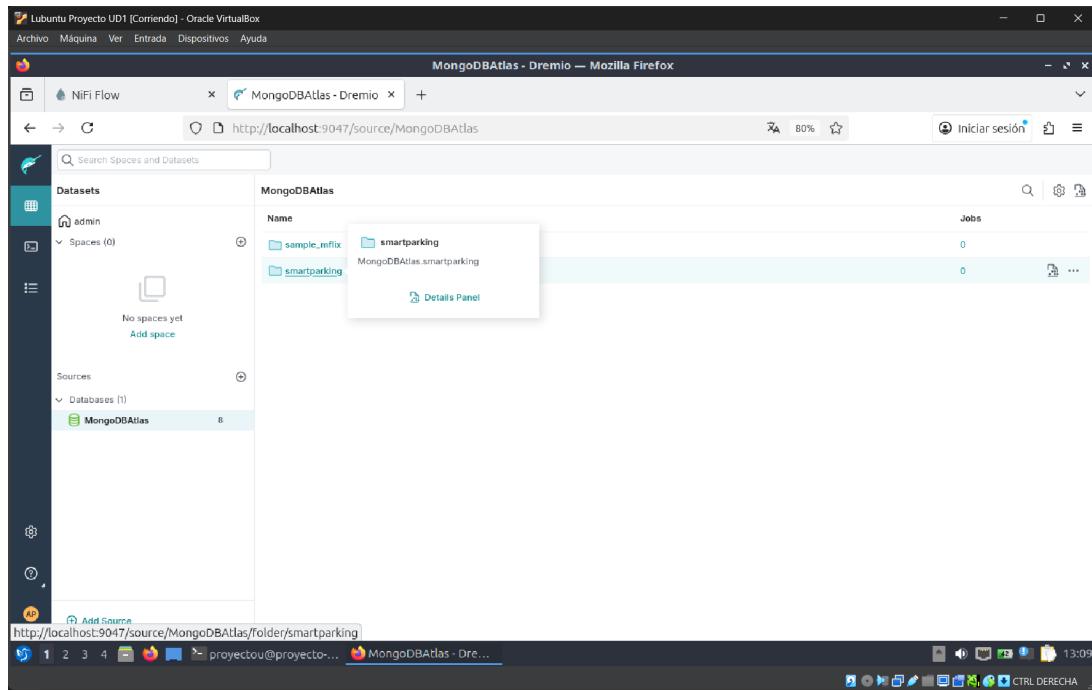


Figura F.8: Fuente de datos 'MongoDBAtlas' conectada, mostrando la base de datos 'smartparking'.

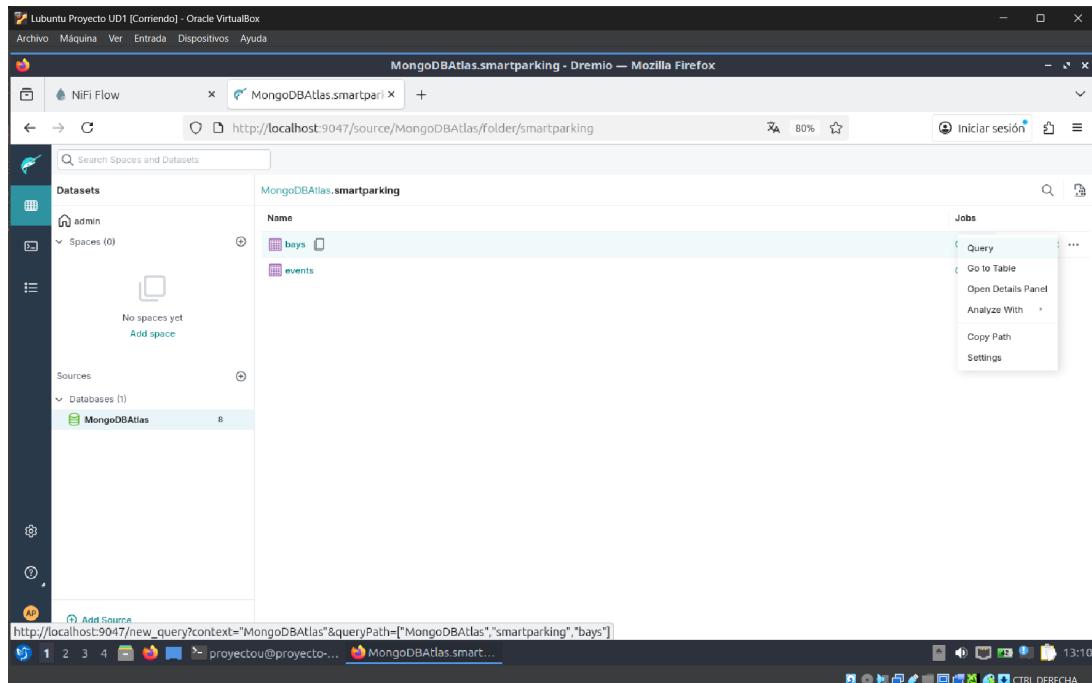
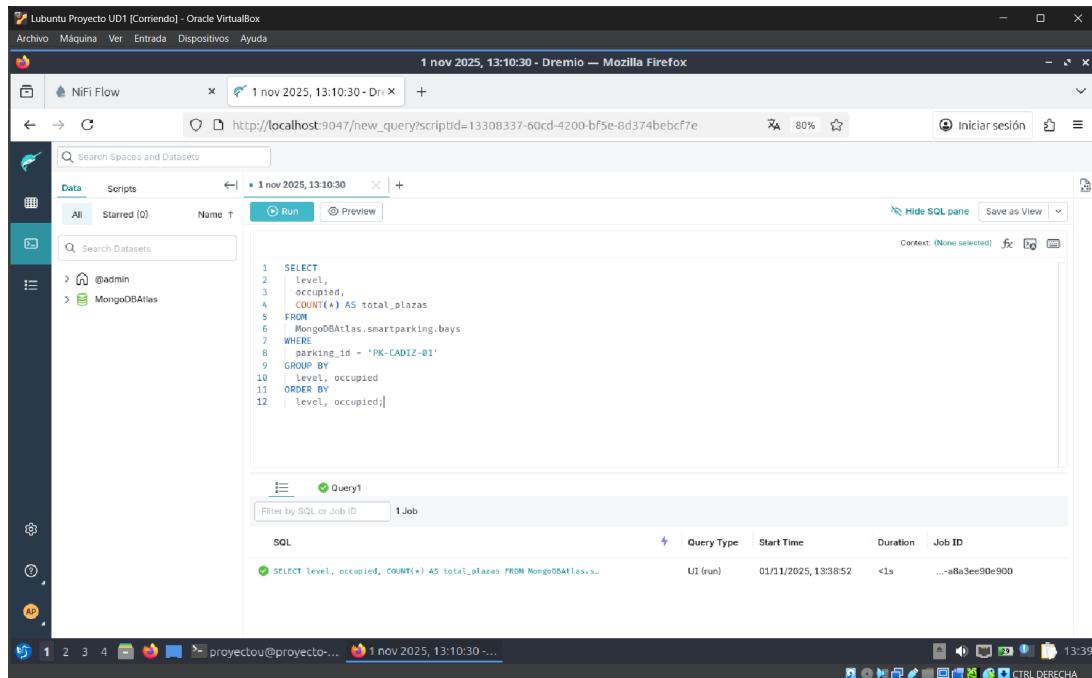


Figura F.9: Navegando en las colecciones 'bays' y 'events' dentro de Dremio.

SmartParking Flow — Monitorización Inteligente



The screenshot shows the Dremio interface running in a Firefox browser window. The URL is http://localhost:9047/new_query?scriptId=1330B337-60cd-4200-bf5e-8d374bebcf7e. The query editor contains the following SQL code:

```
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;
```

The results pane shows one job named "Query1" with the status "UI (run)". The command is the same as the query in the editor.

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

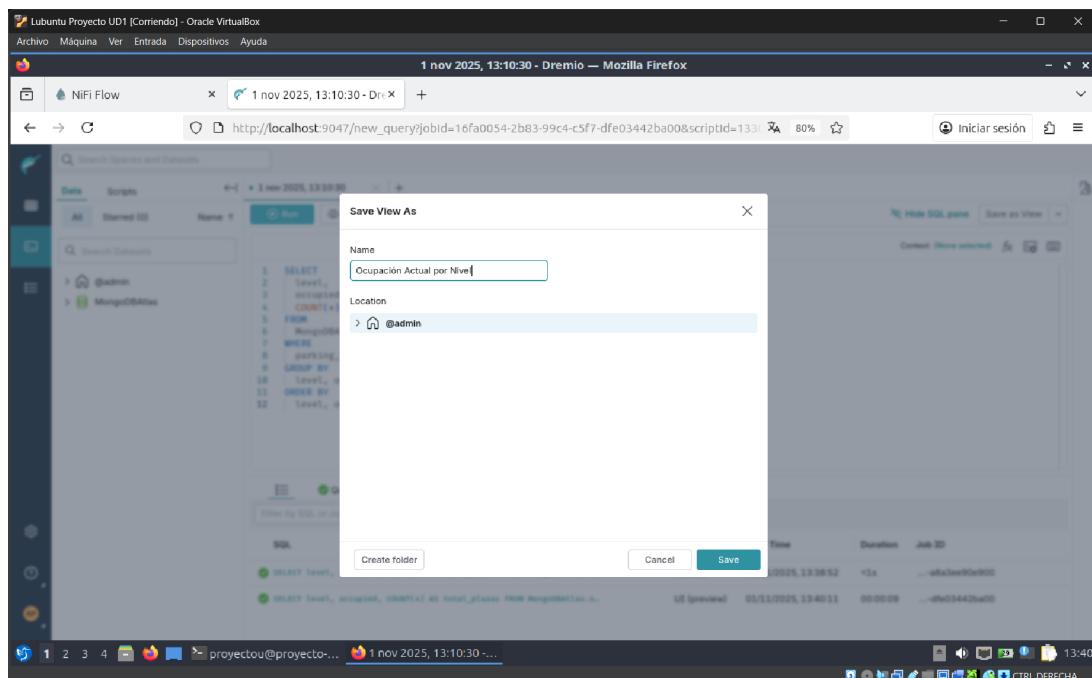
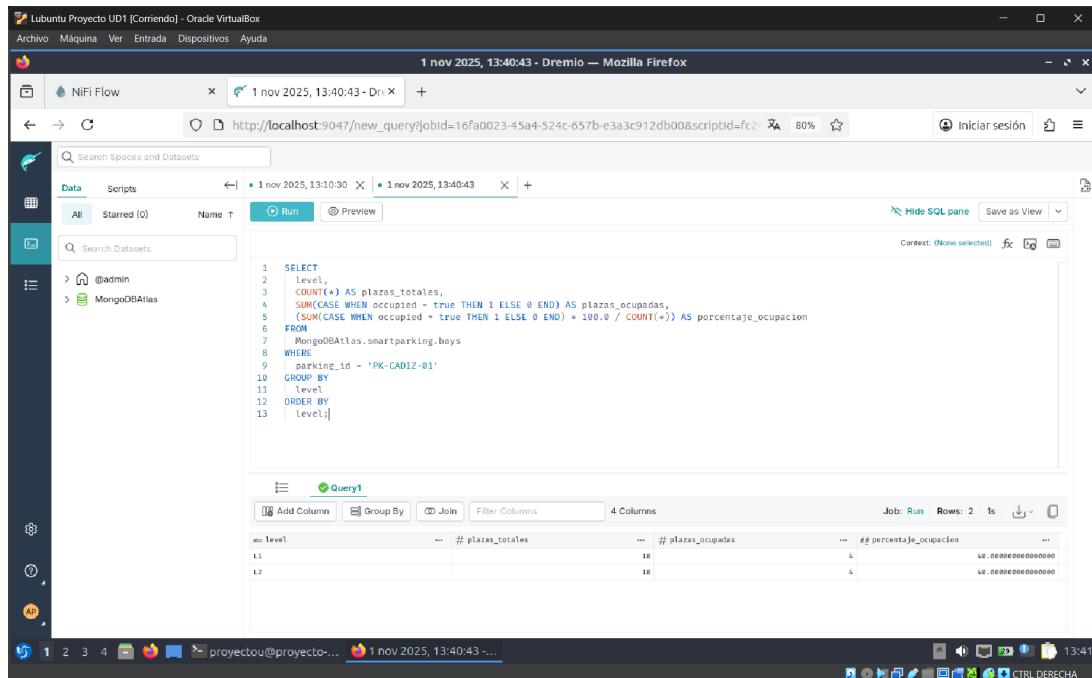


Figura F.11: Guardando la consulta (View) como 'Ocupación Actual por Nivel'.

SmartParking Flow — Monitorización Inteligente



The screenshot shows the Dremio SQL interface in Mozilla Firefox. The URL is http://localhost:9047/new_query?jobId=16fa0023-45a4-524c-657b-e3a3c912db00&scriptId=fca2e3a3c912db00. The query is:

```

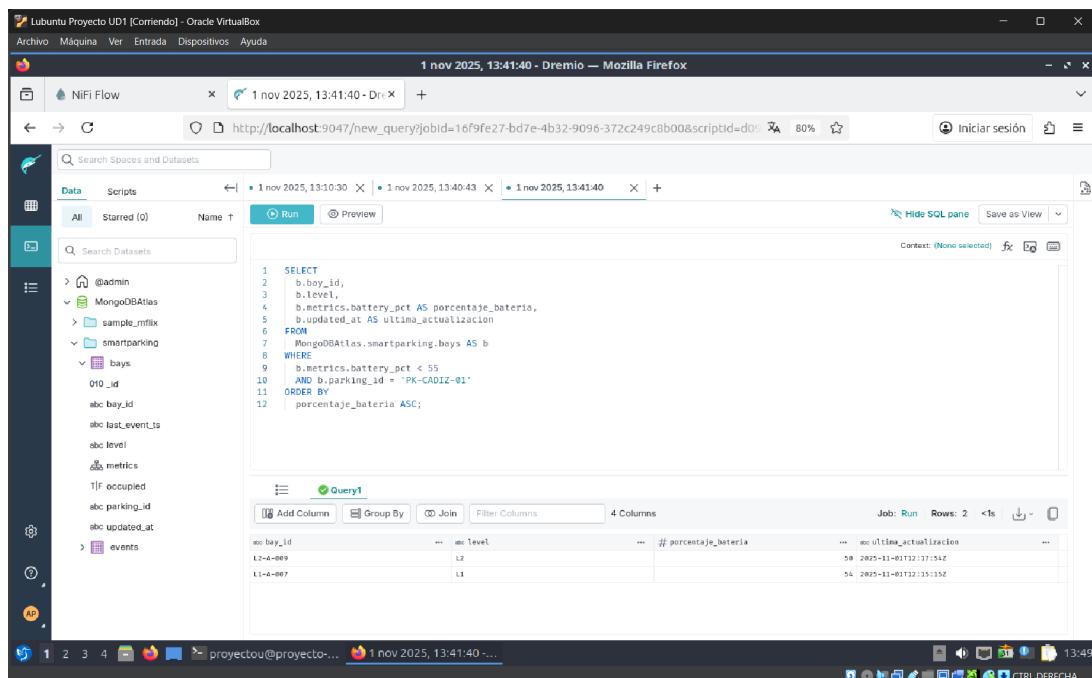
1 SELECT
2     level,
3     COUNT(*) AS plazas_totales,
4     SUM(CASE WHEN occupied = true THEN 1 ELSE 0 END) AS plazas_ocupadas,
5     (SUM(CASE WHEN occupied = true THEN 1 ELSE 0 END) * 100.0 / COUNT(*)) AS porcentaje_ocupacion
6   FROM
7     MongoDBAtlas.smartparking.bays
8   WHERE
9     parking_id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

```

The results table has four columns: `level`, `plazas_totales`, `plazas_ocupadas`, and `porcentaje_ocupacion`. The data is:

level	plazas_totales	plazas_ocupadas	porcentaje_ocupacion
L1	10	4	40.00000000000000
L2	10	6	60.00000000000000

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



The screenshot shows the Dremio SQL interface in Mozilla Firefox. The URL is http://localhost:9047/new_query?jobId=16f9fe27-bd7e-4b32-9096-372c249c8b00&scriptId=d05a2e3a3c912db00. 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`, `level`, `porcentaje_bateria`, and `ultima_actualizacion`. The data is:

bay_id	level	porcentaje_bateria	ultima_actualizacion
L1-A-009	L2	50	2025-11-01T12:17:54Z
L1-A-007	L1	54	2025-11-01T12:15:15Z

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	372
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).

The screenshot shows the Dremio interface running in a Firefox browser window. The title bar indicates it's on a virtual machine named 'Ubuntu Proyecto UD1 [Corriendo] - Oracle VirtualBox'. The main view displays a list of saved queries on the left, with a preview pane on the right showing the results of one specific query.

Left Panel (Saved Queries):

- @admin
- Eficiencia: Plazas Mas Utilizadas (Alta Rotación)
- Estado de Batería de Sensores (Mantenimiento)
- Horas de Mayor Demanda (Picos de Ocupación)
- Ocupación Actual por Nivel
- Porcentaje de Ocupación por Nivel
- MongoDBAtlas
- sample_mflix
- smartparking
- bays
- 010_id
- abc_bay_id
- abc_last_event_ts
- abc_level
- abc_metrics
- 1If occupied
- abc_parking_id
- abc_updated_at

Right Panel (Query Preview):

SQL Query:

```

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;

```

Result Table:

bay_id	level	total_eventos_ocupacion
L1-A-000	L2	35
L1-A-002	L1	34
L1-A-004	L2	31
L1-A-001	L2	31
L1-A-007	L1	31

Figura F.16: Muestra de todas las consultas guardadas (Views) en Dremio.