



ES|QL (ELASTICSEARCH QUERY LANGUAGE)

ES|QL



Revolución en Análisis y
Computación de Datos



Search. Observe. Protect.

La Nueva Frontera en Análisis de Datos

Descubriendo el Poder de ES|QL

Por Customer Architect Team

© Customer Architect Team

Conoce al Equipo de Arquitectos de Cliente (CA), tu aliado estratégico en el universo Elasticsearch. Estamos aquí para asegurar el éxito con Elasticsearch y que obtengas el máximo provecho de cada característica.

Te presentamos este Handbook esencial sobre ES|QL. Este manual no solo facilita tu introducción a ES|QL, sino que también te ayudará a optimizar tus procesos de analítica de datos. Nuestro objetivo es claro: hacer que ES|QL sea accesible, comprensible y valioso para ti.

Prepárate para transformar tu enfoque analítico con la guía experta del Equipo de CA.

www.elastic.co

A todo el equipo de Elasticsearch: su pasión, dedicación y conocimiento son la columna vertebral de este libro.

Gracias por convertir cada página en un reflejo de nuestra colaboración y compromiso con la excelencia.

Juntos, seguimos construyendo más que soluciones, construimos posibilidades.

Prólogo

En un mundo donde los datos no solo abundan sino que fluyen con una velocidad vertiginosa, la necesidad de comprender, procesar y extraer valor de ellos nunca ha sido tan crucial. Elasticsearch ha emergido como una solución líder, ofreciendo un poder y una flexibilidad sin precedentes en el manejo de datos. Sin embargo, como toda herramienta poderosa, desbloquear su verdadero potencial requiere conocimiento, habilidad y, lo más importante, una guía confiable.

Es aquí donde este Handbook sobre ES|QL entra en escena. Concebido desde la experiencia colectiva del equipo de Elasticsearch, este libro no es solo un compendio de instrucciones; es una brújula en el vasto paisaje de la analítica de datos. A través de sus páginas, ES|QL se revela no solo como un lenguaje de consulta, sino como un puente hacia una comprensión más profunda y un manejo más eficiente de sus datos.

Este Handbook es un testimonio de nuestra dedicación no solo a la excelencia tecnológica, sino también al éxito y crecimiento de nuestros usuarios. Cada capítulo, cada ejemplo, cada consejo, se ha diseñado con un propósito: empoderar a quienes lo leen para que no solo ejecuten consultas, sino que también desaten el potencial total de sus datos.

Nos embarcamos en este viaje sabiendo que cada conjunto de datos cuenta una historia, cada consulta desbloquea un misterio, y cada insight puede ser el

comienzo de algo extraordinario. Que este Handbook sea tu compañero en este viaje, iluminando tu camino, enriqueciendo tu comprensión y, finalmente, transformando tu relación con los datos.

Bienvenido a una nueva era de análisis de datos.
Bienvenido al mundo de ES|QL.

Índice

Prólogo	5
Introducción a ES QL.....	8
Primeros pasos con ES QL.....	14
Casos de uso de ES QL.....	18
Sintaxis de ES QL.....	23
Comandos de ES QL.....	26
Funciones y Operadores de ES QL.....	36
Campos metadato de ES QL.....	39
Campos de valores multiples de ES QL	41
Procesando datos con GROK y DISSECT	42
Enriqueciendo datos	45
Usando ES QL	49
Limitaciones	54
Ejemplos.....	57
Recursos	59

Introducción a ES|QL

En la vanguardia de la tecnología de consultas en Elasticsearch, ES|QL brinda una interfaz de consulta sofisticada y potente que transciende los métodos convencionales de análisis de datos. Este lenguaje de consultas, ofrece a los usuarios un medio mas intuitivo y directo para interactuar con sus datos, proporcionando capacidades de filtrado, análisis y transformación sin precedentes. Con su diseño centrado en el usuario, ES|QL demuestra ser una herramienta esencial para quienes buscan profundizar en la compresión de sus datos y obtener insights valiosos de manera eficaz.

Descripción General

Elasticsearch Query Language, es una poderosa herramienta diseñada para filtrar, transformar y analizar datos almacenados en Elasticsearch. Este lenguaje y motor de cómputo se caracteriza por su facilidad de aprendizaje y uso, lo que lo hace accesible para una amplia gama de usuarios, incluyendo equipos como los SRE, desarrolladores y administradores.

Una de las características distintivas de ES|QL es el uso de “pipes” (|) que permite manipular y transformar datos de manera secuencial. Esto posibilita la composición de una serie de operaciones donde el resultado de una se convierte en la entrada de la siguiente, facilitando así transformación de datos complejas y análisis detallados.

Además, ES|QL va más allá de ser simplemente un lenguaje de consulta, es un nuevo motor de cómputo dentro de Elasticsearch. A diferencia de otros lenguajes de consulta, las expresiones ES|QL no se transpiran a Query DSL para su ejecución, sino que son ejecutadas directamente dentro de Elasticsearch, lo que contribuyen a su alto rendimiento y versatilidad.

El motor de ejecución de ES|QL ha sido diseñado con el rendimiento en mente, operando en bloques en lugar de por fila, enfocándose en la vectorización y localidad de caché, y aprovechando la especialización y el multi-threading. Este enfoque asegura que ES|QL sea extremadamente eficiente en la realización de búsquedas, agregaciones y funciones de transformación.

Fundamentos de ES|QL

Diseñadlo para ofrecer una experiencia de consulta avanzada y altamente eficiente. Esta sección busca desentrañar los principios fundamentales de ES|QL, proporcionando una base sólida para aquellos que buscan explorar sus capacidades.

La naturaleza declarativa de ES|QL:

ES|QL es un lenguaje declarativo, lo que significa que se enfoca en el "qué" más que en el "cómo" de la manipulación de datos. Los usuarios especifican el resultado deseado sin detallar los pasos exactos necesarios para obtenerlo, permitiendo que el sistema optimice y determine el proceso más eficiente.

Sintaxis Basada en Pipes:

La sintaxis de ES|QL se centra en el uso de "pipes" (|), un enfoque que permite encadenar múltiples comandos para transformar y analizar datos. Este método promueve la legibilidad y facilita la construcción de consultas complejas, dividiéndolas en pasos secuenciales y manejables.

Composición de Consultas:

ES|QL permite la composición de consultas a través de una serie de comandos y funciones. Cada comando procesa los datos y pasa el resultado al siguiente comando en la

cadena. Esta estructura facilita la realización de operaciones complejas de filtrado, agregación y transformación de datos.

Integración profunda con Elasticsearch:

A diferencia de otros lenguajes de consulta, ES|QL está profundamente integrado con Elasticsearch. Las consultas se ejecutan directamente dentro de Elasticsearch, aprovechando su infraestructura distribuida y capacidades de indexación para proporcionar resultados rápidos y precisos.

Rendimiento y Optimización:

El motor de ejecución de ES|QL está diseñado para alto rendimiento. Operando en bloques y no por fila, se enfoca en la vectorización y localidad de caché. Además, el motor aprovecha la especialización y el multi-threading para optimizar las operaciones de consulta.

Flexibilidad y Extensibilidad:

ES|QL no solo se adapta a una amplia variedad de casos de uso sino que también ofrece la flexibilidad necesaria para extender y personalizar consultas. Soporta una gama de comandos y funciones, y se adapta a distintas necesidades de análisis y procesamiento de datos.

La Infraestructura Computacional de ES|QL

En el corazón de Elasticsearch, la infraestructura computacional de ES|QL se destaca como un pilar fundamental, proporcionando la robustez necesaria para manejar y procesar vastos conjuntos de datos con una eficiencia encomiable. Esta avanzada infraestructura se beneficia enormemente de la arquitectura distribuida de Elasticsearch, permitiendo un escalado eficiente y una rápida ejecución de consultas. Más que solo procesamiento, el motor de ES|QL es sinónimo de inteligencia y adaptabilidad; está dotado con mecanismos avanzados para la optimización dinámica de consultas, ajustando las operaciones en tiempo real para garantizar un procesamiento de datos que no solo es rápido sino también increíblemente eficiente.

Pero la versatilidad de ES|QL no termina ahí. Con un soporte extensivo para operaciones avanzadas, desde windowing hasta el análisis de series temporales, ES|QL se presenta como una herramienta excepcionalmente adaptable y poderosa, adecuada para una diversidad de escenarios y necesidades analíticas. Esta versatilidad se ve aún más enriquecida por su estrecha integración con herramientas de visualización como Kibana, lo que permite a los usuarios no solo indagar en sus datos sino también dar vida a los insights a través de representaciones visuales intuitivas. Este enfoque holístico se ve reforzado por el compromiso constante del equipo de Elasticsearch con la mejora y actualización de ES|QL, asegurando que esta

herramienta no solo esté a la vanguardia en términos de rendimiento y características sino que también permanezca en sintonía con las dinámicas cambiantes del mundo de los datos.

Primeros pasos con ES|QL

Iniciar en el mundo de ES|QL es una aventura emocionante que comienza con la preparación y configuración adecuada del entorno. Antes de sumergirse en las profundidades de este lenguaje de consulta, es fundamental asegurarse de que todas las piezas necesarias estén en su lugar. Esto incluye tener una instancia de Elasticsearch operativa y familiarizarse con las interfaces a través de las cuales ES|QL será utilizado, como Kibana o la API REST de Elasticsearch. Esta etapa inicial es crucial para garantizar un recorrido sin contratiempos en el análisis de datos con ES|QL.

¿Listo para sumergirte en el mundo de ES|QL y ver lo que tus datos pueden hacer realmente? Crea tu cuenta gratuita en Elasticsearch Cloud desde <https://ela.st/startnow>.

Una vez que el entorno está listo, ¡es hora de jugar, experimentar y descubrir! los primeros pasos en ES|QL. Esto implica conocer su sintaxis única, dominar una variedad de comandos específicos, y entender cómo trabajar con funciones y operadores. Además, es crucial familiarizarse con los campos de metadatos y cómo manejar campos multivalor. Aprender a procesar datos con herramientas como Grok y Dissect, y a enriquecer datos son también partes fundamentales de este aprendizaje. Cada aspecto es crucial para aprovechar la potencia de ES|QL en Elasticsearch para el análisis de datos complejos.

Sintaxis:

La sintaxis de ES|QL se centra en una estructura basada en pipes (|) que guía el flujo de datos a través de diversas operaciones, facilitando una construcción lógica y secuencial de consultas. Esta metodología permite a los usuarios ejecutar transformaciones complejas y análisis de datos de manera intuitiva y eficiente. Por ejemplo, una consulta puede comenzar con una extracción de datos usando FROM, seguida de una serie de filtros y transformaciones como WHERE, GROUP BY, y ORDER BY, cada uno conectado por un pipe para una ejecución fluida y ordenada.

Ejemplo:

```
FROM apache-logs
| WHERE url.original == '/login'
| EVAL time_buckets = auto_bucket
(@timestamp, 50,"
```

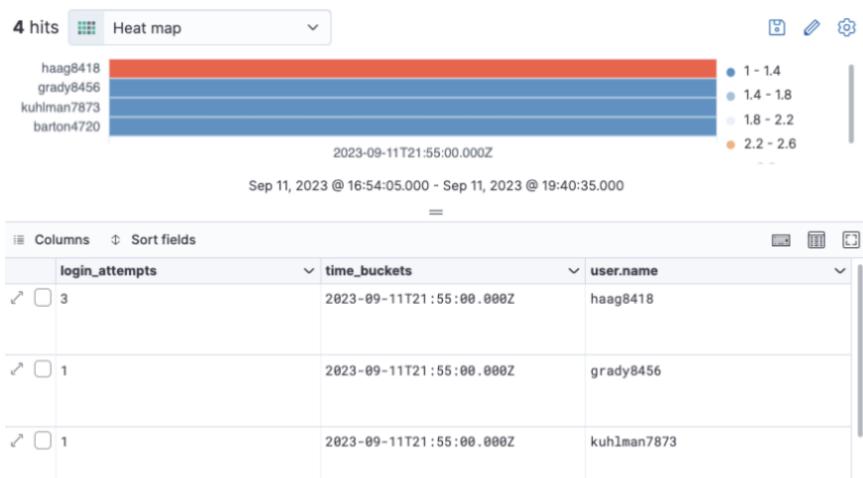
```

2023-09-11T21:54:05.000Z", "
2023-09-12T00:40:35.000Z")
| STATS login_attempts =
count(user.name) by time_buckets,
user.name
| SORT login_attempts desc

```

En este ejemplo podemos ver como seleccionamos el índice ‘apache-logs’ filtrando aquellos mensajes que contienen ‘/login’, evaluamos los datos comprendidos entre dos puntos temporales, repartiéndolos en 50 ‘buckets’, contando los eventos identificados agrupándolos por usuario para finalmente ordenarlos de forma descendente.

Como resultado obtendríamos algo parecido a esto:



Aunque esto representa un ejemplo, nos muestra un caso de uso de seguridad que aporta mucho valor.

Mas adelante iremos profundizando en los casos de usos y ejemplos que nos permitan profundizar poco a poco en esta aventura con ES|QL.

Casos de uso de ES|QL

En el capítulo anterior donde realizamos una introducción de ES|QL y su sintaxis, vimos un ejemplo práctico en un caso de seguridad, donde pudimos evaluar los logins de nuestros usuarios.

En este capítulo veremos los tres casos de uso principales de ES|QL, aunque estamos seguro que podrás aplicarlos a otros muchos.

Observabilidad

Conocido como ‘only’ para abreviar, ES|QL nos ayudará visualizando y analizando datos dentro de Elasticsearch.

Directamente desde la barra de búsqueda, puedes agregar, transformar, calcular y buscar tus métricas, registros y trazas con una sola consulta para optimizar la identificación de cuellos de botella de rendimiento y problemas del sistema, reduciendo el tiempo de resolución.

ES|QL introduce capacidades avanzadas como la definición de campos en el momento de la consulta, búsquedas para el enriquecimiento de datos y procesamiento de consultas concurrente, mejorando la velocidad y eficiencia.

The screenshot shows the Grafana ES|QL editor interface. At the top, there is a code editor window containing the following query:

```
1 from metrics* |
2 stats max_cpu = max(kubernetes.pod.cpu.usage.node.pct),
  avg_mem = max(kubernetes.pod.memory.usage.bytes) by
    kubernetes.pod.name |
3 sort max_cpu desc | limit 10
```

Below the code editor, the status bar indicates "3 lines" and "@timestamp detected". On the right side of the status bar are buttons for "Run query" and "Enter".

Below the status bar is a table header row with columns: "max_cpu", "avg_mem", and "kubernetes.pod.name".

max_cpu	avg_mem	kubernetes.pod.name
-	-	-
0.125	945872896	heartbeat-synthetics-6c9497b68-pljxr
0.117	943742976	heartbeat-synthetics-tokyo-5b9f74dd57-27hlv
0.099	2220580864	relevance-workbench-app-ui-f7cbd657c-dpd7d
0.097	1999900672	elastic-agent-cxjv4
0.09	232505344	kafka-loadgen-deco-green-5cf8cc7988-pxcnp

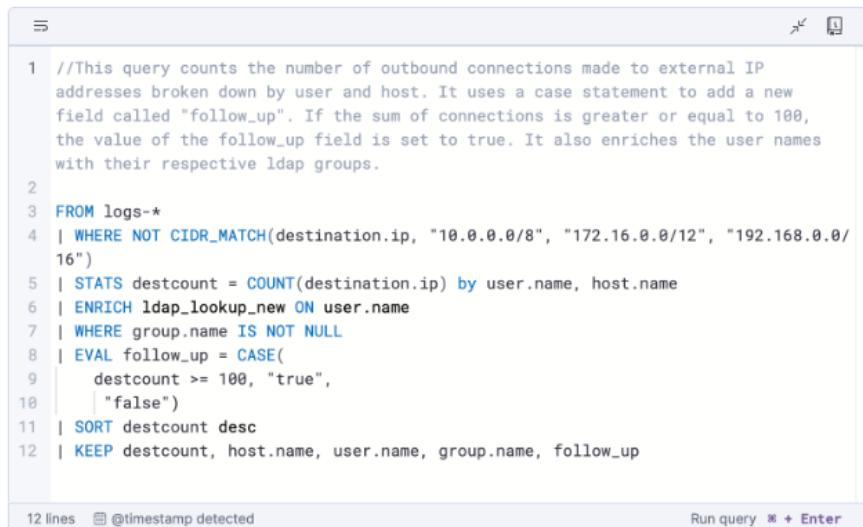
Seguridad

Mejora tu postura de seguridad, acelera los flujos de trabajo de SecOps y mejora la precisión de las alertas, sin importar la fuente y estructura de los datos.

ES|QL permite búsquedas rápidas y flexibles, definición de nuevos campos al vuelo, agregación de resultados, y visualización de patrones significativos, todo desde una única consulta.

Incorpora valores agregados en las reglas de detección para alertar con mayor precisión y reducir la fatiga de alarmas, proporcionando más señales y menos ruido.

Enriquece los datos proporcionando un contexto adicional crucial para investigaciones de seguridad, como la localización geográfica de una dirección IP y su asociación con entidades maliciosas, el usuario y su grupo..., todo desde una sola barra de búsqueda.



```
1 //This query counts the number of outbound connections made to external IP
2 addresses broken down by user and host. It uses a case statement to add a new
3 field called "follow_up". If the sum of connections is greater or equal to 100,
4 the value of the follow_up field is set to true. It also enriches the user names
5 with their respective ldap groups.
6
7 FROM logs-* | WHERE NOT CIDR_MATCH(destination.ip, "10.0.0.0/8", "172.16.0.0/12", "192.168.0.0/16")
8 | STATS destcount = COUNT(destination.ip) by user.name, host.name
9 | ENRICH ldap_lookup_new ON user.name
10 | WHERE group.name IS NOT NULL
11 | EVAL follow_up = CASE(
12     destcount >= 100, "true",
13     "false")
14 | SORT destcount desc
15 | KEEP destcount, host.name, user.name, group.name, follow_up
```

12 lines @timestamp detected Run query + Enter

4 hits [Reset search](#)

destcount	host.name	user.name	group.name	follow_up
213	omm-win-detect	Administrator	local_admins	true
127	omm-win-detect	SYSTEM	system_users	true
98	omm-win-prevent	SYSTEM	system_users	false
86	omm-win-prevent	Administrator	local_admins	false

Búsqueda

Eleva tus capacidades de búsqueda, los desarrolladores encontrarán en ES|QL una experiencia de codificación y consulta simplificada, lo que se traduce en ahorros significativos de tiempo y costos.

ES|QL no solo facilita un mayor entendimiento sobre los datos –qué contienen, cómo organizarlos y cómo solucionar problemas– sino que también optimiza las tareas al consolidarlas en una única consulta procesable de manera concurrente, lo que mejora el rendimiento y reduce el Costo Total de Operación (TCO), permitiendo lograr más con menos..

Esta eficiencia se logra sin sacrificar la profundidad o la calidad del análisis, ofreciendo una solución integral para la manipulación de datos.

```
1 from kibana_sample_data_ecommerce
2 | where products.base_price >15 and geoip.city_name == "New York"
3 | stats avgbaseprice = avg(products.base_price) by category, day_of_week
```

3 lines @timestamp not detected

Run query + Enter

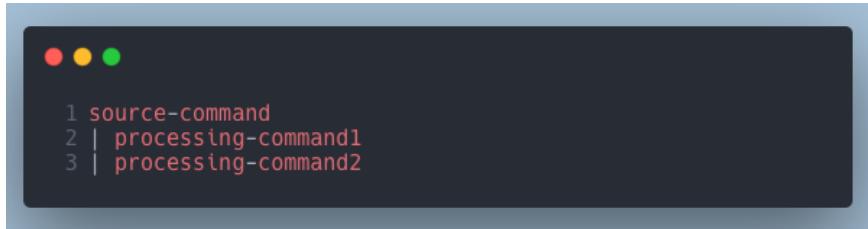
Columns Sort fields

avgbaseprice	category	day_of_week
65	Women's Clothing	Sunday
60	Women's Clothing	Monday
60.83333333333336	Women's Clothing	Tuesday
33	Men's Clothing	Wednesday
61.25	Women's Clothing	Thursday
67.5	Women's Clothing	Friday
65	Women's Clothing	Wednesday

Sintaxis de ES|QL

En este capítulos nos centraremos en la sintaxis de ES|QL e iremos profundizando poco a poco en el entendimiento y aprendizaje.

En líneas generales ES|QL está compuesto de un “source-command” seguido (opcionalmente) de una serie de “processing-commands”



```
1 source-command
2 | processing-command1
3 | processing-command2
```

Aunque todo se puede poner en una única línea, podemos apilarlo para facilitar la lectura.

Identificadores

Son nombres usados para identificar elementos como capos. Pueden usarse tal cual, excepto si contienen caracteres especiales, en cuyo caso deben estar entre comillas invertidas ('`').

```
● ● ●
1 // Retain just one field
2 FROM index
3 | KEEP 1.field
4
5 // Copy one field
6 FROM index
7 | EVAL my_field = `1.field`
```

Literales

Representan valores fijos en las consultas, soportando tanto numéricos como de cadena de texto. Estos últimos deben ir entre comillas dobles ("") y los números pueden ser enteros, decimales o en notación científica.

```
● ● ●
1 // Filter by a string value
2 FROM index
3 | WHERE first_name == "Georgi"
4
5 ROW name = """Indiana "Indy" Jones"""
6
7
8 1969    -- integer notation
9 3.14    -- decimal notation
10 .1234   -- decimal notation starting with decimal point
11 4E5     -- scientific notation (with exponent marker)
12 1.2e-3  -- scientific notation with decimal point
13 -.1e2   -- scientific notation starting with the negative sign
```

Comentarios

Se permite añadir notas o desactivar partes del código, para ello usamos ‘//’ para comentarios de una sola línea o ‘/* ... */’ para comentarios en bloque.

```
1 // Query the employees index
2 FROM employees
3 | WHERE height > 2
4
5 FROM /* Query the employees index */ employees
6 | WHERE height > 2
7
8 FROM employees
9 /* Query the
10 * employees
11 * index */
12 | WHERE height > 2
```

Intervalos de fecha y hora

Los podemos expresar mediante literales de intervalo, combinando un número con un calificador de tiempo como ‘days’, ‘hours’, etc. y se pueden usar para especificar duraciones o periodos de tiempo.

Lista de calificadores compatibles:

```
2 second/seconds
3 minute/minutes
4 hour/hours
5 day/days
6 week/weeks
7 month/months
8 year/years
```

Comandos de ES|QL

Existen dos categorías de comandos en ES|QL, los denominados “source” y los “processing”.

Source commands

Estos comandos producen tablas y son el inicio de cualquier consulta, estando soportados:

- FROM
- ROW
- SHOW

Processing commands

Estos procesan y modifican la entrada recibida añadiendo, eliminado o cambiando las filas o columnas, estando soportados:

- DISSECT
- DROP
- ENRICH
- EVAL
- GROK
- KEEP
- LIMIT
- MV_EXPAND
- RENAME
- SORT
- STATS ... BY
- WHERE

A continuación iremos profundizando en cada uno de ellos. en capítulos posteriores veremos ejemplos prácticos.

FROM

Con ello especificamos la fuente de datos para la consulta, como un índice o alias y retorna una tabla respetando documentos, con filas y columnas correspondientes a documentos y campos.

FROM kibana_sample_data_ecommerce

ROW

Nos permite genera una fila con una o más columnas con los valores concreto. Esto puede resultar útil para realizar pruebas.

```
ROW a = 1, b = "two", c = null
```

SHOW

Nos proporciona información sobre el despliegue (SHOW INFO) y sobre las funciones (SHOW functions) donde podemos profundizar como en el ejemplo de a continuación

```
SHOW functions
```

```
| WHERE STARTS_WITH(name, "sin")
```

DISSECT

Nos permite extraer información estructurada de un string, donde se realizara una coincidencia de un patron como en el ejemplo de a continuación.

```
ROW a = "2023-01-23T12:15:00.000Z -  
some text - 127.0.0.1"
```

```
| DISSECT a "%{date} - %{msg} - %{ip}"
```

DROP

Podremos eliminar columnas (campos) que no necesitemos en el proceso de los datos, siguiendo el ejemplo anterior, una vez procesada la columna “a” podemos hacerle DROP

```

ROW a = "2023-01-23T12:15:00.000Z -  

some text - 127.0.0.1"  

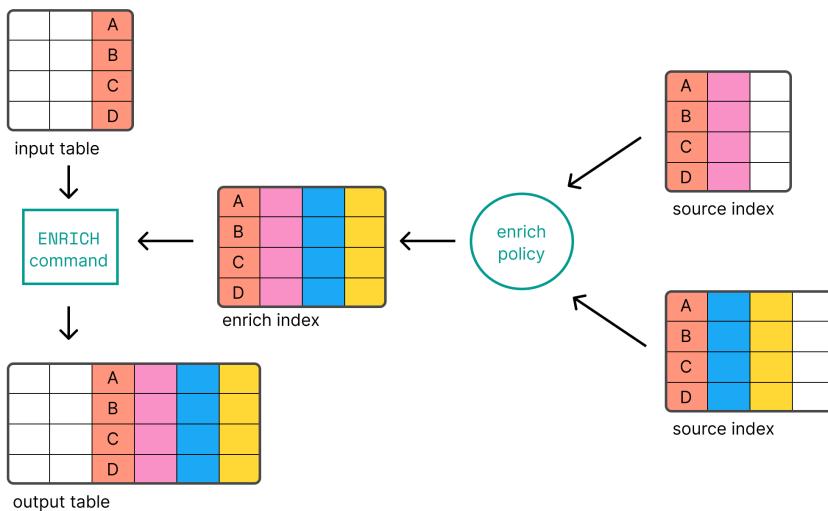
| DISSECT a "%{date} - %{msg} - %{ip}"  

| DROP a

```

ENRICH

Este comando nos permitirá enriquecer en proceso de consulta los datos que estamos operando haciendo uso de políticas de enriquecimiento, permitiendo un gran abanico de posibilidades.



EVAL

Con ello podremos añadir nuevas columnas o campos con valores calculados, gracias al uso de Funciones

```
from kibana_sample_data_ecommerce
| EVAL price_iva =
mv_sum(products.base_price)* 1.21
| KEEP products.base_price, price_iva
```

GROK

Al igual que DISSECT nos permitirá extraer información estructurada de un string o cadena de texto, en base a un patrón

```
ROW a = "2023-01-23T12:15:00.000Z
127.0.0.1 some.email@foo.com 42"
| GROK a "%{TIMESTAMP_ISO8601:date} %
{IP:ip} %{EMAILADDRESS:email} %
{NUMBER:num:int}"
| KEEP date, ip, email, num
```

KEEP

Este comando nos permite seleccionar de un salida, las columnas o campos que deseamos mantener, al igual que DROP nos permite quitar, en ocasiones nos resultará mas sencillo definir que queremos mantener.

```
from kibana_sample_data_ecommerce
| KEEP
customer_full*.keyword, customer_id,
email
```

LIMIT

Limita el numero de filas devueltas por la consulta (por defecto son 500) permitiendo un control mas fino sobre el volumen de datos procesados.

Este límite solo se aplica al número de filas que recupera la consulta. Las consultas y agregaciones se ejecutan en el conjunto de datos completo.

MV_EXPAND

Expande columnas multivalor en una fila por valor, duplicando otras columnas.

```
ROW a=[1,2,3], b="b", j=["a","b"]  
| MV_EXPAND a  
| MV_EXPAND j
```



6 results

#	a	b	j
1	b	a	
1	b	b	
2	b	a	
2	b	b	
3	b	a	
3	b	b	

RENAME

Cambia el nombre de una o más columnas. Si ya existe una columna con el nuevo nombre, será reemplazada por la nueva columna.

```
ROW a=[1,2,3], b="b", j=["a","b"]
| MV_EXPAND a
| MV_EXPAND j
| RENAME a as number, b as b_value, j
as j_value
```

6 results

	#	number	t	b_value	t	j_value
↗	1	1	b	a		
↗	1	1	b	b		
↗	2	2	b	a		
↗	2	2	b	b		
↗	3	3	b	a		
↗	3	3	b	b		

SORT

Nos permite ordenar la tabla o una o mas columnas

```
ROW a=[1,2,3], b="b", j=["a","b"]
| MV_EXPAND a
| MV_EXPAND j
| RENAME a as number, b as b_value, j
as j_value
| SORT number desc
```

 6 results

#	number	t	b_value	t	j_value
3	3	b	a		
3	3	b	b		
2	2	b	a		
2	2	b	b		
1	1	b	b		
1	1	b	a		

STATS ... BY

Agrupa filas de acuerdo con un valor común y calcula uno o más valores agregados sobre las filas agrupadas. Si se omite BY, la tabla de salida contiene exactamente una fila con las agregaciones aplicadas a todo el conjunto de datos

```
FROM kibana_sample_data_ecommerce
| EVAL price_iva =
mv_sum(products.base_price)* 1.21
| KEEP products.base_price, price_iva,
customer_id, customer_first_name
| STATS avg(price_iva) BY
customer_first_name
```

#	avg(price_iva)	customer_first_name
84.45894531249999	Abdulraheem Al	
116.399211328125	Eddie	
84.80904854910713	Mary	
84.74110559682377	Gwen	

WHERE

Produce una tabla que contiene todas las filas de la tabla de entrada para las cuales la condición proporcionada se evalúa como verdadera.

```
FROM kibana_sample_data_ecommerce
```

```
| EVAL price_iva =  
mv_sum(products.base_price)* 1.21  
| KEEP products.base_price, price_iva,  
customer_id, customer_first_name  
| STATS avg(price_iva) BY  
customer_first_name  
| WHERE `avg(price_iva)` < 100 and  
customer_first_name RLIKE "M.*"
```

 4 results

#	avg(price_iva)	customer_first_name
1	84.80904854910713	Mary
2	93.54259732521186	Muniz
3	83.45342051630435	Mostafa
4	80.06774719238281	Marwan

Funciones y Operadores de ES|QL

Las funciones y operadores de ES|QL te permiten manipular y analizar datos de manera eficiente y versátil, facilitando la creación de consultas para buscar eventos específicos, realizar análisis estadísticos y generar visualizaciones.

Funciones

Principalmente tenemos de:

- Agregación
- Matemáticos
- Cadenas de texto
- Fechas y tiempos

- Conversión
- Condicionales
- Valor multiple

Las funciones de agregación

AVG, COUNT, COUNT_DISTINCT, MAX, MEDIAN,
MEDIAN_ABSOLUTE_DEVIATION, MIN, PERCENTILE,
ST_CENTROID, SUM

Las funciones Matemáticas

ABS, ACOS, ASIN, ATAN, ATAN2, CEIL, COS, COSH, E,
FLOOR, LOG, LOG10, PI, POW, ROUND, SIN, SINH, SQRT,
TAN, TANH, TAU

Las funciones con cadenas de texto

CONCAT, LEFT, LENGTH, LTRIM, REPLACE, RIGHT,
RTRIM, SPLIT, SUBSTRING, TO_LOWER, TO_UPPER, TRIM

Las funciones de fecha y hora

AUTO_BUCKET, DATE_DIFF, DATE_EXTRACT,
DATE_FORMA, DATE_PARSE, DATE_TRUNC, NOW

Las funciones de conversión de tipos

TO_BOOLEAN, TO_CARTESIANPOINT,
TO_CARTESIANSHAPE, TO_DATETIME, TO_DEGREES,
TO_DOUBLE, TO_GEOPOINT, TO_GEOSHAPE, TO_INTEGER,

TO_IP, TO_LONG, TO_RADIANS, TO_STRING,
TO_UNSIGNED_LONG, TO_VERSION

Las funciones y expresiones condicionales

CASE, COALESCE, GREATEST, LEAST

Las funciones de valores multiples

MV_AVG, MV_CONCAT, MV_COUNT, MV_DEDUPE,
MV_FIRST, MV_LAST, MV_MAX, MV_MEDIAN, MV_MIN,
MV_SUM

Operadores

Binarios (==, !=, <, <=, >, >=, +, -, *, /, %), Unarios (-),
Lógicos (AND, OR, NOT), IS NULL / IS NOT NULL,
CIDR_MATCH, ENDS_WITH, IN, LIKE, RLIKE,
STARTS_WITH

Campos metadato de ES|QL

Podemos acceder a los campos de metadatos (_index, _id, _version) desde el comando FROM, que es la única directiva que admite este campo

En los posibles casos de uso, accediendo al _id podemos realizar operaciones cuando conocemos previamente el valor de dicho campo, o si este es común entre índices y queremos hacer operaciones conjuntas, de igual forma con _index, cuando, por ejemplo, necesitemos hacer agrupaciones por índices.

```
from kibana* metadata _id, _index  
| stats n_documents = count(_id) by  
_index
```



3 results

n_documents _index

✓ 14074 .ds-kibana_sample_data_logs-2024.01.29-000001

✓ 13014 kibana_sample_data_flights

✓ 4675 kibana_sample_data_ecommerce

Campos de valores multiples de ES|QL

Con ES|QL también podemos operar sobre campos con multiples valores aunque tenemos que tener en cuenta el tipo de dato, keyword elimina duplicados, por lo que si indexamos un documento con campo X que contiene valores [“a”, “b”, “a”] el resultado será [“a”, “b”], mientras que si usamos por ejemplo un campo de tipo long, “x”: [“1”,“2”,“1”] se mantendrán los duplicados.

Procesando datos con GROK y DISSECT

Cuando los datos con los que trabajamos contiene datos no estructurados, podemos usar GROK y DISSECT para analizarlos, como por ejemplo:



Elasticsearch puede estructurar los datos en tiempo de indexado o en tiempo de query, en ambos podemos usar los conocidos GROK y DISSECT.

Elegir cual de los dos usaremos dependerá de nuestros datos y el tipo de análisis que queramos hacer, unas veces será mucho mas eficiente DISSECT si nuestros datos están delimitados o usaremos GROK si queremos usar expresiones regulares.

GROK es el mas potente pero también es el mas lento
DISSECT trabaja bien cuando el dato es repetitivo (sin excepciones)

```
ROW a = "2023-01-23T12:15:00.000Z -  
some text - 127.0.0.1"  
| DISSECT a "%{date} - %{msg} - %{ip}"  
| KEEP date, msg, ip
```

 date	 msg	 ip
2023-01-23T12:15:00.000Z	some text	127.0.0.1

Otro ejemplo donde tratamos la fecha:

```
ROW a = "2023-01-23T12:15:00.000Z -  
some text - 127.0.0.1"  
| DISSECT a "%{date} - %{msg} - %{ip}"  
| KEEP date, msg, ip  
| EVAL date = TO_DATEETIME(date)
```

 msg	 ip	 date
some text	127.0.0.1	2023-01-23T12:15:00.000Z

Y en el otro lado tenemos a GROK:

```
ROW a = "1.2.3.4
[2023-01-23T12:15:00.000Z] Connected"
| GROK a "%{IP:ip} \\[%{TIMESTAMP_ISO8601:@timestamp}\\] %{GREEDYDATA:status}"
```

Document

```
a 1.2 @timestamp 2023-01-23T12:15:00.000Z ip 1.2.3.4 status Connected
```

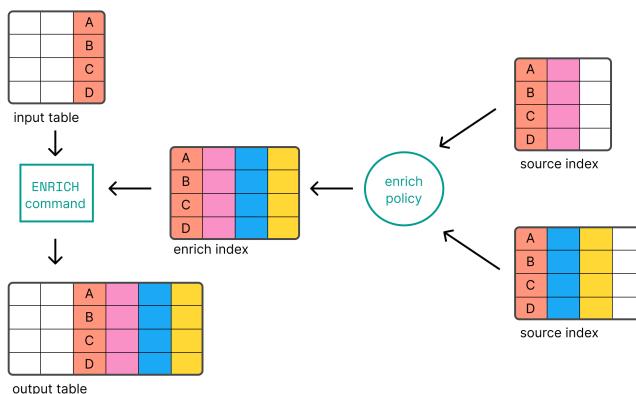
Tenemos unas limitaciones del uso de GROK con ES|QL, no se puede usar “custom patterns” o “múltiple patterns” a parte de que el mismo no esta sujeto al “GROK Watchdog” que interrumpe la ejecución del GROK si este sobrepasa un determinado tiempo.

Aunque podamos realizar procesado de información en tiempo de query, siempre es bueno, una vez se haya identificado el alcance y los patrones, procesar o preprocesar los datos en el momento de ingesta.

Enriqueciendo datos

Como hemos visto anteriormente tenemos un comando de procesamiento que nos permite enriquecer datos en tiempo de query “ENRICH” y ahora vamos a verlo mas en profundidad.

El enriquecimiento de datos es un proceso por el cual en base a unos campos previamente identificados, juntamos datos de diferentes indices:



Política de enriquecimiento

En la política definimos uno o mas indices de origen, de los cuales tomaremos los datos, se determina como se emparejaran los datos que sean procesados por esta política y los campos del indice de origen que se añadirán en el enriquecimiento

```
1 PUT /_enrich/policy/users-policy
2 {
3   "match": {
4     "indices": "users",
5     "match_field": "email",
6     "enrich_fields": ["first_name", "last_name", "city", "zip", "state"]
7   }
8 }
```

Indice origen

Es un índice convencional o regular, lo llamamos origen por el hecho de ser el que contiene los datos que añadiremos al procesado de información.

Indice de enriquecimiento

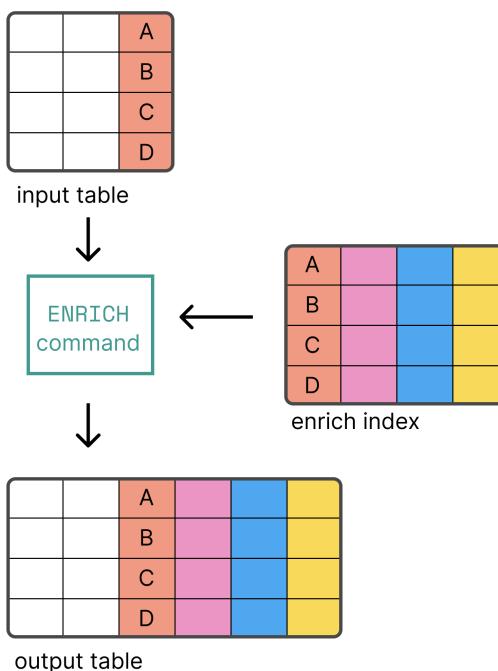
Este es un índice especial de sistema que se genera cuando ejecutamos la política de enriquecimiento, siempre empieza por ".enrich-*" es de solo lectura por lo que no se puede modificar directamente y sobre el que se ha

realizado un “force merge” para mejorar los tiempos de query.

Si efectuamos algún cambio o update en el índice de origen y queremos que se refleje en el enriquecimiento, deberemos volver a ejecutar la política de enriquecimiento para que estos cambios se reflejen en el índice de enriquecimiento.

Uso de la política de enrich

Después de definir y ejecutar la política de enriquecimiento, ya podemos usarla:



Y también la podemos definir y crear desde Kibana cuando necesitemos usarla:



A screenshot of the Kibana Dev Tools interface. The top bar shows the title 'Dev Tools'. Below it is a code editor containing the following Elasticsearch query:

```
1 FROM kibana_sample_data_logs
2 | ENRICH |
3 |STATS | No available policy Click to create
4 | SORT total_bytes DESC
5 | LIMIT 3
```

The status bar at the bottom indicates '5 lines' and '@timestamp detected'. On the right, there are buttons for 'Run query' and 'Enter'.

Create enrich policy

Specify how to retrieve and enrich your incoming data.

Configuration

Policy name

Policy type

Source indices

[Upload a file](#)

 [X](#)

Query (optional)

```
{  
}  
}
```

Defaults to: [match_all](#) query.

[Next >](#)

Usando ES|QL

Tenemos diferentes formas de usar ES|QL, desde la REST API, Discovery de Kibana, Sección de Seguridad de Kibana y a través de varios clústeres.

ES|QL query API

Podemos usarlo desde la REST API como vemos en el siguiente ejemplo:

```
1 POST /_query?format=txt
2 {
3   "query": "FROM library | KEEP author, name, page_count, release_date |
4             SORT page_count DESC | LIMIT 5"
4 }
```

También desde el “Dev Tools” de Kibana:

```
1 POST /_query?format=txt
2 {
3   "query": """
4     FROM library
5     | KEEP author, name, page_count, release_date
6     | SORT page_count DESC
7     | LIMIT 5
8   """
9 }
```

Incluso podemos aplicar filtros propios de “Query DSL” limitando así el alcance o set de documentos donde se ejecutará ES|QL:

```
1 POST /_query?format=txt
2 {
3   "query": """
4     FROM library
5     | KEEP author, name, page_count, release_date
6     | SORT page_count DESC
7     | LIMIT 5
8   """,
9   "filter": {
10     "range": {
11       "page_count": {
12         "gte": 100,
13         "lte": 200
14       }
15     }
16   }
17 }
```

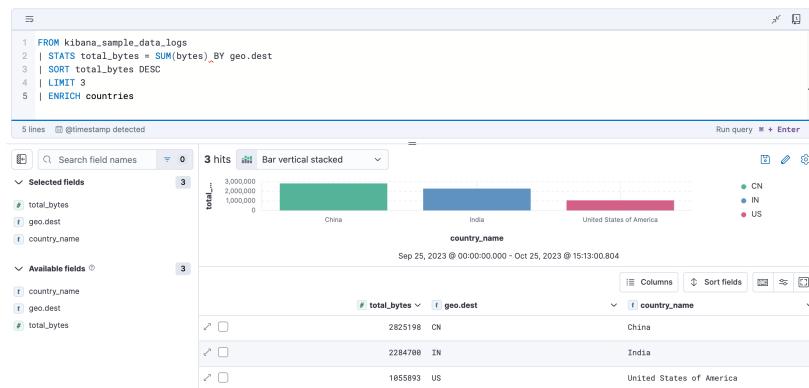
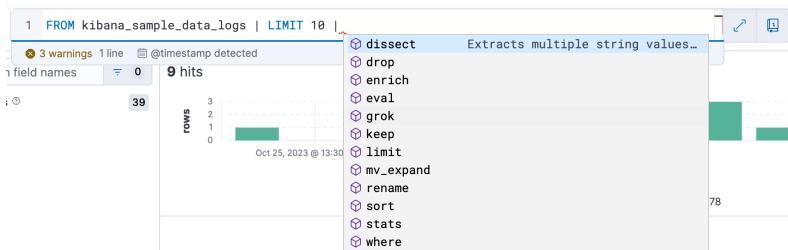
Y también podemos pedirle que nos devuelva los datos en columnas, definir el “locale”, pasara parámetros y ejecutar queries asíncronas.

Usando ES|QL en Kibana

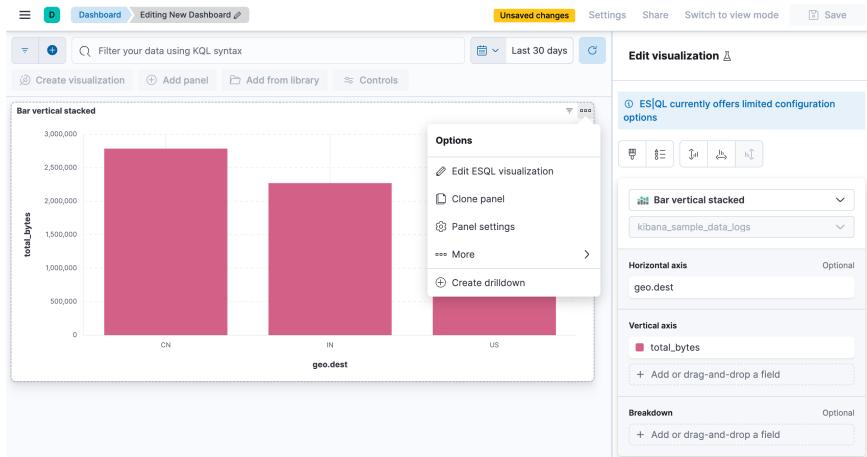
Desde Discovery podemos indicar en la sección del “Data view” a utilizar, que queremos usar ES|QL:

The screenshot shows the Kibana interface with the title "Kibana Sample Data Logs". In the top right, there are three small icons: a gear, a magnifying glass, and a refresh symbol. Below the title, there are two buttons: "Add a field to this data view" and "Manage this data view". Under the title, there is a section titled "Data views" with a search bar and a "Create a data view" button. A list of existing data views is shown, with "Kibana Sample Data Logs" highlighted in blue. At the bottom of the list is a button labeled "Try ES|QL Technical preview".

Y de esta forma usar la barra de query para lanzar nuestras queries de ES|QL



Así como generar gráficos / visualizaciones para usarlas posteriormente en cuadros de mando “Dashboards”



Gestionando ES|QL

Es posible que en algún momento hagamos queries que estén consumiendo demasiado recursos os sean demasiado lentas o tarden demasiado en resolver resultados y necesitemos cancelar peticiones que estén en vuelo.

Para ello usaremos el gestor de tareas como veremos a continuación:

```
1 GET /_tasks?pretty&detailed&group_by=parents&human&actions=*data/read/esql
```

Esto nos devolverá algo como:

```
1 {  
2   "node" : "2j8UKw1bR0283PMwDugNNg",  
3   "id" : 5326,  
4   "type" : "transport",  
5   "action" : "indices:data/read/esql",  
6   "description" : "FROM test | STATS MAX(d) by a, b",  
7   "start_time" : "2023-07-31T15:46:32.328Z",  
8   "start_time_in_millis" : 1690818392328,  
9   "running_time" : "41.7ms",  
10  "running_time_in_nanos" : 41770830,  
11  "cancellable" : true,  
12  "cancelled" : false,  
13  "headers" : {}  
14 }
```

Y podremos actuar sobre esa tarea como con cualquier otra de la siguiente forma:

```
1 POST _tasks/2j8UKw1bR0283PMwDugNNg:5326/_cancel
```

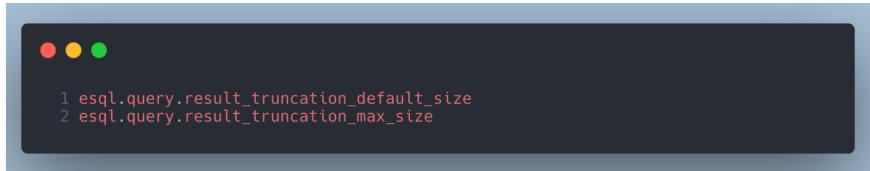
Limitaciones

Existen varias limitaciones, muchas de ellas heredadas como la limitación de la cantidad de datos devueltos, por defecto 1000 y como máximo 10.000, no siendo aplicables estas limitaciones cuando efectuamos agregaciones que se ejecutan sobre todo el set de datos.

Para hacer un uso correcto y evitar llegar a esta limitación en tanto en cuanto al límite de datos, nos centraremos en el uso de “WHERE” y “STATS ... BY”

Aunque podemos cambiar el valor por defecto de forma dinámica con los siguientes parámetros, es siempre

recomendable estudiar el caso de uso para evitar una perdida en el rendimiento.



```
1 esql.query.result_truncation_default_size  
2 esql.query.result_truncation_max_size
```

Tipos de campos soportados

En el tiempo de edición de este handbook los tipos de campos soportados son:

- alias
- boolean
- date
- double (float, half_float, scale_float representados como double)
- ip
- keyword
- int (sort, byte representados como int)
- long
- null
- text
- unsigned_long
- version
- geo_point
- geo_shape
- point
- shape

Tipos de campos no soportados

En el tiempo de edición de este handbook los tipos de campos no soportados son:

- TBS metrics:
 - counter
 - position
 - aggregate_metric_double
- Date/time:
 - date_nanos
 - date_range
- Other types:
 - binary
 - completion
 - dense_vector
 - double_range
 - flattened
 - float_range
 - histogram
 - integer_range
 - ip_range
 - long_range
 - nested
 - rank_feature
 - rank_features
 - search_as_you_type

Ejemplos

A continuación vamos a ver varios ejemplos y en capítulos posteriores tendremos recursos para el día a día usando ES|QL

Agregando y enriqueciendo registros de evento de Windows:

```
● ● ●  
1 FROM logs-*  
2 | WHERE event.code IS NOT NULL  
3 | STATS event_code_count = COUNT(event.code) BY event.code,host.name  
4 | ENRICH win_events ON event.code WITH event_description  
5 | WHERE event_description IS NOT NULL and host.name IS NOT NULL  
6 | RENAME event_description AS event.description  
7 | SORT event_code_count DESC  
8 | KEEP event_code_count,event.code,host.name,event.description
```

Sumando el trafico saliente de un proceso específico (curl.exe)

```
● ● ●  
1 FROM logs-endpoint  
2 | WHERE process.name == "curl.exe"  
3 | STATS bytes = SUM(destination.bytes) BY destination.address  
4 | EVAL kb = bytes/1024  
5 | SORT kb DESC  
6 | LIMIT 10  
7 | KEEP kb,destination.address
```

Procesando los registros DNS para encontrar un elevado número de consultas dns únicas por dominio registrado.

```
● ● ●  
1 FROM logs-*  
2 | GROK dns.question.name "%{DATA}\\.%  
  {GREEDYDATA:dns.question.registered_domain:string}"  
3 | STATS unique_queries = COUNT_DISTINCT(dns.question.name) BY  
  dns.question.registered_domain, process.name  
4 | WHERE unique_queries > 10  
5 | SORT unique_queries DESC  
6 | RENAME unique_queries AS `Unique Queries`, dns.question.registered_domain  
  AS `Registered Domain`, process.name AS `Process`
```

Identificación de un elevado número de conexiones salientes de usuarios

```
● ● ●  
1 FROM logs-*  
2 | WHERE NOT CIDR MATCH(destination.ip, "10.0.0.0/8", "172.16.0.0/12",  
  "192.168.0.0/16")  
3 | STATS destcount = COUNT(destination.ip) BY user.name, host.name  
4 | ENRICH ldap_lookup_new ON user.name  
5 | WHERE group.name IS NOT NULL  
6 | EVAL follow_up = CASE(destcount >= 100, "true","false")  
7 | SORT destcount DESC  
8 | KEEP destcount, host.name, user.name, group.name, follow_up
```

Recursos

En esta sección vamos a ver varios recursos que os ayudarán en el uso de ES|QL

GitHub de ejemplos prácticos:

<https://github.com/elastic/elasticsearch/tree/main/x-pack/plugin/esql/qa/testFixtures/src/main/resources>

Entorno de demo de ES|QL:

<https://esql.demo.elastic.co>

ES|QL Quick Reference

