



Curso de

Introducción a ElasticSearch

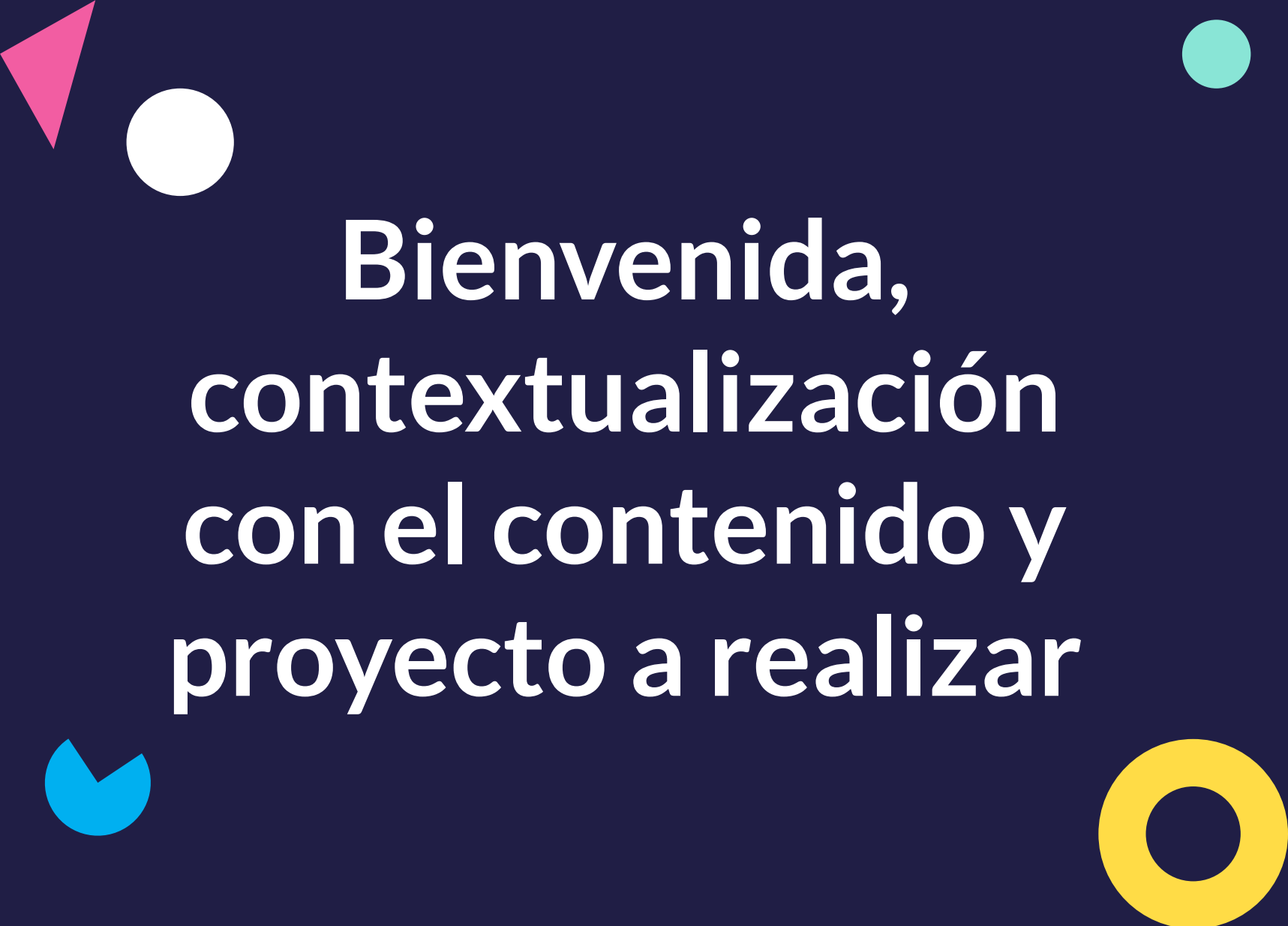
Kevin Sarmiento





Bienvenida al Curso

Conceptos básicos para usar
ElasticSearch



**Bienvenida,
contextualización
con el contenido y
proyecto a realizar**

Kevin Sarmiento

- Desde 2011 trabajo en el mundo del Software.
(90% en el ámbito Web)
- Viajes, Cinefilia, Excursionismo, Senderismo.
- Charlas y Conferencias.

ElasticSearch



Curso 100% práctico
Experiencias con la herramienta en la
industria de pedidos de comida a domicilio

Proyecto



Requerimientos

- *[Requerido]* Conocimientos básicos de **Postman** ya que la herramienta será usada durante todo el curso.
- *[Deseable]* Haber trabajado previamente con cualquier tipo de base de datos.
- *[Deseable]* Conocimientos básicos de **Docker**.



¡Te doy la bienvenida!





¿Qué es ElasticSearch?

Conceptos básicos para usar
ElasticSearch

ElasticSearch

- Elasticsearch provee búsquedas y analítica en **tiempo real**.
- Luego de guardado un documento, es indexado y buscable en **casi tiempo real** (± 1 segundo, usar **?refresh**).
- Interfaz **HTTP** con documentos **JSON**
GET, POST, PUT, DELETE, etc.
- Guardado e indexación eficiente,
la búsqueda es lo más rápida posible.
- Naturaleza distribuida.

Casos de uso

- Búsqueda de información en una app o sitio web
- Motor de almacenamiento para automatizar flujos de negocio
- *Machine Learning* para modelar comportamiento de datos
- Manejar información geoespacial usando Elasticsearch como un GIS

Índice

Documento 1

Documento 2

Documento 3

Documento 4



Cluster

Nodo 1

Primaria 1

Réplica 3

Réplica 2

Nodo 2

Primaria 2

Réplica 1

Réplica 3

Nodo 3

Primaria 3

Réplica 2

Réplica 1





Índices y Documentos

Indexación de datos

Vamos a Postman...



Verbos HTTP

Indexación de datos

Vamos a Postman...



Mapeo de Datos

Indexación de datos

Mapeo de Datos

- Para rendimiento óptimo *indicar un mapeo explícito*
- Para texto: **text** y **keyword**
text: búsquedas de texto completo (*match*)
keyword: valores exactos (*term*)

Ejemplo: platos.estado

- Estados: *activo, pendiente, inactivo*
- Si campo estado es **keyword**
buscar 'activ' no genera resultados
- Si campo estado es **text**
buscar 'activ' retorna activo e inactivo

Tipos de Datos

- Texto: text, keyword.
- Fechas: date.
- Números: integer, long, float, double.
- Booleanos: boolean.
- Objetos: object, nested.
- Geográficos: geo_point, geo_shape.

Vamos a Postman...



Puntaje

Indexación de datos

Puntaje

- Qué tan bien coincide un documento con la búsqueda
- Algoritmo verifica:
de ocurrencias / unicidad de las palabras
- Los resultados vienen ordenados por defecto usando dicho puntaje

Vamos a Postman...



Tipos de Cláusulas

Consultas

Cláusulas

- Must, Filter, Should y Must Not
- Para una sola consulta:
Usar {} (un objeto)
- Para más de una consulta:
Usar [] (una lista de objetos)

Must

- AND lógico
- La consulta debe aparecer en los documentos retornados
- Influye en el puntaje

Filter

- AND lógico también
- La consulta debe aparecer en los documentos retornados
- No Influye en el puntaje
- Usar Filter en vez de Must
Si no nos interesa el puntaje
- Permite caché

Should

- OR lógico
- Alguna de las consultas debería aparecer en los documentos retornados
- Influye en el **puntaje**
- **minimum_should_match** permite alterar el comportamiento

Cuántas consultas deben coincidir

Should

- Si Must o Filter (**AND**) están presentes en la consulta booleana
minimum_should_match=0 (defecto)
(Should se vuelve opcional)
- De lo contrario
minimum_should_match=1 (defecto)
(Al estar sola al menos una condición debe cumplirse)

Must Not

- NOT lógico
- La consulta **no** debe aparecer en los documentos retornados
- No Influye en el **puntaje**
- Permite caché


```
{
  "query": {
    "bool": {
      "must": {} ó [],
      "filter": {} ó [],
      "must_not": {} ó [],
      "should": {} ó [],
      "minimum_should_match": (sí aplica)
    }
  }
}
```



Consultas Booleanas

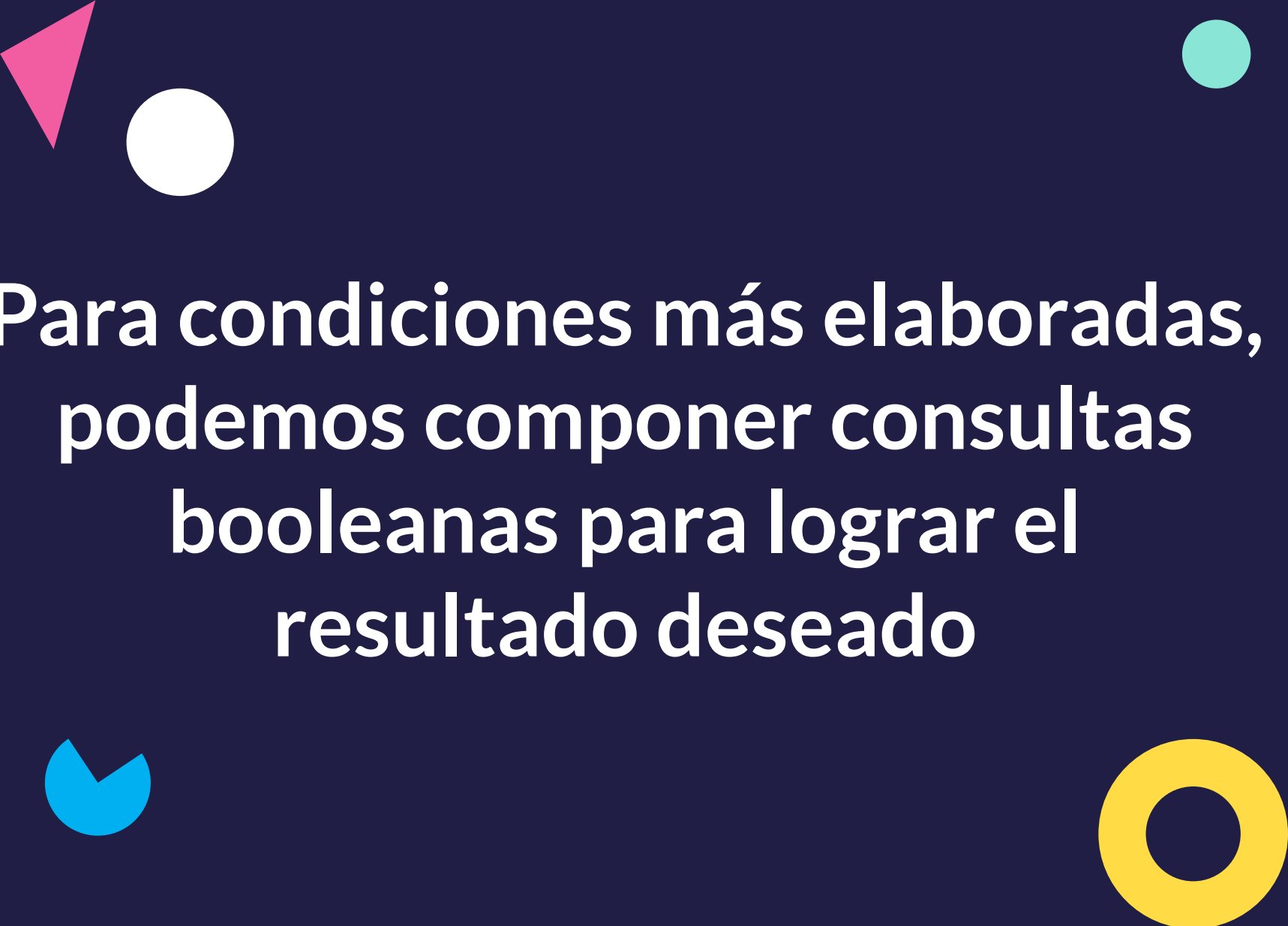
Consultas

Vamos a Postman...



Consultas Compuestas

Consultas



**Para condiciones más elaboradas,
podemos componer consultas
booleanas para lograr el
resultado deseado**

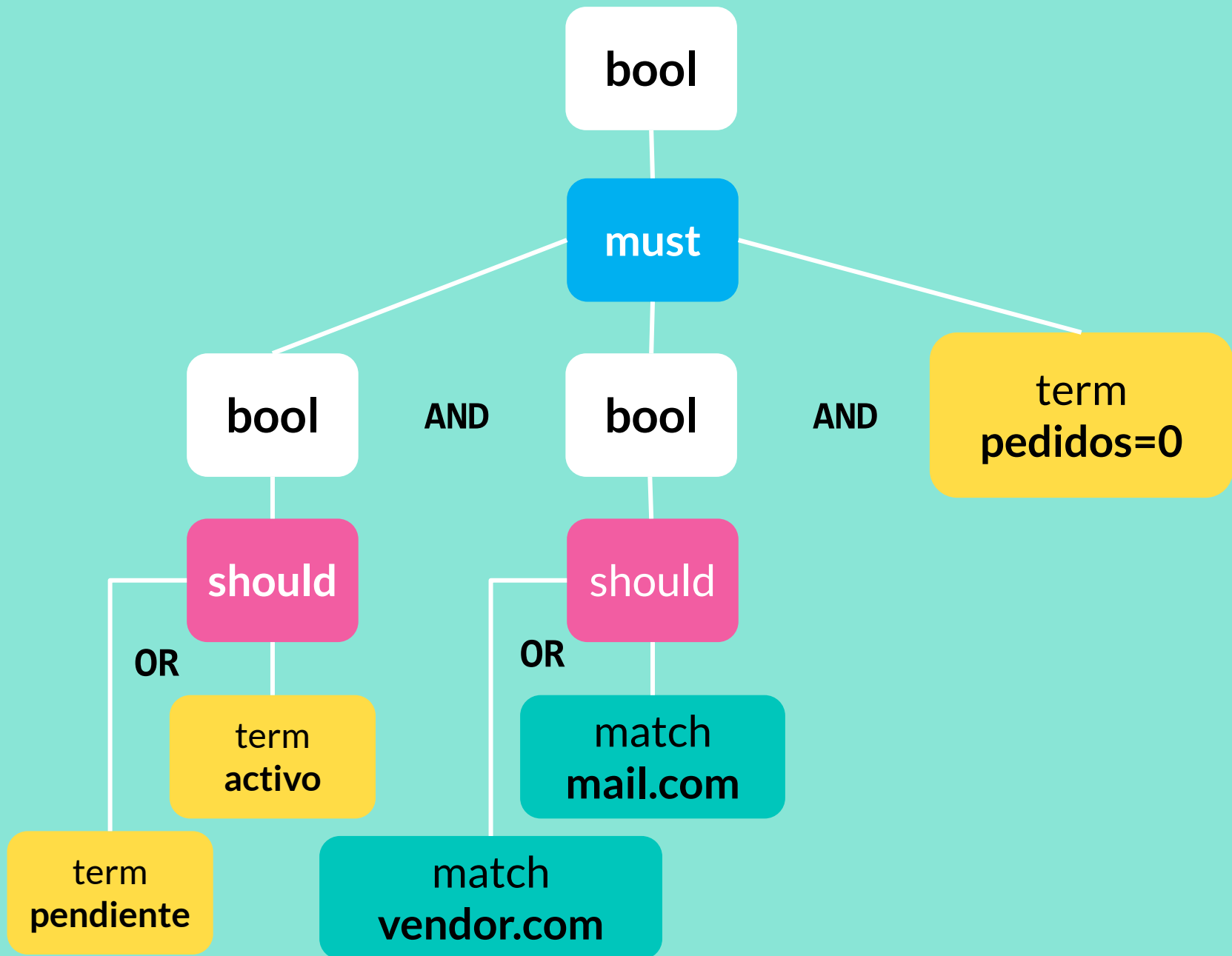
Representemos la siguiente condición...

```
m1 = mail.com
```

```
m2 = vendor.com
```

```
if estado=(activo or pendiente) and  
    ultimaModificacion.usuario like (m1 or m2) and  
    pedidosUltimaHora=0
```

...en Elasticsearch



```
{
  "query": {
    "bool": {
      "must": [
        {
          "bool": {
            //estado activo o pendiente
          }
        },
        {
          "bool": {
            //usuario mail.com o vendor.com
          }
        },
        {
          "term": {
            //pedidos última hora en cero
          }
        }
      ]
    }
  }
}
```




Construyendo una Consulta Compuesta

Consultas

Vamos a Postman...



Consultas Anidadas

Consultas

Anidación

- Guardar una **lista de objetos** dentro de un documento (pertenencia)
- Esta consulta encuentra documentos usando sus **objetos anidados**
- Al encontrar una coincidencia el **documento raíz** es devuelto

Vamos a Postman...



Proyecto: Unificación de Datos

Consultas

Vamos a Postman...



Consultas de Rango y Agregaciones

Consultas

Vamos a Postman...



Proyecto: Revisión final del directorio

Cierre

Vamos a Postman...



Notas Finales

Cierre

Notas Finales (1)

- Tienes un modelo de restaurantes que incluye platos disponibles, una lista de categorías, datos básicos y calificaciones.
- Conoces el funcionamiento de las distintas consultas para filtrar el directorio al detalle.

Notas Finales (2)

- Aprendiste a **calcular métricas** para ver el comportamiento del directorio.
- Dispones del **puntaje** para ordenar los resultados por relevancia.
- Solo te queda integrar esta poderosa herramienta con **tus proyectos...**

Cientes Oficiales

- Java, Javascript, Ruby, Go, .NET, PHP, Perl, Python, entre otros...
- Puedes crear un cliente propio haciendo consultas HTTP a ElasticSearch directamente desde el lenguaje.

```
from datetime import datetime
from elasticsearch import Elasticsearch
es = Elasticsearch()

doc = {
    'author': 'kimchy',
    'text': 'Elasticsearch: cool. bonsai cool.',
    'timestamp': datetime.now(),
}
res = es.index(index="test-index", id=1, body=doc)
print(res['result'])

res = es.get(index="test-index", id=1)
print(res['_source'])

es.indices.refresh(index="test-index")

res = es.search(index="test-index", body={"query": {"match_all": {}}})
print("Got %d Hits:" % res['hits']['total']['value'])
for hit in res['hits']['hits']:
    print("%(timestamp)s %(author)s: %(text)s" % hit["_source"])
```




[FIN]

