# Comprehensive Guide to Elasticsearch Data Types

## 1. Introduction to Elasticsearch Data Types

Elasticsearch supports a wide variety of data types to cater to different use cases. These data types define how data is stored, indexed, and queried.

Basic data types include common ones like integers, floats, strings, and booleans. Specialized data types, such as `ip` for IP addresses or geospatial types for location-based queries, extend the functionality of Elasticsearch. This document explores these data types and their applications.

## 2. Basic Data Types

Elasticsearch provides several basic data types that are analogous to those found in programming languages. Examples include:  
- `integer`, `long`, `short`, `byte`: Numeric types for whole numbers.  
- `float`, `double`: Types for decimal numbers.  
- `boolean`: For true/false values.  
- `date`: To store date values.

Apart from these, Elasticsearch also supports specialized data types like `ip` for IP addresses, and others designed for features like auto-completion and geospatial searches.

## 3. Complex Data Types - Object

1. The `object` data type represents JSON objects, which are fundamental to Elasticsearch.

2. Documents in Elasticsearch are JSON objects and can contain nested objects to form a hierarchy.

3. Mapping for objects uses a `properties` key to define fields and their types, rather than a `type` key.

4. Example:  
{  
 "product": {  
 "name": "Example",  
 "details": {  
 "weight": "1kg",  
 "dimensions": {  
 "length": "10cm",  
 "width": "5cm"  
 }  
 }  
 }  
}

Internally, objects are flattened by Elasticsearch to ensure compatibility with Apache Lucene, using dot notation for hierarchy.

## 4. Challenges with Arrays of Objects

1. When indexing arrays of objects, values are grouped by field name, causing the relationships between fields to be lost.

2. This can lead to unexpected query results, as the boolean AND condition may behave like a boolean OR.

3. Example:  
Searching for products with a review by 'John Doe' and a rating >= 4 may incorrectly match products where the relationship is not preserved.

## 5. Nested Data Type

1. The `nested` data type solves the issue of lost relationships in arrays of objects by storing each object as a separate hidden document.

2. Example Mapping with Nested Data Type:  
PUT /products  
{  
 "mappings": {  
 "properties": {  
 "reviews": {  
 "type": "nested",  
 "properties": {  
 "name": { "type": "text" },  
 "rating": { "type": "double" }  
 }  
 }  
 }  
 }  
}

3. This ensures accurate querying, maintaining relationships between object fields.

## 6. Keyword Data Type

1. The `keyword` data type is designed for fields that require exact matches, such as email addresses, order statuses, and tags.

2. This data type supports filtering, sorting, and aggregations but not full-text searches.

3. Example Mapping:  
PUT /articles  
{  
 "mappings": {  
 "properties": {  
 "status": { "type": "keyword" }  
 }  
 }  
}

4. The `keyword` analyzer ensures that field values are stored unaltered, enabling exact matches.

## 7. Text Data Type

1. The `text` data type is used for full-text searches, allowing partial matches and relevance-based ranking.

2. Example: Searching for articles containing specific words within their content.

3. The distinction between `text` and `keyword` will be explored further in later sections.

## 8. Summary

1. Elasticsearch provides a variety of data types to cater to different use cases, from basic types like `integer` and `boolean` to complex ones like `object` and `nested`.

2. Specialized types like `keyword` and `text` address exact matches and full-text searches, respectively.

3. Understanding these data types and their behavior is crucial for designing efficient and accurate Elasticsearch indices.