# 

# Comprehensive Guide to Arrays in Elasticsearch

## 1. Introduction

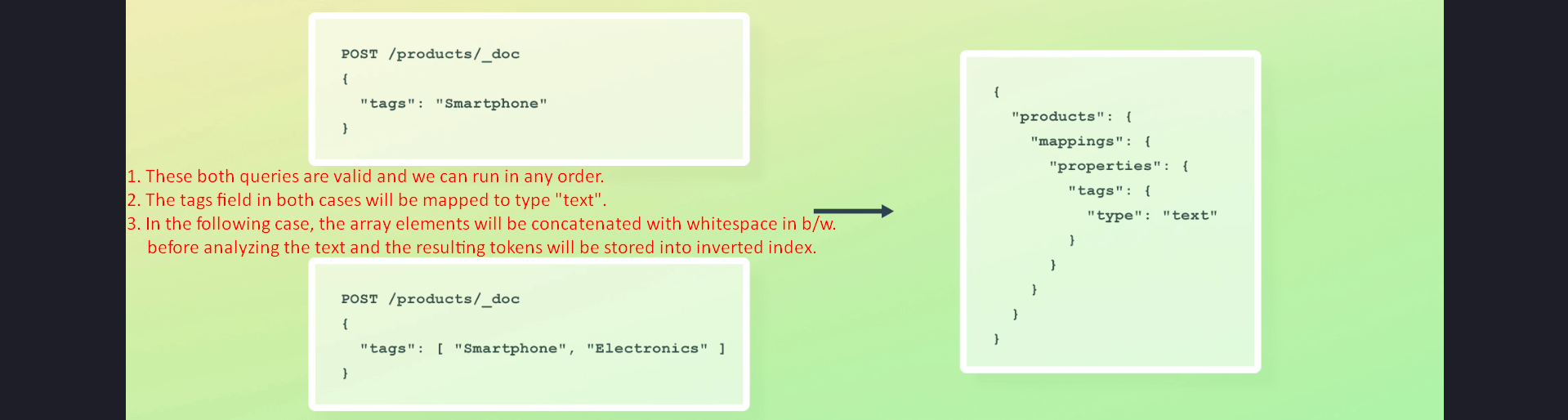
In Elasticsearch, fields can contain multiple values by default, even though there is no explicit 'array' data type. This flexibility allows us to index arrays of values without requiring special configurations.

For instance, in the previous section, we indexed a 'tags' field with an array of values for products. Elasticsearch automatically handled the array without needing specific mappings for multiple values.

## 2. Storing Multiple Values

1. Any field in Elasticsearch can hold zero or more values by default, regardless of the data type.

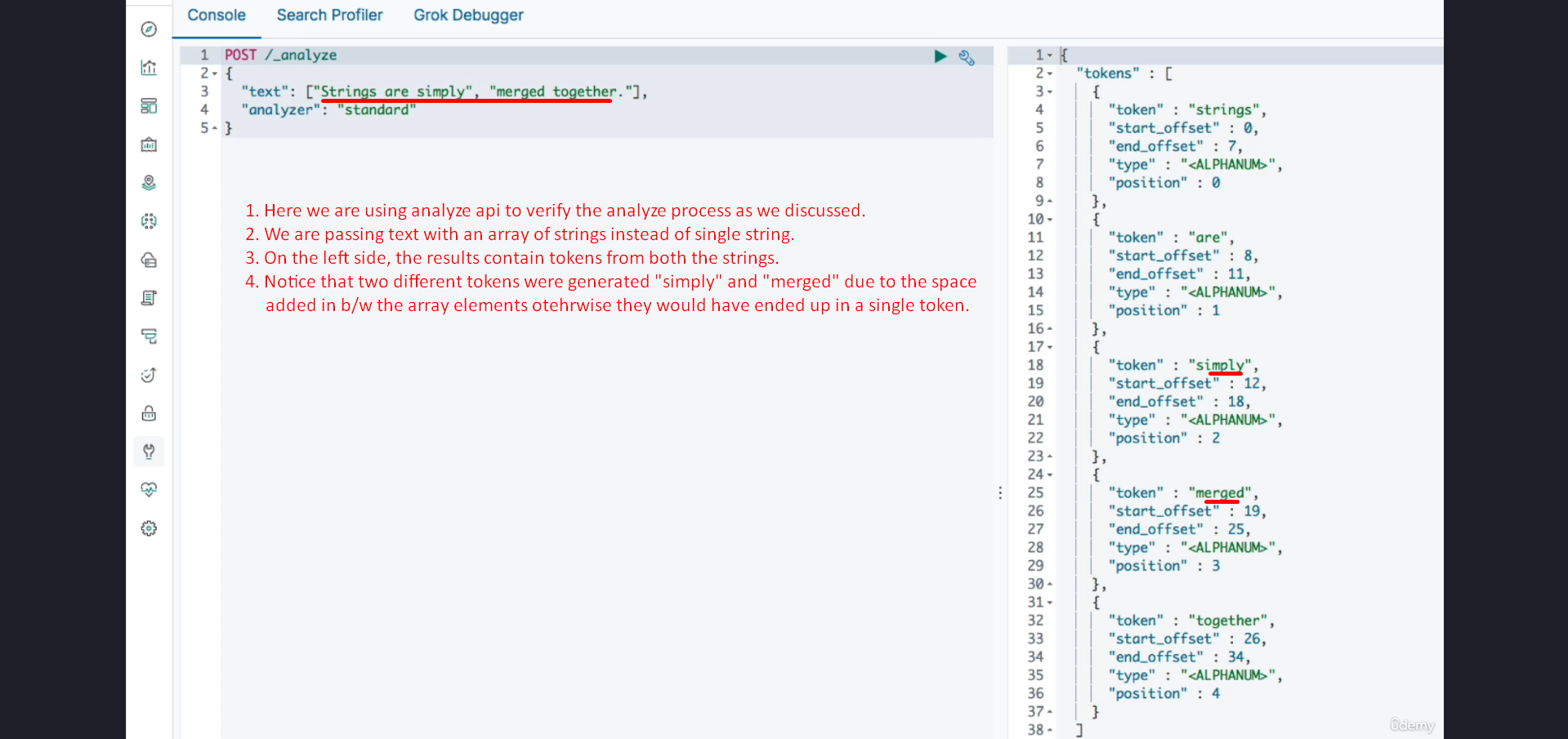
2. Example Mapping:  
PUT /example\_index  
{  
 "mappings": {  
 "properties": {  
 "tags": { "type": "text" }  
 }  
 }  
}

3. Internally, text values in arrays are concatenated with a space in between, analyzed, and stored in an inverted index. Non-text fields, on the other hand, store multiple values within appropriate data structures in Apache Lucene.  


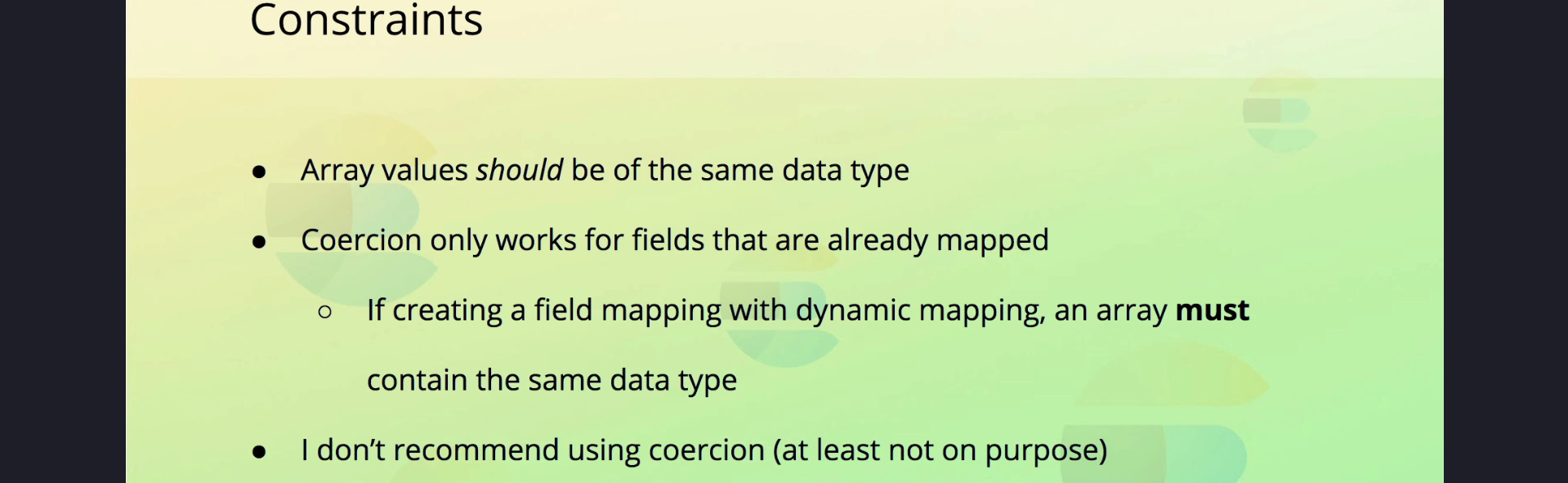
## 3. Analyze API Example

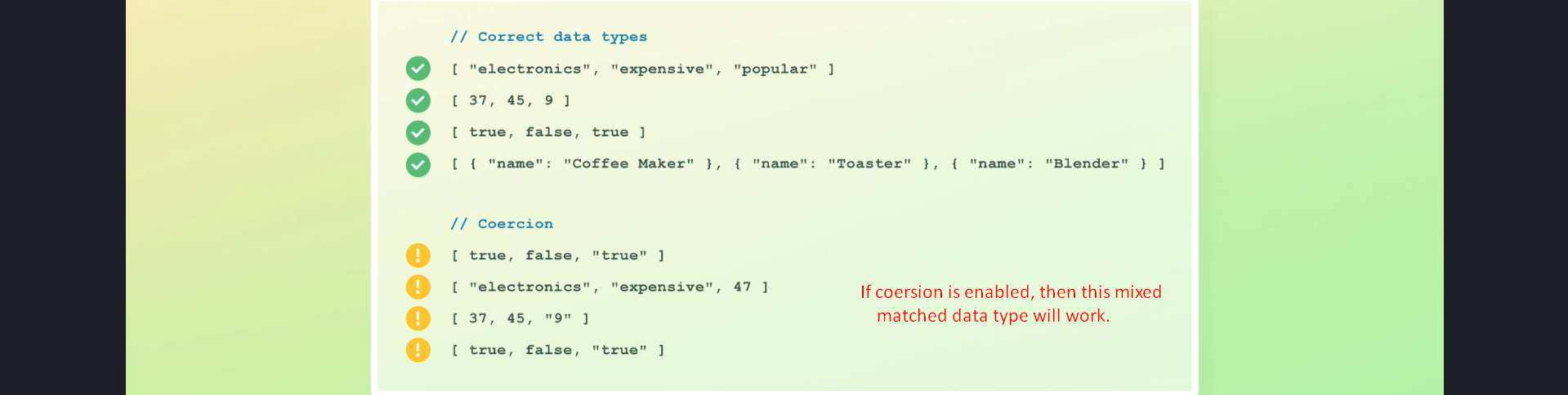
Using the Analyze API, we can observe how text values from arrays are processed. Here is an example where an array of two strings is analyzed:

POST /\_analyze  
{  
 "analyzer": "standard",  
 "text": ["simply", "merged"]  
}

The resulting tokens contain words from both strings, with character offsets continuing from the last offset of the previous string.  
  
Basically the strings in the array are treated as single value not multiple values.

## 4. Constraints and Coercion

1. All values within an array must be of the same data type. Mixing data types is generally not allowed.  


2. Exceptions exist if values can be coerced into the data type defined in the mapping. For example:  
- ["1.5", 2.0] (strings can be coerced into floats)  
- ["true", false] (strings can be coerced into booleans)  
  


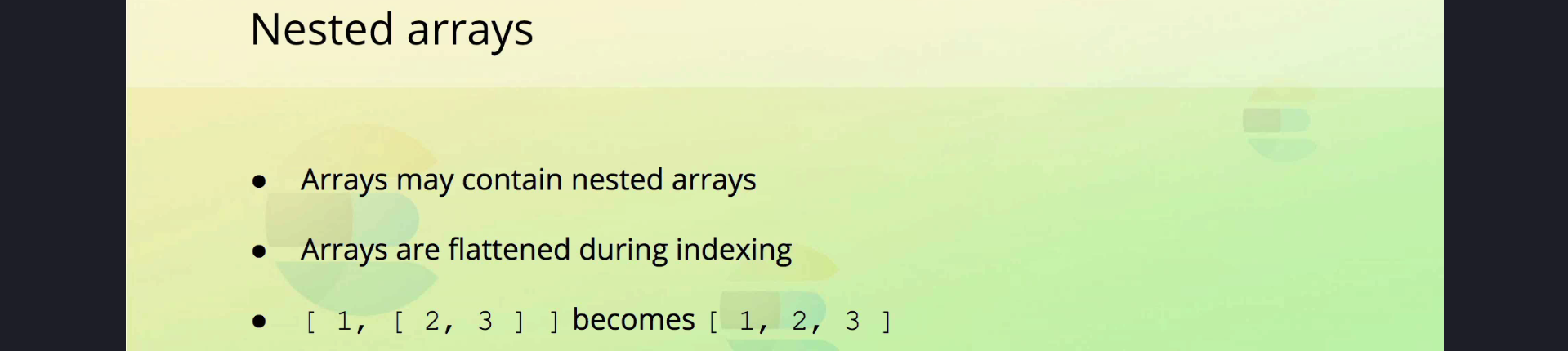
3. If coercion is disabled or if the values cannot be coerced (e.g., mixing objects with booleans), Elasticsearch will return an error.

4. Coercion of array values is only supported after a mapping has been created. If dynamic mapping is used to create a field, you cannot supply mixed data types in the initial indexing.

## 5. Nested Arrays

1. Elasticsearch flattens nested arrays, moving all nested values to the top level.

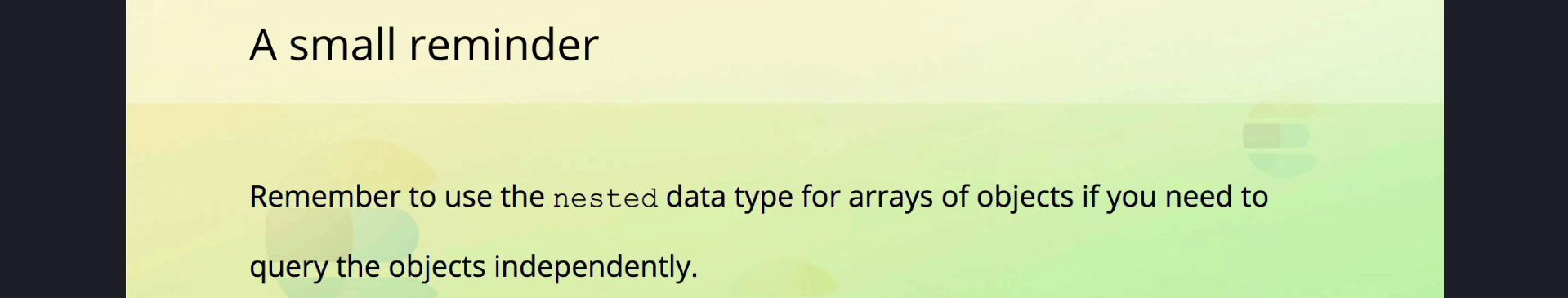
2. Example of Nested Array:  
[  
 [1, 2],  
 [3, 4]  
]

3. Flattening ensures that the array is stored in a consistent format, but this may not preserve the original structure.  


## 6. Arrays of Objects

1. If you need to query objects within an array independently, use the 'nested' data type for the field.

2. If independent querying is not required, the 'object' data type is sufficient.



## 7. Summary

1. Elasticsearch allows fields to contain multiple values without needing an explicit 'array' data type.  
2. Text arrays are concatenated and analyzed, while non-text arrays are stored in suitable Lucene structures.  
3. Coercion enables some flexibility with mixed data types, but it is subject to strict rules and should be used cautiously.  
4. Nested arrays are flattened, and arrays of objects require careful mapping depending on query requirements.