Keys Project - SDE 3/PE

Design a distributed key-value store that can perform the following

Functional requirements

- Store a set of attributes (value) against a particular key (k)
  Fetch the value stored against a particular key (k)
  Delete a key (k)
  Stretch - Perform a secondary index scan to fetch all key along with their attributes where one of the attribute values is v.

Example:

key	value							
	Lati tude	Longi tude	pollution_ level	free_food	capital	population	category	manufac turer
delhi			very high			10 Million		
jakarta	- 6.0	106	high					
bangalore	12.94	77.64	moderate	true				
india					delhi	1.2 Billion		
crocin							Cold & flu	GSK

Key can have a value consisting of multiple attributes.

Each attribute will have name, type associated (primitive types - boolean, double, integer, string) & type has to be identified at run time.

1) Key = delhi has 2 attributes only (pollution_level & population)
2) Key = jakarta has 3 attributes (latitude, longitude, pollution_level)
3) Key = bangalore has 4 attributes (extra - free_food)
4) Key = india has 2 attributes (capital & population)
5) Key = crocin has 2 attributes (category & manufacturer)

Example of Secondary index:

- Get all keys (cities) where pollution_level is high.
  Get all medicines by manufacturer (GSK)

So, in a nutshell, value must be strongly typed when defined.

Attribute

1. Attribute is uniquely identified by its name (latitude, longitude etc.
2. Data type of the attribute is defined at the first insert. (i.e. data type of pollution_level is set when key = delhi is inserted)

3. Once data type is associated with a particular attribute, it cannot be changed.
   (i.e. free_food when defined takes type = boolean, hence, any key when using the attribute - free_food must allow only boolean values on subsequent inserts/updates)

Non-functional requirements

- Highly scalable - Support for high throughput with very low latency
  Highly available
  Shared nothing architecture i.e. Support for Multiple nodes and each node is independent & self-sufficient.
  Stretch - Smart client i.e. clients being aware of available servers & makes smart routing based on that available information.

Notes

Preferable Language - Java
Follow standard OOPs concept and best practices in the industry
Make sure code is thread safe.
Stub the methods for complex logic with comments supporting your

assumptions.
Write at least 1 test case for each functionality.