Updating TableConfig to include a new object called FieldConfig

This document proposes updating TableConfig to include a new object called FieldConfig to store the per column encoding and index info. The document also proposes to deprecate and eventually remove the other objects in TableConfig (e.g. invertedIndexColumns, bloomFilterColumns, etc.) and move these declarations into FieldConfig

Motivation:

As part of text search feature (PR), we need to add a new set of columns to IndexingConfig : something like TextIndexConfig and capture the information required for text indexes: column name and some additional properties used by the implementation.

During initial discussions it was learnt that we have been wanting to cleanup some parts of TableConfig (specifically IndexingConfig) for quite some time to have a cleaner way of specifying the per column indexing and encoding info.

The plan is to start doing the cleanup incrementally and use the change directly for text search feature to begin with.

Current State:

Today we maintain this information as part of several lists in IndexingConfig (which is a section inside TableConfig).

If you look at the code of TableConfig builder:

```
// Indexing config
IndexingConfig indexingConfig = new IndexingConfig();
if (_sortedColumn != null) {
 indexingConfig.setSortedColumn(Collections.singletonList(_sortedColumn));
}
indexingConfig.setInvertedIndexColumns(_invertedIndexColumns);
indexingConfig.setNoDictionaryColumns(_noDictionaryColumns);
indexingConfig.setOnHeapDictionaryColumns(_onHeapDictionaryColumns);
indexingConfig.setBloomFilterColumns(_bloomFilterColumns);
```

All this information can be captured in a separate object “FieldConfig” which can look something like this:

```
private String _name;
private EncodingType _encodingType;
private IndexType _indexType;
private Map<String, String> _properties;

public static String BLOOM_FILTER_COLUMN_KEY = "field.config.bloom.filter";
public static String ON_HEAP_DICTIONARY_COLUMN_KEY =
"field.config.onheap.dictionary";
public static String VAR_LENGTH_DICTIONARY_COLUMN_KEY =
"field.config.var.length.dictionary";
public static String TEXT_INDEX_REALTIME_READER_REFRESH_KEY =
"field.config.realtime.reader.refresh";

// If null, we will create dictionary encoded forward index
public enum EncodingType {
 RAW,
 DICTIONARY
}

// If null, there won't be any index
public enum IndexType {
 INVERTED,
 SORTED,
 TEXT
}
```

- The above change has been implemented as part of PR https://github.com/apache/incubator-pinot/pull/5006
- With this change, TableConfig will now have a List<FieldConfig> member storing the above information for each column.
- IndexingConfig will still continue to be there since it stores additional information that is not made part of FieldConfig. We are not getting rid of IndexingConfig -- just that some parts of it are being moved to FieldConfig.

Usage

IndexLoadingConfig

- IndexLoadingConfig is built off IndexingConfig and is used by PhysicalColumnIndexContainer (to load the offline segments and their indexes appropriately).
- Similarly IndexLoadingConfig is also used by RealtimeSegmentConfig (to specify the config of newly to-be-created MutableSegment).
- IndexLoadingConfig is a simple POJO (non-persistent) and has APIs that allow its users to check for things like “is inverted index enabled on a column” since it maintains a HashSet for each index type.
- The extractFromTableConfig method in IndexLoadingConfig uses IndexingConfig and builds these sets.
- Since we want the users of IndexLoadingConfig to continue to function in the same manner, extractFromTableConfig method will be changed to work off FieldConfig list (and build its internal state) as opposed to using IndexingConfig.

SegmentGeneratorConfig

- SegmentGeneratorConfig provides a constructor for offline data push that takes in a TableConfig and builds its internal state (invertedIndexColumns, varLengthDictionaryColumns etc).
- This constructor will have to be updated for the reasons highlighted above.

Migration plan

1. Once this change goes in, we will start using it for text search feature immediately allowing the user to specify the text index column info using the FieldConfig model.
2. Eventually our plan is to have users migrate to FieldConfig model to specify the encoding and index info in their TableConfig
3. Deprecate the existing APIs (setInvertedColumns, setNoDictionaryColumns etc) for IndexingConfig.
    3.1. This will be followed by deleting these APIs in some release.
    3.2. Update documentation, guidelines etc to educate users on how to use the new model -- we can do this without first deleting.
4. **Proposal:**
    4.1. Do (1) -- this will require minimal changes to IndexLoadingConfig and SegmentGeneratorConfig (as mentioned in the above two sections) specifically to extract the text index column info from FieldConfig. This PR can go in independently and the change is harmless since right now only text search feature will be relying upon information coming via FieldConfig.
    4.2. Do (3)

4.3. Until 4.1 and 4.2 are done, the only purpose/use of FieldConfig will be in text search. Once 4.1 and 4.2 are done, it should take care of all the new TableConfig that Pinot will write into ZK after the release of this change.

4.4. We implement a tool that fetches existing TableConfig from ZK upon controller restart, converts them into FieldConfig model and writes back to ZK.

4.4.1. Also, introduce a version number in TableConfig.

4.4.2. If this version is set, it means the user's TableConfig is using the new FieldConfig based model.

4.4.3. If the version is not set, update to FieldConfig mode, write it back to ZK.

4.4.4. As part of evolving the TableConfig, if a user fetches the config, they will see a changed one as compared to what they had written earlier. They should continue to abide by the FieldConfig model as per the recommendations we release in (3).

Examples:

The following json shows the field config list for 4 columns with TEXT index, INVERTEX index, RAW index and SORTED index respectively. The TEXT index column have additional properties -- these are attributes that aren't related to the indexing/encoding information but are additional properties of the column (e.g: bloom filter creation, var length dictionary, on heap dictionary etc). The "fieldConfigList" is a member of TableConfig object.

```
fieldConfigList":[
  {
   "name":"text_col",
   "encodingType":"RAW",
   "indexType":"TEXT",
   "properties":{
      "field.config.realtime.reader.refresh":"100",
    }
  },
  {
    "name":"inv_index_col",
    "encodingType":"DICTIONARY",
    "indexType":"INVERTED",
    "properties":null
  },
  {
```

```
    "name":"raw_index_col",
    "encodingType":"RAW",
    "indexType":null,
    "properties":null
  },
  {
    "name":"sorted_index_col",
    "encodingType":"DICTIONARY",
    "indexType":"SORTED",
    "properties":{
      "field.config.var.length.dictionary":"true",
    }
  },
]
```

Earlier users would specify this information in disjoint parts of IndexingConfig inside the following sections:

- invertedIndexColumns
- noDictionaryColumns
- varLengthDictionaryColumns
- bloomFilterColumns

With FieldConfig, everything is self-contained. If user doesn’t specify a fieldConfigList, this will be null inside TableConfig and we will just handle each column in the same internal default manner as we do today.