KIP-131 - Add access to OffsetStorageReader from SourceConnector

Status

Current state: *Accepted (vote thread)*

Discussion thread: *here*

JIRA: ⚠ Unable to render Jira issues macro, execution error.

Motivation

Currently the offsets storage is only accessible from SourceTask to able to initialize properly tasks after a restart, a crash or a reconfiguration request.
To implement more complex connectors that need to track the progression of each task it would helpful to have access to an OffsetStorageReader instance from the SourceConnector.

In that way, it would become possible to implement a background thread that could request a tasks reconfiguration based on source offsets.

This improvement proposal comes from a project that needs to periodically scan directories on a shared storage for detecting and for streaming new files into Kafka.

For instance, here is a straightforward implementation is that could be possible with that feature .

A connector which needs to stream files can use a background-thread to periodically scan directories. When new inputs files are detected a tasks reconfiguration is requested. The connector assigns a file subset to each task.
Each task stores sources offsets for the last sent record. The source offsets data are the size of file, the bytes offset and the bytes size.

A Task becomes idle as soon as all the assigned files are completed.
The connector should be able to track offsets for each assigned file. When all tasks has finished the connector can stop them or assigned new files by requesting tasks reconfiguration.

Finally, another advantage of monitoring source offsets from the connector is to detect slow or failed tasks and if necessary to be able to restart all tasks.

Public Interfaces

We propose to add two new interfaces SourceConnectorContext and SinkConnectorContext (similar to SourceTaskContext and SinkTaskContext). Those methods could be used then to expose specifics methods to SourceConnector and SinkConnector.

```
/**
 * SinkConnectorContext is provided to SinkConnector to allow them to interact with the underlying
 * runtime.
 *
 * This interface can be used to add specifics methods to a SinkConnector.
 */
public interface SinkConnectorContext extends ConnectorContext {
}
```

```
/**
 * SourceConnectorContext is provided to SourceConnector to allow them to interact with the underlying
 * runtime.
 *
 * This interface can be used to add specifics methods to a SourceConnector.
 */
public interface SourceConnectorContext extends ConnectorContext {

    /**
     * Get the OffsetStorageReader for this SourceConnectorContext.
     */
    OffsetStorageReader offsetStorageReader();
}
```

The SourceConnectorContext class will provide access to an OffsetStorageReader.

To expose these context objects, we propose to supplement the existing protected field with a new method context() to the Connector class.

```
public ConnectorContext context() {
    return context;
}
```

This method is then overridden by the SourceConnector and SinkConnector classes to change the return type to be SourceConnectorContext and SinkConnectorContext, respectively.

```
/**
 * SinkConnectors implement the Connector interface to send Kafka data to another system.
 */
public abstract class SinkConnector extends Connector {

    /**
     * <p>
     * Configuration key for the list of input topics for this connector.
     * </p>
     * <p>
     * Usually this setting is only relevant to the Kafka Connect framework, but is provided here for
     * the convenience of Connector developers if they also need to know the set of topics.
     * </p>
     */
    public static final String TOPICS_CONFIG = "topics";

    public SinkConnectorContext context() {
        return (SinkConnectorContext) context;
    }

}
```

```
/**
 * SourceConnectors implement the connector interface to pull data from another system and send
 * it to Kafka.
 */
public abstract class SourceConnector extends Connector {

    @Override
    public SourceConnectorContext context() {
        return (SourceConnectorContext) context;
    }
}
```

Proposed Changes

The changes described above are implemented as GitHub PR 4794.

Compatibility, Deprecation, and Migration Plan

This proposal is backward compatible and binary compatible with earlier versions. The current design requires connector implementations to access the existing `ConnectorContext` object through a protected field in the `Connector` base class. There are no plans to change this access, so existing implementations will continue to work as before without changes.

This proposal ***adds new subinterfaces*** of `ConnectorContext` called `SinkConnectorContext` and `SourceConnectorContext` and ***new methods*** to the `SinkConnector` and `SourceConnector` abstract classes that will allow subclasses to access the specific connector's context without having to cast. Connectors that require this version of the API will be able to optionally use these new methods or even check and cast the `context` field.

Note that the new `SinkConnectorContext` and `SourceConnectorContext` interfaces added here will make it easier in the future to provide new methods to access runtime components.

Rejected Alternatives

This proposal is simple and straightforward, and reflects the patterns used in the API in other areas. Another similar approach was considered but evolved to this design following a discussion. Another possibility of moving the `context` field from `Connector` to `SourceConnector` and `SinkConnector` (with the appropriate type) was ruled out as it would not be as binary-compatible as the current approach.