KIP-828: Add the corresponding validator to the configuration where the validator is missing

Status

Current state: *Under Discussion*

Discussion thread: here

JIRA: *here*

Motivation

Aims to add a corresponding validator to a configuration where a validator is missing.
so that the program finds these incorrect configurations during initialization.

Public Interfaces

Public classes that involve changes

1. ConfigDef
   Add a new validator `NonNullAndEmptyString` to the ConfigDef class.

2. SslClientAuth
   SslClientAuth provides a static method `names()` to get a list of all enumeration constant lowercase names.

3. BrokerSecurityConfigs
   Provides a new validator `SaslEnabledMechanismsValidator` for validating the `sasl.enabled.mechanisms` parameter.

4. CompressionType
   CompressionType provides a static method `names()` to get a list of all enumeration constant lowercase names

The implementation class of AbstractConfig involved in changes

The bold methods and classes mentioned in the table below need to be created.

1. AdminClientConfig

config	validator
security.protocol	in(SecurityProtocol.names().toArray(new String[0]))

2. ConsumerConfig

config	validator
`group.instance.id`	new ConfigDef.NonEmptyString()

`key.deserializer`	new ConfigDef.NonNullValidator()
`value.deserializer`	new ConfigDef.NonNullValidator()
security.protocol	in(SecurityProtocol.names().toArray(new String[0]))

3. ProducerConfig

config	validator
`key.serializer`	new ConfigDef.NonNullValidator()
`value.serializer`	new ConfigDef.NonNullValidator()
security.protocol	in(SecurityProtocol.names().toArray(new String[0]))
compression.type	in(CompressionType.**names()**.toArray(new String[0]))

4. SaslConfigs

config	validator
sasl.mechanism	new ConfigDef.**NonNullAndEmptyString()**

5. MirrorClientConfig

config	validator
security.protocol	in(SecurityProtocol.names().toArray(new String[0]))

6. MirrorConnectorConfig

config	validator
security.protocol	in(SecurityProtocol.names().toArray(new String[0]))

7. MirrorMakerConfig

config	validator
security.protocol	in(SecurityProtocol.names().toArray(new String[0]))

8. WorkerConfig

config	validator
ssl.client.auth	in(SslClientAuth.**names()**.toArray(new String[0]))

9. DistributedConfig

config	validator
security.protocol	in(SecurityProtocol.names().toArray(new String[0]))

10. KafkaConfig

config	validator
sasl.mechanism.controller.protocol	new ConfigDef.**NonNullAndEmptyString()**
authorizer.class.name	new ConfigDef.NonNullValidator()
security.inter.broker.protocol	in(SecurityProtocol.names().toArray(new Array[String](0)):_*)
sasl.mechanism.inter.broker.protocol	new ConfigDef.**NonNullAndEmptyString()**
sasl.enabled.mechanisms	new BrokerSecurityConfigs.**SaslEnabledMechanismsValidator()**
compression.type	in(BrokerCompressionCodec.brokerCompressionOptions:_*)

11. StreamsConfig

config	validator

security.protocol	in(SecurityProtocol.names().toArray(new String[0]))

Proposed Changes

1.ConfigDef.NonNullAndEmptyString

NonNullAndEmptyString

```
public static class NonNullAndEmptyString implements Validator {

        @Override
        public void ensureValid(String name, Object o) {
            if (o == null) {
                // Pass in the string null to avoid the spotbugs warning
                throw new ConfigException(name, "null", "entry must be non null");
            }
            String s = (String) o;
            if (s.isEmpty()) {
                throw new ConfigException(name, o, "String must be non-empty");
            }
        }

        @Override
        public String toString() {
            return "non-null and non-empty string";
        }
    }
```

2.SslClientAuth.names()

SslClientAuth.names()

```
    public static final List<SslClientAuth> VALUES;
    private static final List<String> NAMES;

    static {
        SslClientAuth[] sslClientAuths = SslClientAuth.values();
        List<String> names = new ArrayList<>(sslClientAuths.length);
        for (SslClientAuth auth : sslClientAuths) {
            names.add(auth.name().toLowerCase(Locale.ROOT));
        }
        VALUES = Collections.unmodifiableList(Arrays.asList(sslClientAuths));
        NAMES = Collections.unmodifiableList(names);
    }

    public static List<String> names() {
        return NAMES;
    }
```

3.BrokerSecurityConfigs.SaslEnabledMechanismsValidator

BrokerSecurityConfigs.SaslEnabledMechanismsValidator

```
public static class SaslEnabledMechanismsValidator implements ConfigDef.Validator {
        @Override
        public void ensureValid(String name, Object value) {
            if (value == null) {
                throw new ConfigException(name, null, "entry must be non null");
            }

            @SuppressWarnings("unchecked")
            List<String> mechanismStrings = (List) value;

            if (mechanismStrings.isEmpty()) {
                throw new ConfigException(name, null, "entry must be non-empty list");
            }

            mechanismStrings.forEach(mechanism -> {
                if (mechanism == null || mechanism.isEmpty()) {
                    throw new ConfigException(name, mechanism, "enabled mechanism must be non-null or non-empty
string");
                }
            });
        }

        @Override
        public String toString() {
            return "non-empty list, enabled mechanism must be non-null or non-empty string";
        }
    }
```

4.CompressionType.names()

CompressionType.names()

```
    private static final List<String> NAMES;

    static {
        CompressionType[] compressionTypes = CompressionType.values();
        List<String> names = new ArrayList<>(compressionTypes.length);
        for (CompressionType compressionType : compressionTypes) {
            names.add(compressionType.name);
        }
        NAMES = Collections.unmodifiableList(names);
    }

     public static List<String> names() {
        return NAMES;
    }
```

Compatibility, Deprecation, and Migration Plan

For all the configurations mentioned above, if the value set by the user is legal, the user will not perceive any changes.
If the value set by the user is invalid, the corresponding validator will throw a `ConfigException` when the `ConfigDef.parse(Map<?, ?> props)` method is executed. The corresponding Kafka program will fail to initialize.

Rejected Alternatives

I had hoped to add another validator for the following configuration: `ValidList.in("GSSAPI", "PLAIN", "SCRAM-SHA-256", "SCRAM-SHA-512", "OAUTHBEARER")`.

```
sasl.mechanism
sasl.enabled.mechanisms
sasl.mechanism.controller.protocol
sasl.mechanism.inter.broker.protocol
```

But Kafka's sasl module is highly flexible and customizable. Users can use the five sasl mechanisms supported by Kafka, or customize other mechanisms by themselves.