CLOUDURABLE ™

Cassandra / Kafka Support in EC2/AWS. Kafka Training, Kafka Consulting

Tutorial on Kafka Streaming Platform Part 2

Cloudurable provides Cassandra and Kafka Support on AWS/EC2

Kafka Tutorial (Part 2)

Java Producer and Consumer examples

Schema Registry

CLOUDURABLE

StockPrice Producer Java Example

Lab StockPrice Producer Java Example

StockPrice App to demo Advanced Producer

- ***StockPrice*** - holds a stock price has a name, dollar, and cents
- ***StockPriceKafkaProducer -*** Configures and creates ***KafkaProducer***<String, ***StockPrice***>, ***StockSender*** list, ThreadPool (ExecutorService), starts ***StockSender*** runnable into thread pool
- ***StockAppConstants*** - holds topic and broker list
- ***StockPriceSerializer -*** can serialize a ***StockPrice*** into ***byte[]***
- ***StockSender -*** generates somewhat random stock prices for a given ***StockPrice*** name, Runnable, 1 thread per StockSender
  - Shows using ***KafkaProducer*** from many threads

StockPrice domain object

StockPrice.java

```
StockPrice
1  package com.cloudurable.kafka.producer.model;
2
3  import io.advantageous.boon.json.JsonFactory;
4
5  public class StockPrice {
6
7      private final int dollars;
8      private final int cents;
9      private final String name;
10
11     public String toJson() {
12         return "{" +
13                 "\"dollars\": " + dollars +
14                 ", \"cents\": " + cents +
15                 ", \"name\": \"" + name + '\"' +
16                 '}';
17     }
18
```

- has name
- dollars
- cents
- converts itself to JSON

StockPriceKafkaProducer

- Import classes and setup logger
- Create ***createProducer*** method to create ***KafkaProducer*** instance
- Create ***setupBootstrapAndSerializers*** to initialize bootstrap servers, client id, key serializer and custom serializer (***StockPriceSerializer***)
- Write ***main***() method - creates ***producer***, create ***StockSender*** list passing each instance a ***producer,*** creates a thread pool so every stock sender gets it own thread, runs each stockSender in its own thread

StockPriceKafkaProducer imports, createProducer

StockPriceKafkaProducer main()

```
1  package com.cloudurable.kafka.producer;
2
3  import com.cloudurable.kafka.StockAppConstants;
4  import com.cloudurable.kafka.producer.model.StockPrice;
5  import io.advantageous.boon.core.Lists;
6  import org.apache.kafka.clients.producer.*;
7  import org.apache.kafka.common.serialization.StringSerializer;
8  import org.slf4j.Logger;
9  import org.slf4j.LoggerFactory;
10
11 import java.util.List;
12 import java.util.Properties;
13 import java.util.concurrent.ExecutorService;
14 import java.util.concurrent.Executors;
15 import java.util.concurrent.TimeUnit;
16
17 public class StockPriceKafkaProducer {
18     private static final Logger logger =
19             LoggerFactory.getLogger(StockPriceKafkaProducer.class);
20
21     private static Producer<String, StockPrice> createProducer() {
22         final Properties props = new Properties();
23         setupBootstrapAndSerializers(props);
24         return new KafkaProducer<>(props);
25     }
```

Import classes and setup logger

createProducer used to create a KafkaProducer

createProducer() calls setupBoostrapAndSerializers()

Configure Producer Bootstrap and Serializer

StockPriceKafkaProducer

```
27      private static void setupBootstrapAndSerializers(Properties props) {
28          props.put(ProducerConfig.BOOTSTRAP_SERVERS_CONFIG,
29                  StockAppConstants.BOOTSTRAP_SERVERS);
30          props.put(ProducerConfig.CLIENT_ID_CONFIG, "StockPriceKafkaProducer");
31          props.put(ProducerConfig.KEY_SERIALIZER_CLASS_CONFIG,
32                  StringSerializer.class.getName());
33
34          //Custom Serializer - config "value.serializer"
35          props.put(ProducerConfig.VALUE_SERIALIZER_CLASS_CONFIG,
36                  StockPriceSerializer.class.getName());
37      }
```

- Create ***setupBootstrapAndSerializers*** to initialize bootstrap servers, client id, key serializer and custom serializer (***StockPriceSerializer***)
- ***StockPriceSerializer*** will serialize ***StockPrice*** into bytes

StockPriceKafkaProducer.main()

StockPriceKafkaProducer.java

```
StockPriceKafkaProducer  main()
39      public static void main(String... args)
40                                  throws Exception {
41
42          //Create Kafka Producer
43          final Producer<String, StockPrice> producer = createProducer();
44
45          //Create StockSender list
46          final List<StockSender> stockSenders = getStockSenderList(producer);
47
48          //Create a thread pool so every stock sender gets it own.
49          final ExecutorService executorService =
50                  Executors.newFixedThreadPool(stockSenders.size());
51
52          //Run each stock sender in its own thread.
53          stockSenders.forEach(executorService::submit);
54
55      }
```

- ***main*** method - creates ***producer***,
- create ***StockSender*** list passing each instance a ***producer***
- creates a thread pool (executorService)
- every StockSender runs in its own thread

StockAppConstants

```
1  package com.cloudurable.kafka;
2
3  public class StockAppConstants {
4      public final static String TOPIC = "stock-prices";
5      public final static String BOOTSTRAP_SERVERS =
6              "localhost:9092,localhost:9093,localhost:9094";
7
8  }
```

- topic name for Producer example
- List of bootstrap servers

StockPriceKafkaProducer.getStockSenderList

StockPriceKafkaProducer | getStockSenderList()

```
57 @    private static List<StockSender> getStockSenderList(
58                 final Producer<String, StockPrice> producer) {
59             return Lists.list(
60                     new StockSender(StockAppConstants.TOPIC,
61                             new StockPrice( name: "IBM",  dollars: 100,  cents: 99),
62                             new StockPrice( name: "IBM",  dollars: 50,  cents: 10),
63                             producer,
64                              delayMinMs: 1,  delayMaxMs: 10
65                             ),
66                     new StockSender(
67                             StockAppConstants.TOPIC,
68                             new StockPrice( name: "SUN",  dollars: 100,  cents: 99),
69                             new StockPrice( name: "SUN",  dollars: 50,  cents: 10),
70                             producer,
71                              delayMinMs: 1,  delayMaxMs: 10
72                     ),
73                     new StockSender(
74                             StockAppConstants.TOPIC,
75                             new StockPrice( name: "GOOG",  dollars: 500,  cents: 99),
76                             new StockPrice( name: "GOOG",  dollars: 400,  cents: 10),
77                             producer,
78                              delayMinMs: 1,  delayMaxMs: 10
79                     ),
80                     new StockSender(
81                             StockAppConstants.TOPIC,
82                             new StockPrice( name: "INEL",  dollars: 100,  cents: 99),
83                             new StockPrice( name: "INEL",  dollars: 50,  cents: 10),
84                             producer,
85                              delayMinMs: 1,  delayMaxMs: 10
```

StockPriceSerializer

```
1   package com.cloudurable.kafka.producer;
2   import com.cloudurable.kafka.producer.model.StockPrice;
3   import org.apache.kafka.common.serialization.Serializer;
4   import java.nio.charset.StandardCharsets;
5   import java.util.Map;
6
7   public class StockPriceSerializer implements Serializer<StockPrice> {
8
9       @Override
10      public byte[] serialize(String topic, StockPrice data) {
11          return data.toJson().getBytes(StandardCharsets.UTF_8);
12      }
13
14      @Override
15      public void configure(Map<String, ?> configs, boolean isKey) {
16      }
17
18      @Override
19      public void close() {
20      }
21  }
```

- Converts ***StockPrice*** into byte array

StockSender

- Generates random stock prices for a given ***StockPrice*** name,
- StockSender is Runnable
- 1 thread per StockSender
- Shows using ***KafkaProducer*** from many threads
- Delays random time between delayMin and delayMax,
  - then sends random StockPrice between ***stockPriceHigh*** and ***stockPriceLow***

StockSender imports, Runnable

StockSender

```
1  package com.cloudurable.kafka.producer;
2
3  import com.cloudurable.kafka.producer.model.StockPrice;
4  import org.apache.kafka.clients.producer.Producer;
5  import org.apache.kafka.clients.producer.ProducerRecord;
6  import org.apache.kafka.clients.producer.RecordMetadata;
7  import org.slf4j.Logger;
8  import org.slf4j.LoggerFactory;
9
10 import java.util.Date;
11 import java.util.Random;
12 import java.util.concurrent.ExecutionException;
13 import java.util.concurrent.Future;
14
15 public class StockSender implements Runnable{
```

- Imports Kafka ***Producer***, ***ProducerRecord***, ***RecordMetadata, StockPrice***
- Implements Runnable, can be submitted to ***ExecutionService***

StockSender constructor

StockSender

```
17      private final StockPrice stockPriceHigh;
18      private final StockPrice stockPriceLow;
19      private final Producer<String, StockPrice> producer;
20      private final int delayMinMs;
21      private final int delayMaxMs;
22      private final Logger logger = LoggerFactory.getLogger(StockSender.class);
23      private final String topic;
24
25      public StockSender(final String topic, final StockPrice stockPriceHigh,
26                         final StockPrice stockPriceLow,
27                         final Producer<String, StockPrice> producer,
28                         final int delayMinMs,
29                         final int delayMaxMs) {
30          this.stockPriceHigh = stockPriceHigh;
31          this.stockPriceLow = stockPriceLow;
32          this.producer = producer;
33          this.delayMinMs = delayMinMs;
34          this.delayMaxMs = delayMaxMs;
35          this.topic = topic;
36      }
```

- takes a topic, high & low stockPrice, producer, delay min & max

StockSender run()

StockSender run()

```
38
39      public void run() {
40          final Random random = new Random(System.currentTimeMillis());
41          int sentCount = 0;
42
43          while (true) {
44              sentCount++;
45              final ProducerRecord <String, StockPrice> record =
46                                              createRandomRecord(random);
47              final int delay = randomIntBetween(random, delayMaxMs, delayMinMs);
48
49              try {
50                  final Future<RecordMetadata> future = producer.send(record);
51                  if (sentCount % 100 == 0) {displayRecordMetaData(record, future);}
52                  Thread.sleep(delay);
53              } catch (InterruptedException e) {
54                  if (Thread.interrupted()) {
55                      break;
56                  }
57              } catch (ExecutionException e) {
58                  logger.error("problem sending record to producer", e);
59              }
60          }
61      }
```

- In loop, creates random record, ***send*** record, waits random time

StockSender createRandomRecord

StockSender | createRandomRecord()

```
77
78      private final int randomIntBetween(final Random random,
79                                         final int max,
80                                         final int min) {
81          return random.nextInt( bound: max - min + 1) + min;
82      }
83
84      private ProducerRecord<String, StockPrice> createRandomRecord(
85                  final Random random) {
86
87          final int dollarAmount = randomIntBetween(random,
88                  stockPriceHigh.getDollars(), stockPriceLow.getDollars());
89
90          final int centAmount = randomIntBetween(random,
91                  stockPriceHigh.getCents(), stockPriceLow.getCents());
92
93          final StockPrice stockPrice = new StockPrice(
94                  stockPriceHigh.getName(), dollarAmount, centAmount);
95
96          return new ProducerRecord<>(topic, stockPrice.getName(),
97                  stockPrice);
98      }
```

- ***createRandomRecord*** uses ***randomIntBetween***
- creates ***StockPrice*** and then wraps ***StockPrice*** in ***ProducerRecord***

StockSender displayRecordMetaData

StockSender | displayRecordMetaData()

```
62
63      private void displayRecordMetaData(final ProducerRecord<String, StockPrice> record,
64                                         final Future<RecordMetadata> future)
65                                 throws InterruptedException, ExecutionException {
66          final RecordMetadata recordMetadata = future.get();
67          logger.info(String.format("\n\t\t\tkey=%s, value=%s " +
68                          "\n\t\t\tsent to topic=%s part=%d off=%d at time=%s",
69                  record.key(),
70                  record.value().toJson(),
71                  recordMetadata.topic(),
72                  recordMetadata.partition(),
73                  recordMetadata.offset(),
74                  new Date(recordMetadata.timestamp())
75                  ));
76      }
```

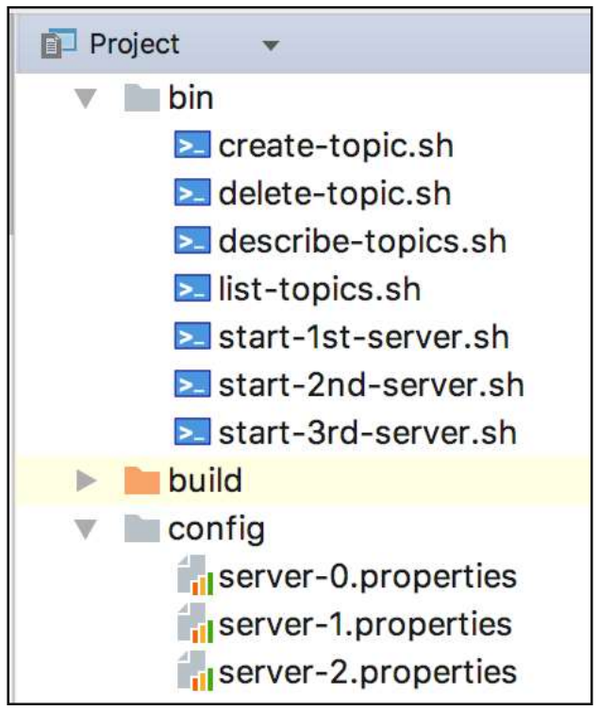
- Every 100 records displayRecordMetaData gets called
- Prints out record info, and recordMetadata info:
  - key, JSON value, topic, partition, offset, time
  - uses Future from call to ***producer.send***()

Run it

- Run ZooKeeper
- Run three Brokers
- run create-topic.sh
- Run ***StockPriceKafkaProducer***

Run scripts

- run ZooKeeper from ~/kafka-training
- use ***bin/create-topic.sh*** to create topic
- use ***bin/delete-topic.sh*** to delete topic
- use ***bin/start-1st-server.sh*** to run Kafka Broker 0
- use ***bin/start-2nd-server.sh*** to run Kafka Broker 1
- use ***bin/start-3rd-server.sh*** to run Kafka Broker 2



Config is under directory called config

server-0.properties is for Kafka Broker 0
server-1.properties is for Kafka Broker 1
server-2.properties is for Kafka Broker 2

Run All 3 Brokers

```
#!/usr/bin/env bash
CONFIG=`pwd`/config
cd ~/kafka-training
## Run Kafka
kafka/bin/kafka-server-start.sh \
    "$CONFIG/server-0.properties"

```

start-2nd-server.sh

```
#!/usr/bin/env bash
CONFIG=`pwd`/config
cd ~/kafka-training
## Run Kafka
kafka/bin/kafka-server-start.sh \
    "$CONFIG/server-1.properties"

```

start-3rd-server.sh

```
#!/usr/bin/env bash
CONFIG=`pwd`/config
cd ~/kafka-training
## Run Kafka
kafka/bin/kafka-server-start.sh \
    "$CONFIG/server-2.properties"

```

server-0.properties

```
broker.id=0
port=9092
log.dirs=./logs/kafka-0
min.insync.replicas=3
compression.type=producer
auto.create.topics.enable=false
message.max.bytes=65536
replica.lag.time.max.ms=5000
delete.topic.enable=true
```

server-1.properties

```
broker.id=1
port=9093
log.dirs=./logs/kafka-1
min.insync.replicas=3
compression.type=producer
```

server-2.properties

```
broker.id=2
port=9094
log.dirs=./logs/kafka-2
min.insync.replicas=3
compression.type=producer
auto.create.topics.enable=false
message.max.bytes=65536
replica.lag.time.max.ms=5000
delete.topic.enable=true
```

Run create-topic.sh script

create-topic.sh

```
1   #!/usr/bin/env bash
2
3   cd ~/kafka-training
4
5   kafka/bin/kafka-topics.sh \
6       --create \
7       --zookeeper localhost:2181 \
8       --replication-factor 3 \
9       --partitions 3 \
10      --topic stock-prices
11
```

Name of the topic is stock-prices

Three partitions

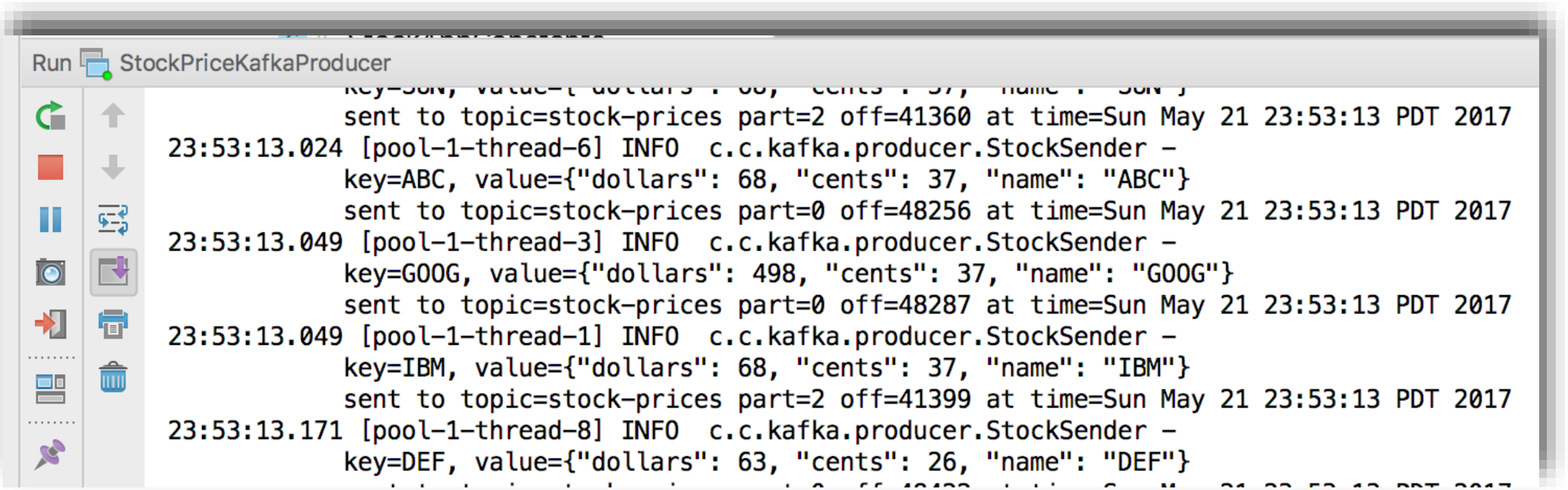
Replication factor of three

Terminal

Local | Local (1) | Local (2) | Local (3) | Local (4)

```
$ ./bin/create-topic.sh
Created topic "stock-prices".
```

Run StockPriceKafkaProducer



- Run ***StockPriceKafkaProducer*** from the IDE
- You should see log messages from ***StockSender***(s) with ***StockPrice*** name, JSON value, partition, offset, and time

Lab Adding an orderly shutdown flush and close

Using flush and close

Shutdown Producer nicely

- Handle ctrl-C shutdown from Java
- Shutdown thread pool and wait
- Flush producer to send any outstanding batches if using batches (***producer.flush()***)
- Close Producer (***producer.close()***) and wait

Nice Shutdown producer.close()

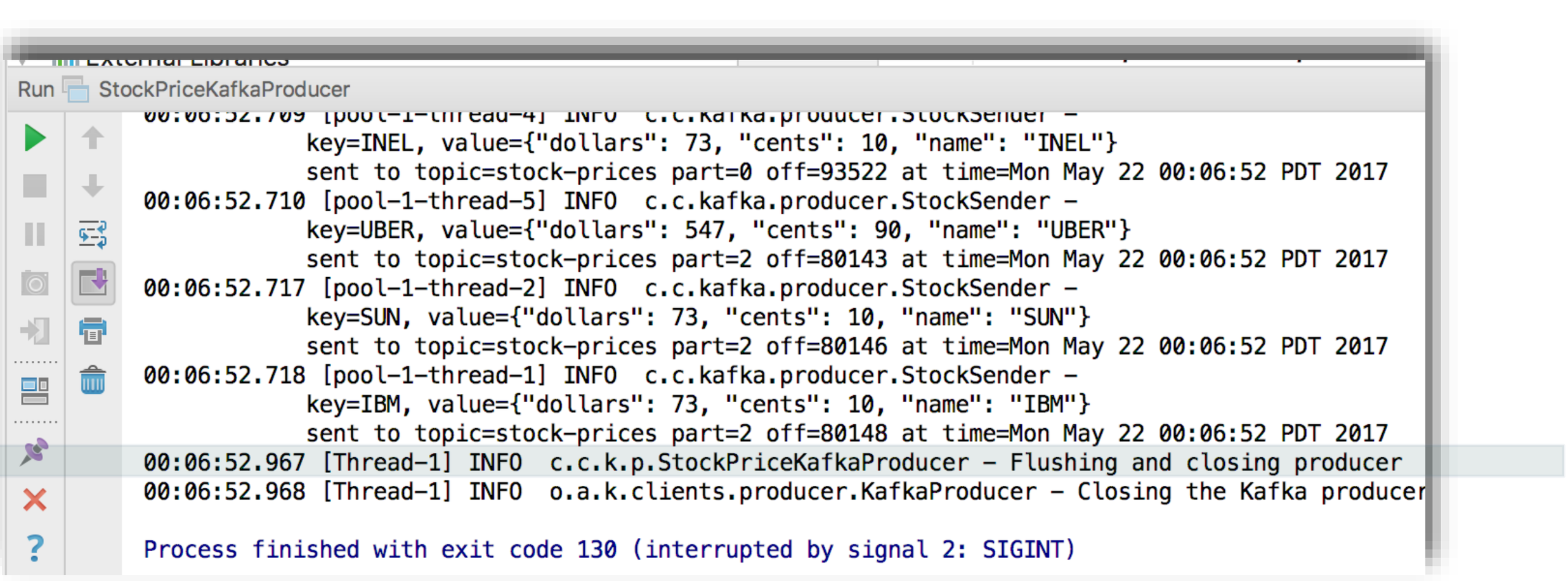
StockPriceKafkaProducer.java

StockPriceKafkaProducer › main()

```
41  public static void main(String... args)
42          throws Exception {
43      //Create Kafka Producer
44      final Producer<String, StockPrice> producer = createProducer();
45      //Create StockSender list
46      final List<StockSender> stockSenders = getStockSenderList(producer);
47      //Create a thread pool so every stock sender gets it own.
48      final ExecutorService executorService =
49              Executors.newFixedThreadPool(stockSenders.size());
50      //Run each stock sender in its own thread.
51      stockSenders.forEach(executorService::submit);
52
53      //Register nice shutdown of thread pool, then flush and close producer.
54      Runtime.getRuntime().addShutdownHook(new Thread(() -> {
55          executorService.shutdown();
56          try {
57              executorService.awaitTermination( timeout: 200, TimeUnit.MILLISECONDS);
58              logger.info("Flushing and closing producer");
59              producer.flush();
60              producer.close( timeout: 10_000, TimeUnit.MILLISECONDS);
61          } catch (InterruptedException e) {
62              logger.warn("shutting down", e);
63          }
64      }));
65  }
```

Restart Producer then shut it down

- Add shutdown hook
- Start StockPriceKafkaProducer
- Now stop it (CTRL-C or hit stop button in IDE)



Lab Configuring Producer Durability

Lab Batching Records

Lab Adding Retries and Timeouts

Kafka Advanced Consumers

Currently Not added to sample

Working with Avro Serialization and the Schema Registry

Schema Management in Kafka

Understanding Avro and the Schema Registry for Kafka

Overview

- First we cover Avro
  - Avro Schema
  - Using a Schema to write data
  - Using a Schema to read data
- Then we cover the Schema Registry
  - Managing Schemas
  - Kafka Avro Serialization

Avro

Introduction to Avro for Big Data and Streaming Architecture

Apache Avro Data Serialization for Hadoop and Kafka

Avro Overview

- Why Avro?
- What is Avro?
- Why are schemas important
- Examples reading and writing with Avro
- Working with Code generation
- Working with Avro Generic Record

Why Avro?

- Direct mapping to JSON
- Compact binary format
- Very Fast
- Polyglot bindings
- Code generation for static languages
- Code generation not needed
- Supports Schema
- Supports compatibility checks
- Allows evolving your data over time

Apache Avro

- Data serialization system
- Data structures
- Binary data format
- Container file format to store persistent data
- RPC capabilities
- Does not require code generation to use

Avro Schemas

- Supports schemas for defining data structure
- Serializing and deserializing data, uses schema
- File schema
  - Avro files store data with its schema
- RPC Schema
  - RPC protocol exchanges schemas as part of the handshake
- Schemas written in JSON

Avro compared to…

- Similar to Thrift, Protocol Buffers, JSON, etc.
- Does not require code generation
- Avro needs less encoding as part of the data since it stores names and types in the schema
- It supports evolution of schemas.

Why schemas?

- Multiple Producers and Consumers evolve over time
- Schemas keep your data clean
  - NoSQL was no schema but not as much: Cassandra has schemas
  - REST/JSON was schema-less and IDL-less but not anymore Swagger and RAML

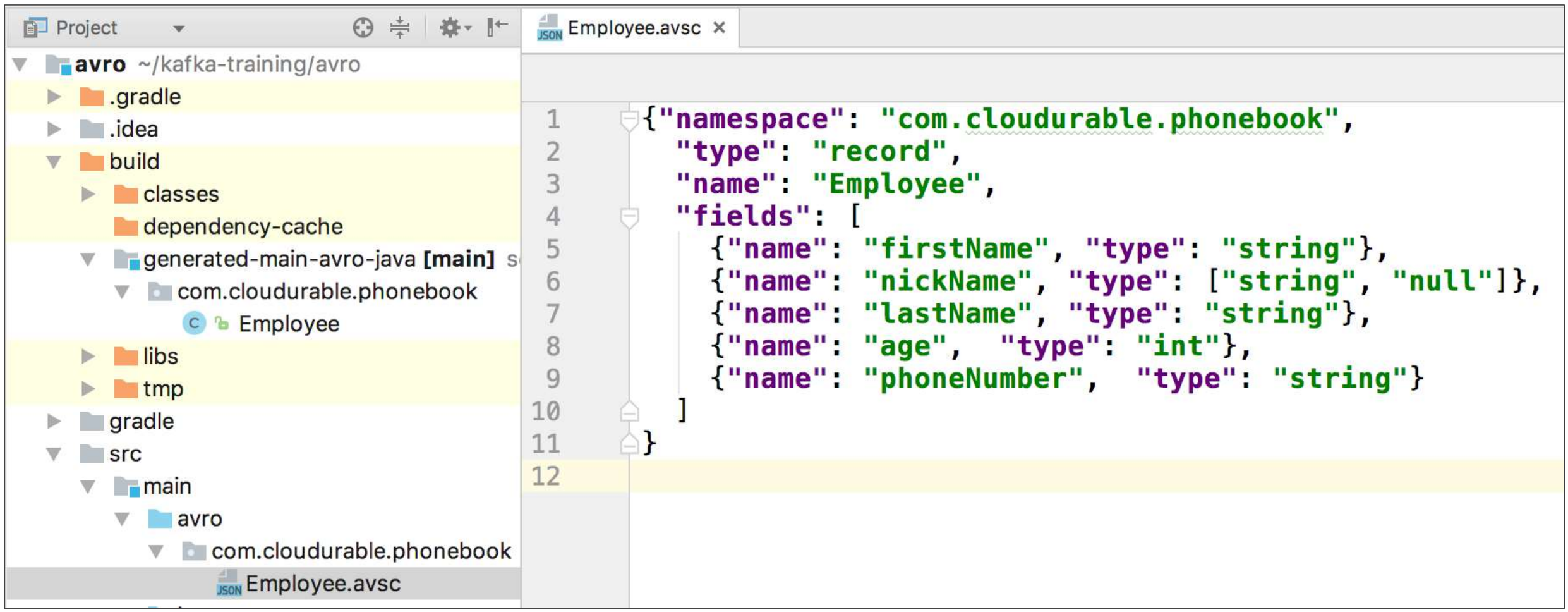
Schema provides Robustness

- Streaming architecture support
  - ***Challenging***: Consumers and Producers ***evolve*** on different timelines.
- Producers send streams of records that 0 to * Consumers read
- All consumers or use cases are ***not known ahead of time***
- Data could go to Hadoop or some store used years later ***in use cases you can't imagine***
- Schemas help ***future proof data***; makes data more robust.
- Avro Schema with support for evolution is important
- ***Schema provides robustness***: provides meta-data about data stored with data

Future Compatibility

- Data record format compatibility is a hard problem to solve
- Schema capture a point in time of what your data when it was stored
- Data will evolve. New fields are added.
- Kafka streams are often recorded in data lakes - needs to be less rigid than schema for operational data
- Producers can have Consumers that they never know about
- You can’t test a Consumer that you don’t know about
- Avro Schemas support agility sakes
- 80% of data science is data cleanup, Schemas help

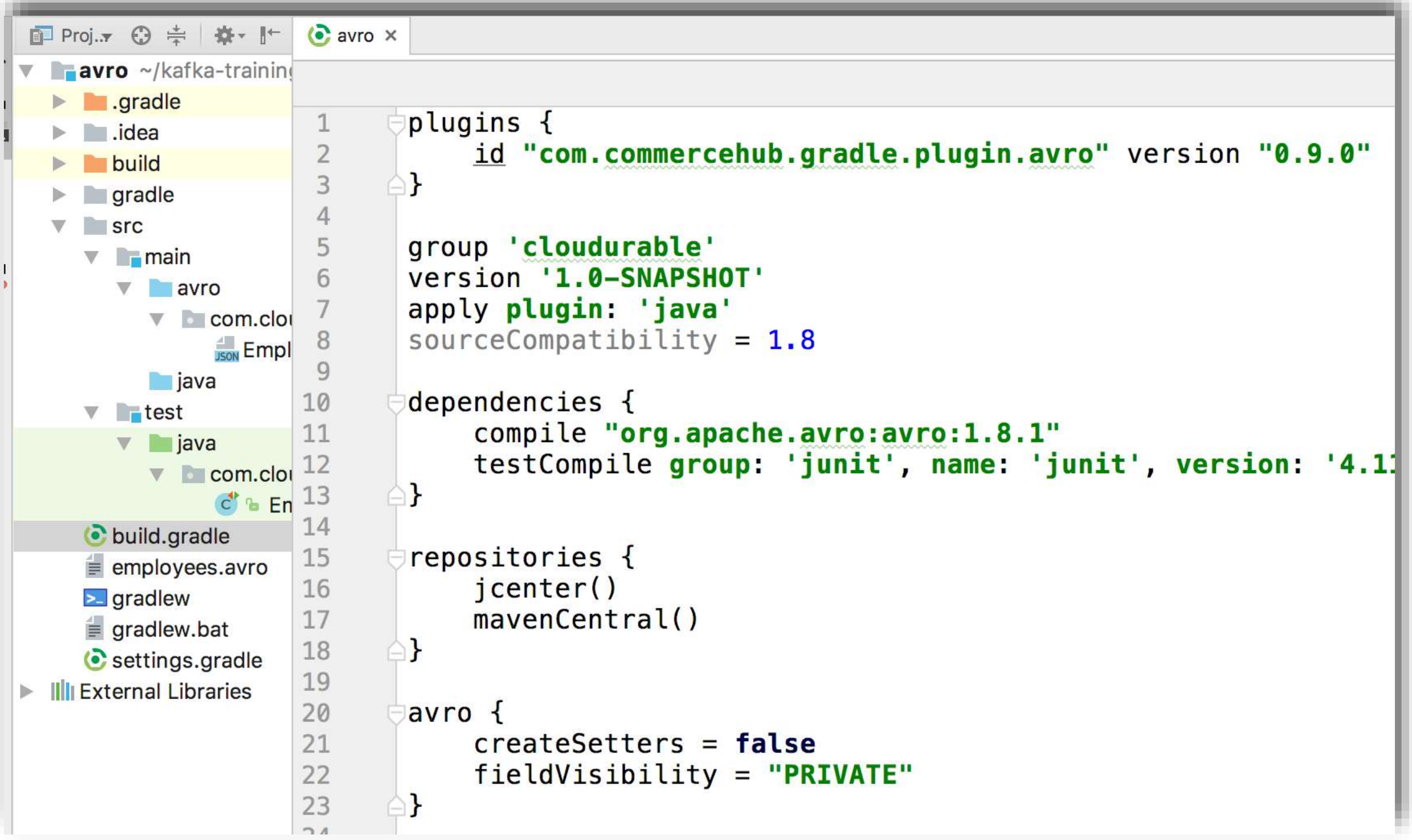
Avro Schema



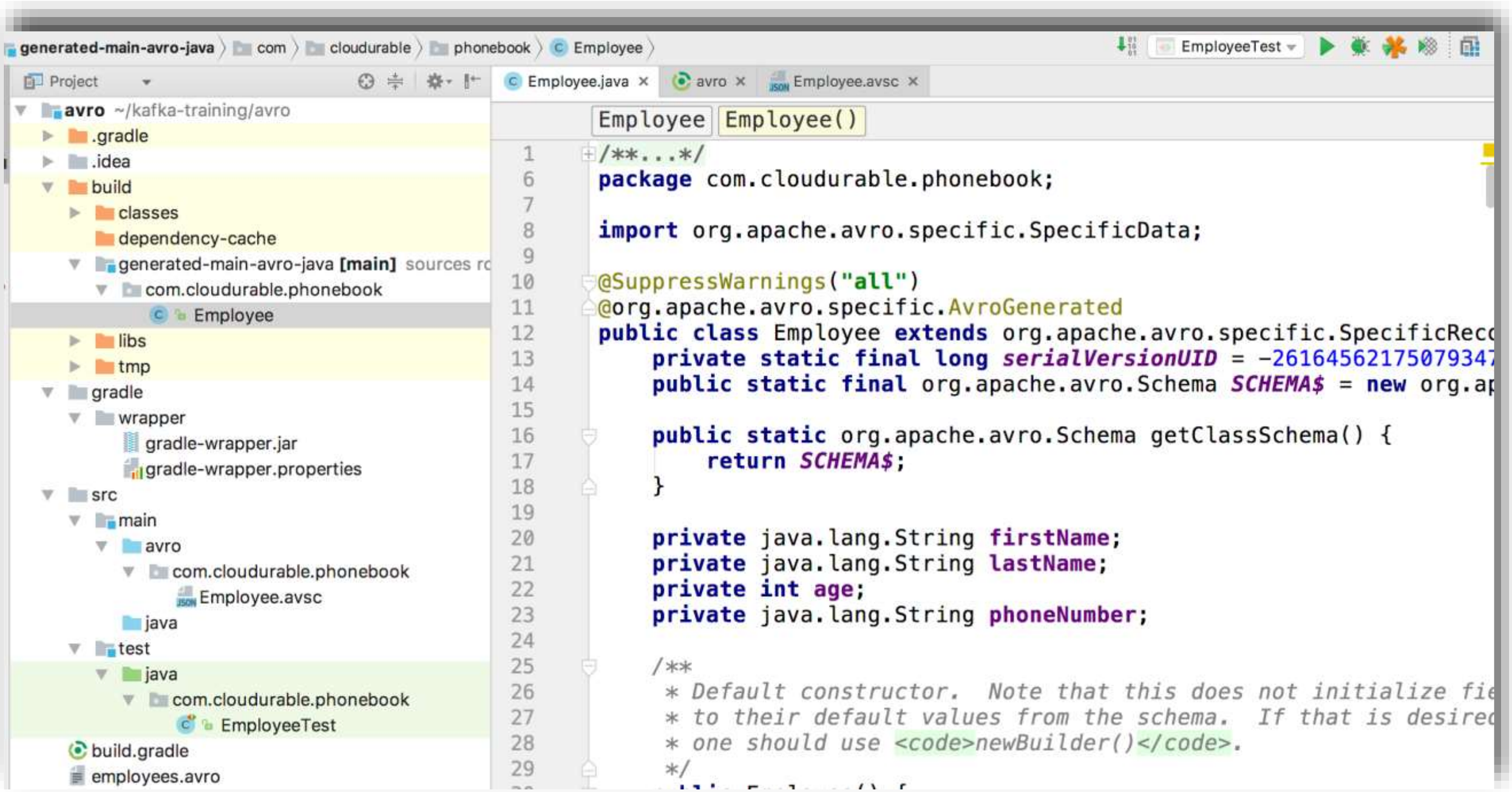
Avro schema stored in src/main/avro by default.

very field and record has a “doc” attribute which you should use to document the purpose of the fi

Code Generation



Employee Code Generation



Using Generated Avro class

```
Employee bob = Employee.newBuilder().setAge(35)
        .setFirstName("Bob")
        .setLastName("Jones")
        .setPhoneNumber("555-555-1212")
        .build();

assertEquals( expected: "Bob", bob.getFirstName());
```

Writing employees to an Avro File

```
final DatumWriter<Employee> datumWriter = new SpecificDatumWriter<>(Employee.class);
final DataFileWriter<Employee> dataFileWriter = new DataFileWriter<>(datumWriter);

try {
    dataFileWriter.create(employeeList.get(0).getSchema(),
            new File( pathname: "employees.avro"));
    employeeList.forEach(employee -> {
        try {
            dataFileWriter.append(employee);
        } catch (IOException e) {
            throw new RuntimeException(e);
        }

    });
} finally {
    dataFileWriter.close();
}
```

Reading employees From a File

```
final File file = new File( pathname: "employees.avro");
final List<Employee> employeeList = new ArrayList<>();
final DatumReader<Employee> empReader = new SpecificDatumReader<>(Employee.class);
final DataFileReader<Employee> dataFileReader = new DataFileReader<>(file, empReader);

while (dataFileReader.hasNext()) {
    employeeList.add(dataFileReader.next(new Employee()));
}

employeeList.forEach(System.out::println);
```

All 2 tests passed – 333ms

```
/Library/Java/JavaVirtualMachines/jdk1.8.0_66.jdk/Contents/Home/bin/java ...
{"firstName": "Bob0", "lastName": "Jones0", "age": 25, "phoneNumber": "555-555-1212"}
{"firstName": "Bob1", "lastName": "Jones1", "age": 26, "phoneNumber": "555-555-1212"}
{"firstName": "Bob2", "lastName": "Jones2", "age": 27, "phoneNumber": "555-555-1212"}
{"firstName": "Bob3", "lastName": "Jones3", "age": 28, "phoneNumber": "555-555-1212"}
{"firstName": "Bob4", "lastName": "Jones4", "age": 29, "phoneNumber": "555-555-1212"}
{"firstName": "Bob5", "lastName": "Jones5", "age": 30, "phoneNumber": "555-555-1212"}
{"firstName": "Bob6", "lastName": "Jones6", "age": 31, "phoneNumber": "555-555-1212"}
{"firstName": "Bob7", "lastName": "Jones7", "age": 32, "phoneNumber": "555-555-1212"}
{"firstName": "Bob8", "lastName": "Jones8", "age": 33, "phoneNumber": "555-555-1212"}
{"firstName": "Bob9", "lastName": "Jones9", "age": 34, "phoneNumber": "555-555-1212"}
{"firstName": "Bob10", "lastName": "Jones10", "age": 35, "phoneNumber": "555-555-1212"}
{"firstName": "Bob11", "lastName": "Jones11", "age": 36, "phoneNumber": "555-555-1212"}
{"firstName": "Bob12", "lastName": "Jones12", "age": 37, "phoneNumber": "555-555-1212"}
```

Using GenericRecord

```
final String schemaLoc = "src/main/avro/com/cloudurable/phonebook/Employee.avsc";
final File schemaFile = new File(schemaLoc);
final Schema schema = new Schema.Parser().parse(schemaFile);

GenericRecord bob = new GenericData.Record(schema);
bob.put( key: "firstName",  v: "Bob");
bob.put( key: "lastName",  v: "Smith");
bob.put( key: "phoneNumber",  v: "555-555-1212");
bob.put( key: "age",  v: 35);


assertEquals( expected: "Bob", bob.get("firstName"));
```

Writing Generic Records

```
final List<GenericRecord> employeeList = new ArrayList<>();
```

```
final DatumWriter<GenericRecord> datumWriter = new GenericDatumWriter<>(schema);
final DataFileWriter<GenericRecord> dataFileWriter = new DataFileWriter<>(datumWriter);

try {
    dataFileWriter.create(employeeList.get(0).getSchema(),
            new File( pathname: "employees2.avro"));
    employeeList.forEach(employee -> {
        try {
            dataFileWriter.append(employee);
        } catch (IOException e) {
            throw new RuntimeException(e);
        }
    });
} finally {
    dataFileWriter.close();
}
```

Reading using Generic Records

```
final File file = new File( pathname: "employees2.avro");
final List<GenericRecord> employeeList = new ArrayList<>();
final DatumReader<GenericRecord> empReader = new GenericDatumReader<>();
final DataFileReader<GenericRecord> dataFileReader = new DataFileReader<>(file, empReader);

while (dataFileReader.hasNext()) {
    employeeList.add(dataFileReader.next( reuse: null));
}

employeeList.forEach(System.out::println);
```

```
All 2 tests passed – 307ms
/Library/Java/JavaVirtualMachines/jdk1.8.0_66.jdk/Contents/Home/bin/java ...
{"firstName": "Bob0", "nickName": null, "lastName": "Smith0", "age": 25, "phoneNumber": "555-555-1212"}
{"firstName": "Bob1", "nickName": null, "lastName": "Smith1", "age": 26, "phoneNumber": "555-555-1212"}
{"firstName": "Bob2", "nickName": null, "lastName": "Smith2", "age": 27, "phoneNumber": "555-555-1212"}
{"firstName": "Bob3", "nickName": null, "lastName": "Smith3", "age": 28, "phoneNumber": "555-555-1212"}
{"firstName": "Bob4", "nickName": null, "lastName": "Smith4", "age": 29, "phoneNumber": "555-555-1212"}
{"firstName": "Bob5", "nickName": null, "lastName": "Smith5", "age": 30, "phoneNumber": "555-555-1212"}
{"firstName": "Bob6", "nickName": null, "lastName": "Smith6", "age": 31, "phoneNumber": "555-555-1212"}
{"firstName": "Bob7", "nickName": null, "lastName": "Smith7", "age": 32, "phoneNumber": "555-555-1212"}
{"firstName": "Bob8", "nickName": null, "lastName": "Smith8", "age": 33, "phoneNumber": "555-555-1212"}
{"firstName": "Bob9", "nickName": null, "lastName": "Smith9", "age": 34, "phoneNumber": "555-555-1212"}
```

Avro Schema Validation

```
GenericRecord employee = new GenericData.Record(schema);
employee.put( key: "firstName",  v: "Bob" + index);
employee.put( key: "lastName",  v: "Smith"+ index);
employee.put( key: "phoneNumber",  v: "555-555-1212");
//employee.put("age", index % 35 + 25);
employee.put( key: "age",  v: "old");
employeeList.add(employee);
```

2 tests done: 1 failed – 395ms

```
org.apache.avro.file.DataFileWriter$AppendWriteException: java.lang.ClassCastException: java.lang.String cann

    at org.apache.avro.file.DataFileWriter.append(DataFileWriter.java:308)
    at com.cloudurable.phonebook.EmployeeTestNoGen.lambda$testWrite$0(EmployeeTestNoGen.java:71)
    at java.util.ArrayList.forEach(ArrayList.java:1249)
    at com.cloudurable.phonebook.EmployeeTestNoGen.testWrite(EmployeeTestNoGen.java:69) <27 internal calls>
Caused by: java.lang.ClassCastException: java.lang.String cannot be cast to java.lang.Number
    at org.apache.avro.generic.GenericDatumWriter.writeWithoutConversion(GenericDatumWriter.java:117)
    at org.apache.avro.generic.GenericDatumWriter.write(GenericDatumWriter.java:73)
    at org.apache.avro.generic.GenericDatumWriter.writeField(GenericDatumWriter.java:153)
    at org.apache.avro.generic.GenericDatumWriter.writeRecord(GenericDatumWriter.java:143)
    at org.apache.avro.generic.GenericDatumWriter.writeWithoutConversion(GenericDatumWriter.java:105)
```

Avro supported types

- Records
- Arrays
- Enums
- Unions
- Maps
- String, Int, Boolean, Decimal, Timestamp, Date

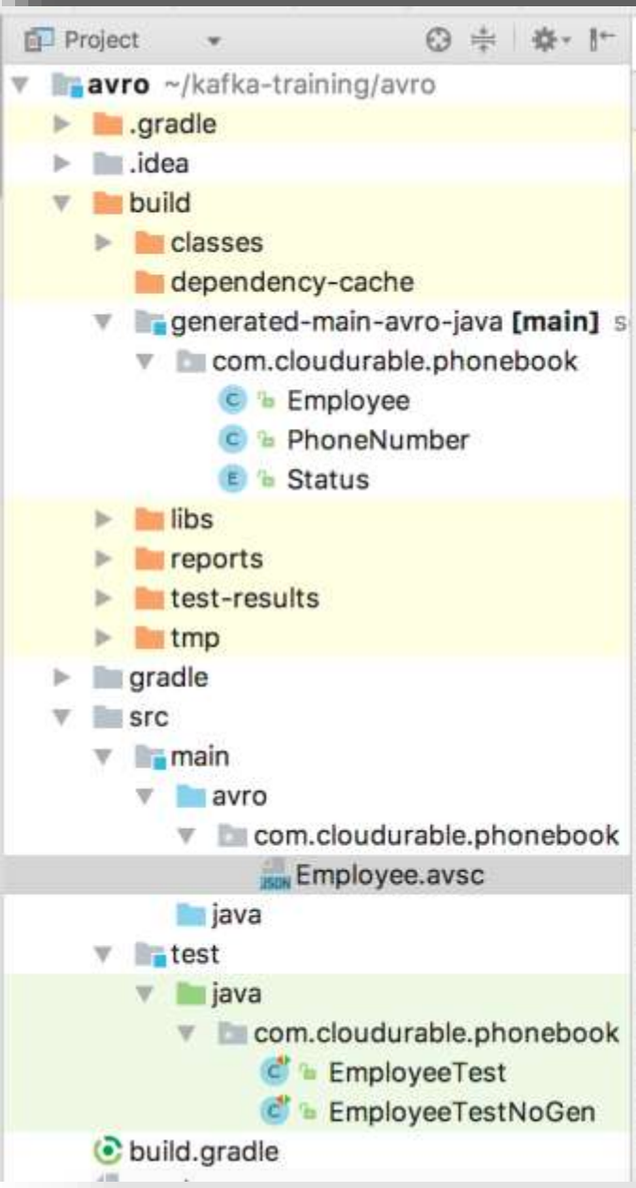
Avro Types 1

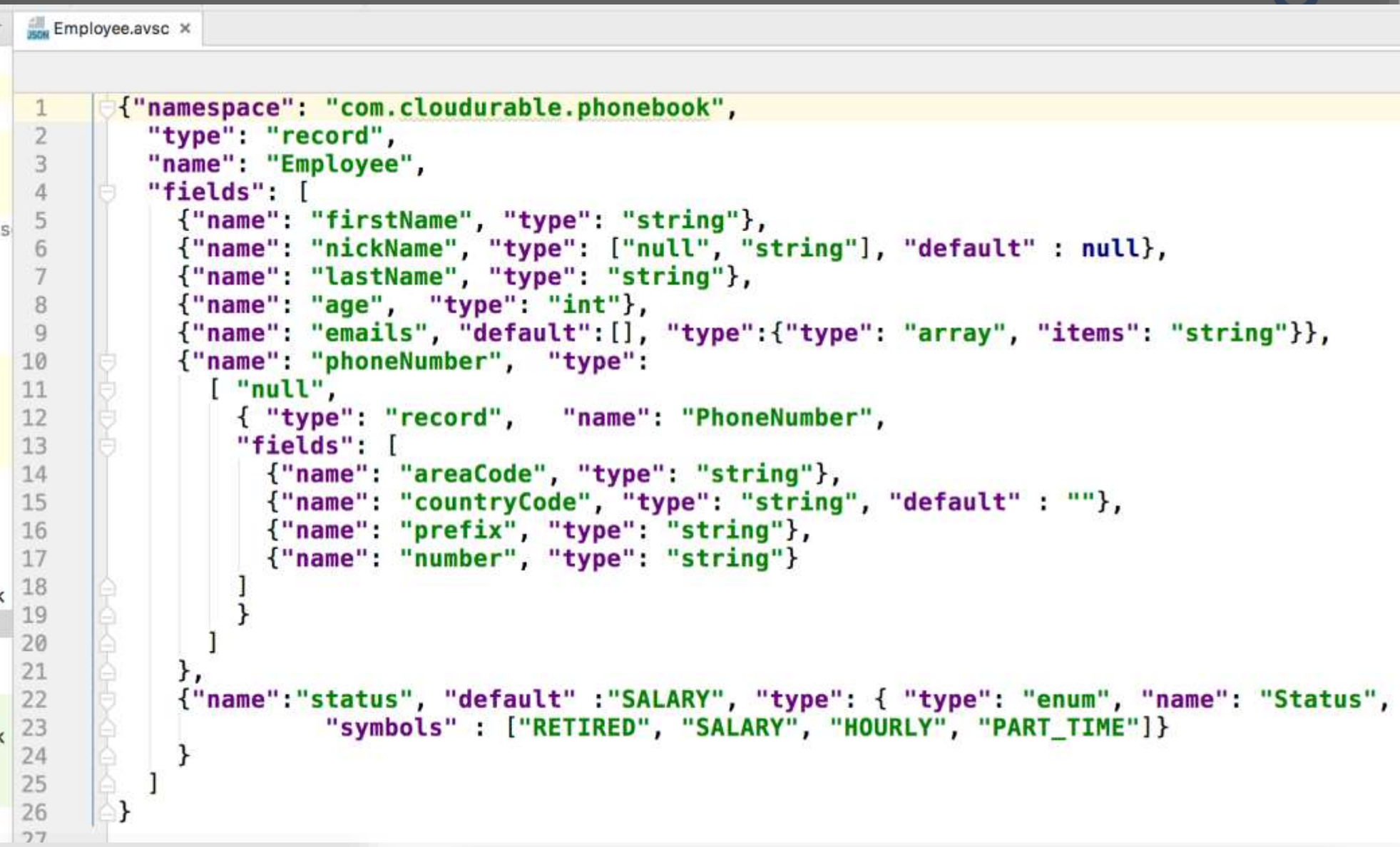
avro type	json type	example	description
null	null	null	no value
boolean	boolean	true	true or false
int	integer	1	32 bit signed int
long			64 bit signed int
float	number	1.1	single precision (32-bit) floating-point number
double	number	1.2	double precision (64-bit) floating-point number
bytes	string	"\u00FF"	
string	string	"foo"	
record	object	{"a": 1}	complex type / object
enum	string	“Hi Mom"	Enumerated type
array	array	[1]	array of any value
union			can be used to define nullable fields
map	object	{"a": 1}	
fixed	string	"\u00ff"	fixed length string

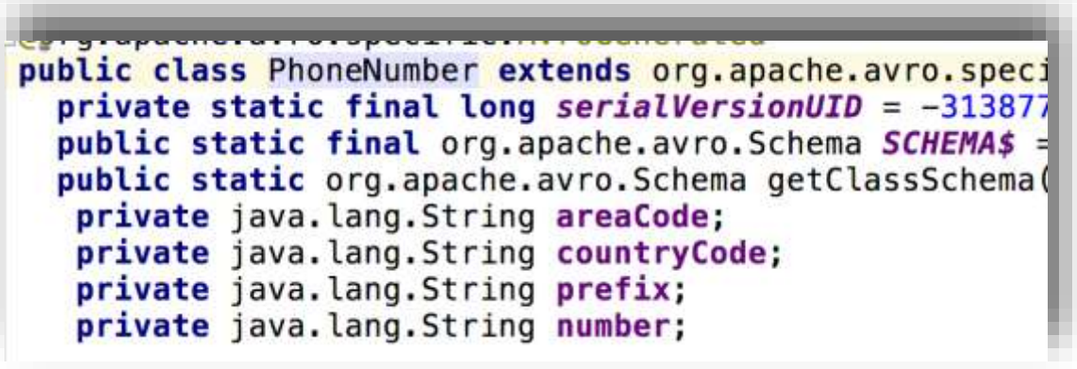
Avro Types 2

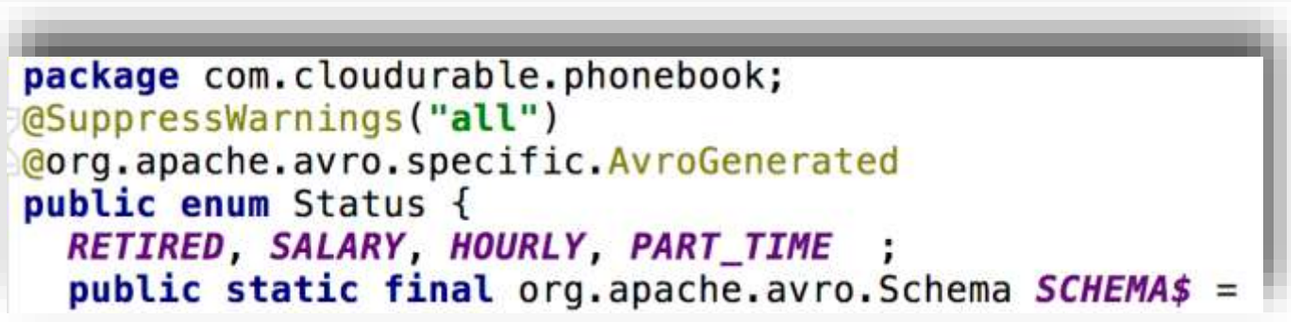
avro type	json type	example	description
date	int	2000	stores the number of days from the unix epoch, 1 January 1970
time-millis	int	10800000	stores the number of milliseconds after midnight
time-micros	long	1000000	microseconds after midnight
timestamp-milis	long	1494437796880	milliseconds since epoch
decimal	complex/object	22.555	specifies scale and precision like BigDecimal in Java
duration	complex/object		Specifies duration in months, days and milliseconds

Fuller example Avro Schema









Avro Record Attributes

- ***name***: name of the record (required).
- ***namespace***: equates to packages or modules
- ***doc***: documentation for future user of this schema
- ***aliases***: array aliases (alias names)
- ***fields***: an array of fields

Avro Field Attributes

- ***name***: name of the field (required)
- ***doc***: description of field (important for future usage)
- ***type***: JSON object defining a schema, or a JSON string naming a record definition (required)
- ***default***: Default value for this field
- ***order***: specifies sort ordering of record (optional).
  - "ascending" (the default), "descending", "ignore"
- ***aliases***: array of alternate names

Tips for Using Avro

- Avoid advance features which are not supported by polyglot language mappings
  - Think simple data objects
- Instead of magic strings use enums (better validation)
- Document all records and fields in the schema
- Avoid complex union types (use Unions for nullable fields only)
- Avoid recursive types
- Use reasonable field names
  - employee_id instead of id and then use employee_id in all other records that have a field that refer to the employee_id

Avro

- Fast data serialization
- Supports data structures
- Supports Records, Maps, Array, and basic types
- You can use it direct or use Code Generation

- Read more.

Avro

Confluent Schema Registry and Kafka Avro Serialization

Managing Record Schema in Kafka

Overview

- What is the Schema Registry?
- Why do you want to use it?
- Understanding Avro Schema Evolution
- Setting up and using Schema Registry
- Managing Schemas with REST interface
- Writing Avro Serializer based Producers
- Writing Avro Deserializer based Consumers

Confluent Schema Registry

- Confluent Schema Registry manage Avro Schemas for Kafka clients
- Provides REST interface for putting and getting Avro schemas
- Stores a history of schemas
  - versioned
  - allows you to configure compatibility setting
  - supports evolution of schemas
- Provides serializers used by Kafka clients which handles schema storage and serialization of records using Avro
- Don’t have to send schema just the schema uuid, schema can be looked up in Registry. Helps with Speed!

Avro Producer Schema Process

- Producer creates a record/message, which is an Avro record
- Record contains schema and data
- Schema Registry Avro Serializer serializes the data and schema id (just id)
  - Keeps a cache of registered schemas from Schema Registry to ids
- Consumer receives payload and deserializes it with Kafka Avro Deserializers which use Schema Registry
- Deserializer looks up the full schema from cache or Schema Registry based on id

Why Schema Registry?

- Consumer has its schema, one it is expecting record/message to conform to
  - Compatibility check is performed or two schemas
    - if no match, but are compatible, then payload transformation happens aka Schema Evolution
    - if not failure
- Kafka records have Key and Value and schema can be done on both

Schema Registry Actions

- Register schemas for key and values of Kafka records
- List schemas (subjects); List all versions of a subject (schema)
- Retrieve a schema by version or id; get latest version of schema
- Check to see if schema is compatible with a certain version
- Get the compatibility level setting of the Schema Registry
  - BACKWARDS, FORWARDS, FULL, NONE
- Add compatibility settings to a subject/schema

Schema Compatibility

- ***Backward compatibility***
  - data written with older schema can be read with newer schema
- ***Forward compatibility***
  - data written with newer schema can be read with an older schema
- ***Full compatibility***
  - New version of a schema is backward and forward compatible
- ***None***
  - Schema will not be validated for compatibility at all

Schema Registry Config

- Compatibility can be configured globally or per schema
- Options are:
  - NONE - don’t check for schema compatibility
  - FORWARD - check to make sure last schema version is forward compatible with new schemas
  - BACKWARDS (default) - make sure new schema is backwards compatible with latest
  - FULL - make sure new schema is forwards and backwards compatible from latest to new and from new to latest

Schema Evolution

- Avro schema is changed after data has been written to store using an older version of that schema, then Avro might do a Schema Evolution
- Schema evolution happens only during deserialization at Consumer
- If Consumer's schema is different from Producer's schema, then value or key is automatically modified during deserialization to conform to consumers reader schema

Schema Evolution 2

- Schema evolution is automatic transformation of Avro schema
  - transformation is between version of consumer schema and what the producer put into the Kafka log
  - When Consumer schema is not identical to the Producer schema used to serialize the Kafka Record, then a data transformation is performed on the Kafka record (key or value)
  - If the schemas match then no need to do a transformation

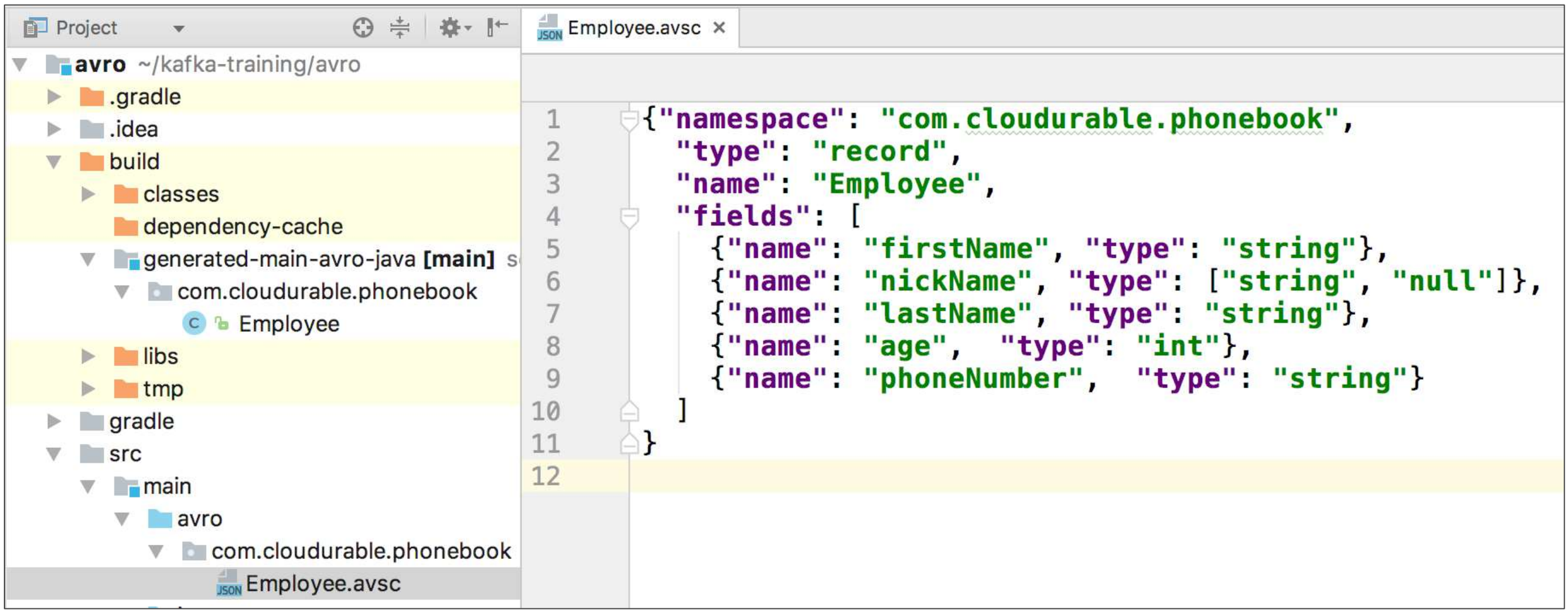
Allowed Schema Modifications

- Add a field with a default
- Remove a field that had a default value
- Change a field's order attribute
- Change a field's default value
- Remove or add a field alias
- Remove or add a type alias
- Change a type to a union that contains original type

Rules of the Road for modifying Schema

- Provide a default value for fields in your schema
  - Allows you to delete the field later
- Don’t change a field's data type
- When adding a new field to your schema, you have to provide a default value for the field
- Don’t rename an existing field
  - You can add an alias

Remember our example Employee



Avro covered in Avro/Kafka Tutorial

Let's say

- Employee did not have an age in version 1 of the schema
- Later we decided to add an age field with a default value of -1
- Now let's say we have a Producer using version 2, and a Consumer using version 1

Scenario adding new field age w/ default value

- Producer uses version 2 of the Employee schema and creates a com.cloudurable.Employee record, and sets age field to 42, then sends it to Kafka topic new-employees
- Consumer consumes records from new-employees using version 1 of the Employee Schema
  - Since Consumer is using version 1 of schema, age field is removed during deserialization

Scenario adding new field age w/ default value 2

- Same consumer modifies some fields, then writes record to a NoSQL store
  - When it does this, the age field is missing from record that it writes to NoSQL store
- Another client using version 2 reads the record from the NoSQL store
  - Age field is missing from the record (because the Consumer wrote it with version 1), age is set to default value of -1

Schema Registry Actions REST

- Register schemas for key and values of Kafka records
- List schemas (subjects)
- List all versions of a subject (schema)
- Retrieve a schema by version or id
  - get latest version of schema

Schema Registry Actions 2

- Check to see if schema is compatible with a certain version
- Get the compatibility level setting of the Schema Registry
  - BACKWARDS, FORWARD, FULL, NONE
- Add compatibility settings to a subject/schema

Register a Schema

```
private final static MediaType SCHEMA_CONTENT =
        MediaType.parse("application/vnd.schemaregistry.v1+json");

private final static String EMPLOYEE_SCHEMA = "{\n" +
        "  \"schema\": \"" +
        "  {" +
        "    \\\"namespace\\\": \\\"com.cloudurable.phonebook\\\"," +
        "    \\\"type\\\": \\\"record\\\"," +
        "    \\\"name\\\": \\\"Employee\\\"," +
        "    \\\"fields\\\": [" +
        "        {\\\"name\\\": \\\"firstName\\\", \\\"type\\\": \\\"string\\\"}," +
        "        {\\\"name\\\": \\\"lastName\\\", \\\"type\\\": \\\"string\\\"}," +
        "        {\\\"name\\\": \\\"age\\\",  \\\"type\\\": \\\"int\\\"}," +
        "        {\\\"name\\\": \\\"phoneNumber\\\",  \\\"type\\\": \\\"string\\\"}" +
        "    ]" +
        "  }\"" +
        "}";
```

Register a Schema

```
final OkHttpClient client = new OkHttpClient();

//POST A NEW SCHEMA
Request request = new Request.Builder()
        .post(RequestBody.create(SCHEMA_CONTENT, EMPLOYEE_SCHEMA))
        .url("http://localhost:8081/subjects/Employee/versions")
        .build();

String output = client.newCall(request).execute().body().string();
System.out.println(output);
```

```
curl -X POST -H "Content-Type: application/vnd.schemaregistry.v1+json"
    --data '{"schema": "{\"type\": …}' \
    http://localhost:8081/subjects/Employee/versions
```

{"id":2}

List All Schema

```
//LIST ALL SCHEMAS
request = new Request.Builder()
        .url("http://localhost:8081/subjects")
        .build();

output = client.newCall(request).execute().body().string();
System.out.println(output);
```

curl -X GET http://localhost:8081/subjects

["Employee","Employee2","FooBar"]

Working with versions

```
//SHOW ALL VERSIONS OF EMPLOYEE
request = new Request.Builder()
        .url("http://localhost:8081/subjects/Employee/versions/")
        .build();

output = client.newCall(request).execute().body().string();
System.out.println(output);
```

[1,2,3,4,5]

```
//SHOW VERSION 2 OF EMPLOYEE
request = new Request.Builder()
        .url("http://localhost:8081/subjects/Employee/versions/2")
        .build();

output = client.newCall(request).execute().body().string();
System.out.println(output);
```

{“subject”:"Employee","version":2,"id":4,"schema":"
{\"type\":\"record\",\"name\":\"Employee\",
\”namespace\”:\"com.cloudurable.phonebook\", …

```
//SHOW THE SCHEMA WITH ID 3
request = new Request.Builder()
        .url("http://localhost:8081/schemas/ids/3")
        .build();

output = client.newCall(request).execute().body().string();
System.out.println(output);
```

{“subject”:"Employee","version":1,"id":3,"schema":"
{\"type\":\"record\",\"name\":\"Employee\",
\”namespace\”:\"com.cloudurable.phonebook\", …

Working with Schemas

```
//SHOW THE LATEST VERSION OF EMPLOYEE 2
request = new Request.Builder()
        .url("http://localhost:8081/subjects/Employee/versions/latest")
        .build();

output = client.newCall(request).execute().body().string();
System.out.println(output);
```

```
//CHECK IF SCHEMA IS REGISTERED
request = new Request.Builder()
        .post(RequestBody.create(SCHEMA_CONTENT, EMPLOYEE_SCHEMA))
        .url("http://localhost:8081/subjects/Employee")
        .build();

output = client.newCall(request).execute().body().string();
System.out.println(output);

//TEST COMPATIBILITY
request = new Request.Builder()
        .post(RequestBody.create(SCHEMA_CONTENT, EMPLOYEE_SCHEMA))
        .url("http://localhost:8081/compatibility/subjects/Employee/versions/latest")
        .build();
```

Changing Compatibility Checks

```
// SET TOP LEVEL CONFIG
// VALUES are none, backward, forward and full
request = new Request.Builder()
        .put(RequestBody.create(SCHEMA_CONTENT, "{\"compatibility\": \"none\"}"))
        .url("http://localhost:8081/config")
        .build();

output = client.newCall(request).execute().body().string();
System.out.println(output);

// SET CONFIG FOR EMPLOYEE
// VALUES are none, backward, forward and full
request = new Request.Builder()
        .put(RequestBody.create(SCHEMA_CONTENT, "{\"compatibility\": \"backward\"}"))
        .url("http://localhost:8081/config/Employee")
        .build();

output = client.newCall(request).execute().body().string();
System.out.println(output);
```

Incompatible Change

```
private final static String EMPLOYEE_SCHEMA = "{\n" +
        "  \"schema\": \"" +
        "  {" +
        "    \\\"namespace\\\": \\\"com.cloudurable.phonebook\\\"," +
        "    \\\"type\\\": \\\"record\\\"," +
        "    \\\"name\\\": \\\"Employee\\\"," +
        "    \\\"fields\\\": [" +
        "        {\\\"name\\\": \\\"fName\\\", \\\"type\\\": \\\"string\\\"}," +
        "        {\\\"name\\\": \\\"lName\\\", \\\"type\\\": \\\"string\\\"}," +
        "        {\\\"name\\\": \\\"age\\\",  \\\"type\\\": \\\"int\\\"}," +
        "        {\\\"name\\\": \\\"phoneNumber\\\",  \\\"type\\\": \\\"string\\\"}" +
        "    ]" +
        "  }\"" +
        "}";
```

```
//POST A NEW SCHEMA
Request request = new Request.Builder()
        .post(RequestBody.create(SCHEMA_CONTENT, EMPLOYEE_SCHEMA))
        .url("http://localhost:8081/subjects/Employee/versions")
        .build();

String output = client.newCall(request).execute().body().string();
System.out.println(output);
```

{“error_code":409,"
message":"Schema being registered is incompatible with an e

Incompatible Change

```
private final static String EMPLOYEE_SCHEMA = "{\n" +
        "  \"schema\": \"" +
        "  {" +
        "    \\\"namespace\\\": \\\"com.cloudurable.phonebook\\\"," +
        "    \\\"type\\\": \\\"record\\\"," +
        "    \\\"name\\\": \\\"Employee\\\"," +
        "    \\\"fields\\\": [" +
        "        {\\\"name\\\": \\\"fName\\\", \\\"type\\\": \\\"string\\\"}," +
        "        {\\\"name\\\": \\\"lName\\\", \\\"type\\\": \\\"string\\\"}," +
        "        {\\\"name\\\": \\\"age\\\",  \\\"type\\\": \\\"int\\\"}," +
        "        {\\\"name\\\": \\\"phoneNumber\\\",  \\\"type\\\": \\\"string\\\"}" +
        "    ]" +
        "  }\"" +
        "}";
```

```
//TEST COMPATIBILITY
request = new Request.Builder()
        .post(RequestBody.create(SCHEMA_CONTENT, EMPLOYEE_SCHEMA))
        .url("http://localhost:8081/compatibility/subjects/Employee/versions/latest")
        .build();
```

{"is_compatible":false}

Use Schema Registry

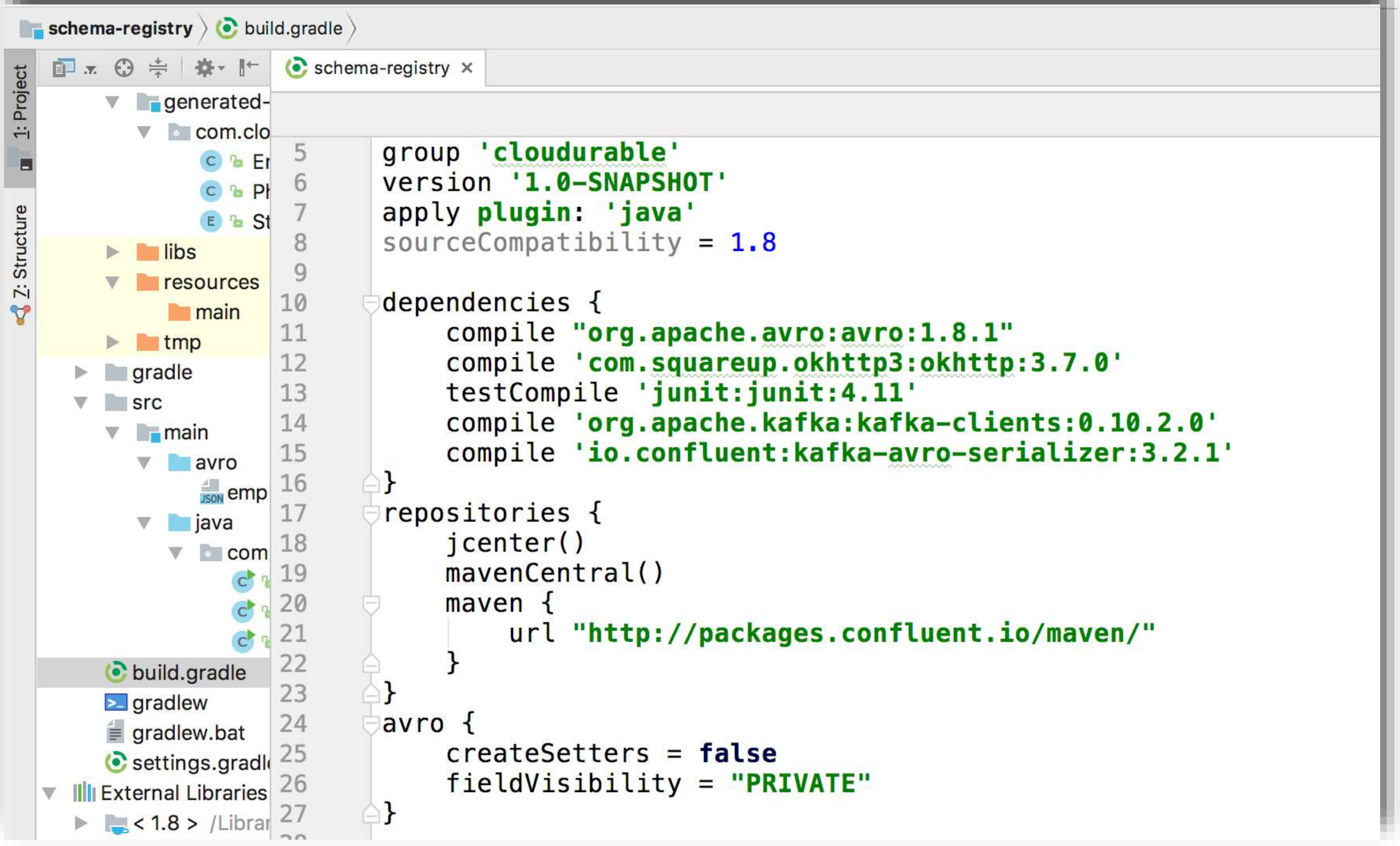
- Start up Schema Registry server pointing to Zookeeper cluster
- Import Kafka Avro Serializer and Avro Jars
- Configure Producer to use Schema Registry
- Use KafkaAvroSerializer from Producer
- Configure Consumer to use Schema Registry
- Use KafkaAvroDeserializer from Consumer

Start up Schema Registry Server

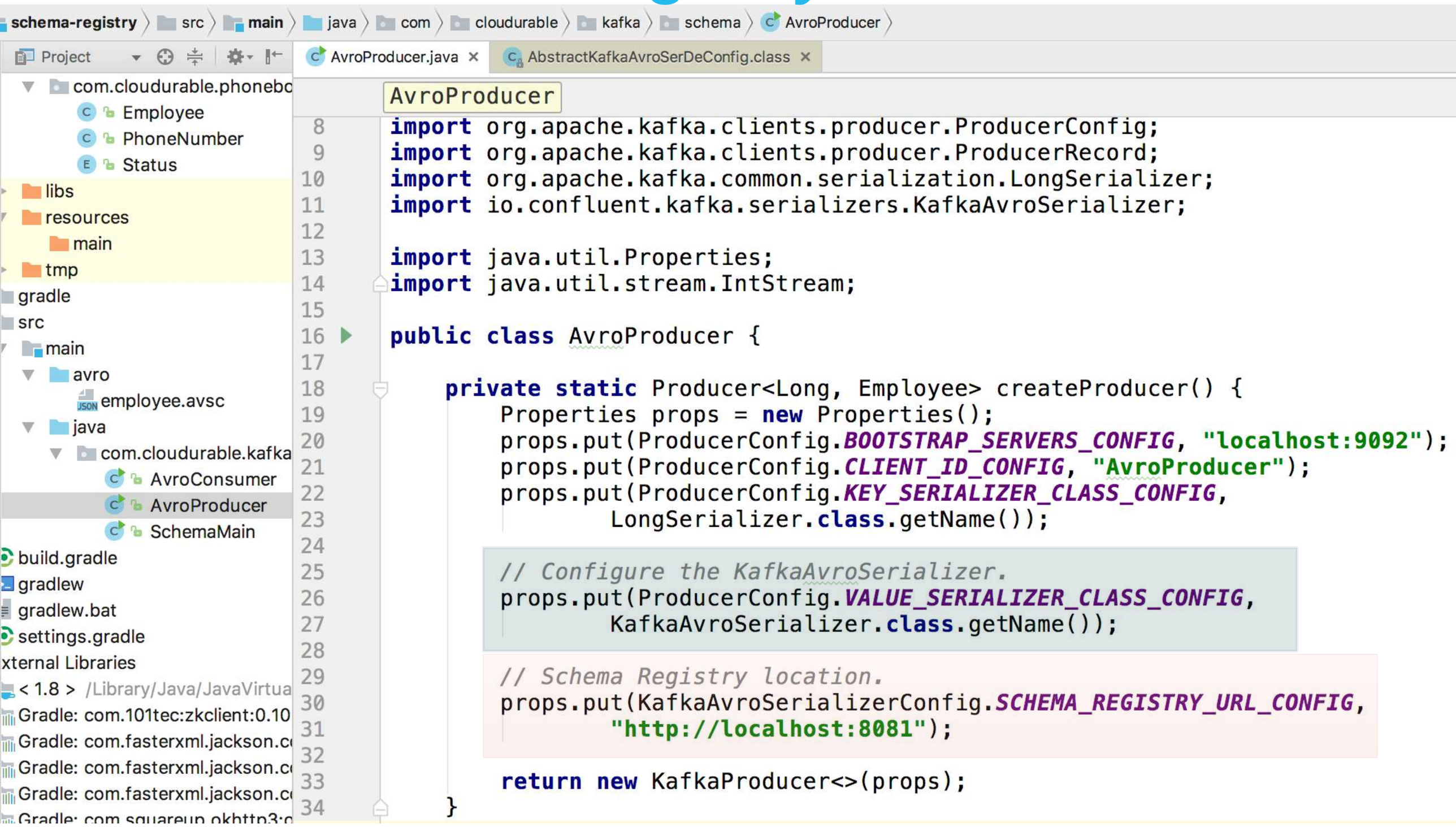
```
cat ~/tools/confluent-3.2.1/etc/schema-registry/schema-registry.properties

listeners=http://0.0.0.0:8081
kafkastore.connection.url=localhost:2181
kafkastore.topic=_schemas
debug=false
```

```
$ bin/schema-registry-start  ~/tools/confluent-3.2.1/etc/schema-registry/schema-registry.properties
[2017-05-09 15:45:34,055] INFO SchemaRegistryConfig values:
        metric.reporters = []
        kafkastore.sasl.kerberos.kinit.cmd = /usr/bin/kinit
        response.mediatype.default = application/vnd.schemaregistry.v1+json
        kafkastore.ssl.trustmanager.algorithm = PKIX
        authentication.realm =
        ssl.keystore.type = JKS
        kafkastore.topic = _schemas
        metrics.jmx.prefix = kafka.schema.registry
```

Import Kafka Avro Serializer & Avro Jars



Configure Producer to use Schema Registry



Use KafkaAvroSerializer from Producer

schema-registry › src › main › java › com › cloudurable › kafka › schema › AvroProducer

Project

- com.cloudurable.phonebc
  - Employee
  - PhoneNumber
  - Status
- libs
- resources
  - main
- tmp
- gradle
- src
  - main
    - avro
      - employee.avsc
    - java
      - com.cloudurable.kafka
        - AvroConsumer
        - AvroProducer
        - SchemaMain
- build.gradle
- gradlew
- gradlew.bat
- settings.gradle

xternal Libraries

- < 1.8 > /Library/Java/JavaVirtua
- Gradle: com.101tec:zkclient:0.10
- Gradle: com.fasterxml.jackson.c
- Gradle: com.fasterxml.jackson.c
- Gradle: com.fasterxml.jackson.c

AvroProducer.java

AvroProducer

```
36      private final static String TOPIC = "new-employees";
37
38      public static void main(String... args) {
39
40          Producer<Long, Employee> producer = createProducer();
41
42          Employee bob = Employee.newBuilder().setAge(35)
43                  .setFirstName("Bob")
44                  .setLastName("Jones")
45                  .setPhoneNumber(
46                          PhoneNumber.newBuilder()
47                                  .setAreaCode("301")
48                                  .setCountryCode("1")
49                                  .setPrefix("555")
50                                  .setNumber("1234")
51                                  .build())
52                  .build();
53
54          IntStream.range(1, 100).forEach(index->{
55              producer.send(new ProducerRecord<>(TOPIC,  key: 1L * index, bob));
56
57          });
58
59          producer.flush();
60          producer.close();
61      }
```

Configure Consumer to use Schema Registry

AvroConsumer.java

```
AvroConsumer
16  public class AvroConsumer {
17
18      private final static String BOOTSTRAP_SERVERS = "localhost:9092";
19      private final static String TOPIC = "new-employees";
20
21      private static Consumer<Long, Employee> createConsumer() {
22          Properties props = new Properties();
23          props.put(ConsumerConfig.BOOTSTRAP_SERVERS_CONFIG, BOOTSTRAP_SERVERS);
24          props.put(ConsumerConfig.GROUP_ID_CONFIG, "KafkaExampleAvroConsumer");
25          props.put(ConsumerConfig.KEY_DESERIALIZER_CLASS_CONFIG,
26                  LongDeserializer.class.getName());
27
28          //Use Kafka Avro Deserializer.
29          props.put(ConsumerConfig.VALUE_DESERIALIZER_CLASS_CONFIG,
30                  KafkaAvroDeserializer.class.getName());  //<----------------------
31
32          //Use Specific Record or else you get Avro GenericRecord.
33          props.put(KafkaAvroDeserializerConfig.SPECIFIC_AVRO_READER_CONFIG, "true");
34
35
36          //Schema registry location.
37          props.put(KafkaAvroDeserializerConfig.SCHEMA_REGISTRY_URL_CONFIG,
38                  "http://localhost:8081"); //<----- Run Schema Registry on 8081
39
40
41          return new KafkaConsumer<>(props);
42      }
43
```

Use KafkaAvroDeserializer from Consumer

```
public static void main(String... args) {

    final Consumer<Long, Employee> consumer = createConsumer();
    consumer.subscribe(Collections.singletonList(TOPIC));

    IntStream.range(1, 100).forEach(index -> {

        final ConsumerRecords<Long, Employee> records =
                consumer.poll( timeout: 100);

        if (records.count() == 0) {
            System.out.println("None found");
        } else records.forEach(record -> {

            Employee employeeRecord = record.value();

            System.out.printf("%s %d %d %s \n", record.topic(),
                    record.partition(), record.offset(), employeeRecord);
        });
    });
}
```

Schema Registry

- Confluent provides Schema Registry to manage Avro Schemas for Kafka Consumers and Producers
- Avro provides Schema Migration
- Confluent uses Schema compatibility checks to see if Producer schema and Consumer schemas are compatible and to do Schema evolution if needed
- Use KafkaAvroSerializer from Producer
- Use KafkaAvroDeserializer from Consumer