Axon 2->3 Migration

Class and package changes

o.a. = org.axonframework

o.a.commandhandling.annotation.CommandHandler	o.a.commandhandling.CommandHandler
o.a.eventhandling.annotation.EventHandler	o.a.eventhandling.EventHandler
o.a.domain.IdentifierFactory	o.a.common.IdentifierFactory
o.a.commandhandling.annotation.TargetAggregateIdentifier	o.a.commandhandling.TargetAggregateIdentifier
o.a.eventsourcing.annotation.AggregateIdentifier	o.a.commandhandling.model.AggregateIdentifier
o.a.eventsourcing.annotation.EventSourcingHandler	o.a.eventsourcing.EventSourcingHandler
o.a.domain.EventMessage	o.a.eventhandling.EventMessage
o.a.saga.annotation.StartSaga	o.a.eventhandling.saga.StartSaga
o.a.saga.annotation.EndSaga	o.a.eventhandling.saga.EndSaga
o.a.saga.annotation.SagaEventHandler	o.a.eventhandling.saga.SagaEventHandler
o.a.saga.AssociationValue	o.a.eventhandling.saga.AssociationValue
o.a.saga.AssociationValues	o.a.eventhandling.saga.AssociationValues
o.a.saga.annotation.AssociationValuesImpl	o.a.eventhandling.saga.AssociationValuesImpl

Aggregates

Aggregates no longer extend the AbstractAnnotatedAggregateRoot.

To apply events a static import is needed:

```
import org.axonframework.commandhandling.model.AggregateLifecycle;
```

When using Spring, don't forget to annotate the aggregates with @Aggregate.

Sagas

Sagas no longer extend the AbstractAnnotatedSaga.

If you need to call the end() method, it needs to be imported like:

```
import org.axonframework.eventhandling.saga.SagaLifecycle.end;
```

When using Spring, don't forget to annotate the aggregates with @Saga.

Snapshotter

When you want to enable snapshots (or caching for that matter) on your aggregates, you need to define the repositories explicitly. (See Configuration)

Mongo – dependency & cleanup

o.a.mongo3.eventstore.DefaultMongoTemplate	o.a.mongo.eventsourcing.eventstore.DefaultMongoTemplate
o.a.mongo3.eventstore.MongoEventStore	o.a.mongo.eventsourcing.eventstore.MongoEventStorageEngine

We are using MongoDB as our datastore.

Before running the Axon 3 version of the project the database needs to be cleaned because the mongo client will try to recreate indexes that already exist (but slightly different of course).

We always dropped all indexes on the domainevents and sagas tables. And all snapshots are unusable, so we removed them too.

Axon tests

The Axon test classes have been refactored a lot. We re-wrote our test changing the fixtures from the FixtureConfiguration class to AggregateTestFixture classes. Then instead of:

```
fixture = Fixtures.newGivenWhenThenFixture(ToBeTested.class);
```

you can instantiate it like:

```
fixture = new AggregateTestFixture(ToBeTested.class);
```

The same applies to Saga testing where the AnnotatedSagaTestFixture is replaced by a SagaTestFixture. We neede to re-write some test because of the changes in the fixtures. As an example:

Old code:

```
fixture.givenAggregate(aggregateId).published()
        .whenAggregate(aggregateId).publishes(new Event(x))
        .expectActiveSagas(1)
        .expectDispatchedCommandsEqualTo(
                new ValidateAggregateCommand(aggregateId)
        );
```

New code:

```
fixture.whenPublishingA(new Event(x))
        .expectActiveSagas(1)
        .expectDispatchedCommands(
                new ValidateAggregateCommand(aggregateId)
        );
```

Most of the time the changes are pretty straightforward to make, but it will take some time and tweaking.

CommandCallback

The CommandCallback now also contains a CommandMessage parameter.
We needed to change our CustomCallback classes to cope with it, without actually looking at the messages returned ☺.

UnitOfWork

o.a.unitofwork.UnitOfWork	o.a.messaging.unitofwork.UnitOfWork
o.a.unitofwork.CurrentUnitOfWork	o.a.messaging.unitofwork.CurrentUnitOfWork

Working with UnitOfWork has also changed a bit. There is no longer the need to call registerListener. Instead directly implement the methods. Example:

```
CurrentUnitOfWork.get().registerListener(new UnitOfWorkListenerAdapter() {
    @Override
    public void afterCommit(UnitOfWork unitOfWork) {
        super.afterCommit(unitOfWork);
        // do other work
    }
});
```

New code:

```
CurrentUnitOfWork.get().onCommit(unitOfWork ->  {
        // do other work
});
```

Configuration

Axon 3 introduces the Spring Boot Autoconfiguration, which can easy the configuration on new projects. On an existing project you can use it to replace a lot of custom configuration code in your project.

We basically removed all our axon configuration code and started from the autoconfiguration again.

The autoconfiguration gives a working system, but needs to be adapted to your needs.
To use snapshot and/or caching you need to define your own repository bean. There is no option (yet) to add this to the annotation of the aggregate.

```
@Bean
public Repository<MyAggregate> myAggregateRepository(
                EventStore eventStore,
                SnapshotTriggerDefinition snapshotTriggerDefinition,
                Cache cache) {
    return new CachingEventSourcingRepository<>(
        new GenericAggregateFactory<>(MyAggregate.class), eventStore,
        cache, snapshotTriggerDefinition);
}
```

Using this naming convention, the autoconfiguration will pick up the repository for your aggregate automatically.

Saga serialization

One of the issues we ran into was the saga-serialization.

In Axon 2.x, the sagas are serialized by default with a JavaSerializer. This is not very upwards compatible, so we had to implement a migration serializer before the upgrade to Axon 3.

We choose to create a serializer that would try to deserialize using the XStreamSerializer and if that would fail, try it with a JavaSerializer. Serializing would always be with the XStreamSerializer. By loading and saving all saga-instances all sagas were serialized using the XStream xml format, that can be used in Axon 3 too.

To avoid loading errors in Axon 3 we needed to tell the serializer to ignore the fields from the AbstractAnnotatedSaga that are present in the Axon 2 serialized form:

```
@Bean(name = "sagaStore")
public MongoSagaStore sagaStore() throws Exception {
    XStreamSerializer serializer =  new XStreamSerializer();
    serializer.getXStream().registerConverter(new JodaConverter());
    //fields are present in Axon 2.x sagas, so ignore them when reading
    serializer.getXStream().ignoreUnknownElements("associations");
    serializer.getXStream().ignoreUnknownElements("identifier");
    serializer.getXStream().ignoreUnknownElements("isActive");
    return new MongoSagaStore(getAxonMongoSagaTemplate(), serializer);
}
```

The JodaConverter is there because of next issue:

DateTime serialization

In Axon2 the Date/DateTime classes were serialized using an ISO8601 representation. In Axon3 however this is changed to a child element <iMillis>, due to the move to the newer java.time classes. . Loading aggregates using the old notation will lead to empty date fields (or 1-1-1970). To be able to

read the old events without changing the data we implemented a custom Joda DateTime converter that will (de-)serialize using the ISO8601 format again.

The serializer needs to be added to both the saga- and the eventstore serializer.