How (Not) to Start With Cassandra

By **Nenad Bozic**, dzone.com
March 19th, 2016

Within several previous projects, we have held consultations for development teams coming from the relational world and adopting Apache Cassandra. Experience gained from those projects has triggered me to publish my learning guidelines for this database. We see a pattern when adopting Cassandra and it involves taking a shortcut rather than in-depth understanding of how Cassandra works. I will try to address those shortcuts and red flags and give our advice on how to start.

The pattern is as follows: some developer in the team follows the trends and reads that Cassandra works great with a write-heavy data load, sensor data, messages from IoT etc. There is a requirement that part of the system will need distributed scalable storage. The most senior developer makes a decision, Cassandra is hot and popular—it fits the use case—and so, they go with it. The background of the development team is usually in the relational world. So, they go with it. They install Cassandra on one node, do not have a need for more at the moment, and start developing **(red flag one)**. Data modeling is the next task and, from a relational world experience, that is something picked along the way, you do not need to put too much effort in it since database objects resemble the domain objects **(red flag two)**. So, they start to model tables for single entities, each of them having a generated id, and join data by id at the application level. It is said that Cassandra is scalable and fast, so adding more nodes will overcome the poorly modeled data. The data model starts talking, joining data at the application level in huge lists becomes hard, making queries to many tables becomes overly complex, the team is working hard to keep the data in different tables in sync. Even when problems start manifesting in slow reads, the development team decides not to change the data model because it is common in the relational world to stick with your tables after you initially model them **(red flag number three)**. The application is becoming more and more popular, and then there is a need for scaling. The development team starts adding nodes, and they realize that Cassandra is not as scalable as it should be. The truth is, if you start with 3 nodes instead of one, you will choose NetworkTopologyStrategy over SimpleStrategy so you will have one less problem scaling out. This is just one example from many of those complicating the process of scaling out when using single node at the beginning.

Red Flag Number One — Using Cassandra on Single Node

This problem is not that specific to Cassandra itself, but it is a problem of every distributed product. Whenever you work with distributed systems, make sure you work with at least three nodes from the beginning. Problems in distributed systems are completely different from problems on single node systems, just to name a few: consuming messages exactly once, saving files to file systems, scheduling, locking. The point to take here is that you should always start developing against a truly distributed system, if you have a need for distributed technology like Cassandra, start a three node cluster from the beginning and develop against it. If you use a single node cluster, you will pay in the long run when you have the need to scale out.

Red Flag Number Two — Relational World Data Modeling

Cassandra is specific, just as every NoSQL database is. In order to achieve efficiency, you need to know how it works, both externally and internally. Relational databases hide a lot of details from developers, and you do not need to know how data is stored on disk to make efficient queries. In Cassandra, it is important to use query based modeling, you need to know how you will read your data before modeling it. It is a distributed system, so data should be denormalized and replicated, with a lot of duplications. It is a common thing to see the same data spread across many tables, each optimized for a different read query.

Red Flag Number Three — Evolving Data Model

Before we started working with Cassandra, we had solved problems of a not-so-perfect data model with an additional query or additional join. Here you do not have the commodity of joins on the one hand, and you must optimize the data model for reading on the other hand. As stated in the previous red flag, you need the best data model for each feature, so if the feature requirements are changed, the data model should be changed also. In query based modeling, the data model is exposed all the way up (this is a tradeoff to achieve performance) so it should evolve as feature requirements evolve. That was the prime reason why we came up with Cassandra Migration Tool, a supporting library which stores the database version and eases up Schema and Data changes.

Learning Guidelines

So, how to overcome these obstacles and make the adoption of this new technology a smooth ride? First of all, watch the Introduction to NoSQL by Martin Fowler; this is not a prerequisite for Cassandra, but it is a great video explaining NoSQL databases and why they exist. Cassandra was developed based on two storage engines, Google BigTable and Amazon DynamoDB, so my suggestion is to follow up with these two whitepapers.

Now that you have enough background information, you are ready for DataStax Academy. They have free online courses which will bring you up to speed in no time. As a bare minimum, I would advise watching DS201: Cassandra core concepts and DS220: Data Modeling. As the last step, there is a great four video series from Patrick McFadin, DataStax Cassandra Evangelist, on Data Modeling. He talks a lot about real world examples and advanced techniques for achieving great data models.

This is our regular learning path in SmartCat; it is just tip of the iceberg, but it gets you to a place where you understand decisions you make regarding Cassandra. Hope it will help in overcoming some of the obstacles when starting to work with this great database. It is more of a time-consuming path, but after watching those videos, it will definitely pay off in the long run. It will be much clearer why some decisions were made, why the data model needs to evolve with the way the data is read, and why knowing internals is so important for its performance.

We are interested to hear your experiences and would like to hear which learning path you have taken.