**Database Delivery – State based vs Migration based**

BY [SAMIR BEHARA](https://dotnetvibes.com/author/samirbehara/) *on* [APRIL 16, 2018](https://dotnetvibes.com/2018/04/16/database-delivery-state-based-vs-migration-based/) • ( [1](https://dotnetvibes.com/2018/04/16/database-delivery-state-based-vs-migration-based/#comments) )

 https://dotnetvibes.com/2018/04/16/database-delivery-state-based-vs-migration-based/

2 Votes

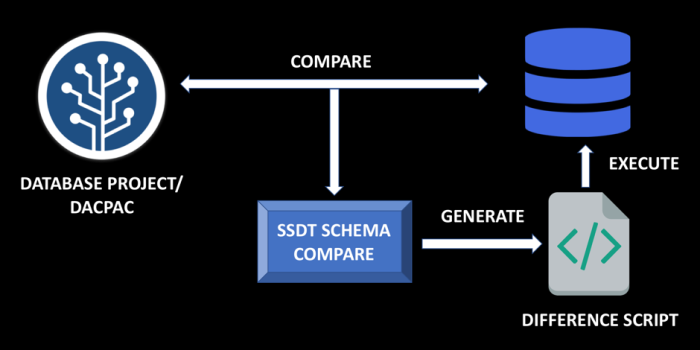
For database development and deployment there are primarily 2 delivery mechanisms — State based and Migration based approach. In this article, we will understand in details about these 2 approaches and identify use cases in real world & scenarios where they fit and not.

**State Based Approach**

In State based database delivery, you will need to store a snapshot of the current state of the database in your source control. Every table, stored procedure, view, trigger will be saved as separate sql files which will be the real representation of the state of your database objects.

When you want to modify the database schema or reference data, all you need to do is explicitly mention the expected database state. The Compare Tool does all the magic and generates the difference script which then gets executed against the database.

**The most important thing to note is that in State based approach the system of truth is the Source Code itself.** Application developers would love this because it is very similar to making changes to your class files, compile and build it. Another good thing is that you do not have to deal with ALTER scripts with State based approach – the Schema/Data Compare Tool takes care of generating the ALTER scripts and runs it against the target database without any manual intervention.



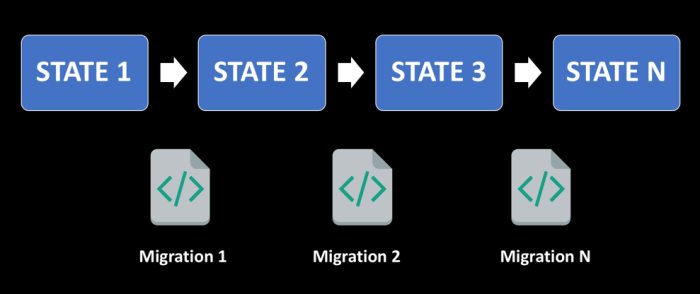
*If you want to implement State based approach, you can go with Database Projects in Visual Studio which uses SQL Server Data Tools.*

**Migration Based Approach**

In Migration Based Database Delivery, when you start working on a project you capture the current state of the database in a single migration script stored inside source control and that becomes the base/initial state.

As the project evolves over time, you will need to make changes to your database schema and reference data. On every instance, you will need to create a migration script with an incremental version number. If you want to determine the current state of your database or recreate the database in another environment, you will need to run all the migration scripts in the correct order.

**The system of truth in Migration based approach is the Database itself.** This approach provides more fine grain control on your scripts if you have to deal with data intensive operations and TSQL querying. You have the authority to review the scripts which gets executed against your destination database and modify as required.



*If you want to implement Migration based approach, you can go with RedGate ReadyRoll – which is part of the Visual Studio 2017 Enterprise edition or even the ReadyGate SQL Toolbelt. You can also go with Flyway – which is a very popular open source database migration tool.*

You can read more about Flyway here —  
[Continuous Database Integration with Flyway](https://dotnetvibes.com/2018/04/06/continuous-database-integration-with-flyway/)

**State Based vs Migration Based and its Use Cases**

**Scenario 1**  
You are working on a Greenfield Project, and the development effort is approximately 6 months before it is deployed to Production. During this period, you will need to make multiple changes to the database schema.

In this scenario, having Migration based approach might not be a good choice, since you will end up having a long list of migration scripts which might not be beneficial to keep track of. State based approach provides the developers with lot of flexibility in modifying the schema and not worry about maintaining a number of scripts in the source control.

**Scenario 2**  
With State based approach, you can have your schema inside the Source Control which gives developer a better insight and more convenience to look at the current state of the database. If there are compile time errors inside your SQL files, you will know it. It enhances the overall development experience for developers from inside the Visual Studio IDE.

**Scenario 3**  
You are working on a team who has less expertise on SQL. Having them write the ALTER scripts or some customized refactoring scripts might have a chance of being error-prone. State based migration is preferred in such cases.

**Scenario 4**  
As the project evolves, the number of migration scripts inside the source control increases. In Migration based approach the ordering of scripts in which the scripts need to run is very critical.   
With State Based approach you do not need to deal with either these additional migration scripts or the order of execution.

**Scenario 5**  
If you have a large sized Team working on a data intensive project, where multiple developers might make changes on the same file, then Migration based approach might create additional issues.  
It might result in either overriding the changes made by another developer or it might also result in Merge conflicts and having to deal with it manually is additional complexity and error prone.

**Scenario 6**  
Over a period of time, deployments for Migration based approach will take a longer amount of time, since there are chances that a number of changes on an object are applied one by one. In State based approach, the source control stores the final version of the object and hence can be applied directly to the database in no time – as a result it is fast.

**Scenario 7**  
If you need more fine grain control on your migration scripts, then Migration based approach is the way to go. In State based approach, you cannot modify the difference script which is generated by SSDT before getting executed against the database.

**Scenario 8**  
If you have to deal with more data intensive refactoring and scripting, Migration based approach can handle it like a champ – since you have the ability to review and edit the migration scripts before it gets executed against the database. You can leverage your T-SQL expertise to write better scripts than you think the internal compare team would generate. With State Based approach, complex refactoring might need multiple steps to complete.

To wrap up this article, you learned that both State based and Migration based approach helps in doing seamless database deployments. It is a common question as to which approach is better. I would say that it depends upon your project and team requirements. Each approach has its strengths, so go with the approach which will allow you to be more productive and assist in doing reliable database deployments.

Which database delivery approach are you using in your projects? I would be happy to know your thoughts and have a discussion in case you have any questions.

Great post. It hits on a topic that many are thinking about these days.

There has been a lot of analysis on what approach is best for Agile / DevOps teams. You cover a lot of the benefits of the state-based approach, but by and far, the migration-based approach actually has a lot of advantages over the state-based approach for teams looking to go fast. See Martin Fowler’s take on this: <https://www.martinfowler.com/articles/evodb.html>. The key points are: by making the change a first class artifact, you can “build once, deploy often” (as opposed to something new that is generated prior to each deployment). Another key point is that with migrations, you encourage small, incremental changes (per Agile/DevOps philosophy). Lastly, it’s much easier to support parallel development strategies with migrations – in part because the migrations themselves are small, incremental changes.

Also, it’s worth listing Liquibase (<http://www.liquibase.org/>) as a key open source project – it has a substantial following. Beyond ReadyRoll, on the commercial side, Datical (<https://www.datical.com/>) is a major migration-based solution.