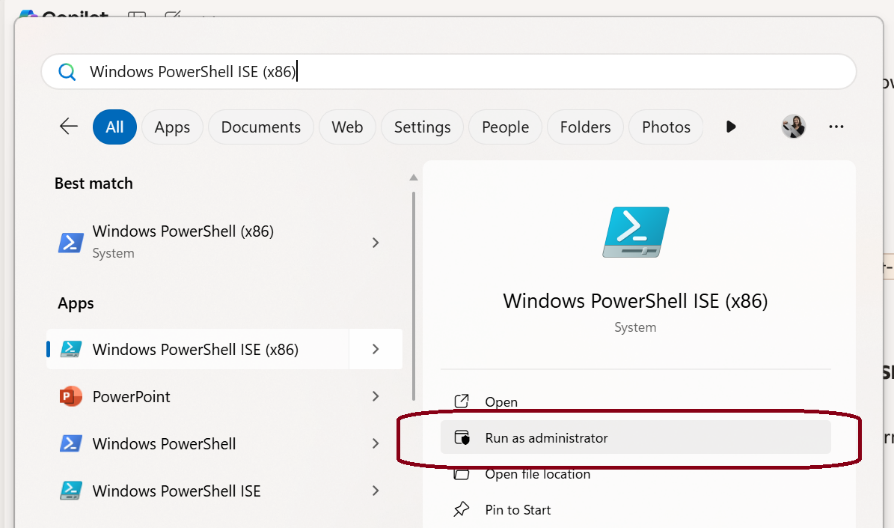
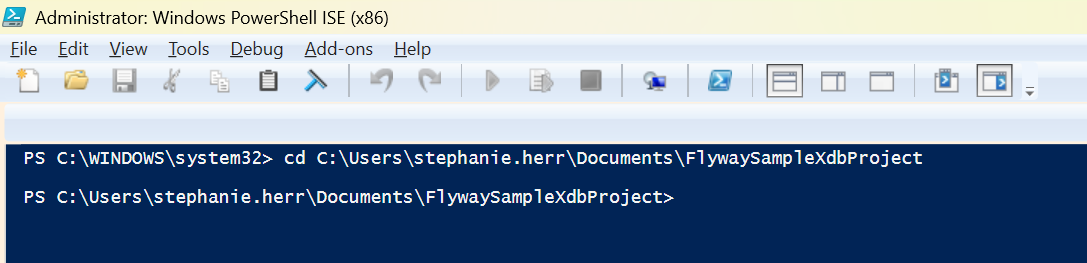
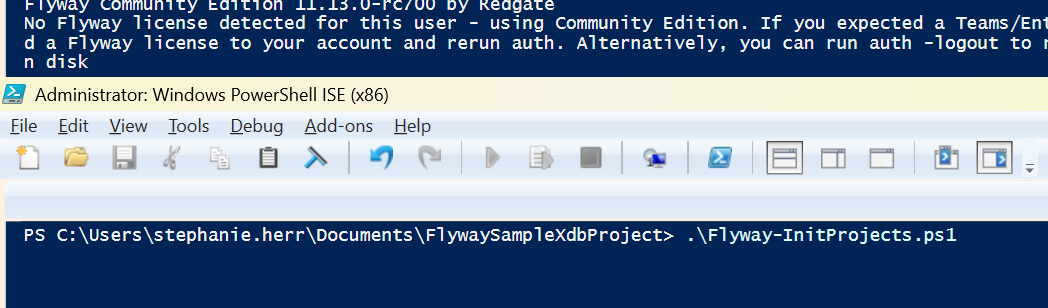
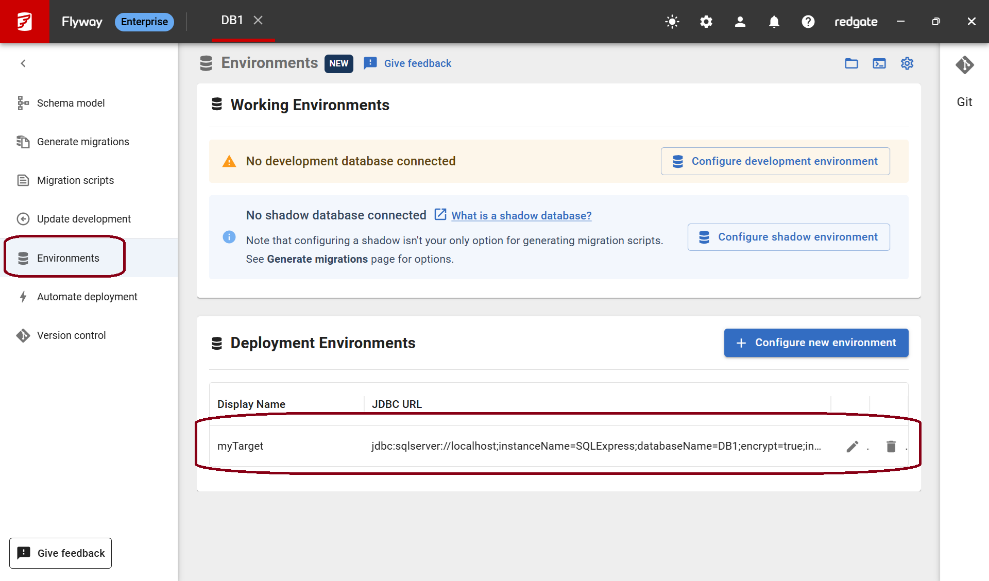
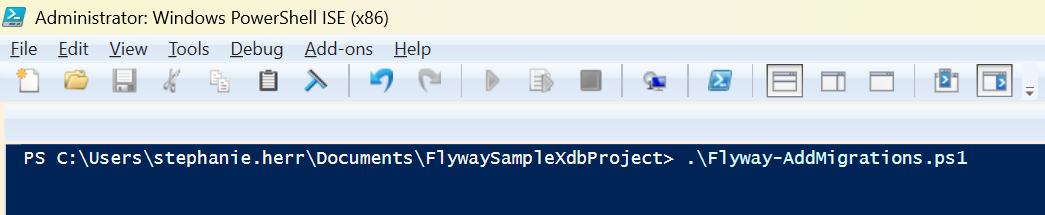
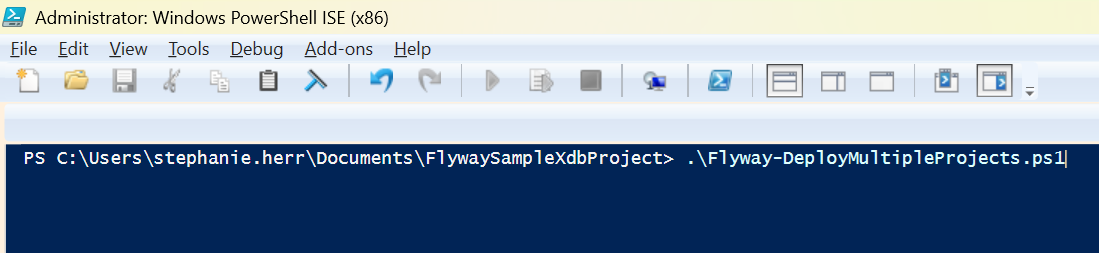
# Dealing with x-database dependencies in Flyway

Flyway is an amazing solution for tracking changes to databases and automating database deployments. But, Flyway is limited to working on only one database at a time in SQL Server. We can work around this by scripting flyway to deploy to many different targets. With some retry logic, we can fix *most* dependency issues between the databases.

Note: In some databases, like Oracle and PostgreSQL, Flyway can work across multiple schemas, which are sometimes treated the same as separate databases in SQL Server speak. If you are using pdbs in Oracle, then this trick may also help you.)

We are going to work through a very simple example of how this can work in Flyway.

1. In SQL Server, **create 5 sample databases**:  
   *This example assumes you’re on localhost. If you are not using localhost, you’ll need to update connection strings later.*  
   CREATE DATABASE DB1;  
   CREATE DATABASE DB2;  
   CREATE DATABASE DB3;  
   CREATE DATABASE DB4;  
   CREATE DATABASE DB5;
2. Clone this repository to a folder called **FlywaySampleXdbProject** on your machine. For this example, we use the Documents folder.
3. Open PowerShell by right-clicking and **Running as administrator**, so there will not be any problems creating directories. I’m using the Windows PowerShell ISE (x-86) on my Windows machine:  
   
4. Navigate to your **FlywaySampleXdbProject** directory:  
   cd C:\Users\stephanie.herr\Documents\FlywaySampleXdbProject  
   
5. Run the **Flyway-InitProject.ps1** script:  
     
   1. You should now have 5 new directories in your FlywaySampleXdbProjects folder.  
      A screenshot of a computer

      AI-generated content may be incorrect.
   2. These projects can be opened in Flyway Desktop. You’ll see the connection information for the target database.  
      
6. Run the **Flyway-AddMigrations.ps1 script**:  
     
     
   This script is adding a versioned migration script (V1 script) in each project that will create a Sample table and a procedure that selects everything from that table.  
     
   It also creates an additional versioned migration script (V2 script) in the DB2 project. This script is creating a view for the Sample table in DB4. This is creating a dependency between DB2 and DB4. DB4 needs to be migrated before DB2 in this case since DB2’s new view depends on the table in DB4.
7. Run the **Flyway-DeployMultipleProjects.ps1 script**:  
     
     
   This script tries to deploy each project. If a deployment fails, it will try again, up to 5 times. You can change the number of times it retries in the file.  
     
   In this example, DB2 fails on the first pass since DB4 hasn’t been migrated yet. On the second pass, the deployment to DB2 succeeds and the script stops.

This works well for databases like SQL Server that work with explicit transactions. Flyway tracks every migration script in the target database (flyway\_schema\_history table) to know what scripts have been executed against it (and when) and also what scripts are pending, meaning they still need to be executed on that target. So, in this example, the V2 script fails, any changes are rolled back, and nothing is entered in the tracking table. The script would still be pending execution from Flyway’s perspective.

**Things to note**

1. If you know you have a core database that other databases rely on, make sure you deploy to the core database first before the other ones.
2. This example handles dependencies between databases, but it might not handle circular dependencies. You may have to break your release up into multiple steps, deploying to DB1 first, then DB2, then back to DB1 and back to DB2. There could be some additional tricks with [executing scripts outside of transactions](https://documentation.red-gate.com/fd/flyway-execute-in-transaction-setting-277578997.html) and idempotent scripts, but you would need to be sure you’re happy with this. Executing outside of transactions is not recommended in case something unexpected happens.
3. This script is executing each deployment in sequence. You could use a CI/CD system, like Azure DevOps, GitHub, GitLab, etc., to have the deployments run in parallel and retry the deployments to all environments after the first round completes. This could help speed up the deployment time.
4. Finally, and probably most importantly, if a deployment is still failing after the maximum number of retries, you need to be ready for this. Now, some of your databases are deployed, but another one may still be broken. How will you recover in this instance? Are the changes independent and this is ok until you get a fix? Can you restore previous versions of all your databases? Can you rollback or rollforward across all your databases? If possible, try to have a pre-Production or Staging environment that gives you a realistic rehearsal of the multi-database deployment to catch and fix any issues before you try to deploy to Production.

**Conclusion**

We hope this simple example gave you some ideas about how to handle more complicated deployments that involve multiple databases, especially when the scripts have dependencies across the different databases. If you have this problem, or any suggestions on improving this approach, [we’d love to hear from you](mailto:DatabaseDevOps@redgate.com?subject=Deploying%20x-database%20dependencies).