Migrate PostgreSQL Single Server to Flexible Server using Azure CLI

Contents

Migrate PostgreSQL Single Server to Flexible Server using Azure CLI 1
- **Pre-requisites** 1
- **CLI Commands** 1
- **Create Migration** 2
- **List Migrations** 6
- **Show Details** 6
- **Update Migration** 8
- **Delete Migration** 9
- **Post Migration** 9

This article shows you how to create a migration from your Azure database for PostgreSQL single server to flexible server using our automated migration service through Azure CLI.

Pre-requisites

Azure CLI Setup

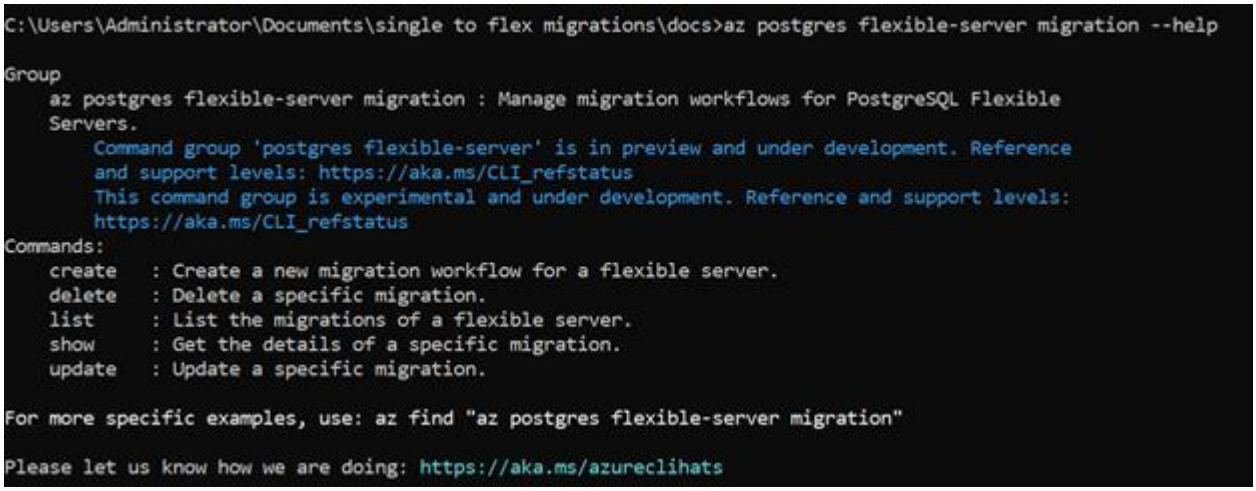
- Firstly, install the latest Azure CLI for your corresponding operating system from the Azure CLI install page
- In case Azure CLI is already installed, check the version by issuing **az version** command. The version should be at least **2.28.0 or above** to use the CLI commands. If not, please update your Azure CLI using the following link.
- Once installed, run the **az login** command. This will open the default browser and load an Azure sign-in page to authenticate. Pass in your Azure credentials to do a successful authentication. For other ways to sign with Azure CLI, visit this link.

Make sure to take care of the pre-requisites listed here which are necessary to get started with the automated migration service.

CLI Commands

The automated migration service comes with a list of easy-to-use CLI commands to do migration-related tasks. All the CLI commands start with **az postgres flexible-server migration**. There are also help statements provided that assist you in understanding the various options and in framing the right syntax for the CLI commands.

az postgres flexible-server migration --help gives you the following output.



It lists out the name of the commands and the corresponding verbs that are supported.

Create Migration

The create migration command helps in creating a migration workflow from the source to target servers. The CLI command to create a migration is given below

```
az postgres flexible-server migration create --subscription <sub_id> --resource-group <rg_name> --name <target_server_name> --migration-name <name_of_migration> --properties "<absolute_path_of_jsonfile>"
```

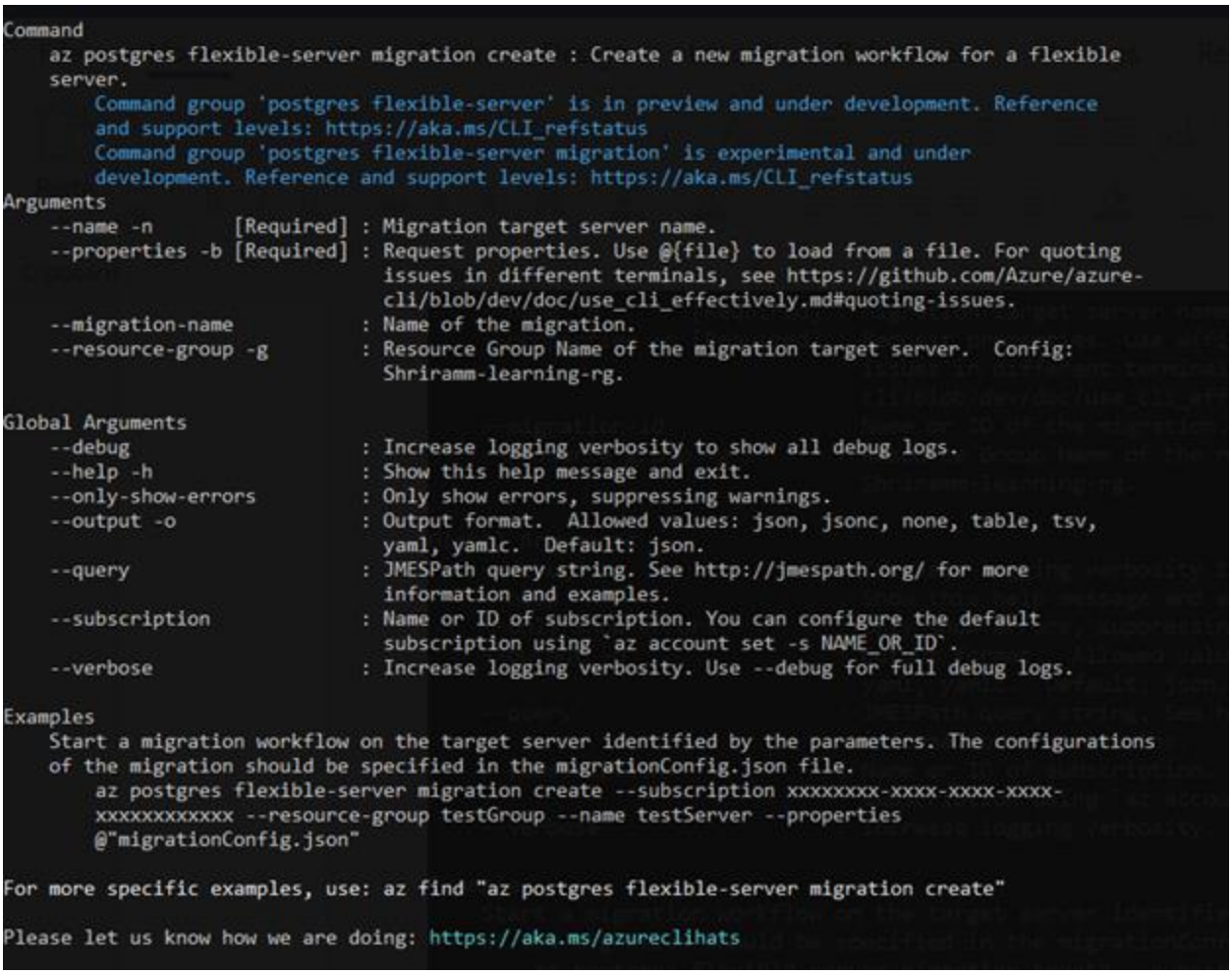
The parameters marked in yellow needs to be replaced with corresponding values of your target server.

- **sub_id -** Subscription ID of the target flexible server
- **rg_name** - Resource group of the target flexible server
- **target_server_name** - Name of the target flexible server
- **name_of_migration** - Unique name to identify a migration between the source and target servers
- **absolute_path_of_jsonfile -** Path to the json file which has information about the source single server

For example:

```
az postgres flexible-server migration create --subscription 5c5037e5-d3f1-4e7b-b3a9-f6bf9asd2b30 --resource-group my-learning-rg --name myflexibleserver --migration-name migration1 --properties "C:\Users\Administrator\Documents\migrationBody.json"
```

az postgres flexible-server migration create -- help gives the following result



It clearly calls out the expected arguments and has an example syntax that needs to be used to create a successful migration attempt from the source to target server. You can assign a name to the migration workflow by passing a value to the **migration-name** argument. The **migration-name** argument is unique to the target server. No two migrations for a target server can have the same migration name. The **migration-name** argument is also used in other CLI commands such as **update, delete, show** as an identifier to the migration workflow to perform the corresponding actions.

The **create migration** command needs a json file to be passed as part of its **properties** argument.

The structure of the json is given below.

Contents of migrationBody.json:

```
{
"properties": {

"SourceDBServerResourceId":"/subscriptions/<subscriptionid>/resourceGroups/<s
rc_ rg_name>/providers/Microsoft.DBforPostgreSQL/servers/<source server
name>",

"SourceDBServerFullyQualifiedDomainName": "fqdn of the source server as per
the custom DNS server",
"TargetDBServerFullyQualifiedDomainName": "fqdn of the target server as per
the custom DNS server"

"SecretParameters": {
    "AdminCredentials":
    {
       "SourceServerPassword": "<password>",
       "TargetServerPassword": "<password>"
    },
"AADApp":
    {
        "ClientId": "<client id>",
        "TenantId": "<tenant id>",
        "AadSecret": "<secret>"
     }
},

"MigrationResourceGroup":
    {
      "ResourceId":"/subscriptions/<subscriptionid>/resourceGroups/<temp_rg_n
ame>",
      "SubnetResourceId":"/subscriptions/<subscriptionid>/resourceGroups/<rg_
name>/providers/Microsoft.Network/virtualNetworks/<Vnet_name>/subnets/<subnet
_name>
    },

"DBsToMigrate":
   [
                  "<db1>","<db2>"
   ],

"SetupLogicalReplicationOnSourceDBIfNeeded": "true",

"OverwriteDBsInTarget": "true"

}

}
```

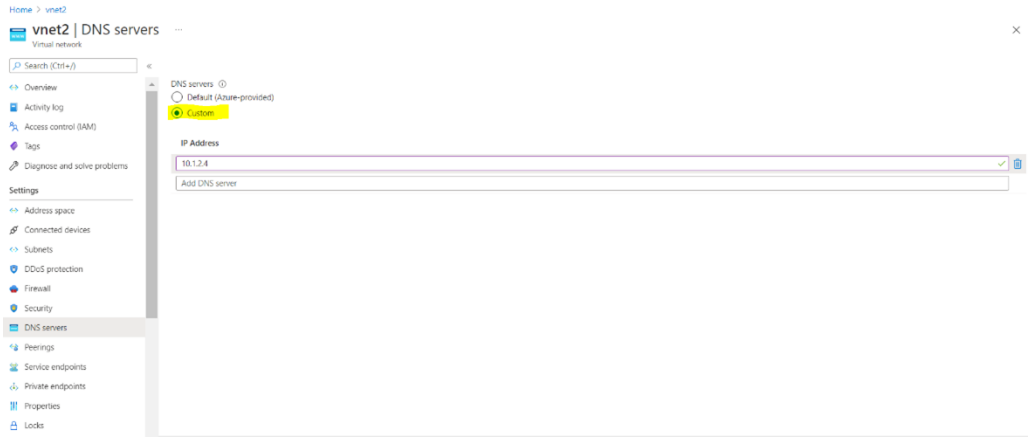
Please note that all the properties marked in yellow are optional.

Let us take a deep dive of the various parameters in the json file.

- **SourceDBServerResourceId** – This is the property for the resource ID of the single server and is mandatory
- **SourceDBServerFullyQualifiedDomainName** - This is an optional property and should only be used when a custom DNS server is used for name resolution inside a VNet. The FQDN of the single server as per the custom DNS server should be provided for this property.

  How to figure out if a custom DNS server is being used for the name resolution?

  Navigate to your VNet in the portal and click on DNS server. It should indicate if it is using a custom DNS server or default Azure provided DNS server.



- **TargetDBServerFullyQualifiedDomainName -** This is an optional property and should only be used when a custom DNS server is used for name resolution inside a VNet. The FQDN of the flexible server as per the custom DNS server should be provided for this property.
- **SecretParameters** – All the fields under the SecretParameters sections are mandatory. They help to authenticate against the source and target servers and help in checking proper authorization access to the resources.
- **MigrationResourceGroup** – This section consists of two properties
  - **ResourceID -** The automated migration service creates DMS and other network infrastructure components on the fly to migrate data and schema from the source to target. By default, all the components created by this service will be under the resource group of the target server. If you wish to put them under a different resource group, then you can assign the resource ID of that resource group to this property.
  - **SubnetResourceID** - In case if your source has public access turned OFF or if your target server is deployed inside a VNet, then you would want to specify a subnet under which DMS needs to be created so that it can connect to both source and target servers. This property comes in handy to specify the subnet in which the DMS should be created in case of private access of your source or target servers.

- **DBsToMigrate -** This is the property where you specify the list of databases you want to migrate to the flexible server. You can include a maximum of 8 database names.
- **SetupLogicalReplicationOnSourceDBIfNeeded -** Logical replication pre-requisite can be enabled on the source server automatically by setting this property to **true**.
- **OverwriteDBsinTarget -** If the target database happens to have an existing database with the same name as the one you are trying to migrate, the migration will pause until the you acknowledge that overwrites in the target DBs are allowed. This can be avoided by giving the migration service the permissions to automatically overwrite databases by setting the value of this property to **true**.

You can download the template json files using the following link

- JSON template for Public Access(both source and target)
- JSON template for Private Access(target inside a VNet or public access turned off at the source with a private endpoint configured)

List Migrations

This command lists down the migration attempts that were done to a particular target flexible server. The CLI command to list migrations is given below

```
az postgres flexible-server migration list --subscription <sub_id> --
resource-group <rg_name> --name <target_server_name> --filter Active
```

There is a parameter called filter and it can take **Active** and **All** as values.

- **Active** – Lists down the current active migration attempts for the target server. It does not include the migrations that have failed/cancelled/succeeded before.
- **All** – Lists down all the migration attempts to the target server, which includes both the active and past migrations irrespective of the state.

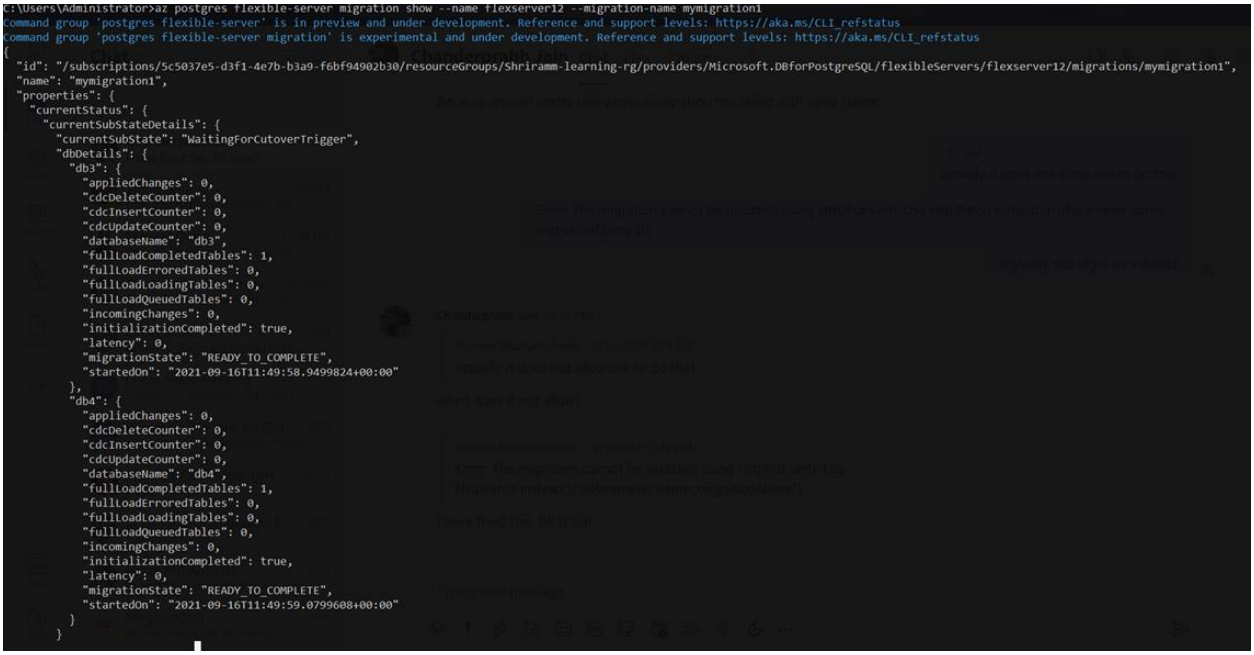
Use the **az postgres flexible-server migration list -- help** for any additional information.

Show Details

This command gets the details of a specific migration. This includes information on the current state and substate of the migration workflow. The CLI command to show the details of a migration is given below:

```
az postgres flexible-server migration show --subscription <sub_id> --
resource-group <rg_name> --name <target_server_name> --migration-name
<migration_name>
```

The **migration_name** is the name assigned to the migration workflow during the **create migration** command. Here is a snapshot of the sample response from the **Show Details** CLI command.



Some important points to note on the command response –

- As soon as the **create migration** command is triggered, the migration moves to the **InProgress** state and **PerformingPreRequisiteSteps** substate. In general, it takes around 8 to 10 minutes for the migration workflow to move out of this substate since it takes some time to create a DMS, add its IP on firewall list of source and target servers and to perform a few maintenance tasks.

- After the **PerformingPreRequisiteSteps** substate, the migration moves to the substate of **Migrating Data** where the dump and restore of the databases take place.

- Each DB being migrated has its own section with all migration details such as table count, success rate, number, and type of changes, etc.

- Possible migration states include
    - **InProgress**: The migration infrastructure is being setup or the actual data migration is in progress.
    - **Canceled**: The migration has been cancelled or deleted.
    - **Failed**: The migration has failed.
    - **Succeeded**: The migration has succeeded and is complete.
    - **WaitingForUserAction** : Migration is waiting on a user action. This state has a list of substates, which are given under the sub-states section. The substates for this state start with the word **Waiting**.

- Possible migration substates include
    - **PerformingPreRequisiteSteps**: Infrastructure is being set up and is being prepped for data migration
    - **MigratingData** : Data is being migrated
    - **CompletingMigration** : Migration cutover in progress

- **WaitingForLogicalReplicationSetupRequestOnSourceDB** : Waiting for logical replication enablement. You can manually enable this manually or enable via the update migration CLI command covered in the next section.
- **WaitingForCutoverTrigger** : Migration is ready for cutover. You can start the cutover when ready.
- **WaitingForTargetDBOverwriteConfirmation** : Waiting for confirmation on target overwrite as data is present in the target database being migrated into. You can enable this via the **update migration** CLI command covered in the next section.
- **Completed** - Cutover was successful, and migration is complete.

Use the **az postgres flexible-server migration show -- help** for any additional information.

Update Migration

As soon as the infrastructure setup is complete, the migration activity will pause with appropriate messages seen in the **show details** CLI command response if some pre-requisites are missing. At this point, the migration goes into a state called **WaitingForUserAction**. The **update migration** command is used to set values for parameters, which help the migration to move to the next stage in the migration process. Let us look at each of the sub-states.

- **WaitingForLogicalReplicationSetupRequestOnSourceDB** - The migration is waiting for logical replication to be enabled at the source. This can be enabled by the following CLI command

```
az postgres flexible-server migration update --subscription <sub_id> --
resource-group <rg_name> --name <target_server_name> --migration-name
<migration_name> --initiate-data-migration true
```

The **initiate-data-migration** flag is set to true, which will turn on the logical replication flag at the source. This requires a reboot of the server and might take some time to move to the next stage of the migration. A user can also enable this manually by going to the source server in the Azure portal and changing the Azure Replication Support flag to **Logical**. This would restart the server. In case you have enabled it manually, **you would still need to issue the above update command** for the migration to move out of the **WaitingForUserAction** state. The server does not do a reboot again since it was already done via the portal action.

- **WaitingForCutoverTrigger** - The migration cutover starts the cutover process which makes the target in full sync with the source. At this stage, you should stop writes to your source single server. The CLI command for the cutover is given below

```
az postgres flexible-server migration update --subscription <sub_id> --
resource-group <rg_name> --name <target_server_name> --migration-name
<migration_name> --cutover
```

After issuing the above command, use the **show details** command to make sure that the migration moved to **Completed** state. Once the cutover is complete, run tests to make sure that the data and

schema in target server matches the source. After the tests are completed, update your application to point to the new target flexible server.

- **WaitingForTargetDBOverwriteConfirmation** - The migration is waiting for confirmation on target overwrite as data is present in the target database being migrated into. This can be enabled by the following CLI command.

```
az postgres flexible-server migration update --subscription <sub_id> --resource-group <rg_group> --name <target_server_name> --migration-name <migration_name> --overwrite-dbs true
```

Use the **az postgres flexible-server migration update -- help** for any additional information.

Delete Migration

Any ongoing migration attempts can be deleted using the **delete migration** command. This command just stops any more migration activity on your target server. It will not drop or rollback any changes on your target server that was done by this migration attempt. Below is the CLI command to delete a migration

```
az postgres flexible-server migration delete --subscription <sub_id> --resource-group <rg_name> --name <target_server_name> --migration-name <migration_name>
```

Use the **az postgres flexible-server migration delete -- help** for any additional information.

Post Migration

- Note that all the resources created by this migration solution will be automatically cleaned up irrespective of whether the migration has **succeeded/failed/cancelled**. There is no action required from your end.
- If your migration has failed and if you want to retry the migration, then you need to create a new migration with a different name and try running it again. For now, there is no option of retry on a failed migration.
- If you have more than eight databases on your single server and want to migrate all of them, it is recommended to create multiple migrations between the same single server and flexible server with each migration migrating a set of eight databases each.
- For security reasons, it is highly recommended to delete the Azure Active Directory app once the migration completes.
- Post data validations and making your application point to flexible server, you can consider deleting your single server.