**Welcome to Spawn**

Spawn makes it easy to create **disposable copies of databases for development and test**. The database copies are **hosted in the cloud** by us so you don't need database servers installed locally.

You can **create, save and destroy your databases** as often as you like, even spinning up multiple copies for different branches and CI pipelines. It uses a command line interface, making it **simple to script and automate** in DevOps pipelines.

[**Getting started**](https://docs.spawn.cc/getting-started/installation)[**#**](https://docs.spawn.cc/#getting-started)

We recommend you begin with our tutorial to learn how to:

* install the Spawn client and sign up for free
* create copies of a sample database
* save revisions of the database and easily undo mistakes
* upload your own database and make copies of it

[**Commands**](https://docs.spawn.cc/commands/spawnctl)[**#**](https://docs.spawn.cc/#commands)

Alternatively jump straight into the commands section to learn about spawnctl, the command line client.

[**Database engine support**](https://docs.spawn.cc/engine-support)[**#**](https://docs.spawn.cc/#database-engine-support)

Spawn currently supports the following database engines:

* MSSQL
* PostgreSQL
* MySQL
* MariaDB
* MongoDB
* Redis

# Getting started: Installation

Spawn makes it easy to create disposable copies of databases for your development environment, and to automate them within DevOps pipelines. It uses a command line interface to create and manage those copies, and to make scripting and automation simple.

This tutorial will walk you through installing Spawn, using a database copy and creating an image of your own data from which you can further copies.

## Create a Spawn account[#](https://docs.spawn.cc/getting-started/installation#create-a-spawn-account)

Login to the Spawn web app to create your account.

[LOGIN WITH GITHUB](https://app.spawn.cc/login)

## Installation[#](https://docs.spawn.cc/getting-started/installation#installation)

* macOS or Linux
* Windows

1. Download [spawnctl.exe](https://run.spawn.cc/spawnctl.exe) and save to it a folder of your choosing.

##### NOTE

spawnctl.exe is not an installer, it's a command line executable that can be run directly.

We suggest creating the folder C:\Program Files\Spawn and saving it there.

1. Add the folder containing spawnctl.exe to your path.

##### NOTE

Search for View advanced system settings from the Windows start menu and click the Environment Variables button to edit the Path variable. You may need to restart your terminal after doing this.

## Authenticate spawnctl with your Spawn account[#](https://docs.spawn.cc/getting-started/installation#authenticate-spawnctl-with-your-spawn-account)

Authenticate spawnctl to link your Spawn account

spawnctl auth

Copy

## Create a copy of a sample database[#](https://docs.spawn.cc/getting-started/installation#create-a-copy-of-a-sample-database)

Spawn instantly creates copies of databases as cloud-hosted containers and passes the connection details back to you. These copies are called [data containers](https://docs.spawn.cc/concepts/data-container). The data containers are created from [data images](https://docs.spawn.cc/concepts/data-image) -- prepared images of the database you want to copy, complete with all the data and configuration.

To make it easy to get started we've created some [public data images](https://docs.spawn.cc/other/public-data-images) based on familiar sample databases.

* PostgreSQL
* MySQL
* SQL Server

Create a copy of the PostgreSQL [Pagila](https://github.com/devrimgunduz/pagila) demo database by running

spawnctl create data-container --image postgres-pagila:v11.0 --name spawn-tutorial --lifetime 1h

Copy

##### NOTE

The --name flag is optional. If you don't specify a name a random one will be generated for you based on the image name.

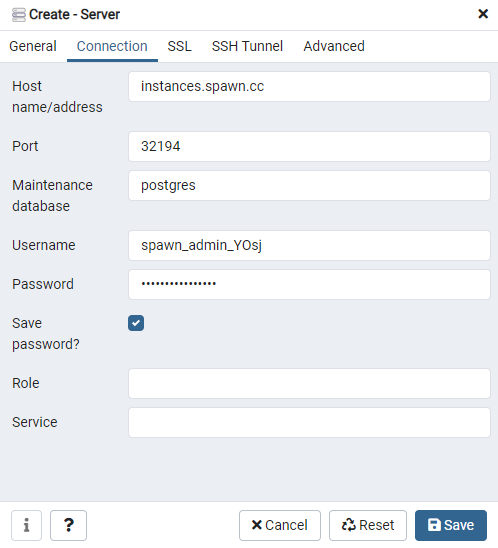
Spawn creates the database copy and returns the connection details:

Data container 'spawn-tutorial' (97477) created!

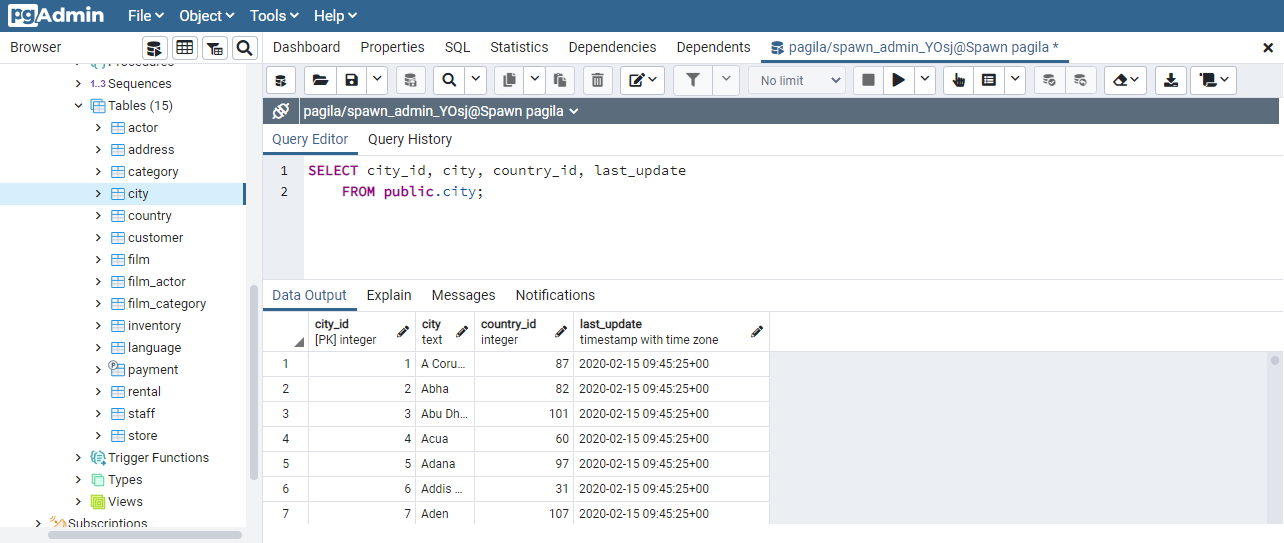
-> Host=instances.spawn.cc;Port=32194;Username=<some\_user\_id>;Database=postgres;Password=<some\_password>

Copy

Using [pgAdmin](https://www.pgadmin.org/) (or your preferred client) you can connect to this database as normal. Copy in the server name and port, the username and password and click Connect.



This database copy is a full version of PostgreSQL running in a container. You can use it just like any other, querying and updating as normal.



List data containers to see the one that was just created:

spawnctl get data-containers

Copy

ID Name Revision Status Engine CreatedAt ExpiresAt

97477 spawn-tutorial rev.0 Completed PostgreSQL:11.0 29 seconds ago 59 minutes from now

Copy

In the next section, we'll learn how to do more with your data container than just connect to it.

# Getting started: Using data containers

The state of a Spawn data container can easily be saved so that you can

* undo mistakes
* return to previous states
* store the state to make further copies from

##### NOTE

Ensure you've completed the Spawn [installation guide](https://docs.spawn.cc/getting-started/installation) before going further.

## Accidentally delete data from the database[#](https://docs.spawn.cc/getting-started/containers#accidentally-delete-data-from-the-database)

We're going to pretend to make a mistake with the database, deleting some data that we didn't intend to. Using the connection details, connect to the database using your preferred client and delete all the records from a table:

* PostgreSQL
* MySQL
* SQL Server

*DELETE* *FROM* *public*.film\_actor;

Copy

## Reset the database to its original state[#](https://docs.spawn.cc/getting-started/containers#reset-the-database-to-its-original-state)

Normally we'd now need to restore the data from a backup, but Spawn allows you return the database to its original state in just a few seconds using the [reset data-container](https://docs.spawn.cc/commands/spawnctl-dc-reset) command:

spawnctl reset data-container spawn-tutorial

Copy

Reconnect to the database and check -- the data we just deleted is restored, undoing the mistake.

## Make some changes you'd like to keep[#](https://docs.spawn.cc/getting-started/containers#make-some-changes-youd-like-to-keep)

What if you want to keep changes that you've made, so that a reset will only return to that point? You can save the state of your database at any time using the [save data-container](https://docs.spawn.cc/commands/spawnctl-dc-save) command.

Let's add a new table and some data to our database. Use your database client to execute the following:

* PostgreSQL
* MySQL
* SQL Server

*CREATE* *TABLE* *public*.crew

(

crew\_id *integer* NOT NULL,

first\_name *text* *COLLATE* pg\_catalog."default" NOT NULL,

last\_name *text* *COLLATE* pg\_catalog."default" NOT NULL,

*CONSTRAINT* crew\_pkey *PRIMARY* *KEY* (crew\_id)

);

*INSERT* *INTO* *public*.crew *VALUES* (1, 'John', 'Smith');

*INSERT* *INTO* *public*.crew *VALUES* (2, 'Jane', 'Doe');

Copy

## Save the new state as a "revision"[#](https://docs.spawn.cc/getting-started/containers#save-the-new-state-as-a-revision)

We can save the current state of the database as a new revision. This is the default state that reset will return to. Each time you use the [save data-container](https://docs.spawn.cc/commands/spawnctl-dc-save) command it will create a new revision.

spawnctl save data-container spawn-tutorial

Copy

Successfully saved data container 'spawn-tutorial'

New revision is 'rev.1'

Copy

##### NOTE

You can use the [load data-container](https://docs.spawn.cc/commands/spawnctl-dc-load) command to return to states other than the current revision.

## Store the state to make further copies[#](https://docs.spawn.cc/getting-started/containers#store-the-state-to-make-further-copies)

The revisions that we've created only exist for as long as the data container lives -- if we delete the data container the revisions will be lost. To store a revision for later use we need to use the [graduate data-container](https://docs.spawn.cc/commands/spawnctl-dc-graduate) command, creating a persistent data image that can be used to create new data containers.

spawnctl graduate data-container spawn-tutorial --revision rev.1 --name spawn-tutorial-new-table

Copy

Successfully graduated data container 'spawn-tutorial' at revision 'rev.1' to a new data-image

New image 'spawn-tutorial-new-table' (97480) available!

Copy

The new image appears in the list of available data images:

spawnctl get data-images

Copy

This data image includes the new table that we added to our database, and can be used to create new data containers whenever we need them:

spawnctl create data-container --image spawn-tutorial-new-table --lifetime 1h

**Getting started: Making data images**

In the previous exercise we made a new data image by making changes to a Spawn database and then graduating it. We can also create new data images from scratch. Data images can be created from [empty databases](https://docs.spawn.cc/data-image-configuration/empty), from [backups](https://docs.spawn.cc/data-image-configuration/backup) and from [scripts](https://docs.spawn.cc/data-image-configuration/scripts).

**Create a data image from an existing backup**[**#**](https://docs.spawn.cc/getting-started/images#create-a-data-image-from-an-existing-backup)

Data images can also be created directly from backup files. The backup is first uploaded to the Spawn service and restored to a new database, before being used to create our re-usable data image. For a large backup file the initial upload might take some time, but after that new copies can be created in just a few seconds.

* PostgreSQL
* MySQL
* SQL Server

We'll use a sample backup file for this example -- it's just a single table with a few rows of data, generated by [pg\_dump](https://www.postgresql.org/docs/current/app-pgdump.html).

Save [dump.sql](https://docs.spawn.cc/assets/files/dump-3f4002df30d2609188acaef5e8a44dfe.sql) to a local folder (you might need to rename it to dump.sql).

Save the configuration file [database-from-backup.yaml](https://docs.spawn.cc/assets/files/database-from-backup-eb53b3bd4e1af34855c3adfba342b98d.yaml) alongside the backup file, renaming it if needed.

database-from-backup.yaml

sourceType: backup

name: database-from-backup

engine: postgresql

version: 12.0

backups:

- folder: .

file: dump.sql

Copy

This configuration file tells Spawn to create a data image from the backup file dump.sql in the current folder, restoring it to PostgreSQL 12.0, and to name the data image database-from-backup

Call [create data-image](https://docs.spawn.cc/commands/spawnctl-di-create), passing in the path to the configuration file we created:

spawnctl create data-image --file ./database-from-backup.yaml

Copy

Data image 'database-from-backup' (10001) created!

Copy

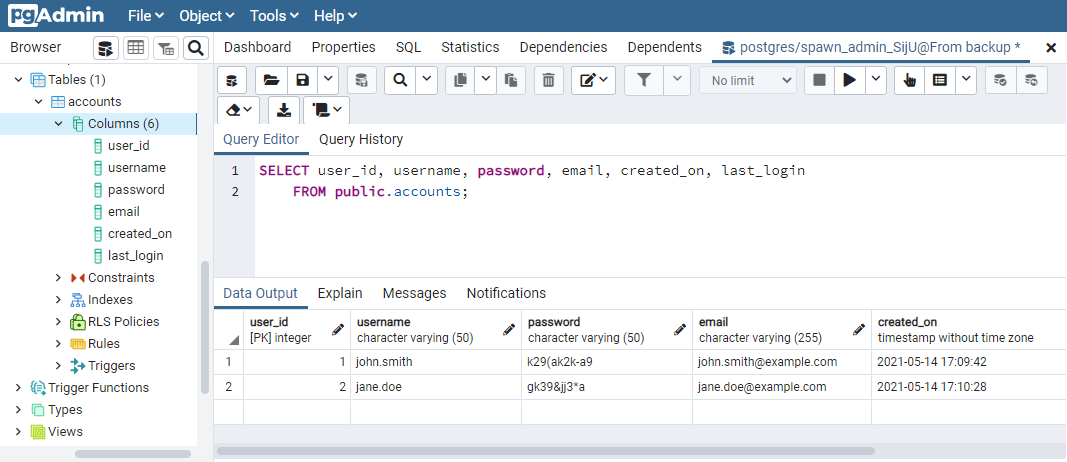
This data image can now be used to create multiple database copies, using spawnctl create data-container as before:

spawnctl create data-container --image database-from-backup --lifetime 1h

Copy

Connect to this new data container and you'll find the database just as though it was restored from the backup file. Unlike restoring a database backup though, once the data image has been created Spawn will create new copies in seconds, regardless of the size of the backup.

* PostgreSQL
* MySQL
* SQL Server



Besides backups, data images can also be created from empty or from scripts. See [source configuration](https://docs.spawn.cc/data-image-configuration/intro) for more information about creating data images from different sources.

**Next steps**[**#**](https://docs.spawn.cc/getting-started/images#next-steps)

That's the basics of Spawn covered:

* Creating disposable copies of databases for development and testing
* Resetting those database and creating revisions to undo changes
* Creating your own data images from which to make further database copies

Next learn more about the [**spawnctl commands**](https://docs.spawn.cc/commands/spawnctl)

**spawnctl**

**Introduction**[**#**](https://docs.spawn.cc/commands/spawnctl#introduction)

Use the Spawn command-line tool, spawnctl (spawn control), to manage databases on Spawn. Using spawnctl, you can create, delete, save, reset, and make other operations on databases. Every section will introduce to a new Spawn command.

To get the most from this section, you should already have [installed Spawn](https://docs.spawn.cc/how-tos/installation).

Each command begins by calling Spawn Control using spawnctl in a terminal window. To test it run the version command.

spawnctl version

Client Version: 0.0.1

Server Version: 0.0.3.0

Copy

The first thing you'll need to do is authenticate with the Spawn service.

**auth**

**Overview**[**#**](https://docs.spawn.cc/commands/authenticate#overview)

The Spawn service takes care of all of your data, storage and compute.

Authenticate with the Spawn service to be able to create and control your resources.

**Command**[**#**](https://docs.spawn.cc/commands/authenticate#command)

spawnctl auth

Copy

**Tutorial**[**#**](https://docs.spawn.cc/commands/authenticate#tutorial)

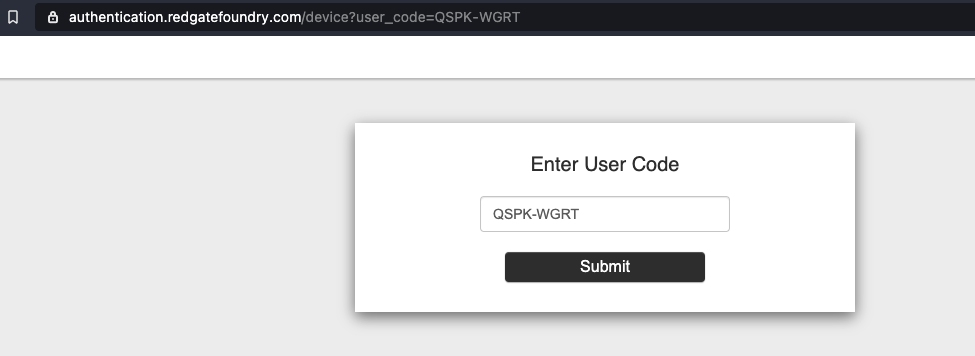
As a prerequisite you should have followed the instructions to [install spawnctl](https://docs.spawn.cc/how-tos/installation)

1. Open a terminal window and type spawnctl auth to begin authentication

spawnctl auth

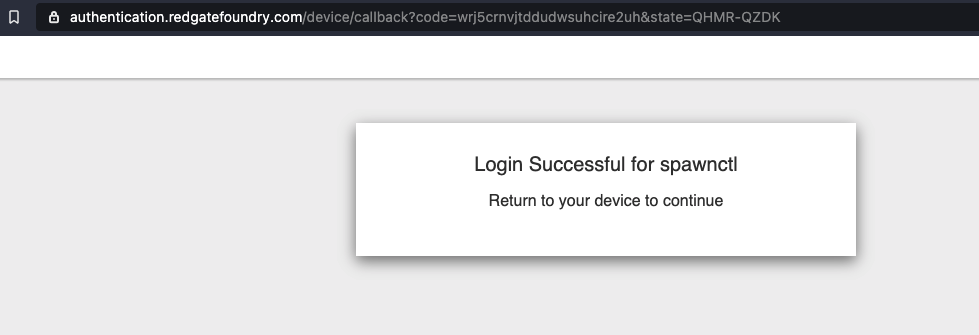
Copy

1. Hit Submit to validate the auth code:



If this is your first time using Spawn, you will have to authorize Spawn against your Github account at this point.

1. The authentication code is replaced with a success message:



1. Return to the terminal to continue using spawnctl.

**If this is your first time using Spawn, you will need to login to the Spawn web app to create your account. Please see the**[**getting started installation guide**](https://docs.spawn.cc/getting-started/installation)

You are now authenticated with the Spawn service. Verify that you are authenticated by running:

spawnctl get data-images

Copy

**Additional information**[**#**](https://docs.spawn.cc/commands/authenticate#additional-information)

spawnctl automatically renews your authentication session when it expires (24 hours after first authentication). If you'd like to prevent further session refreshes, you will need to revoke access to Redgate Foundry Authentication in your chosen authentication provider.

# onboard (Deprecated)

##### CAUTION

The onboard command has been deprecated. It is still available for older spawnctl versions for backwards compatibility.

**Onboarding is now performed via the Spawn web app. Please see the**[**getting started installation guide**](https://docs.spawn.cc/getting-started/installation)

## Overview[#](https://docs.spawn.cc/commands/onboard#overview)

In order to perform operations with the Spawn service, you must first "onboard".

This **must happen before any other operations** as it creates your user account on the Spawn service.

## Command[#](https://docs.spawn.cc/commands/onboard#command)

spawnctl onboard

Copy

## Additional information[#](https://docs.spawn.cc/commands/onboard#additional-information)

You only need to perform this operation once.

# create data-image

## Overview[#](https://docs.spawn.cc/commands/spawnctl-di-create#overview)

In order for Spawn to create a database to work against, you'll first need to create a data image.

For more information about data images please see [data image concepts](https://docs.spawn.cc/concepts/data-image)

### Lifetime[#](https://docs.spawn.cc/commands/spawnctl-di-create#lifetime)

You can specify a lifetime for a data image via the lifetime flag. This will automatically delete your data image after the time specified, in time duration format (number followed by a unit suffix. Valid units are "h", "m", "s").

If no value is provided, your data image will have an unlimited lifetime, and will only be deleted if you explicitly request it.

If there are data containers running for the expired data image, the data image will not be removed

You can [update](https://docs.spawn.cc/commands/spawnctl-di-update) data images after creation to alter their lifetime

## Command[#](https://docs.spawn.cc/commands/spawnctl-di-create#command)

spawnctl create data-image -f ./development.yaml

Copy

To override any Tags specified in the yaml file, use the --tag option.

spawnctl create data-image -f ./development.yaml --tag v1.0

Copy

To override the name specified in the .yaml file, use the --name option.

spawnctl create data-image -f ./development.yaml --name new-name

Copy

To create a data-image with a lifetime define it in the .yaml file or use --lifetime option

spawnctl create data-image -f ./development.yaml --lifetime 48h

Copy

## Tutorial[#](https://docs.spawn.cc/commands/spawnctl-di-create#tutorial)

In this tutorial we will create a data image.

As a prerequisite you should've followed the instructions to [install spawnctl](https://docs.spawn.cc/how-tos/installation)

1. Create a file development.yaml with your data image specifications.

sourceType: empty

name: dev

engine: postgresql

version: 11.0

Copy

In this case we want to create a PostgreSQL data image that is completely empty and is named dev.

Interested in other **engines**? Go to our [data image page](https://docs.spawn.cc/concepts/data-image#supported-engines) to see how you can use other engines we support.

1. Run the following command to create a data image.

$ spawnctl create data-image -f ./development.yaml

Data image 'dev' (10001) created!

Copy

1. You can verify your data image by running the following command.

$ spawnctl get data-images

NAME IMAGE ID ENGINE STATUS MESSAGE CREATED

dev 10001 PostgreSQL 2 Created 2 minutes ago

# get data-image

## Overview[#](https://docs.spawn.cc/commands/spawnctl-di-get#overview)

You can review your data images by using the get command.

## Commands[#](https://docs.spawn.cc/commands/spawnctl-di-get#commands)

spawnctl get data-images

Copy

spawnctl get data-image <ImageName[:Tag]\_Or\_ImageID>

Copy

## Tutorial[#](https://docs.spawn.cc/commands/spawnctl-di-get#tutorial)

In this tutorial we will create a data image, then get the data image. We will then review the data image information.

As a prerequisite you should've followed the instructions to [install spawnctl](https://docs.spawn.cc/how-tos/installation)

1. Create a file development.yaml with your data image specifications.

sourceType: empty

name: dev

engine: postgresql

version: 11.0

Copy

In this case we want to create a PostgreSQL data image that is completely empty and is named dev.

1. Run the following command to create a data image.

$ spawnctl create data-image -f ./development.yaml

Data image 'dev' (10001) created!

Copy

1. You can verify your data image was properly created by running the following command.

$ spawnctl get data-images

ID Name Tags Engine Status CreatedAt Teams

10001 dev PostgreSQL:11.0 Completed 3 hours ago spawn:team1

Copy

### Extra information[#](https://docs.spawn.cc/commands/spawnctl-di-get#extra-information)

To see extra information on images, add the flag --output wide or use the alias -o.

$ spawnctl get data-images --output wide

ID Name Tags Engine SourceType Status CreatedAt Teams Owner Size

10001 dev PostgreSQL:11.0 empty Completed 3 hours ago spawn:team1 Jane Doe 29MB

Copy

### Public images[#](https://docs.spawn.cc/commands/spawnctl-di-get#public-images)

Spawn hides [default public images](https://docs.spawn.cc/other/public-data-images) once you have your own data images. You can show them again by passing the --public flag to the spawnctl get data-images command.

**delete data-image**

**Overview**[**#**](https://docs.spawn.cc/commands/spawnctl-di-delete#overview)

To delete a data image you can use the delete command. You cannot delete data images that have data containers.

**Command**[**#**](https://docs.spawn.cc/commands/spawnctl-di-delete#command)

spawnctl delete data-image <ImageNames[:Tag]\_Or\_ImageIDs>

Copy

If you are an admin of your organisation, you can delete data-images within your organisation using the --org flag.

**Examples**[**#**](https://docs.spawn.cc/commands/spawnctl-di-delete#examples)

Delete a data-image with id 10001:

spawnctl delete data-image 10001

Copy

Delete a data-image with name dev:

spawnctl delete data-image dev

Copy

Delete multiple data-images with ids 10001, 10002 and 10003:

spawnctl delete data-image 10001 10002 10003

Copy

Delete multiple data-images with names dev1, dev2:

spawnctl delete data-image dev1 dev2

Copy

Delete data-image with named dev and tag testing:

spawnctl delete data-image dev:testing

Copy

Delete a data-image with id 10001 that you do not own but is part of your organisation:

spawnctl delete data-image 10001 --org

Copy

**Tutorial**[**#**](https://docs.spawn.cc/commands/spawnctl-di-delete#tutorial)

In this tutorial we will create a data image and delete it.

As a prerequisite you should've followed the instructions to [install spawnctl](https://docs.spawn.cc/how-tos/installation)

1. Identify the data image you plan to delete.

$ spawnctl get data-images

NAME IMAGE ID ENGINE STATUS MESSAGE CREATED

dev 10001 PostgreSQL 2 Created 2 minutes ago

Copy

In this case we want to delete the data image 10001.

1. Run the delete command to remove the data image.

$ spawnctl delete data-image 10001

Deleting image....

10001

Copy

1. You can verify the data image was deleted by running the following command.

$ spawnctl get data-images

No resources found.

# update data-image

## Overview[#](https://docs.spawn.cc/commands/spawnctl-di-update#overview)

You can update your data images to enable or disable sharing with others, and add or remove Tags by using the update command.

Team sharing currently only works with GitHub authentication. Microsoft authentication is **not supported**.

### Lifetime[#](https://docs.spawn.cc/commands/spawnctl-di-update#lifetime)

You can specify a lifetime for a data image via the lifetime flag. This will automatically delete your data image after the time specified, in time duration format (number followed by a unit suffix. Valid units are "h", "m", "s").

If no value is provided, your data image will have an unlimited lifetime, and will only be deleted if you explicitly request it.

If there are data containers running for the expired data image, the data image will not be removed.

### Organisations[#](https://docs.spawn.cc/commands/spawnctl-di-update#organisations)

If you are an admin of your organisation, you can use the --org command to update data-images that are part of your organisation.

## Commands[#](https://docs.spawn.cc/commands/spawnctl-di-update#commands)

spawnctl update data-image <ImageName[:Tag]\_Or\_ImageID> --team <GitHub\_Team1> --team <GitHub\_Team2>

Copy

spawnctl update data-image <ImageName[:Tag]\_Or\_ImageID> --team <GitHub\_Team1> --remove

Copy

spawnctl update data-image <ImageName[:Tag]\_Or\_ImageID> --tag <Tag1> --tag <Tag2>

Copy

spawnctl update data-image <ImageName[:Tag]\_Or\_ImageID> --tag <Tag1> --remove

Copy

spawnctl update data-image <ImageName[:Tag]\_Or\_ImageID> --lifetime 2h

Copy

To rename an existing data image use the --name option:

spawnctl update data-image <OldImageName[:Tag]\_Or\_ImageID> --name <NewImageName>

Copy

## Tutorial[#](https://docs.spawn.cc/commands/spawnctl-di-update#tutorial)

In this tutorial we will create a data image, then share the data image with a GitHub team and add a Tag.

As a prerequisite you should've followed the instructions to [install spawnctl](https://docs.spawn.cc/how-tos/installation)

1. Create a file development.yaml with your data image specifications.

sourceType: empty

name: dev

engine: postgresql

version: 11.0

Copy

In this case we want to create a PostgreSQL data image that is completely empty and is named dev.

1. Run the following command to create a data image.

$ spawnctl create data-image -f ./development.yaml

Data image 'dev' (10986) created!

Copy

1. You can verify your data image was properly created by running the following command.

$ spawnctl get data-images

ID Name Tags Engine SourceType Status CreatedAt Teams Owner

10986 dev PostgreSQL Empty Completed 2019-03-26T11:50:35Z true

Copy

Notice that both the tags and teams columns have no values.

1. Run the following command to share the image with a specified team and add a tag.

$ spawnctl update data-image dev --team red-gate:spawn-developers --tag v1.0

Image : 'dev' updated

Copy

1. You can now verify that the image is shared with the specified team

$ spawnctl get data-images

ID Name Tags Engine SourceType Status CreatedAt Teams Owner

10986 dev v1.0 PostgreSQL Empty Completed 2019-03-26T11:50:35Z red-gate:spawn-developers true

Copy

Providing the --remove flag will remove the specified teams and/or tags from the data image.

# create data-container

## Overview[#](https://docs.spawn.cc/commands/spawnctl-dc-create#overview)

In order to Spawn a database to work against, you'll need to create a data container. A data container is created from a data image.

### Lifetime[#](https://docs.spawn.cc/commands/spawnctl-dc-create#lifetime)

You can specify a lifetime for a data container via the lifetime flag. This will automatically delete your data container after the time specified, in time duration format (number followed by a unit suffix. Valid units are "h", "m", "s").

If no value is provided, your data container will have an unlimited lifetime, and will only be deleted if you explicitly request it.

You can [update](https://docs.spawn.cc/commands/spawnctl-dc-update) data containers after creation to alter their lifetime

## Command[#](https://docs.spawn.cc/commands/spawnctl-dc-create#command)

spawnctl create data-container --image <ImageName[:Tag]\_Or\_ImageID> --lifetime <LifetimeDurationString>

Copy

## Tutorial[#](https://docs.spawn.cc/commands/spawnctl-dc-create#tutorial)

In this tutorial we will create a data image and then use it to create a data container.

As a prerequisite you should've followed the instructions to [install spawnctl](https://docs.spawn.cc/how-tos/installation)

1. Create a file development.yaml with your data image specifications.

sourceType: empty

name: dev

engine: postgresql

version: 11.0

Copy

In this case we want to create a PostgreSQL data image that is completely empty and is named dev.

1. Run the following command to create a data image.

$ spawnctl create data-image -f ./development.yaml

Data image 'dev' (10001) created!

Copy

1. You can verify your data image by running the following command.

$ spawnctl get data-images

NAME IMAGE ID ENGINE STATUS MESSAGE CREATED

dev 10001 PostgreSQL 2 Created 2 minutes ago

Copy

1. Create a data container from the newly created data image.

$ spawnctl create data-container --image dev --lifetime 1h30m

Data container 'dev-rambbomj' (10001) created!

-> Host=instances.spawn.cc;Port=53223;User ID=<some\_user\_id>;Password=<some\_password>;

Copy

1. You can verify your data container was properly created by running the following command.

$ spawnctl get data-containers

NAME CONTAINER ID REVISION STATUS MESSAGE ENGINE CREATED EXPIRES AT

dev-rambbomj 10001 rev.0 2 Running PostgreSQL 1 minute ago 1 hour from now

Copy

1. You should now be able to connect to your database and execute queries.

In this example we connect to the PostgreSQL data container (database) using [psql](http://postgresguide.com/utilities/psql.html).

$ psql -h instances.spawn.cc -p 53223 -U <some\_user\_id>

Password for user <some\_user\_id>:

psql (10.5, server 11.0 (Debian 11.0-1.pgdg90+2))

SSL connection (protocol: TLSv1.2, cipher: ECDHE-RSA-AES256-GCM-SHA384, bits: 256, compression: off)

Type "help" for help.

<some\_user\_id>=# CREATE TABLE customers(id INT);

CREATE TABLE

<some\_user\_id>=# \dt

List of relations

Schema | Name | Type | Owner

--------+-----------+-------+------------------

public | customers | table | <some\_user\_id>

(1 row)

**get data-container**

**Overview**[**#**](https://docs.spawn.cc/commands/spawnctl-dc-get#overview)

You can review your data container by using the get command.

**Commands**[**#**](https://docs.spawn.cc/commands/spawnctl-dc-get#commands)

spawnctl get data-containers

Copy

spawnctl get data-container <ContainerName\_Or\_ContainerID>

Copy

You can change the output format when getting data containers to return more information. -o json will print the data container information out as JSON, including connection details (hostname, port, username, password). This makes it easier to consume Spawn as part of a startup script for an application as you can dynamically configure database connection details.

spawnctl get data-container <ContainerName\_Or\_ContainerID> -o json

Copy

If you are an admin of your organisation, you can use the --org command to check all data-containers created within your organisation.

**Examples**[**#**](https://docs.spawn.cc/commands/spawnctl-dc-get#examples)

Get all data-containers you own:

spawnctl get data-containers

Copy

Get specific data-containers you own:

spawnctl get data-container 10001

Copy

Get all data-containers within your organisation:

spawnctl get data-containers --org

Copy

Get specific data-container within your organisation:

spawnctl get data-container 10001 --org

Copy

**Tutorial**[**#**](https://docs.spawn.cc/commands/spawnctl-dc-get#tutorial)

In this tutorial we will create a data image, then create a data container from that image. We will then review the data container information.

As a prerequisite you should've followed the instructions to [install spawnctl](https://docs.spawn.cc/how-tos/installation)

1. Create a file development.yaml with your data image specifications.

sourceType: empty

name: dev

engine: postgresql

version: 11.0

Copy

In this case we want to create a PostgreSQL data image that is completely empty and is named dev.

1. Run the following command to create a data image.

$ spawnctl create data-image -f ./development.yaml

Data image 'dev' (10001) created!

Copy

1. You can verify your data image by running the following command.

$ spawnctl get data-images

NAME IMAGE ID ENGINE STATUS MESSAGE CREATED

dev 10001 PostgreSQL 2 Created 2 minutes ago

Copy

1. Create a data container from the newly created data image.

$ spawnctl create data-container --image dev

Data container 'dev-rambbomj' (10001) created!

-> Host=instances.spawn.cc;Port=53223;User ID=<some\_user\_id>;Password=<some\_password>;

Copy

1. You can verify your data container was properly created by running the following command.

$ spawnctl get data-containers

ID Name Revision Status Engine CreatedAt ExpiresAt

10001 dev-rambbomj rev.0 Completed PostgreSQL:11.0 1 minute ago Never

Copy

1. You can also get only that specific container.

$ spawnctl get data-container dev-rambbomj

ID Name Revision Status Engine CreatedAt ExpiresAt

10001 dev-rambbomj rev.0 Completed PostgreSQL:11.0 1 minute ago Never

**delete data-container**

**Overview**[**#**](https://docs.spawn.cc/commands/spawnctl-dc-delete#overview)

To delete a data container use the delete command.

**Command**[**#**](https://docs.spawn.cc/commands/spawnctl-dc-delete#command)

spawnctl delete data-container <ContainerNames\_Or\_ContainerIDs>

Copy

If you are an admin of your organisation, you can delete data-containers within your organisation using the --org flag.

**Examples**[**#**](https://docs.spawn.cc/commands/spawnctl-dc-delete#examples)

Delete a data-container with id 10001:

spawnctl delete data-container 10001

Copy

Delete a data-container with name dev-rambbomj:

spawnctl delete data-container dev-rambbomj

Copy

Delete multiple data-container with ids 10001, 10002 and 10003:

spawnctl delete data-container 10001 10002 10003

Copy

Delete multiple data-container with names dev-rambbomj, dev-wtejgstr:

spawnctl delete data-container dev-rambbomj dev-wtejgstr

Copy

Delete a data-container with id 10001 that you do not own but is part of your organisation:

spawnctl delete data-container 10001 --org

Copy

**Tutorial**[**#**](https://docs.spawn.cc/commands/spawnctl-dc-delete#tutorial)

In this tutorial we will create a data image, then create a data container from that image and finally delete the data-container.

As a prerequisite you should've followed the instructions to [install spawnctl](https://docs.spawn.cc/how-tos/installation)

1. Create a file development.yaml with your data image specifications.

sourceType: empty

name: dev

engine: postgresql

version: 11.0

Copy

In this case we want to create a PostgreSQL data image that is completely empty and is named dev.

1. Run the following command to create a data image.

$ spawnctl create data-image -f ./development.yaml

Data image 'dev' (10001) created!

Copy

1. You can verify your data image by running the following command.

$ spawnctl get data-images

NAME IMAGE ID ENGINE STATUS MESSAGE CREATED

dev 10001 PostgreSQL 2 Created 2 minutes ago

Copy

1. Create a data container from the newly created data image.

$ spawnctl create data-container --image dev

Data container 'dev-rambbomj' (10001) created!

-> Host=instances.spawn.cc;Port=53223;User ID=<some\_user\_id>;Password=<some\_password>;

Copy

1. You can verify your data container was properly created by running the following command.

$ spawnctl get data-containers

NAME CONTAINER ID REVISION STATUS MESSAGE ENGINE CREATED

dev-rambbomj 10001 rev.0 2 Running PostgreSQL 1 minute ago

Copy

1. You should now be able to delete your data container.

$ spawnctl delete data-container dev-rambbomj

Deleting container......

dev-rambbomj

# update data-container

## Overview[#](https://docs.spawn.cc/commands/spawnctl-dc-update#overview)

You can update your data containers to change their lifetime by using the update command.

### Lifetime[#](https://docs.spawn.cc/commands/spawnctl-dc-update#lifetime)

You can specify a lifetime for a data container via the lifetime flag. This will automatically delete your data container after the time specified, in time duration format (number followed by a unit suffix. Valid units are "h", "m", "s").

If no value is provided, your data container will have an unlimited lifetime, and will only be deleted if you explicitly request it.

### Organisations[#](https://docs.spawn.cc/commands/spawnctl-dc-update#organisations)

If you are an admin of your organisation, you can use the --org command to update data-containers that are part of your organisation.

## Commands[#](https://docs.spawn.cc/commands/spawnctl-dc-update#commands)

spawnctl update data-container <ContainerName\_Or\_ContainerID> --lifetime <LifetimeDurationString>

Copy

To rename an existing data container use the --name option:

spawnctl update data-container <OldContainerName\_Or\_ContainerID> --name <NewContainerName>

Copy

## Tutorial[#](https://docs.spawn.cc/commands/spawnctl-dc-update#tutorial)

In this tutorial we will create a data container with an unlimited lifetime, then update it to expire in 2 hours.

As a prerequisite you should've followed the instructions to [install spawnctl](https://docs.spawn.cc/how-tos/installation) and [created a data image](https://docs.spawn.cc/commands/spawnctl-di-create)

1. Create a data container from a data image.

$ spawnctl create data-container --image dev

Data container 'dev-rambbomj' (10001) created!

-> Host=instances.spawn.cc;Port=53223;User ID=<some\_user\_id>;Password=<some\_password>;

Copy

1. You can verify your data container was properly created by running the following command.

$ spawnctl get data-containers

NAME CONTAINER ID REVISION STATUS MESSAGE ENGINE CREATED EXPIRES AT

dev-rambbomj 10001 rev.0 2 Running PostgreSQL 1 minute ago Never

Copy

1. Run the update command to change the lifetime of the container to expire 2 hours from now

$ spawnctl update data-container dev-rambbomj --lifetime 2h

Container: 'dev-rambbomj' updated!

Copy

1. Verify that the lifetime of the container has been updated

$ spawnctl get data-containers

NAME CONTAINER ID REVISION STATUS MESSAGE ENGINE CREATED EXPIRES AT

dev-rambbomj 10001 rev.0 2 Running PostgreSQL 1 minute ago 2 hours from now

**save data-container**

**Overview**[**#**](https://docs.spawn.cc/commands/spawnctl-dc-save#overview)

Saving a data container creates a new revision for that data container. To save a data container use the save command. Revisions are necessary for operations like reset, graduate and load.

**Command**[**#**](https://docs.spawn.cc/commands/spawnctl-dc-save#command)

spawnctl save data-container <ContainerName\_Or\_ContainerID>

Copy

**Tutorial**[**#**](https://docs.spawn.cc/commands/spawnctl-dc-save#tutorial)

In this tutorial we will create a data image, then create a data container from that image. We will then make some changes to the data-container and finally save those changes.

As a prerequisite you should've followed the instructions to [install spawnctl](https://docs.spawn.cc/how-tos/installation)

1. Create a file development.yaml with your data image specifications.

sourceType: empty

name: dev

engine: postgresql

version: 11.0

Copy

In this case we want to create a PostgreSQL data image that is completely empty and is named dev.

1. Run the following command to create a data image.

$ spawnctl create data-image -f ./development.yaml

Data image 'dev' (10001) created!

Copy

1. You can verify your data image by running the following command.

$ spawnctl get data-images

NAME IMAGE ID ENGINE STATUS MESSAGE CREATED

dev 10001 PostgreSQL 2 Created 2 minutes ago

Copy

1. Create a data container from the newly created data image.

$ spawnctl create data-container --image dev

Data container 'dev-rambbomj' (10001) created!

-> Host=instances.spawn.cc;Port=53223;User ID=<some\_user\_id>;Password=<some\_password>;

Copy

1. You can verify your data container was properly created by running the following command.

Notice the created data container has revision rev.0.

$ spawnctl get data-containers

NAME CONTAINER ID REVISION STATUS MESSAGE ENGINE CREATED

dev-rambbomj 10001 rev.0 2 Running PostgreSQL 1 minute ago

Copy

1. You should now be able to connect to your database and execute queries.

In this example we connect to the PostgreSQL data container (database) using [psql](http://postgresguide.com/utilities/psql.html).

$ psql -h instances.spawn.cc -p 53223 -U <some\_user\_id>

Password for user <some\_user\_id>:

psql (10.5, server 11.0 (Debian 11.0-1.pgdg90+2))

SSL connection (protocol: TLSv1.2, cipher: ECDHE-RSA-AES256-GCM-SHA384, bits: 256, compression: off)

Type "help" for help.

<some\_user\_id>=# CREATE TABLE customers(id INT);

CREATE TABLE

<some\_user\_id>=# \dt

List of relations

Schema | Name | Type | Owner

--------+-----------+-------+------------------

public | customers | table | <some\_user\_id>

(1 row)

Copy

1. We can now perform a save operation on this data container.

$ spawnctl save data-container dev-rambbomj

Saving container....

Data container 'dev-rambbomj' saved!

New revision is 'rev.1'

Copy

1. You can also verify that a new revision was created by running the following command.

$ spawnctl get data-containers

NAME CONTAINER ID REVISION STATUS MESSAGE ENGINE CREATED

dev-rambbomj 10001 rev.1 2 Running PostgreSQL 1 minute ago

Copy

In this case the revision changed from rev.0 to rev.1.

**reset data-container**

**Overview**[**#**](https://docs.spawn.cc/commands/spawnctl-dc-reset#overview)

Resetting a data container allows you to remove all changes (schema and data) made to it since the last revision. This is useful when you want to discard the last changes you have made, especially ones that would be difficult to revert otherwise (e.g. dropping a table with data).

**Command**[**#**](https://docs.spawn.cc/commands/spawnctl-dc-reset#command)

spawnctl reset data-container <ContainerNames\_Or\_ContainerIDs>

Copy

**Examples**[**#**](https://docs.spawn.cc/commands/spawnctl-dc-reset#examples)

Reset a data-container with id 10001:

spawnctl reset data-container 10001

Copy

Reset a data-container with name dev:

spawnctl reset data-container dev

Copy

Reset multiple data-containers with ids 10001, 10002 and 10003:

spawnctl reset data-container 10001 10002 10003

Copy

Reset multiple data-container with names dev1, dev2:

spawnctl reset data-container dev1 dev2

Copy

**Tutorial**[**#**](https://docs.spawn.cc/commands/spawnctl-dc-reset#tutorial)

In this tutorial we will create a data image, then create a data container from that image. We'll then make some changes to the data container (or database) followed by a reset operation. We will finally inspect the state of this data container.

As a prerequisite you should've followed the instructions to [install spawnctl](https://docs.spawn.cc/how-tos/installation)

1. Create a file development.yaml with your data image specifications.

sourceType: empty

name: dev

engine: postgresql

version: 11.0

Copy

In this case we want to create a PostgreSQL data image that is completely empty and is named dev.

1. Run the following command to create a data image.

$ spawnctl create data-image -f ./development.yaml

Data image 'dev' (10001) created!

Copy

1. You can verify your data image by running the following command.

$ spawnctl get data-images

NAME IMAGE ID ENGINE STATUS MESSAGE CREATED

dev 10001 PostgreSQL 2 Created 2 minutes ago

Copy

1. Create a data container from the newly created data image.

$ spawnctl create data-container --image dev

Data container 'dev-rambbomj' (10001) created!

-> Host=instances.spawn.cc;Port=53223;User ID=<some\_user\_id>;Password=<some\_password>;

Copy

1. You can verify your data container was properly created by running the following command.

Notice the created data container has revision rev.0.

$ spawnctl get data-containers

NAME CONTAINER ID REVISION STATUS MESSAGE ENGINE CREATED

dev-rambbomj 10001 rev.0 2 Running PostgreSQL 1 minute ago

Copy

1. You should now be able to connect to your database and execute queries.

In this example we connect to the PostgreSQL data container (database) using [psql](http://postgresguide.com/utilities/psql.html).

$ psql -h instances.spawn.cc -p 53223 -U <some\_user\_id>

Password for user <some\_user\_id>:

psql (10.5, server 11.0 (Debian 11.0-1.pgdg90+2))

SSL connection (protocol: TLSv1.2, cipher: ECDHE-RSA-AES256-GCM-SHA384, bits: 256, compression: off)

Type "help" for help.

<some\_user\_id>=# CREATE TABLE customers(id INT);

CREATE TABLE

<some\_user\_id>=# \dt

List of relations

Schema | Name | Type | Owner

--------+-----------+-------+------------------

public | customers | table | <some\_user\_id>

(1 row)

Copy

1. We can now perform a reset operation on this data container.

$ spawnctl reset data-container dev-rambbomj

Data container 'dev-rambbomj' reset!

Copy

1. You can now connect to this data container (database) and verify its content.

$ psql -h instances.spawn.cc -p 53223 -U <some\_user\_id>

psql (10.5, server 11.0 (Debian 11.0-1.pgdg90+2))

SSL connection (protocol: TLSv1.2, cipher: ECDHE-RSA-AES256-GCM-SHA384, bits: 256, compression: off)

Type "help" for help.

<some\_user\_id>=# \dt

Did not find any relations.

Copy

Notice that there are no relations even though we just added the table customers. This is because all data changes that happened after our last save are removed and the data container is restarted for the data at that checkpoint.

**load data-container**

**Overview**[**#**](https://docs.spawn.cc/commands/spawnctl-dc-load#overview)

Loading a data container allows you to run your data container (database) with the data saved at a moment in time.

**Command**[**#**](https://docs.spawn.cc/commands/spawnctl-dc-load#command)

spawnctl load data-container <ContainerName\_Or\_ContainerID> --revision=<RevisionID>

Copy

**Tutorial**[**#**](https://docs.spawn.cc/commands/spawnctl-dc-load#tutorial)

In this tutorial we will create a data image, then create a data container from that image. We will then make some changes to the data container and save those changes. After this we will load the previous revision and inspect the data container.

As a prerequisite you should've followed the instructions to [install spawnctl](https://docs.spawn.cc/how-tos/installation)

1. Create a file development.yaml with your data image specifications.

sourceType: empty

name: dev

engine: postgresql

version: 11.0

Copy

In this case we want to create a PostgreSQL data image that is completely empty and is named dev.

1. Run the following command to create a data image.

$ spawnctl create data-image -f ./development.yaml

Data image 'dev' (10001) created!

Copy

1. You can verify your data image by running the following command.

$ spawnctl get data-images

NAME IMAGE ID ENGINE STATUS MESSAGE CREATED

dev 10001 PostgreSQL 2 Created 2 minutes ago

Copy

1. Create a data container from the newly created data image.

$ spawnctl create data-container --image dev

Data container 'dev-rambbomj' (10001) created!

-> Host=instances.spawn.cc;Port=53223;User ID=<some\_user\_id>;Password=<some\_password>;

Copy

1. You can verify your data container was properly created by running the following command.

Notice the created data container has revision rev.0.

$ spawnctl get data-containers

NAME CONTAINER ID REVISION STATUS MESSAGE ENGINE CREATED

dev-rambbomj 10001 rev.0 2 Running PostgreSQL 1 minute ago

Copy

1. You should now be able to connect to your database and execute queries.

In this example we connect to the PostgreSQL data container (database) using [psql](http://postgresguide.com/utilities/psql.html).

$ psql -h instances.spawn.cc -p 53223 -U <some\_user\_id>

Password for user <some\_user\_id>:

psql (10.5, server 11.0 (Debian 11.0-1.pgdg90+2))

SSL connection (protocol: TLSv1.2, cipher: ECDHE-RSA-AES256-GCM-SHA384, bits: 256, compression: off)

Type "help" for help.

<some\_user\_id>=# CREATE TABLE customers(id INT);

CREATE TABLE

<some\_user\_id>=# \dt

List of relations

Schema | Name | Type | Owner

--------+-----------+-------+------------------

public | customers | table | <some\_user\_id>

(1 row)

Copy

1. We can now perform a save operation on this data container.

$ spawnctl save data-container dev

Saving container....

Data container 'dev-rambbomj' saved!

New revision is 'rev.1'

Copy

1. You can now load the data container at revision rev.0.

$ spawnctl load data-container dev-rambbomj --revision=rev.0

Data container 'dev-rambbomj' loaded!

New revision is 'rev.0-ciys.0'

Copy

Notice that your existing data container now has revision rev.0-ciys.0

1. You can now connect to this data container (database) and verify its content.

$ psql -h instances.spawn.cc -p 53223 -U <some\_user\_id>

psql (10.5, server 11.0 (Debian 11.0-1.pgdg90+2))

SSL connection (protocol: TLSv1.2, cipher: ECDHE-RSA-AES256-GCM-SHA384, bits: 256, compression: off)

Type "help" for help.

<some\_user\_id>=# \dt

Did not find any relations.

Copy

Notice that there are no relations even though our data container at revision rev.1 had one table customers.

**graduate data-container**

**Overview**[**#**](https://docs.spawn.cc/commands/spawnctl-dc-graduate#overview)

Graduating a data container allows you to create a data image from it. This is useful when you make changes to a data container and want to use its current state as the baseline for future work. A good example of when to use the graduate command would be when you were able to reproduce a bug and want to share the state of your database with others. You can graduate the data container and share the resulting data image with your team.

**Command**[**#**](https://docs.spawn.cc/commands/spawnctl-dc-graduate#command)

spawnctl graduate data-container <ContainerName\_Or\_ContainerID> --revision <RevisionID> [--name <GraduatedImageName> --team <GitHub\_Team1> --tag <Tag1>]

Copy

**Tutorial**[**#**](https://docs.spawn.cc/commands/spawnctl-dc-graduate#tutorial)

In this tutorial we will create a data image, then create a data container from that image. We will then make some changes to the database, save them and perform a graduation. This will create a new data image. We will use this image to create a new data container and inspect its current state.

As a prerequisite you should've followed the instructions to [install spawnctl](https://docs.spawn.cc/how-tos/installation)

1. Create a file development.yaml with your data image specifications.

sourceType: empty

name: dev

engine: postgresql

version: 11.0

Copy

In this case we want to create a PostgreSQL data image that is completely empty and is named dev.

1. Run the following command to create a data image.

$ spawnctl create data-image -f ./development.yaml

Data image 'dev' (10001) created!

Copy

1. You can verify your data image by running the following command.

$ spawnctl get data-images

NAME IMAGE ID ENGINE STATUS MESSAGE CREATED

dev 10001 PostgreSQL 2 Created 2 minutes ago

Copy

1. Create a data container from the newly created data image.

$ spawnctl create data-container --image dev

Data container 'dev-rambbomj' (10001) created!

-> Host=instances.spawn.cc;Port=53223;User ID=<some\_user\_id>;Password=<some\_password>;

Copy

1. You can verify your data container was properly created by running the following command.

Notice the created data container has revision rev.0.

$ spawnctl get data-containers

NAME CONTAINER ID REVISION STATUS MESSAGE ENGINE CREATED

dev-rambbomj 10001 rev.0 2 Running PostgreSQL 1 minute ago

Copy

1. You should now be able to connect to your database and execute queries.

In this example we connect to the PostgreSQL data container (database) using [psql](http://postgresguide.com/utilities/psql.html).

$ psql -h instances.spawn.cc -p 53223 -U <some\_user\_id>

Password for user <some\_user\_id>:

psql (10.5, server 11.0 (Debian 11.0-1.pgdg90+2))

SSL connection (protocol: TLSv1.2, cipher: ECDHE-RSA-AES256-GCM-SHA384, bits: 256, compression: off)

Type "help" for help.

<some\_user\_id>=# CREATE TABLE customers(id INT);

CREATE TABLE

<some\_user\_id>=# \dt

List of relations

Schema | Name | Type | Owner

--------+-----------+-------+------------------

public | customers | table | <some\_user\_id>

(1 row)

Copy

1. We can now perform a save operation on this data container.

$ spawnctl save data-container dev-rambbomj

Saving container....

Data container 'dev-rambbomj' saved!

New revision is 'rev.1'

Copy

1. We can now perform a graduate operation on this data container.

Now that our new changes are saved, we want to make the current state of this data container (database) the baseline for future work.

$ spawnctl graduate data-container dev-rambbomj --revision rev.1

Successfully graduated data container 'dev-rambbomj' at revision 'rev.1' to a new data-image

New image 'dev-rambbomj-graduate-1' (10002) available!

Copy

1. You can verify the new data image by running the following command.

$ spawnctl get data-images

NAME IMAGE ID ENGINE STATUS MESSAGE CREATED

dev 10001 PostgreSQL 2 Created 2 hours ago

dev-rambbomj-graduate-1 10002 PostgreSQL 2 Created 2 minutes ago

Copy

Notice the name of the new image is dev-rambbomj-graduate-1. This is because it was created from the graduation of a data container.

1. You can now create a data container from that image.

$ spawnctl create data-container --image dev-rambbomj-graduate-1

Creating data container......

Data container 'dev-rambbomj-graduate-1-hztarabe' created!

-> Host=instances.spawn.cc;Port=53224;User ID=<some\_user\_id>;Password=<password>;

Copy

1. Connect to this data container and verify its content.

$ psql -h instances.spawn.cc -p 53224 -U <some\_user\_id>

psql (10.5, server 11.0 (Debian 11.0-1.pgdg90+2))

SSL connection (protocol: TLSv1.2, cipher: ECDHE-RSA-AES256-GCM-SHA384, bits: 256, compression: off)

Type "help" for help.

<some\_user\_id>=# \dt

List of relations

Schema | Name | Type | Owner

--------+-----------+-------+------------------

public | customers | table | spawn\_admin\_jmij

(1 row)

Copy

Notice that the table customers we created previously is present. This is because after creating the table, we *saved* the data container and then *graduated*. The graduation operation created a new image. All subsequent data containers based on this image will contain a copy of the data.

**logs data-container**

**Overview**[**#**](https://docs.spawn.cc/commands/spawnctl-dc-logs#overview)

Using spawnctl, you can get the logs of a running data container.

**Command**[**#**](https://docs.spawn.cc/commands/spawnctl-dc-logs#command)

spawnctl logs data-container <ContainerName\_Or\_ContainerID>

Copy

**Tutorial**[**#**](https://docs.spawn.cc/commands/spawnctl-dc-logs#tutorial)

In this tutorial we will create a data image and then use it to create a data container. Then, we will get the logs of that data container.

As a prerequisite you should've followed the instructions to [install spawnctl](https://docs.spawn.cc/how-tos/installation)

1. Create a file development.yaml with your data image specifications.

sourceType: empty

name: dev

engine: postgresql

version: 11.0

Copy

In this case we want to create a PostgreSQL data image that is completely empty and is named dev.

1. Run the following command to create a data image.

$ spawnctl create data-image -f ./development.yaml

Data image 'dev' (10001) created!

Copy

1. You can verify your data image by running the following command.

$ spawnctl get data-images

NAME IMAGE ID ENGINE STATUS MESSAGE CREATED

dev 10001 PostgreSQL 2 Created 2 minutes ago

Copy

1. Create a data container from the newly created data image.

$ spawnctl create data-container --image dev

Data container 'dev-rambbomj' (10001) created!

-> Host=instances.spawn.cc;Port=53223;User ID=<some\_user\_id>;Password=<some\_password>;

Copy

1. You can verify your data container was properly created by running the following command.

$ spawnctl get data-containers

NAME CONTAINER ID REVISION STATUS MESSAGE ENGINE CREATED

dev-rambbomj 10001 rev.0 2 Running PostgreSQL 1 minute ago

Copy

1. You can now get the logs of the data container using the spawnctl logs data-container command.

$ spawnctl logs data-container dev-rambbomj

2019-08-19 12:10:45.302 UTC [1] LOG: listening on IPv4 address "0.0.0.0", port 5432

2019-08-19 12:10:45.302 UTC [1] LOG: listening on IPv6 address "::", port 5432

2019-08-19 12:10:45.306 UTC [1] LOG: listening on Unix socket "/var/run/postgresql/.s.PGSQL.5432"

2019-08-19 12:10:45.358 UTC [21] LOG: database system was interrupted; last known up at 2019-08-19 12:10:23 UTC

2019-08-19 12:10:46.670 UTC [22] FATAL: the database system is starting up

2019-08-19 12:10:49.679 UTC [23] FATAL: the database system is starting up

2019-08-19 12:10:52.009 UTC [21] LOG: database system was not properly shut down; automatic recovery in progress

2019-08-19 12:10:52.026 UTC [21] LOG: invalid record length at 0/1652570: wanted 24, got 0

2019-08-19 12:10:52.026 UTC [21] LOG: redo is not required

2019-08-19 12:10:52.065 UTC [1] LOG: database system is ready to accept connections

# proxy data-container

## Overview[#](https://docs.spawn.cc/commands/spawnctl-dc-proxy#overview)

Running spawnctl proxy data-container <container> will configure a localhost proxy on your machine on the default port for the database engine your data container is running on. For example, if you have a Postgres data container running spawnctl proxy data-container mypostgrescontainer will open port 5432 on your machine over localhost and forward all connections to the upstream data container in the Spawn service.

This can be useful for environments where you've already got localhost connection strings configured and you don't want to manually update the port when data containers are created.

The proxy will run indefinitely until you cancel the command (e.g with CTRL+C).

## Command[#](https://docs.spawn.cc/commands/spawnctl-dc-proxy#command)

spawnctl proxy data-container <ContainerNames\_Or\_ContainerIDs>

Copy

You can also optionally specify the local port to use

spawnctl proxy data-container <ContainerNames\_Or\_ContainerIDs> --port 9999

Copy

##### NOTE

For SQL Server the address for the proxied data container is 127.0.0.1 (not localhost). This is due to SQL Server treating localhost as a "shared memory" connection which is not supported by the proxy command.

## Tutorial[#](https://docs.spawn.cc/commands/spawnctl-dc-proxy#tutorial)

In this tutorial we will create a data image, then create a data container from that image. We'll then connect to the data container via the localhost connection spawnctl proxy will provide.

As a prerequisite you should've followed the instructions to [install spawnctl](https://docs.spawn.cc/how-tos/installation)

1. Create a file development.yaml with your data image specifications.

sourceType: empty

name: dev

engine: postgresql

version: 11.0

Copy

In this case we want to create a PostgreSQL data image that is completely empty and is named dev.

1. Run the following command to create a data image.

$ spawnctl create data-image -f ./development.yaml

Data image 'dev' (10001) created!

Copy

1. You can verify your data image by running the following command.

$ spawnctl get data-images

NAME IMAGE ID ENGINE STATUS MESSAGE CREATED

dev 10001 PostgreSQL 2 Created 2 minutes ago

Copy

1. Create a data container from the newly created data image.

$ spawnctl create data-container --image dev

Data container 'dev-rambbomj' (10001) created!

-> Host=instances.spawn.cc;Port=53223;User ID=<some\_user\_id>;Password=<some\_password>;

Copy

1. In a separate terminal, we'll set up the local proxy for this data container.

$ spawnctl proxy data-container dev-rambbomj

container 'dev-rambbomj' is now accessible at localhost:5432

Copy

1. You should now be able to connect to your database and execute queries over localhost on the default engine port.

In this example we connect to the PostgreSQL data container (database) using [psql](http://postgresguide.com/utilities/psql.html).

$ psql -h localhost -U <some\_user\_id>

Password for user <some\_user\_id>:

psql (10.5, server 11.0 (Debian 11.0-1.pgdg90+2))

SSL connection (protocol: TLSv1.2, cipher: ECDHE-RSA-AES256-GCM-SHA384, bits: 256, compression: off)

Type "help" for help.

<some\_user\_id>=# CREATE TABLE customers(id INT);

CREATE TABLE

<some\_user\_id>=# \dt

List of relations

Schema | Name | Type | Owner

--------+-----------+-------+------------------

public | customers | table | <some\_user\_id>

(1 row)

**create access-token**

**Overview**[**#**](https://docs.spawn.cc/commands/spawnctl-accesstoken-create#overview)

To support non-interactive environments, you can generate access tokens to authenticate against the Spawn API.

**Command**[**#**](https://docs.spawn.cc/commands/spawnctl-accesstoken-create#command)

spawnctl create access-token --purpose <Purpose>

Copy

**Tutorial**[**#**](https://docs.spawn.cc/commands/spawnctl-accesstoken-create#tutorial)

In this tutorial we create an access token for the purpose of using the spawnctl in a CI environment.

As a prerequisite you should've followed the instructions to [install spawnctl](https://docs.spawn.cc/how-tos/installation)

1. Run the following command to create an access token.

$ spawnctl create access-token --purpose "CI system Spawn access token"

Access token generated: <access-token-string>

Copy

This access token **cannot be retrieved** after this command. So make sure you store it somewhere safe to retrieve later.

1. Future spawnctl commands can use the access token to avoid requiring the interactive flow provided by [spawnctl auth](https://docs.spawn.cc/commands/authenticate).

spawnctl create data-image -f ./development.yaml --accessToken <access-token-string>

**get access-token**

**Overview**[**#**](https://docs.spawn.cc/commands/spawnctl-accesstoken-get#overview)

You can retrieve the list of access tokens you have generated for the Spawn API.

**Command**[**#**](https://docs.spawn.cc/commands/spawnctl-accesstoken-get#command)

spawnctl get access-tokens

Copy

spawnctl get access-token <AccessTokenId>

Copy

**Tutorial**[**#**](https://docs.spawn.cc/commands/spawnctl-accesstoken-get#tutorial)

In this tutorial we get the list of all access tokens for the current user.

As a prerequisite you should've followed the instructions to [install spawnctl](https://docs.spawn.cc/how-tos/installation)

1. Run the following command to get the list of access tokens.

$ spawnctl get access-tokens

ID CreatedAt Purpose

01c8fb177fd348b9be7d10173e5e48b3 2019-03-26T13:33:31Z CI system Spawn access token

**delete access-token**

**Overview**[**#**](https://docs.spawn.cc/commands/spawnctl-accesstoken-delete#overview)

You can delete access tokens if you want to prevent them from authenticating a user against the Spawn API.

**Command**[**#**](https://docs.spawn.cc/commands/spawnctl-accesstoken-delete#command)

spawnctl delete access-token <AccessTokenId>

Copy

**Tutorial**[**#**](https://docs.spawn.cc/commands/spawnctl-accesstoken-delete#tutorial)

In this tutorial we delete an access token that has been generated for a CI system.

As a prerequisite you should've followed the instructions to [install spawnctl](https://docs.spawn.cc/how-tos/installation)

1. Run the following command to get the list of access tokens.

$ spawnctl get access-tokens

ID CreatedAt Purpose

01c8fb177fd348b9be7d10173e5e48b3 2019-03-26T13:33:31Z CI system Spawn access token

Copy

1. Delete the access token by specifying the ID.

spawnctl delete access-token 01c8fb177fd348b9be7d10173e5e48b3

Copy

1. Get the list of access tokens to confirm it has been deleted.

$ spawnctl get access-tokens

No resources found.

**get organisation**

**Overview**[**#**](https://docs.spawn.cc/commands/spawnctl-organisation-get#overview)

You can retrieve the list of accounts that are part of your organisation for reference purposes.

**Command**[**#**](https://docs.spawn.cc/commands/spawnctl-organisation-get#command)

spawnctl get organisation

Copy

**Tutorial**[**#**](https://docs.spawn.cc/commands/spawnctl-organisation-get#tutorial)

In this tutorial we get the list of all members of the organistion the current user is a part of.

As a prerequisite you should've followed the instructions to [install spawnctl](https://docs.spawn.cc/how-tos/installation)

1. Run the following command to get the list of organisation members.

$ spawnctl get organisation

Name Email Onboarded

Spawn User spawn.user@example.com true

Spawn CI User spawn.ci.user@example.com true

**Create user within your organisation**

**Overview**[**#**](https://docs.spawn.cc/commands/spawnctl-user-create#overview)

You can create a user within your organisation if you're an admin.

**Command**[**#**](https://docs.spawn.cc/commands/spawnctl-user-create#command)

spawnctl create user

Copy

If you include the --admin flag, new user will be an admin of your organisation.

**Tutorial**[**#**](https://docs.spawn.cc/commands/spawnctl-user-create#tutorial)

In this tutorial we create a user within the current user's organisation.

As a prerequisite you should've followed the instructions to [install spawnctl](https://docs.spawn.cc/how-tos/installation)

1. Run the following command to get the list of organisation members.

$ spawnctl get organisation

Name Email Onboarded

Spawn User spawn.user@example.com true

Copy

1. Create a new user

spawnctl create user --name "Spawn CI" --email spawn.ci.user@example.com --admin=true

Copy

1. Run the following command to get the list of organisation members.

$ spawnctl get organisation

Name Email Onboarded

Spawn User spawn.user@example.com true

Spawn CI spawn.ci.user@example.com false

**Update user within your organisation**

**Overview**[**#**](https://docs.spawn.cc/commands/spawnctl-user-update#overview)

You can update a user from your organisation if you're an admin.

**Command**[**#**](https://docs.spawn.cc/commands/spawnctl-user-update#command)

spawnctl update user

Copy

**Examples**[**#**](https://docs.spawn.cc/commands/spawnctl-user-update#examples)

spawnctl update user your@email.com --admin=true

spawnctl update user your@email.com -n name

Copy

**Tutorial**[**#**](https://docs.spawn.cc/commands/spawnctl-user-update#tutorial)

In this tutorial we update a user within the current user's organisation.

As a prerequisite you should've followed the instructions to [install spawnctl](https://docs.spawn.cc/how-tos/installation)

1. Run the following command to get the list of organisation members.

$ spawnctl get organisation

Name Email Onboarded

Spawn User spawn.user@example.com true

Spawn CI spawn.ci.user@example.com true

Copy

1. Update user

spawnctl update user spawn.ci.user@example.com --admin=false

Copy

1. Run the following command to get the list of organisation members.

$ spawnctl get organisation

Name Email Onboarded

Spawn User spawn.user@example.com true

Spawn CI spawn.ci.user@example.com true

**Delete a user from your organisation**

**Overview**[**#**](https://docs.spawn.cc/commands/spawnctl-user-delete#overview)

You can delete a user from your organisation if you're an admin.

**Command**[**#**](https://docs.spawn.cc/commands/spawnctl-user-delete#command)

spawnctl delete user

Copy

**Tutorial**[**#**](https://docs.spawn.cc/commands/spawnctl-user-delete#tutorial)

In this tutorial we delete a user within the current user's organisation.

As a prerequisite you should've followed the instructions to [install spawnctl](https://docs.spawn.cc/how-tos/installation)

1. Run the following command to get the list of organisation members.

$ spawnctl get organisation

Name Email Onboarded

Spawn User spawn.user@example.com true

Spawn CI spawn.ci.user@example.com true

Copy

1. Delete user

spawnctl delete user spawn.ci.user@example.com

Copy

1. Run the following command to get the list of organisation members.

$ spawnctl get organisation

Name Email Onboarded

Spawn User spawn.user@example.com true

**Spawnctl shell completion**

**Overview**[**#**](https://docs.spawn.cc/commands/spawnctl-completion#overview)

You can get spawnctl shell completion for your shell of choice by invoking the spawnctl completion command.

**Commands**[**#**](https://docs.spawn.cc/commands/spawnctl-completion#commands)

Run spawnctl completion --help for instructions on how to configure shell completion for your shell.

Completion is supported for the following shells:

* bash
* zsh
* fish
* Powershell

**Usage**[**#**](https://docs.spawn.cc/commands/spawnctl-completion#usage)

Once configured, spawnctl will autocomplete sub-commands, image names and container names by pressing <Tab>.

**Spawnctl environment variables**

**Overview**[**#**](https://docs.spawn.cc/commands/spawnctl-env-vars#overview)

You can override spawnctl defaults by setting environment variables.

* SPAWNCTL\_API\_ENDPOINT to override default spawn api endpoint
* SPAWNCTL\_AUTH\_ENDPOINT to override default spawn authentication endpoint
* SPAWNCTL\_ACCESS\_TOKEN to override the current user of spawn and run commands as another user (for CI purposes for example)

Some points about overriding access tokens:

* SPAWNCTL\_ACCESS\_TOKEN env var will override the interactive token obtained from spawnctl auth
* The --accessToken flag will override both the interactive token obtained from spawnctl auth AND the SPAWNCTL\_ACCESS\_TOKEN env var

Note: access tokens can be created through the [create access-token](https://docs.spawn.cc/commands/spawnctl-accesstoken-create) command.

**Data Image configuration introduction**

You can create data images from different sources:

* [data images that are empty](https://docs.spawn.cc/data-image-configuration/empty)
* [data images that are created from scripts](https://docs.spawn.cc/data-image-configuration/scripts)
* [data images that are created from a backup](https://docs.spawn.cc/data-image-configuration/backup)

Depending on which source you want to use, you will need to define a different type of configuration file.

The source is defined in your configuration file by using the sourceType property.

For example:

sourceType: empty

name: dev

engine: postgresql

version: 11.0

Copy

In this case we want to create a PostgreSQL data image whose source is empty (has no data or schema) and is named dev.

**Empty**

**Overview**[**#**](https://docs.spawn.cc/data-image-configuration/empty#overview)

If you define the sourceType property as empty you will be able to create a data image that is completely empty, with no content on it.

sourceType: empty

name: dev

engine: postgresql

version: 11.0

Copy

In this case we want to create a PostgreSQL data image whose source is empty and is named dev.

**Tutorial**[**#**](https://docs.spawn.cc/data-image-configuration/empty#tutorial)

In this tutorial we will create an empty data image and then use it to create a data container. We will then inspect the data container.

1. Create a file development.yaml with your data image specifications.

sourceType: empty

name: dev

engine: postgresql

version: 11.0

Copy

1. Run the following command to create a data image.

$ spawnctl create data-image -f ./development.yaml

Data image 'dev' (10001) created!

Copy

1. You can verify your data image by running the following command.

$ spawnctl get data-images

NAME IMAGE ID ENGINE STATUS MESSAGE CREATED

dev 10001 PostgreSQL 2 Created 2 minutes ago

Copy

1. Create a data container from the newly created data image.

$ spawnctl create data-container --image dev

Data container 'dev-rambbomj' (10001) created!

-> Host=instances.spawn.cc;Port=53223;User ID=<some\_user\_id>;Password=<some\_password>;

Copy

1. You can verify your data container was properly created by running the following command.

$ spawnctl get data-containers

NAME CONTAINER ID REVISION STATUS MESSAGE ENGINE CREATED

dev-rambbomj 10001 rev.0 2 Running PostgreSQL 1 minute ago

Copy

1. You should now be able to connect to your database and execute queries.

In this example we connect to the PostgreSQL data container (database) using [psql](http://postgresguide.com/utilities/psql.html).

$ psql -h instances.spawn.cc -p 53223 -U <some\_user\_id>

Password for user <some\_user\_id>:

psql (10.5, server 11.0 (Debian 11.0-1.pgdg90+2))

SSL connection (protocol: TLSv1.2, cipher: ECDHE-RSA-AES256-GCM-SHA384, bits: 256, compression: off)

Type "help" for help.

<some\_user\_id>=# \dt

Did not find any relations.

Copy

Notice the created database/data container has no tables.

**Scripts**

**Overview**[**#**](https://docs.spawn.cc/data-image-configuration/scripts#overview)

If you define the sourceType property as scripts you will be able to create a data image whose content will be what you defined in your scripts.

sourceType: scripts

name: dev

engine: postgresql

version: 11.0

scriptsFolders:

- ./Databases/Postgres

Copy

In this case we want to create a PostgreSQL data image whose source is scripts and is named dev.

The path given to the scriptsFolder property is relative to the location of the image definition .yaml file.

Spawn will execute scripts inside each folder in alphanumeric order.

You can specify multiple scripts folders in the yaml file. Spawn will execute scripts from each folder in the order specified.

**Tutorial**[**#**](https://docs.spawn.cc/data-image-configuration/scripts#tutorial)

In this tutorial we will create a data image from scripts and then use it to create a data container. We will then inspect the data container.

1. Create a file development.yaml with your data image specifications.

sourceType: scripts

name: dev

engine: postgresql

version: 11.0

scriptsFolders:

- ./Database

Copy

We are defining the source of our data image to be scripts and the scripts folders to be imported will be the ones present in the ./Database folder.

If we inspect the Database folder we find two .sql files:

$ tree

.

├── 001.sql

├── 002.sql

0 directories, 2 files

Copy

If we inspect the files we get the following:

$ cat Database/001.sql

CREATE DATABASE mycompany;

$ cat Database/002.sql

connect mycompany;

CREATE TABLE customers

(

id BIGSERIAL PRIMARY KEY,

daterecorded TIMESTAMP

WITH TIME ZONE,

description TEXT

);

Copy

1. Run the following command to create a data image.

$ spawnctl create data-image -f ./development.yaml

Data image 'dev' (10001) created!

Copy

1. Create a data container from the newly created data image.

$ spawnctl create data-container --image dev

Data container 'dev-rambbomj' (10001) created!

-> Host=instances.spawn.cc;Port=53223;User ID=<some\_user\_id>;Password=<some\_password>;

Copy

1. You should now be able to connect to your database and execute queries.

In this example we connect to the PostgreSQL data container (database) using [psql](http://postgresguide.com/utilities/psql.html).

$ psql -h instances.spawn.cc -p 53223 -U <some\_user\_id>

Password for user <some\_user\_id>:

psql (10.5, server 11.0 (Debian 11.0-1.pgdg90+2))

SSL connection (protocol: TLSv1.2, cipher: ECDHE-RSA-AES256-GCM-SHA384, bits: 256, compression: off)

Type "help" for help.

<some\_user\_id>=# \l

List of databases

Name | Owner | Encoding | Collate | Ctype | Access privileges

------------------+------------------+----------+------------+------------+---------------------------------------

postgres | <some\_user\_id> | UTF8 | en\_US.utf8 | en\_US.utf8 |

<some\_user\_id> | <some\_user\_id> | UTF8 | en\_US.utf8 | en\_US.utf8 |

mycompany | <some\_user\_id> | UTF8 | en\_US.utf8 | en\_US.utf8 |

(5 rows)

<some\_user\_id>=# \c mycompany

mycompany=# \dt

List of relations

Schema | Name | Type | Owner

--------+------------------+-------+------------------

public | customers | table | <some\_user\_id>

Copy

Notice the created database/data container has a database called mycompany as well as the table customers that where present in our scripts.

# Backup

## Overview[#](https://docs.spawn.cc/data-image-configuration/backup#overview)

If you define the sourceType property as backup you will be able to create a data image whose content will be what you defined in your backup.

sourceType: backup

name: dev

engine: mysql

version: 5.7

backups:

- folder: mybackup

file: backup.sql

Copy

In this case we want to create a MySQL data image whose source is backup and is named dev.

The folder is relative to the location of your yaml file.

You can restore multiple backup files by adding multiple entries in the backups section.

sourceType: backup

name: dev

engine: mysql

version: 5.7

backups:

- folder: mybackup

file: backup1.sql

- folder: mybackup

file: backup2.sql

Copy

In this case we want to create a MySQL data image by restoring two backup files: backup1.sql and backup2.sql.

## Size limit[#](https://docs.spawn.cc/data-image-configuration/backup#size-limit)

Spawn imposes a 500Gi limit on the size of data images.

## Tutorial[#](https://docs.spawn.cc/data-image-configuration/backup#tutorial)

As a prerequisite you should've followed the instructions to [install spawnctl](https://docs.spawn.cc/how-tos/installation)

* PostgreSQL
* MySQL
* MariaDB
* SQL Server

Spawn currently supports two formats of PostgreSQL backups:

* Plain
* Custom

You can read more about these in the [PostgreSQL documentation](https://www.postgresql.org/docs/current/app-pgdump.html).

Depending on the type of pg\_dump backup you create, your data image definition file will require different configuration.

* Plain
* Custom

Plain backups are the default type. If you omit the format value in the data image definition file, Spawn will assume the postgres backup is of the "plain" format.

For plain backups, you'll need to specify the following items in the data image definition file:

* folder
* file

## Tutorial (video)[#](https://docs.spawn.cc/data-image-configuration/backup#tutorial-video-2)

Watch this video for how to create data images from a PostgreSQL database via a backup. Or follow the step-by-step instructions below.

## Tutorial (step-by-step)[#](https://docs.spawn.cc/data-image-configuration/backup" \l "tutorial-step-by-step-2" \o "Direct link to heading)

### Create a backup[#](https://docs.spawn.cc/data-image-configuration/backup#create-a-backup-3)

Create a subdirectory for the backup file and data image YAML definition file:

mkdir -p ~/spawn/postgres-backup-image/ && cd ~/spawn/postgres-backup-image/

Copy

Assuming there is a database called pagila running at prod-db.example.com on port 5432 and is accessible to the user admin, you can run the following pg\_dump command:

pg\_dump -h prod-db.example.com -U admin --create pagila > dump.sql

Copy

### Create a data image yaml file[#](https://docs.spawn.cc/data-image-configuration/backup#create-a-data-image-yaml-file-3)

Now that we've got a dump.sql file produced by pg\_dump we can create our Spawn data image definition YAML file called image.yaml in the current directory with the following contents:

*# ~/spawn/postgres-backup-image/image.yaml*

sourceType: backup

name: pagila

engine: postgresql

version: 12.0

backups:

- folder: /<YOUR\_HOME\_DIRECTORY>/spawn/postgres-backup-image/

file: dump.sql

tags:

- production

Copy

This instructs Spawn to create a data image called pagila on Postgres 12. It will upload the file /<YOUR\_HOME\_DIRECTORY>/spawn/postgres-backup-image/dump.sql to Spawn and restore that backup using psql, snapshot the state of the restored database and then produce the data image ready for consumption.

The data image will also have a production tag associated with it, so you later know that this data image represents a backup of production.

### Create the image[#](https://docs.spawn.cc/data-image-configuration/backup#create-the-image)

Run

spawnctl create data-image --file=./image.yaml

Copy

This will create the data image from the backup defined in the image.yaml file we created.

**Database engine support**

Spawn currently supports the following database engines:

| **Engine** | **Version** |
| --- | --- |
| MSSQL | 2017, 2019 |
| PostgreSQL | 11.0, 12.0, 13.2 |
| MySQL | 5.7, 8.0 |
| MariaDB | 10.6.3 |
| MongoDB | 3.6.8 |
| Redis | 6.0.9 |

For more information on how to specify which engine and version to use in your data images, please follow this [link](https://docs.spawn.cc/concepts/data-image#supported-engines)

**Current support**[**#**](https://docs.spawn.cc/engine-support#current-support)

| **command** | **PostgreSQL** | **MySQL** | **MariaDB** | **MSSQL** | **MongoDB** | **Redis** |
| --- | --- | --- | --- | --- | --- | --- |
| spawnctl create data-image empty | ✅ | ✅ | ✅ | ✅ | ✅ | ✅ |
| spawnctl create data-image scripts | ✅ | ✅ | ✅ | ✅ | ✅ | ⛔ |
| spawnctl create data-image backup | ✅ | ✅ | ✅ | ✅ | ⛔ | ⛔ |
| spawnctl delete data-image | ✅ | ✅ | ✅ | ✅ | ✅ | ✅ |
| spawnctl get data-image | ✅ | ✅ | ✅ | ✅ | ✅ | ✅ |
| spawnctl create data-container | ✅ | ✅ | ✅ | ✅ | ✅ | ✅ |
| spawnctl delete data-container | ✅ | ✅ | ✅ | ✅ | ✅ | ✅ |
| spawnctl get data-container | ✅ | ✅ | ✅ | ✅ | ✅ | ✅ |
| spawnctl save data-container | ✅ | ✅ | ✅ | ✅ | ✅ | ✅ |
| spawnctl reset data-container | ✅ | ✅ | ✅ | ✅ | ✅ | ✅ |
| spawnctl load data-container | ✅ | ✅ | ✅ | ✅ | ✅ | ✅ |
| spawnctl graduate data-container | ✅ | ✅ | ✅ | ✅ | ✅ | ✅ |

**Is there something missing?**[**#**](https://docs.spawn.cc/engine-support#is-there-something-missing)

Let us know, drop us a line at spawn@red-gate.com and tell us which database engines we should support.

The next section explains how to use the command-line to control Spawn.

**Spawn and Github Actions**

Spawn is easy to integrate with [Github Actions](https://github.com/features/actions). We provide actions that plug directly into your workflows allowing you to use Spawn functionality without having to install and script spawnctl on your agents.

Most of the [spawnctl commands](https://docs.spawn.cc/commands/spawnctl) have their own action:

* [Create data image](https://github.com/red-gate/create-spawn-data-image)
* [Delete data image](https://github.com/red-gate/delete-spawn-data-image)
* [Create data container](https://github.com/red-gate/create-spawn-data-container)
* [Save data container](https://github.com/red-gate/save-spawn-data-container)
* [Reset data container](https://github.com/red-gate/reset-spawn-data-container)
* [Delete data container](https://github.com/red-gate/delete-spawn-data-container)

If you've used [spawnctl](https://docs.spawn.cc/commands/spawnctl), the action inputs and outputs should be familiar. See the README.md in each action's repository to learn what inputs each action requires and what outputs it provides, or examine the example workflow below.

**Example workflow**[**#**](https://docs.spawn.cc/cicd/github-actions#example-workflow)

The following example workflow shows how each of the available Github actions is used. The workflow runs through the following steps:

* Create a data image
* Create a data container from the image
* Display the connection details for the new data container
* Save and reset the data container
* Delete the container and the image

name: Demo Spawn Actions

*# Set up the workflow with a manual trigger.*

on:

workflow\_dispatch: {}

jobs:

demo-actions:

name: Demo Spawn Github Actions

runs-on: ubuntu-latest

steps:

- uses: actions/checkout@v2

- name: create data image

id: create-image

uses: red-gate/create-spawn-data-image/@v1

with:

dataImageYaml: demo/github/image.yaml

lifetime: '5h'

- name: create data container

id: create-container

uses: red-gate/create-spawn-data-container/@v1

with:

dataImage: ${{ steps.create-image.outputs.dataImageName }}

lifetime: '5h'

- name: display connection details

run: |

echo "Connection details for the new container."

echo "Plug these into eg Flyway to do migration testing."

echo "These values are sensitive and will be masked."

echo "container name: $containerName"

echo "container host: $containerHost"

echo "container port: $containerPort"

echo "container username: $containerUsername"

echo "container password: $containerPassword"

env:

containerName: ${{ steps.create-container.outputs.dataContainerName }}

containerHost: ${{ steps.create-container.outputs.dataContainerHost }}

containerPort: ${{ steps.create-container.outputs.dataContainerPort }}

containerUsername: ${{ steps.create-container.outputs.dataContainerUsername }}

containerPassword: ${{ steps.create-container.outputs.dataContainerPassword }}

- name: save data container

uses: red-gate/save-spawn-data-container/@v1

with:

dataContainer: ${{ steps.create-container.outputs.dataContainerName }}

- name: reset data container

uses: red-gate/reset-spawn-data-container/@v1

with:

dataContainer: ${{ steps.create-container.outputs.dataContainerName }}

- name: delete data container

if: always()

uses: red-gate/delete-spawn-data-container/@v1

with:

dataContainer: ${{ steps.create-container.outputs.dataContainerName }}

- name: delete data image

if: always()

uses: red-gate/delete-spawn-data-image/@v1

with:

dataImage: ${{ steps.create-image.outputs.dataImageName }}

env:

SPAWNCTL\_ACCESS\_TOKEN: ${{ secrets.SPAWNCTL\_ACCESS\_TOKEN }}

Copy

Note that all of the actions require a SPAWNCTL\_ACCESS\_TOKEN [secret](https://docs.github.com/en/actions/reference/encrypted-secrets) available to the repository in which the workflow runs.

**Feedback**[**#**](https://docs.spawn.cc/cicd/github-actions#feedback)

We welcome all feedback on the Github actions. Please open an issue or a pull request on the individual action repositories.

# How to - Installation and onboarding

Spawn makes it easier to include the database in your dev environments and DevOps pipelines.

Spin up instant copies of your databases in Spawn for each project, branch and pipeline. Self service and automated copies means you never need to be blocked by sharing a database with other team members.

This is the first in a series of short guides for getting up and running quickly with Spawn.

## Create a Spawn account[#](https://docs.spawn.cc/how-tos/installation#create-a-spawn-account)

Login to the Spawn web app to create your account.

[LOGIN WITH GITHUB](https://app.spawn.cc/login)

## Installation[#](https://docs.spawn.cc/how-tos/installation#installation)

### Windows[#](https://docs.spawn.cc/how-tos/installation#windows)

1. Download [spawnctl](https://run.spawn.cc/spawnctl.exe)
2. Add the folder containing spawnctl to the path

### Mac and Linux[#](https://docs.spawn.cc/how-tos/installation#mac-and-linux)

1. Run this

curl -sL https://run.spawn.cc/install | sh

Copy

1. Add spawnctl to your path

export PATH=$PATH:$HOME/.spawnctl/bin

Copy

## Authenticate spawnctl with your Spawn account[#](https://docs.spawn.cc/how-tos/installation#authenticate-spawnctl-with-your-spawn-account)

Authenticate spawnctl to link your Spawn account

spawnctl auth

Copy

## Using Spawn[#](https://docs.spawn.cc/how-tos/installation#using-spawn)

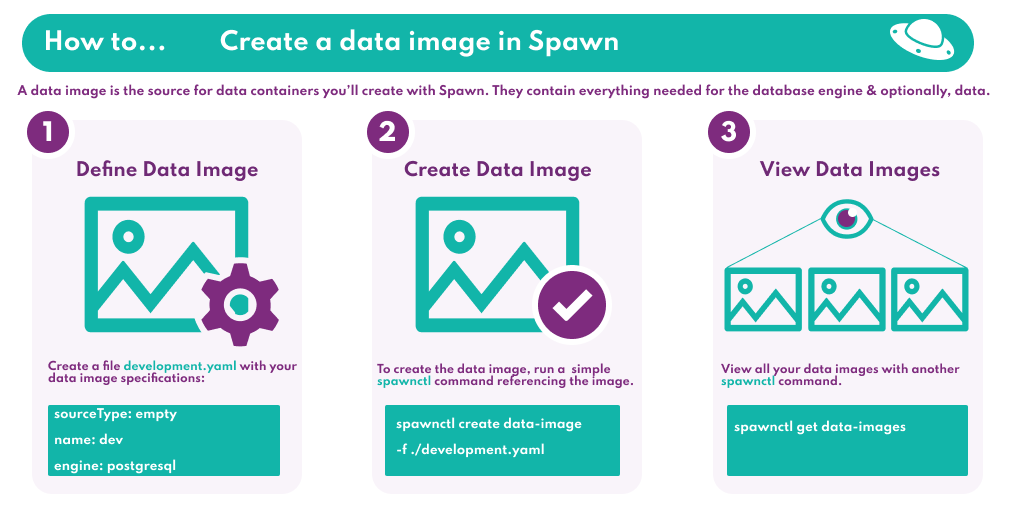
You are now ready to start using spawn, create [data images](https://docs.spawn.cc/commands/spawnctl-di-create) and [data containers](https://docs.spawn.cc/commands/spawnctl-dc-create) to begin your journey.

**How to - Creating a data image**

A data image is the source for future data containers that you will create with Spawn, they contain everything needed for the database engine as well as the databases and optionally, data. Data images can be created as [empty](https://docs.spawn.cc/data-image-configuration/empty) or from [backup](https://docs.spawn.cc/data-image-configuration/backup)/[script](https://docs.spawn.cc/data-image-configuration/scripts) files.

Spawn is currently in open beta. [Complete the installation instructions](https://docs.spawn.cc/how-tos/installation) to get access.

This is the second in a series of short guides for getting up and running quickly with Spawn.



**Image definition**[**#**](https://docs.spawn.cc/how-tos/dataimage#image-definition)

Similar to [Docker](https://www.docker.com/), an image is created based on a definition contained in a file. To get started quickly, we will create a simple image which will allow us to create an empty PostgreSQL container.

* Create a file development.yaml with your data image specifications.

sourceType: empty

name: dev

engine: postgresql

version: 11.0

Copy

Interested in other **engines**? Go to our [data image page](https://docs.spawn.cc/concepts/data-image#supported-engines) to see how you can use other engines we support.

**Create a data image**[**#**](https://docs.spawn.cc/how-tos/dataimage#create-a-data-image)

To create the data image, we simply run a [spawnctl](https://docs.spawn.cc/how-tos/installation) command referencing the file created above.

spawnctl create data-image -f ./development.yaml

Copy

**Seeing your image**[**#**](https://docs.spawn.cc/how-tos/dataimage#seeing-your-image)

View all your data images using the following command.

spawnctl get data-images

Copy

**Using your data image**[**#**](https://docs.spawn.cc/how-tos/dataimage#using-your-data-image)

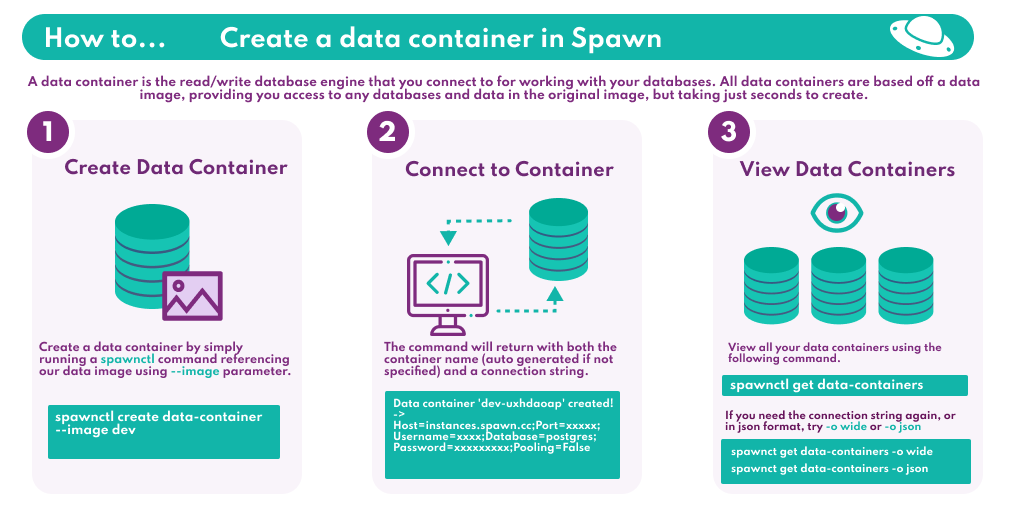
Now you have a data image available, you can create [data containers](https://docs.spawn.cc/commands/spawnctl-dc-create).

**How to - Creating a data container**

A data container is the read/write database engine that you connect to for working with your databases. All data containers are based off a [data image](https://docs.spawn.cc/how-tos/dataimage), providing you access to any databases and data in the original image, but taking just seconds to create. All data containers that you create are isolated from any others, allowing you or your team members to make changes without worrying about impacting anything else.

Spawn is currently in open beta. [Complete the installation instructions](https://docs.spawn.cc/how-tos/installation) to get access.

This is part of a series of short guides for getting up and running quickly with Spawn.



**Create a data container**[**#**](https://docs.spawn.cc/how-tos/datacontainer#create-a-data-container)

Similar to [Docker](https://www.docker.com/), a data container is created based on a data image. To create a data container in this example, you should have already created a [data image](https://docs.spawn.cc/how-tos/dataimage) named dev.

To create the data container, we simply run a [spawnctl](https://docs.spawn.cc/how-tos/installation) command referencing our data image using the --image parameter.

spawnctl create data-container --image dev

Copy

**Connecting to your data container**[**#**](https://docs.spawn.cc/how-tos/datacontainer#connecting-to-your-data-container)

Once the data container has been created successfully, the command will return with both the container name (auto generated if not specified) and a connection string. You can use this connection string in an application that needs to connect to the database, or use the values such as Host,Port,Username and Password to connect to your data container using your usual client tools.

Data container 'dev-uchdwpkp' (00003) created!

-> Host=instances.spawn.cc;Port=xxxxx;Username=xxxx;Database=postgres;Password=xxxxxxxxx

Copy

**Seeing your data containers**[**#**](https://docs.spawn.cc/how-tos/datacontainer#seeing-your-data-containers)

View all your data containers using the following command.

spawnctl get data-containers

Copy

ID Name Revision Status Engine CreatedAt

00001 dev-cxmwgjeo rev.0 Completed PostgreSQL 15 minutes ago

00002 dev-eukjbnkm rev.0 Completed PostgreSQL 10 minutes ago

00003 dev-uchdwpkp rev.0 Completed PostgreSQL 7 minutes ago

Copy

If you need the connection string again for a data container, add -o wide.

spawnctl get data-containers -o wide

Copy

To return your data containers, including connection details in a format more suitable for parsing programmatically, add -o json

spawnctl get data-containers -o json

Copy

**Using your data container**[**#**](https://docs.spawn.cc/how-tos/datacontainer#using-your-data-container)

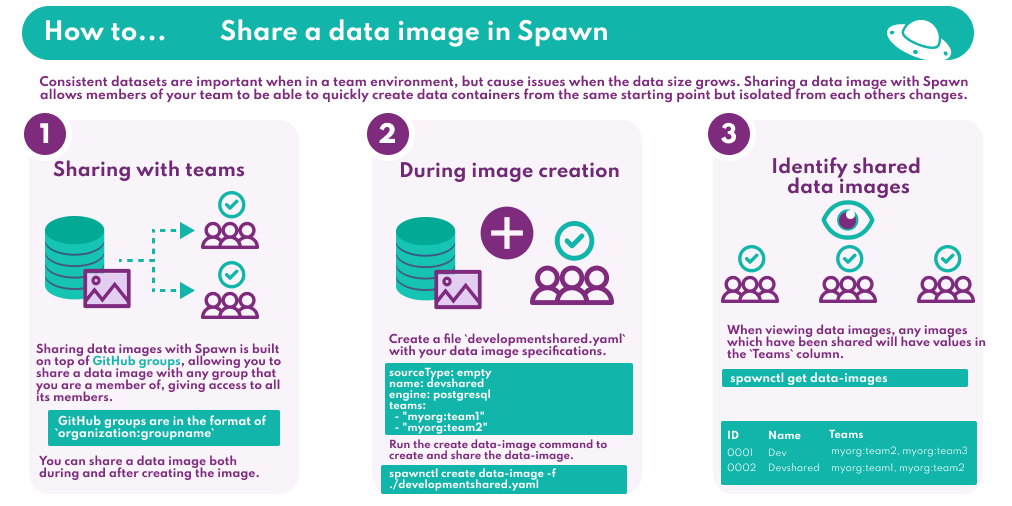
Now you are able to create data containers, you can perform advanced actions such as [save](https://docs.spawn.cc/commands/spawnctl-dc-save), [graduate](https://docs.spawn.cc/commands/spawnctl-dc-graduate), [reset](https://docs.spawn.cc/commands/spawnctl-dc-reset) and [load](https://docs.spawn.cc/commands/spawnctl-dc-load).

# How to - Sharing a data image

Working with a consistent dataset is important when in a team environment, but can cause issues when the data size grows. Sharing a [data image](https://docs.spawn.cc/concepts/data-image) with Spawn allows all members of your team to be able to quickly create [data containers](https://docs.spawn.cc/concepts/data-container), all getting the same starting point but being isolated from each other's changes. You can also share data images created using the [graduate](https://docs.spawn.cc/commands/spawnctl-dc-graduate) command, allowing you to share changes you've made to your data container such as recreating a bug.

Spawn is currently in open beta. [Complete the installation instructions](https://docs.spawn.cc/how-tos/installation) to get access.

This is part of a series of short guides for getting up and running quickly with Spawn.



## Prerequisites[#](https://docs.spawn.cc/how-tos/share-dataimage#prerequisites)

In order to be able to share data images within your team, you and your teammates must be members of the same Spawn organization. Once you have successfully signed up with the Spawn service, you can add further users to your Spawn organization by running:

spawnctl create user --name <teammate's name> --email <teammate's email>

Copy

Now when the new user signs up with Spawn, they will be placed in your Spawn organization. Any other members of your team can be added to your Spawn organization in the same way. You can then follow the rest of the steps below to start sharing data images with your team.

##### NOTE

If the users you want to share images with have already signed up to Spawn, please contact us and we will move those users into your Spawn organization manually.

## Sharing with teams[#](https://docs.spawn.cc/how-tos/share-dataimage#sharing-with-teams)

Sharing data images with Spawn is built on top of [GitHub groups](https://docs.github.com/en/free-pro-team@latest/github/setting-up-and-managing-organizations-and-teams/collaborating-with-groups-in-organizations), allowing you to share a data image with any group that you are a member of, giving access to all its members.

GitHub groups are in the format of organization:groupname

You can share a data image both during and after creating the image, we'll cover both below.

## During image creation[#](https://docs.spawn.cc/how-tos/share-dataimage#during-image-creation)

When creating an image, we can share it with teams at the same time by adding them to the image definition file.

1. Create a file developmentshared.yaml with your data image specifications.

sourceType: empty

name: devshared

engine: postgresql

version: 11.0

teams:

- "myorg:team1"

- "myorg:team2"

Copy

1. Run the create data-image command to create and share the data-image.

spawnctl create data-image -f ./developmentshared.yaml

Copy

## After image creation[#](https://docs.spawn.cc/how-tos/share-dataimage#after-image-creation)

Share an existing data image using the [update](https://docs.spawn.cc/commands/spawnctl-di-update) command.

spawnctl update data-image dev --team myorg:team2 --team myorg:team3

Copy

## Identifying shared data images[#](https://docs.spawn.cc/how-tos/share-dataimage#identifying-shared-data-images)

When viewing data images, any images which have been shared will have values in the Teams column.

spawnctl get data-images

Copy

ID Name Tags Engine Status CreatedAt Teams

00001 dev PostgreSQL Completed 1 day ago myorg:team2, myorg:team3

00002 devshared PostgreSQL Completed 2 minutes ago myorg:team1, myorg:team2

Copy

## Using your data image[#](https://docs.spawn.cc/how-tos/share-dataimage#using-your-data-image)

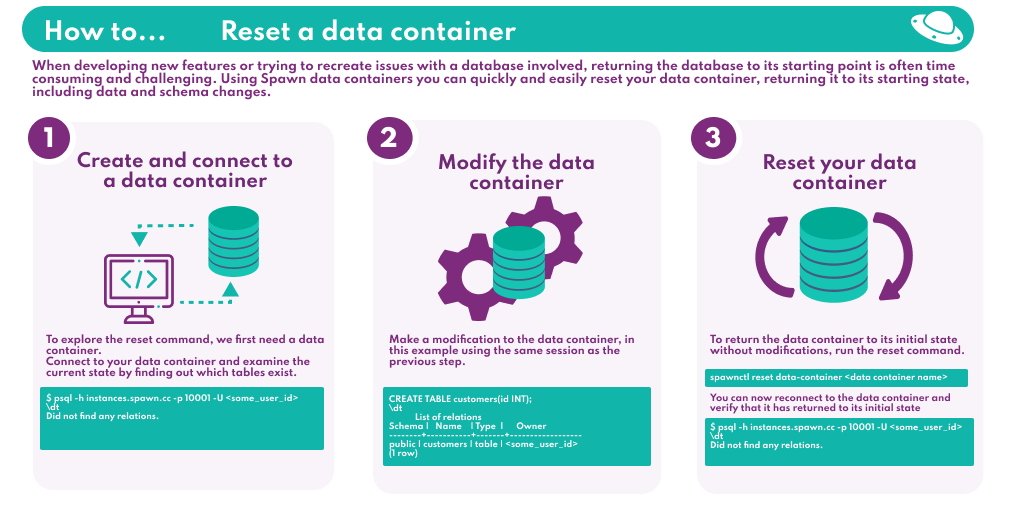
Now you have a data image available and shared with team members, you can all create [data containers](https://docs.spawn.cc/commands/spawnctl-dc-create) and have access to the same data, in your own isolated environments.

**How to - Resetting a data container**

When developing new features or trying to recreate issues with a database involved, returning the database to its starting point is often time consuming and challenging. Using Spawn [data containers](https://docs.spawn.cc/how-tos/datacontainer) you can quickly and easily [reset](https://docs.spawn.cc/commands/spawnctl-dc-reset) your data container, returning it to its starting state, including data and schema changes.

Spawn is currently in open beta. [Complete the installation instructions](https://docs.spawn.cc/how-tos/installation) to get access.

This is part of a series of short guides for getting up and running quickly with Spawn.



**Create and connect to a data container**[**#**](https://docs.spawn.cc/how-tos/reset-datacontainer#create-and-connect-to-a-data-container)

To explore the [reset](https://docs.spawn.cc/commands/spawnctl-dc-reset) command, we first need a [data container](https://docs.spawn.cc/how-tos/datacontainer). If you don't have a data container available, you can [follow this guide](https://docs.spawn.cc/how-tos/datacontainer).

This example uses an empty PostgeSQL data container, but you can follow along with any type by modifying the connection and queries accordingly. Connect to your data container and examine the current state by finding out which tables exist. Connect using whatever tool you would normally for the database engine type you are working with.

$ psql -h instances.spawn.cc -p 10001 -U <some\_user\_id>

Password *for* user <some\_user\_id>:

psql (10.5, server 11.0 (Debian 11.0-1.pgdg90+2))

<some\_user\_id>=*# \dt*

Did not find any relations.

Copy

In this case, the data container is empty so no tables exist.

**Modify your data container**[**#**](https://docs.spawn.cc/how-tos/reset-datacontainer#modify-your-data-container)

Make a modification to the data container, in this example using the same session as the previous step.

<some\_user\_id>=*# CREATE TABLE customers(id INT);*

CREATE TABLE

<some\_user\_id>=*# \dt*

List of relations

Schema | Name | Type | Owner

--------+-----------+-------+------------------

public | customers | table | <some\_user\_id>

(1 row)

Copy

There is now a table created in the data container that wasn't present when it was first created.

**Reset your data container**[**#**](https://docs.spawn.cc/how-tos/reset-datacontainer#reset-your-data-container)

To return the data container to its initial state without modifications, run the reset command.

$ spawnctl reset data-container <data container name>

Successfully reset data container '<data container name>'

Copy

You can now reconnect to the data container (the connection details remain the same but the previous connection will have been terminated) and verify that it has returned to its initial state.

$ psql -h instances.spawn.cc -p 10001 -U <some\_user\_id>

Password *for* user <some\_user\_id>:

psql (10.5, server 11.0 (Debian 11.0-1.pgdg90+2))

<some\_user\_id>=*# \dt*

Did not find any relations.

Copy

**Using your data container**[**#**](https://docs.spawn.cc/how-tos/reset-datacontainer#using-your-data-container)

Now you are able to create and reset data containers, other advanced actions are available such as [save](https://docs.spawn.cc/commands/spawnctl-dc-save), [graduate](https://docs.spawn.cc/commands/spawnctl-dc-graduate) and [load](https://docs.spawn.cc/commands/spawnctl-dc-load).

# How to - Scheduling data image creation

One of the many benefits of Spawn is the ability to work with production-like datasets in all environments regardless of the size due to instant data container creation.

However to take advantage of this you first need to have a data image containing that data.

In this guide, you'll explore how to set up a scheduled pipeline in a CI environment to regularly create data images from your production-like datasets.

Spawn is currently in open beta. [Complete the installation instructions](https://docs.spawn.cc/how-tos/installation) to get access.

## Prerequisites[#](https://docs.spawn.cc/how-tos/ci-scheduled-image-creation#prerequisites)

* We'll assume you already have a masked backup of your production environment.
* We'll assume that the agent you're using to invoke Spawn has access to the masked production backup file.

### Backing up your database[#](https://docs.spawn.cc/how-tos/ci-scheduled-image-creation#backing-up-your-database)

Depending on your environment, you'll have to get hold of a backup of your database you'd like to create an image from. The following table gives some suggestions of how you can do this depending on your environment. This table is by no means exhaustive, but following these instructions to generate a backup file has been tested and confirmed to work with Spawn.

| **Engine** | **Documentation** |
| --- | --- |
| PostgreSQL | [pg\_dump](https://www.postgresql.org/docs/current/app-pgdump.html) |
| MySQL | [mysqldump](https://dev.mysql.com/doc/refman/5.7/en/mysqldump-sql-format.html) |
| MSSQL RDS | [Native MSSQL RDS Backups](https://aws.amazon.com/premiumsupport/knowledge-center/native-backup-rds-sql-server/) |
| MSSQL On-prem | [MSSQL Backups](https://docs.microsoft.com/en-us/sql/relational-databases/backup-restore/create-a-full-database-backup-sql-server?view=sql-server-ver15) |
| Mongo | [mongodump](https://docs.mongodb.com/database-tools/mongodump/#mongodb-binary-bin.mongodump) |

## Setting up Spawn in CI[#](https://docs.spawn.cc/how-tos/ci-scheduled-image-creation#setting-up-spawn-in-ci)

### Authenticating[#](https://docs.spawn.cc/how-tos/ci-scheduled-image-creation#authenticating)

When you're using Spawn interactively, you'll start off by running spawnctl auth. The authentication token you receive has a configured expiration time. **This is no good for CI environments as an interactive authentication workflow is impossible**.

Therefore, we must use [access tokens](https://docs.spawn.cc/commands/spawnctl-accesstoken-create) to authenticate against Spawn as these have no expiration ([though can be revoked if necessary](https://docs.spawn.cc/commands/spawnctl-accesstoken-delete)).

spawnctl create access-token --purpose "Scheduled data image creation from masked production backup"

Access token generated: <long\_access\_token\_string>

Copy

This command will create an access token with a given purpose. It's best practice to give clear, human-readable purposes for your access tokens so you can understand what they're used for in the future.

Now that we have this access token, you should set it up as a **secret in your CI pipeline of choice** so that your agents can access it.

## Creating the data image[#](https://docs.spawn.cc/how-tos/ci-scheduled-image-creation#creating-the-data-image)

Spawn can be used in any CI environment that supports running scripts. In this case, we're using a Bash script on a Linux agent, but you could use whichever OS and scripting language you like.

### Defining the data image to create[#](https://docs.spawn.cc/how-tos/ci-scheduled-image-creation#defining-the-data-image-to-create)

First, we'll need a file in source control that represents the data image we'd like to create:

name: WidgetStore

sourceType: backup

engine: postgresql

version: 11.0

teams:

- myorg:developers

- myorg:dbas

tags:

- latest-production

backups:

- folder: /backups/

file: production-masked-latest.bak

Copy

There's some important best practices to mention in this yaml:

* The image is [**shared with multiple teams**.](https://docs.spawn.cc/how-tos/share-dataimage) In this case, Developers and DBAs in my organisation
* The image is tagged with latest-production
  + This means that consumers can always run spawnctl create data-container --image WidgetStore:latest-production and they'll **receive a data container with the latest production data**

As called out in the prerequisites, we've assumed this agent can access the masked production backup. The yaml assumes those backups reside in the /backups/ directory on the CI agent.

### Creating the data image[#](https://docs.spawn.cc/how-tos/ci-scheduled-image-creation#creating-the-data-image-1)

Now we have the data image yaml defined in source control, we'll actually create it in our CI pipeline.

Here's an example of the script we'll use to do just that:

#!/bin/bash

*# Install the latest version of spawnctl on the agent*

curl https://run.spawn.cc/install | sh

export PATH=$PWD:$HOME/.spawnctl/bin

*# Create the data image*

spawnctl create data-image \

-f $GIT\_CHECKOUT\_DIR/widgetstore-backup.yaml \

--accessToken $SPAWNCTL\_ACCESS\_TOKEN \

--tag $PIPELINE\_RUN\_ID \

--lifetime 48h \

-q

Copy

This script is very short, as we're only downloading spawnctl and then [creating a data image](https://docs.spawn.cc/commands/spawnctl-di-create).

The data image YAML file contains all the information about how to construct that image.

This pipeline can be configured to run as often as you'd like to refresh your data images.

### Authenticating[#](https://docs.spawn.cc/how-tos/ci-scheduled-image-creation#authenticating-1)

The $SPAWNCTL\_ACCESS\_TOKEN environment variable is the access token we created and made available to the agents in previous steps.

### An extra tag for tracing[#](https://docs.spawn.cc/how-tos/ci-scheduled-image-creation#an-extra-tag-for-tracing)

You'll notice that we've also appended the --tag $PIPELINE\_RUN\_ID flag to the command. This is another best practice, as it will add a tag in addition to latest-production defined in the YAML file. In this case, the additional tag is the pipeline run identifier that triggered this data image creation. This means you'll be able to identify which images were created by which pipeline invocation.

### Image lifetimes[#](https://docs.spawn.cc/how-tos/ci-scheduled-image-creation#image-lifetimes)

We've also specified a [lifetime for the image](https://docs.spawn.cc/commands/spawnctl-di-create##lifetime).

This sets a retention period for the data image. In this case, our data image is only valid for 7 days before automatically being cleaned up by Spawn. This prevents us from having stale data images that would no longer be useful.

### Suppressing progress output[#](https://docs.spawn.cc/how-tos/ci-scheduled-image-creation#suppressing-progress-output)

We've also added the -q flag to suppress output from spawnctl to avoid polluting the CI pipeline logs with progress messages.

## Reviewing the new images[#](https://docs.spawn.cc/how-tos/ci-scheduled-image-creation#reviewing-the-new-images)

As a developer in my organisation, I can now see these newly created data images and start using them in development:

$ spawnctl get data-images

ID Name Tags Engine Status CreatedAt Teams ExpiresAt

10001 WidgetStore 234 PostgreSQL:11.0 Completed 3 days ago myorg:developers, myorg:dbas 4 days from now

10002 WidgetStore 235 PostgreSQL:11.0 Completed 2 days ago myorg:developers, myorg:dbas 5 days from now

10003 WidgetStore 236, latest-production PostgreSQL:11.0 Completed

**How to - Run spawnctl using docker**

To pull and run the spawnctl container image, [spawnctl](https://hub.docker.com/r/redgatefoundry/spawnctl)

docker run --rm redgatefoundry/spawnctl

Copy

Standard [interactive authentication](https://docs.spawn.cc/commands/authenticate) using spawnctl auth will not work for docker containers

Use one of the following methods to access spawn using [access token](https://docs.spawn.cc/commands/spawnctl-accesstoken-create):

* Using SPAWNCTL\_ACCESS\_TOKEN environment variable

docker run --rm -e SPAWNCTL\_ACCESS\_TOKEN:<access-token-string> redgatefoundry/spawnctl

Copy

* Append --accessToken to spawnctl command

docker run --rm redgatefoundry/spawnctl get all --accessToken <access-token-string>

**Introduction**

This section will introduce you to the fundamental components and concepts of Spawn:

| **Source** | **Data Image** | **Team** | **Tag** | **Data Container** | **Branch** |
| --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |

If you're already familiar with these concepts, and want to work through some examples, you can skip to the Commands section of this documentation and [learn about Spawn commands](https://docs.spawn.cc/commands/spawnctl).

# Source

## Definition[#](https://docs.spawn.cc/concepts/source#definition)

Source describes the place from which data in the Data image comes, and its type.

### Supported sources[#](https://docs.spawn.cc/concepts/source#supported-sources)

* empty
* scripts
* backup

The source of data and schema is defined in the yaml file that represents the Data image.

## Example[#](https://docs.spawn.cc/concepts/source#example)

In this example, the source is an empty PostgreSQL database.

name: dev

sourceType: empty

engine: postgresql

version: 11.0

# Data Image

## Definition[#](https://docs.spawn.cc/concepts/data-image#definition)

A data image (data-image) is a fundamental component that contains the necessary information to Spawn a database. It is analogous to a Docker Image.

A data image definition is composed of at least of a name (name), a source type (sourceType), an engine (engine), and an engine version (version) as seen in the example below.

## Example[#](https://docs.spawn.cc/concepts/data-image#example)

In this example, we have an application (myApp) that uses a MySQL database to store customer data.

The data image is defined in a single yaml file.

myApp.yaml

Copy

The yaml file contains all the instructions necessary to create a data image.

name: dev

sourceType: backup

engine: mysql

version: 5.7

backups:

- folder: ./data

file: backup.sql

Copy

You can't connect to a data-image, it's a read-only representation of your database.

Spawn imposes a 500Gi limit on the size of data images.

### Supported engines[#](https://docs.spawn.cc/concepts/data-image#supported-engines)

Please use the following strings when choosing which engine to use in your data image definition:

* mssql
* postgresql
* mysql
* mariadb
* mongodb
* redis

### Supported engine versions[#](https://docs.spawn.cc/concepts/data-image#supported-engine-versions)

Please visit our [engine support page](https://docs.spawn.cc/engine-support) to view the current versions we support for each engine.

### Supported source types[#](https://docs.spawn.cc/concepts/data-image#supported-source-types)

Please use the following strings when choosing which source type to use in your data image definition:

* empty | [details](https://docs.spawn.cc/data-image-configuration/empty)
* scripts | [details](https://docs.spawn.cc/data-image-configuration/scripts)
* backup | [details](https://docs.spawn.cc/data-image-configuration/backup)

You can connect to a data container data-container, created from the data image.

**Team**

**Definition**[**#**](https://docs.spawn.cc/concepts/data-image-team#definition)

A Team is used to share a data image (data-image) with other users. You can share your data image with [GitHub groups](https://help.github.com/en/articles/collaborating-with-groups-in-organizations) that you are a member of. A data image can be shared with multiple teams.

Team sharing currently only works with GitHub authentication. Microsoft authentication is **not supported**.

**Example**[**#**](https://docs.spawn.cc/concepts/data-image-team#example)

In this example, we will add two teams to the yaml file defining our data image.

name: dev

sourceType: backup

engine: mysql

version: 5.7

backups:

- folder: ./data

file: backup.sql

teams:

- "red-gate:spawn-developers"

- "red-gate:spawn-admins"

**Tag**

**Definition**[**#**](https://docs.spawn.cc/concepts/data-image-tag#definition)

A Tag differentiates multiple data images (data-image) with the same name. It is analogous to a Docker Tag.

data images can have multiple tags, however a data image name and tag combination must be unique. Creating a new data image with a name and tag combination that already exists will remove said tag from the existing data image.

Referencing an existing data image with a tag in commands can be done in the format name:tag.

Tags must be less than or equal to 128 characters, lower case, not starting with a period or dash and containing only the following characters: A-z 0-9 - \_ .

**Example**[**#**](https://docs.spawn.cc/concepts/data-image-tag#example)

In this example, we will add two tags to the yaml file defining our data image.

name: dev

sourceType: backup

engine: mysql

version: 5.7

backups:

- folder: ./data

file: backup.sql

teams:

- "red-gate:spawn-developers"

- "red-gate:spawn-admins"

tags:

- "v1.0"

- "development"

# Data Container

## Definiton[#](https://docs.spawn.cc/concepts/data-container#definiton)

A data container (data-container) is created from the instructions described in the data image. It's a running instance of your data-image; it's a database server.

When Spawn returns connection details, it's the data-container that you're working against.

Any changes you make apply only to the data-container, not to the data-image.

**Branch**

**Definition**[**#**](https://docs.spawn.cc/concepts/branch#definition)

The first time a data-container is created from a data-image, a new timeline is created by default.

You can think of the default timeline as the default Git branch (main).

Changes made to your database (data-container) can be recorded along this timeline as revisions.

When you load a previous revision, a new timeline is created.

You can think of this as a new branch.

**Example**[**#**](https://docs.spawn.cc/concepts/branch#example)

In this example we create a new data-container and make 3 revisions to it.

$ spawnctl create data-image -f ./development.yaml

Data image 'dev' (10001) created!

$ spawnctl get data-images

NAME IMAGE ID ENGINE STATUS MESSAGE CREATED

dev 10001 PostgreSQL 2 Created 2 minutes ago

$ spawnctl create data-container --image dev

Data container 'dev-rambbomj' (10001) created!

-> Host=instances.spawn.cc;Port=53223;User ID=<some\_user\_id>;Password=<some\_password>;

$ spawnctl get data-containers

NAME CONTAINER ID REVISION STATUS MESSAGE ENGINE CREATED

dev-rambbomj 10001 rev.0 2 Running PostgreSQL 1 minute ago

Copy

Notice the created data container has revision rev.0.

You can now make changes to that container by executing any SQL you like.

In this example we connect to the PostgreSQL data container (database) using [psql](http://postgresguide.com/utilities/psql.html).

$ psql -h instances.spawn.cc -p 53223 -U <some\_user\_id>

Password for user <some\_user\_id>:

psql (10.5, server 11.0 (Debian 11.0-1.pgdg90+2))

SSL connection (protocol: TLSv1.2, cipher: ECDHE-RSA-AES256-GCM-SHA384, bits: 256, compression: off)

Type "help" for help.

<some\_user\_id>=# CREATE TABLE customers(id INT);

CREATE TABLE

<some\_user\_id>=# \dt

List of relations

Schema | Name | Type | Owner

--------+-----------+-------+------------------

public | customers | table | <some\_user\_id>

(1 row)

Copy

We can now perform a save operation on this data container.

$ spawnctl save data-container dev

Saving container....

Data container 'dev-rambbomj' saved!

New revision is 'rev.1'

Copy

Notice the data container has now revision rev.1.

$ spawnctl get data-containers

NAME CONTAINER ID REVISION STATUS MESSAGE ENGINE CREATED

dev-rambbomj 10001 rev.1 2 Running PostgreSQL 1 minute ago

Copy

If you perform more save operations, revisions will be created as you do so.

After 3 save operations, this branch now contains rev1, rev2 and rev3.

You can now load any revision you like

In our case we load rev1. A new branch is created.

$ spawnctl load data-container dev-rambbomj --revision=rev.1

Data container 'dev-rambbomj' loaded!

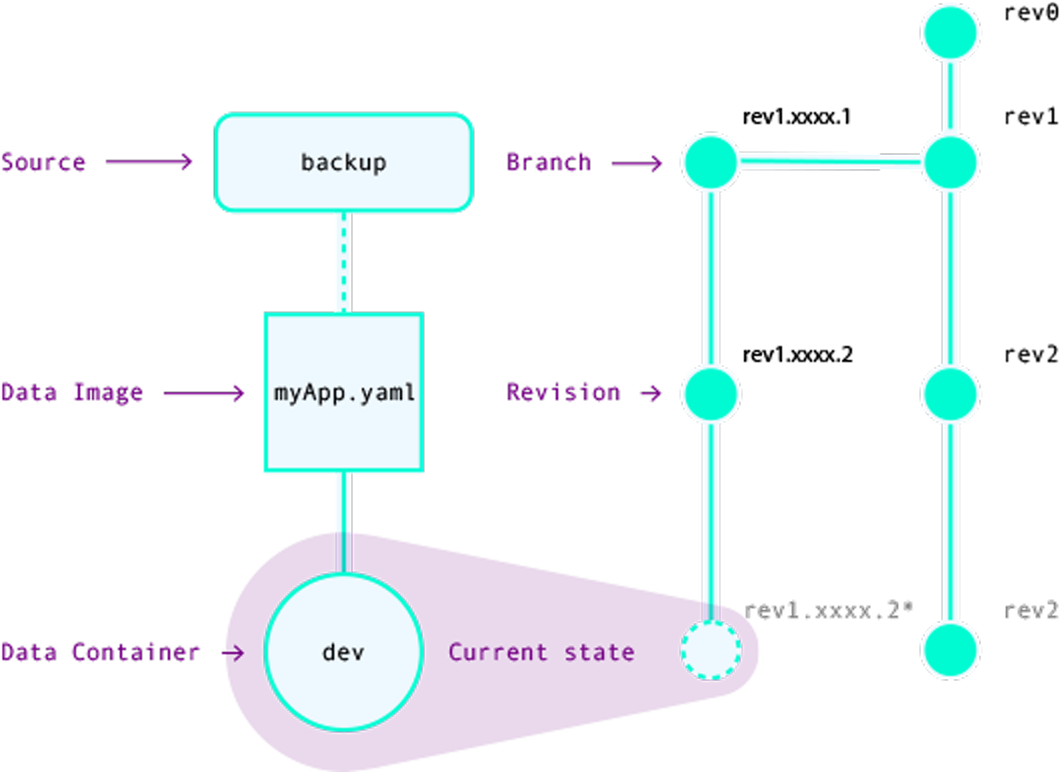
New revision is 'rev.1-ciys.0'

Copy

If we connect to that new container and made more saves, your new branch would now have rev1.xxxx.1, rev1.xxxx.2, and rev1.xxxx.3

| **rev1** | **xxxx** | **3** |
| --- | --- | --- |
| name of the data container | branch identifier | revision number on this branch |

**Summary**



**Review of concepts**[**#**](https://docs.spawn.cc/concepts/summary#review-of-concepts)

You should now be familiar with the core concepts of Spawn:

1. Where the original data and schema comes from (Source)
2. Where this data, schema and instructions are kept/defined (Data Image)
3. How to share a data image with other team members (Team)
4. Differentiating data images with the same name (Tag)
5. What an instantiation of a data image looks like (Data Container)
6. How it's possible to keep multiple states from which to work (Branch)

The next section explains how you can change the source configuration to create different data images.

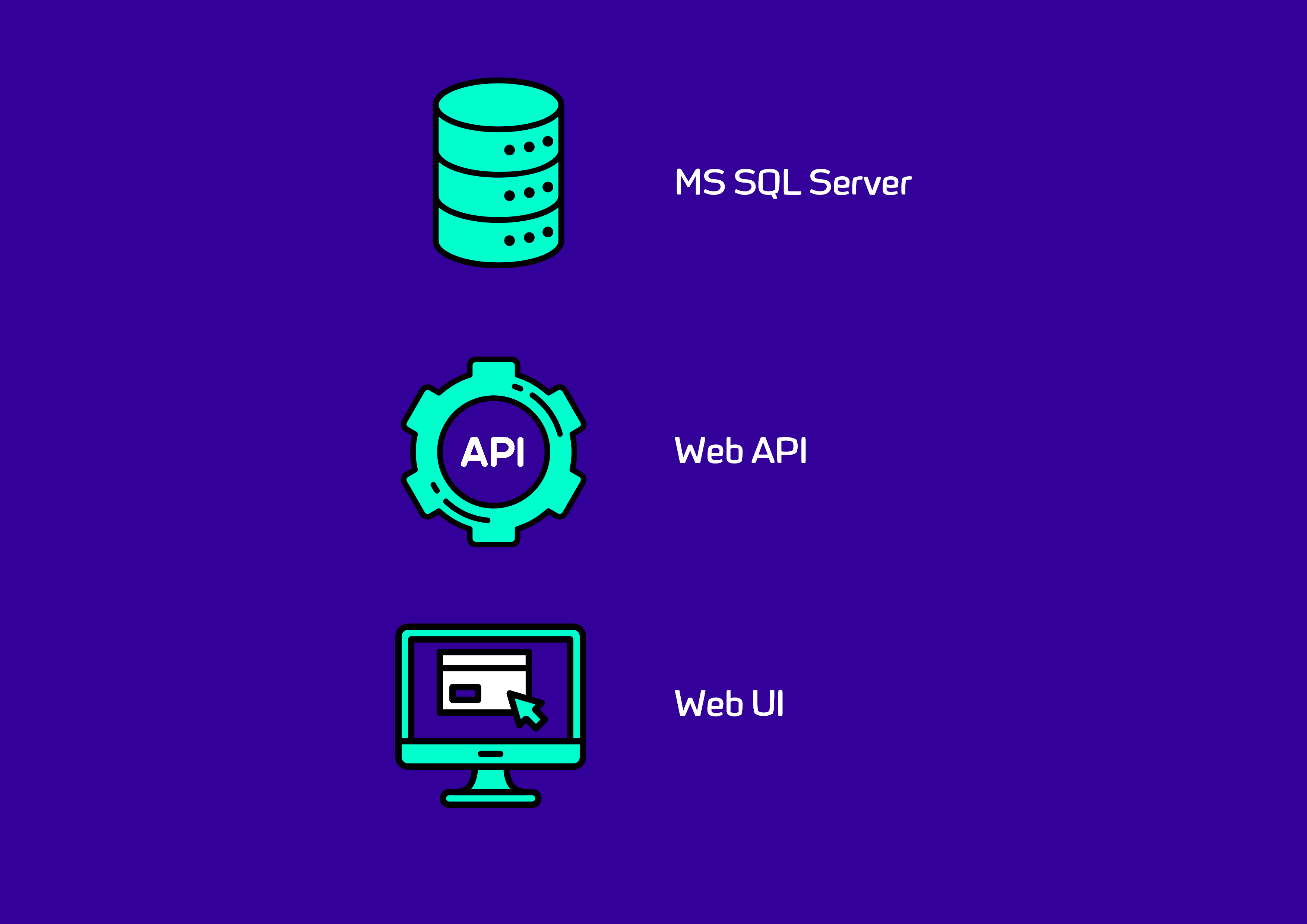
**Spawn for development teams**

**Introduction**[**#**](https://docs.spawn.cc/other/spawn-for-a-team#introduction)

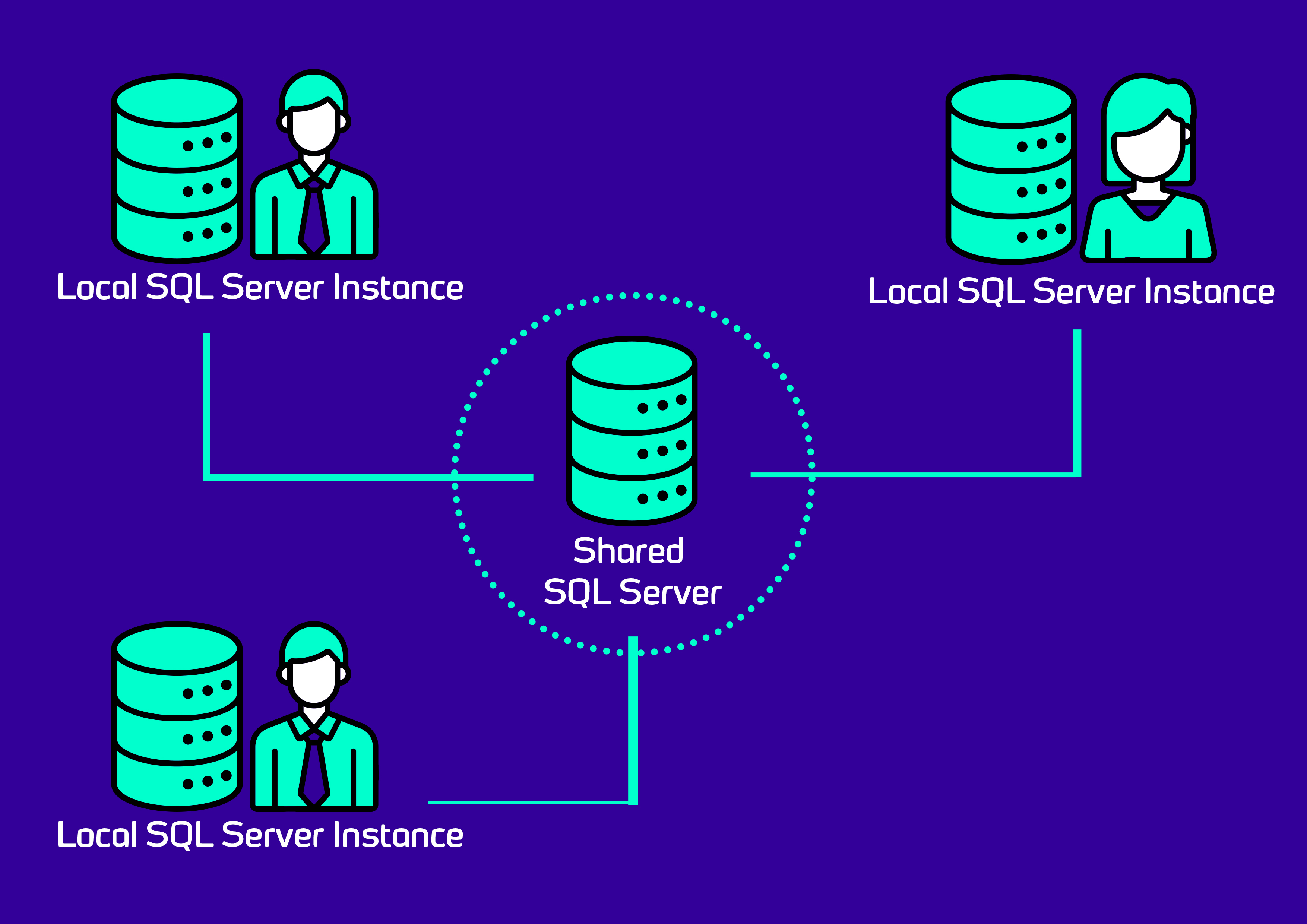
Spawn offers many advantages for teams that might be used to using shared development environments or local database servers running on developers' machines. This article gives a case study of how a team at Redgate made the transition from using local database servers and shared environments to using spawn to create ephemeral, reproducible database servers on demand.

**Current setup**[**#**](https://docs.spawn.cc/other/spawn-for-a-team#current-setup)

The application is a simple web app - there is a backend API server that needs to talk to a SQL Server instance and web frontend:



Each developer runs a local instance of SQL Server on their development machine. These instances contain databases that are typically quite small and contain only 'toy' data; they are not representative of the databases that will be used in production. In addition to these dedicated local instances, there are also some SQL Server instances that are available to the team as a whole, and are shared by all developers on the team. These shared instances typically contain larger databases that are more realistic and representative of the kinds of databases that will be found in production. The situation is summarized in this image:



**Problems solved by spawn**[**#**](https://docs.spawn.cc/other/spawn-for-a-team#problems-solved-by-spawn)

spawn is a good fit to improve the developer experience in such a setup for these reasons:

1. Reproducibility of development environments: with each team member running their own SQL Server instance there is no easy way to share database environments between team members. Consequently, a bug that is reproducible on one developer's machine may not be reproducible on one someone else's. spawn allows all team members to use the same data image, thus ensuring consistency in the database layer across all developers' machines.
2. Cheaper integration tests: The [test pyramid](https://martinfowler.com/articles/practical-test-pyramid.html) is predicated on the assumption that unit tests are the fastest and most reliable tests to run. While this still holds true with spawn and other container technologies, they make integration tests easier, less flaky and no longer require maintenance of external infrastructure. This makes integration testing more viable, increasing confidence in the system as a whole.

**Creating and sharing a development image**[**#**](https://docs.spawn.cc/other/spawn-for-a-team#creating-and-sharing-a-development-image)

The first step to adopting spawn within a development team is to create a [data image](https://spawn.cc/docs/concepts-data-image). While it is possible to use a backup as the source for the image, it is preferable to use scripts to create it so that those scripts can be placed under source control.

With the scripts folder in place, the next step is to use it to create the image. This is done by creating a simple .yaml file describing the image to be created:

sourceType: scripts

name: development-environment

engine: mssql

version: 2017

teams:

- "red-gate:dev-team-one"

scriptsFolders:

- ./scripts

Copy

We can indicate which GitHub teams should have access to the image by specifying the teams field. An image can be shared with multiple GitHub teams; here the image is shared with just one team, red-gate:dev-team-one.

The source scripts folder and this .yaml file should be versioned alongside the application code.

With this .yaml file created, use spawn to generate the data image:

spawnctl create data-image -f development-environment.yaml

Copy

**Creating your data container**[**#**](https://docs.spawn.cc/other/spawn-for-a-team#creating-your-data-container)

Once the image is created, each member of the team can create new data containers from it. To create a new data container from the image:

spawnctl create data-container -i development-environment -n dev-environment-for-joe-bloggs

Copy

Once the command completes you should see a connection string. This is the connection string for your own private data container based on the shared data image.

**Using your data container**[**#**](https://docs.spawn.cc/other/spawn-for-a-team#using-your-data-container)

One of the key advantages of working with spawn hosted instances over instances that you manage yourself is the ability to snapshot and roll back the state of your databases. This ensures that your instance is always in a consistent, known state for your development workflow. To create a snapshot of your data container:

spawnctl save data-container <your container name>

Copy

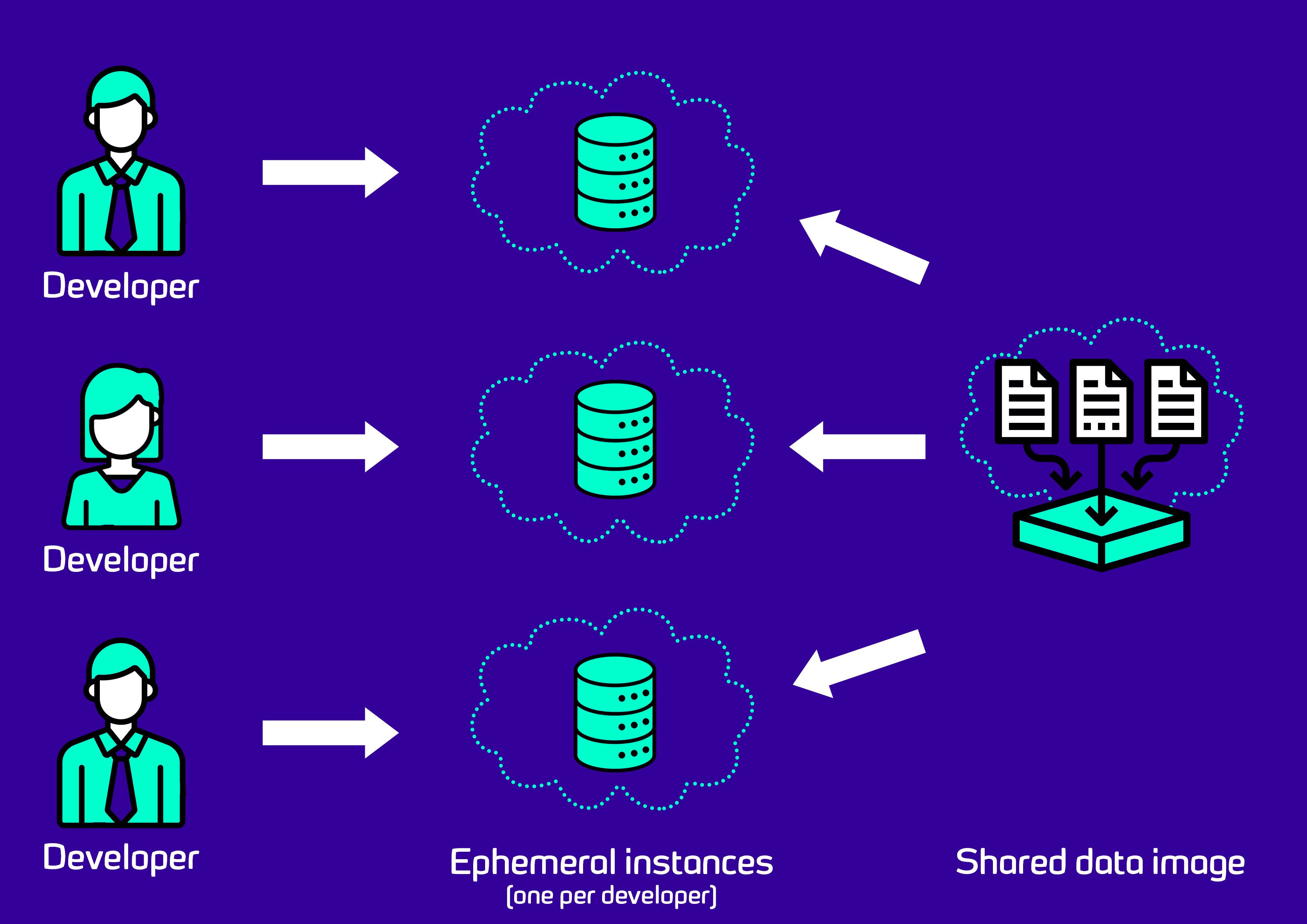
Once this snapshot is created, you can continue to work with your data container as normal but with the added safety net that you can restore your instance to the last known state at any time. To restore your data container to the last saved revision:

spawnctl reset <your container name>

Copy

**Advantages of working in this way**[**#**](https://docs.spawn.cc/other/spawn-for-a-team#advantages-of-working-in-this-way)

Having followed these steps, the database setup for each developer on the team looks like this:



Each developer is now using their own cloud-hosted development database server. These instances can be snapshotted and rolled back independently of each other. Each instance shares the same base data image, ensuring consistency of schema and data between the instances.

By using spawn to host reproducible, ephemeral database instances the developer experience has improved significantly:

* There is no longer a need to run a SQL Server instance locally in order to do application development. This makes setup and onboarding of new team members less onerous.
* Development SQL Server instances are no longer shared between team members; such shared instances can suffer from having multiple team members treading on each other's toes, and could quickly get into inconsistent states when accessed by different versions of the application.
* There is no infrastructure to manage; all database instances are hosted and ephemeral - they can be taken down and recreated in seconds.
* All developers can use the same data image for their development databases, but have separate data containers. Having everyone on the team using the same image helps with reproducibility of issues caused by the data in the database - bugs should be easily reproducible by all team members because they all have the same test data.
* The shared data image only needs to be created once, so it can be a large, realistic example of a customer environment. Recreating such an environment from scratch multiple times, once per developer, can be very time consuming.
* Taking snapshots of data containers and periodically resetting back to them allows development databases to always be in a clean, consistent known state.

Icons made by [Freepik](https://www.flaticon.com/authors/freepik), [Becris](https://www.flaticon.com/authors/becris) & [Prettycons](https://www.flaticon.com/authors/prettycons) from [www.flaticon.com](https://www.flaticon.com/)

# Public data images

Spawn has some public data images available to all users. You can list these by running spawnctl get data-images --public. (They will also be listed if you don't have any data images of your own.) [Data images](https://docs.spawn.cc/how-tos/dataimage) that you create yourself are never made public and are restricted to you and any teams you [share](https://docs.spawn.cc/how-tos/share-dataimage) them with.

## Available public data images[#](https://docs.spawn.cc/other/public-data-images#available-public-data-images)

### Empty databases[#](https://docs.spawn.cc/other/public-data-images#empty-databases)

| **imageName:tag** | **Database** | **Version** |
| --- | --- | --- |
| postgres-empty | PostgreSQL | v11 |
| mysql-empty:v8.0 | MySQL | v8.0 |
| mysql-empty:v5.7 | MySQL | v5.7 |
| mssql-empty:v2017 | SQL Server | v2017 |
| mssql-empty:v2019 | SQL Server | v2019 |
| mariadb-empty:v10.6 | MariaDB | v10.6 |

### Sample databases[#](https://docs.spawn.cc/other/public-data-images#sample-databases)

| **imageName:tag** | **Database** | **Version** | **Notes** |
| --- | --- | --- | --- |
| postgres-pagila:v11 | PostgreSQL | v11 | [Pagila](https://github.com/devrimgunduz/pagila) by Devrim Gündüz |
| mysql-sakila:v5.7 | MySQL | v5.7 | [Sakila](https://dev.mysql.com/doc/sakila/en/) from MySQL |
| mssql-wideworldimporters:v2017 | SQL Server | v2017 | [Wide World Importers (OLTP)](https://docs.microsoft.com/en-us/sql/samples/wide-world-importers-what-is?view=sql-server-ver15) from Microsoft |
| mssql-adventureworks:v2017 | SQL Server | v2017 | [AdventureWorks (OLTP)](https://docs.microsoft.com/en-us/sql/samples/adventureworks-install-configure?view=sql-server-ver15&tabs=ssms) from Microsoft |
| mssql-adventureworks:v2019 | SQL Server | v2019 | [AdventureWorks (OLTP)](https://docs.microsoft.com/en-us/sql/samples/adventureworks-install-configure?view=sql-server-ver15&tabs=ssms) from Microsoft |
| mariadb-nation:v10.6 | MariaDB | v10.6 | [Nation](https://www.mariadbtutorial.com/getting-started/mariadb-sample-database/) from MariaDB |

### Databases for Flyway[#](https://docs.spawn.cc/other/public-data-images#databases-for-flyway)

Spawn is partnered with [Flyway](https://flywaydb.org/), where Flyway uses Spawn for a seamless getting started experience. Public images provided for Flyway are listed below:

| **imageName:tag** | **Database** | **Version** |
| --- | --- | --- |
| postgres-flyway-getting-started | PostgreSQL | v11 |
| postgres-flyway-existing-database | PostgreSQL | v11 |
| mysql-flyway-getting-started | MySQL | v5.7 |
| mssql-flyway-getting-started | SQL Server | v2017 |
| postgres-flyway-getting-started-complete | PostgreSQL | v11 |
| mysql-flyway-getting-started-complete | MySQL | v5.7 |
| mssql-flyway-getting-started-complete | SQL Server | v2017 |

## Requests for other databases[#](https://docs.spawn.cc/other/public-data-images#requests-for-other-databases)

Are there other databases that you'd like to see added to this list? Let us know at [spawn@red-gate.com](mailto:spawn@red-gate.com).

# How to setup Spawn in existing projects

You've probably already got a project set up to use a database in your development environment. It might be a database engine installed and running locally, a shared instance in your network, or a dedicated instance via something like Docker or a VM.

It's easy to replace those instances with a Spawn data container. In this page, you'll learn how.

## Programmatically pulling out connection details[#](https://docs.spawn.cc/existing-projects/existing-projects-intro#programmatically-pulling-out-connection-details)

The key is to get the Spawn data container connection details and supply that to your application. This is easy by using the -o json flag on the spawnctl get data-container command.

* MacOS/Linux
* Windows

$ spawnctl get data-container -o json my-data-container

{"id":54321,"imageId":98765,"name":"my-data-container","revision":"rev.0","status":2,"engine":"PostgreSQL","engineVersion":"11.0","statusMessage":"Running data container 'my-data-container' (54321)","connectionString":"Host=instances.spawn.cc;Port=31895;Username=spawn\_admin;Database=spawn;Password=my-secure-password","host":"instances.spawn.cc","port":31895,"user":"spawn\_admin","password":"my-secure-password","createdAt":1629121793,"expiresAt":0}

Copy

As a human, this isn't much use. Thankfully, [jq](https://stedolan.github.io/jq/) is an excellent way to parse this output and pull out the useful parts as part of a script.

$ spawnctl get data-container my-data-container -o json | jq

{

"id": 54321,

"imageId": 98765,

"name": "my-data-container",

"revision": "rev.0",

"status": 2,

"engine": "PostgreSQL",

"engineVersion": "11.0",

"statusMessage": "Running data container my-data-container (54321)",

"connectionString": "Host=instances.spawn.cc;Port=31895;Username=spawn\_admin;Database=spawn;Password=my-secure-password",

"host": "instances.spawn.cc",

"port": 31895,

"user": "spawn\_admin",

"password": "my-secure-password",

"createdAt": 1629121793,

"expiresAt": 0

}

Copy

Already, we can see that jq has pretty-printed the json for us. But we can even pull out the individual components now:

$ spawnctl get data-container my-data-container -o json | jq -r '.host'

instances.spawn.cc

$ spawnctl get data-container my-data-container -o json | jq -r '.port'

31895

$ spawnctl get data-container my-data-container -o json | jq -r '.user'

spawn\_admin

$ spawnctl get data-container my-data-container -o json | jq -r '.password'

my-secure-password

Copy

As you can see, it's very easy to pull out the individual parts. We can use this to construct environment variables or flags that we can supply to our applications.

dbHost=$(spawnctl get data-container my-data-container -o json | jq -r '.host')

dbPort=$(spawnctl get data-container my-data-container -o json | jq -r '.port')

dbUser=$(spawnctl get data-container my-data-container -o json | jq -r '.user')

dbPassword=$(spawnctl get data-container my-data-container -o json | jq -r '.password')

export DB\_CONNECTION\_STRING="Host=$dbHost;Port=$dbPort;User ID=$dbUser;Password=$dbPassword;"

./my-web-api --dbConnectionString=$DB\_CONNECTION\_STRING

Copy

Depending on your development approach, setting this environment variable once for the session may be enough. On the other hand, it might make sense to create a wrapper script for launching your application that runs these spawnctl commands to construct the connection string on startup. If you've got other task runners that are part of your development process, then running spawnctl commands as part of that process should be straightforward.

##### SEE THIS IN ACTION

To see an example of this working in a repository you can try, check out <https://github.com/red-gate/spawn-demo>

Still not sure how to get Spawn up and running in development? Something missing from this documentation? Let us know by [joining our Slack](https://spawn.cc/slack) or emailing [spawn@red-gate.com](mailto:spawn@red-gate.com) and we'll work with you to get you up and running.

**Service Information**

**Cloud hosting**[**#**](https://docs.spawn.cc/Service%20Information#cloud-hosting)

Spawn is a hosted service currently running entirely within AWS infrastructure. At the moment, we operate Spawn from the London data center. Please [contact us via email](mailto:spawn@red-gate.com) or [ask us directly in our Slack workspace](https://spawn-cc.slack.com/join/shared_invite/zt-ob7zd1uf-X45y8K0ank1LfNZHkSU7Vg#/shared-invite/email) if you'd like us to operate Spawn in multiple regions.

**Security Measures**[**#**](https://docs.spawn.cc/Service%20Information#security-measures)

All spawn resources (data images and data containers) are stored in the Spawn cloud which is a multitenant service. Here’s what we do to protect your data:

* All user data and running instances are isolated from each other through industry-standard approaches:
  + Process and filesystem containerisation of running workloads
  + Network isolation between running workloads
* Connections to the Spawn API use OpenID Connect authentication and JSON Web Tokens to ensure users only see their own data images and containers
  + Therefore, users may only see and connect to their own data images and data containers
  + Access to data or workloads that are not associated with your user is prohibited through the isolation measures described above
* Data containers only accept connections on a randomly-allocated TCP port (though you should not rely on this alone for security)
* Connections to running data containers are enforced through TLS, and non-encrypted connections are not permitted to ensure secure transmission of data between users and their Spawn data containers
* Data containers created from data images are always configured with unique, cryptographically generated random passwords, but you should ensure any additional SQL user accounts you create have strong passwords

If you have any questions, please [contact us via email](mailto:spawn@red-gate.com) or [ask us directly in our Slack workspace](https://spawn-cc.slack.com/join/shared_invite/zt-ob7zd1uf-X45y8K0ank1LfNZHkSU7Vg#/shared-invite/email).