Should You Run Your Database in Docker?

> *Someone mentioned not running databases in containers, but I don't understand why it would be bad…*

What problems can be caused by running PostgreSQL in a container? Should you be worried even if you don't have a lot of traffic? Is it a general thing, and if so, why are so many apps setting up databases in their docker-compose files?

Let's dig into this question, see what factors should influence your decision and whether you could do better or should start worrying.

Before Diving Deeper - Do You Know What Do You Need?

You wouldn't think twice about using (or trying) Docker in a local development environment. After all, what's the worst that can happen, apart from having to go back to your usual setup?

Using Docker becomes a less easy choice when it comes to more critical environments. In the end, it all depends on your needs. Not those of other people. Yours.

Production means different things to different people. Decisions get easier if you know what it means for **you**. What guarantees do you need? What risks are you willing to take? What upsides do you need, and what tradeoffs are you willing to take?

Docker For Development Environment Databases

Should you run databases in Docker? If you're doing so in your development environment, there's nothing to be concerned about.

You don't have important data to lose. In case anything goes wrong, you simply recreate your environment from scratch. (You can get your dev env up in a single command, right?)

Let's look at a few upsides of using containers in this setting:

- There's less **clutter** on your development machine
- You can work on multiple projects side by side, which depend on slightly different database versions
- You can create a development environment on any OS in a reliable fashion
- Everything is "documented" through automation and reproducible

Personally, my favourite way to develop right now, is to have backing services be defined in a docker-compose.yaml file for each project and bind to local ports of the host. This way, I can run a dev server locally, using the complete power of my tools, and have it interact with databases which live in containers.

The only downside could be, that the team is not yet familiar with the toolchain, and needs to invest some research and adjustment time into it.

What About My Simple Live App?

If you’re working on a small project, and are deploying to a single machine, it’s completely okay to run your database in a Docker container.

Be sure to mount a volume to make the data persistent, and have backup processes in place. Try to restore them every once in a while to make sure your backups are any good.

A lot of people are running their small projects using Docker containers, or using docker-compose.yaml files to bring up their complete stack. It’s convenient, and perfectly fine for small projects handling non-crucial data. In the worst case - you restore a backup and are back in the game.

If you have a production system which people depend on, please take this recommendation with a grain of salt, and take a look further below.

Pitfall: Scheduling Across Multiple Machines

Here’s a pretty bad anti-pattern, which can cause you a lot of trouble, even if you’re just working on a small project. You should not run stateful applications in orchestration tools which are built for stateless apps.

Orchestration tools are designed to handle Docker containers running stateless applications. Such applications don’t mind being terminated at any time, any number can run at the same time without communicating with each other and nobody will really notice if a new container will take over on a different machine.

This is obviously **not true at all** for databases! They need their data to be available and being interrupted can cause all kinds of havoc.

If you're using Kubernetes and your database runs in a ReplicaSet or ReplicationController, that's a serious problem. You'd need to make use of StatefulSets and PersistentVolumeClaims. This means your cluster should provide a way to create PersistentVolumes, which can be accessed from all nodes.

But even then, you can't just put any stateful app into a StatefulSet and be done with it. You have to tune it and make sure that the assumptions which are required actually are respected by the application.

Production

Okay, but what about responsible production environments? Is it a good idea to run your important databases in Docker containers? In general, I'd say **don't use Docker for production databases**.

The rule of thumb here is: how can you reduce complexity? The less unknowns are in your stack, the easier it'll be for you to maintain things and react to incidents.

Databases are **critical services**. They take effort to operate, and even more effort to do so reliably. If you really need your data to stick around and be safe no matter what, you **don't want unnecessary risks**.

Running databases with valuable data in Docker has been known to cause trouble in the olden days (2016).

> *Docker WILL crash. Docker WILL destroy everything it touches.*

The time of weird bugs causing data corruption may be past, but one more layer in your tech stack can lead to unnecessary gotchas when operating a crucial service and performing maintenance tasks.

Especially if there are alternative ways to get more stability, spend less effort and reduce your risk. Database services provided by your **cloud provider** are a very good way to go for production (and that means also staging due to prod/staging parity) databases. Use **RDS** if you’re on AWS, **Aurora** on Azure, or an equivalend hosted Database service of your cloud provider. This will simplify a lot of management tasks such as updating minor versions, handling regular backups and even scaling up.

What If Managed Database Services Are Out Of The Question?

So it’s super important data, which is too sensitive to go to a managed service?

Is there a *really* well-maintained internal Kubernetes cluster? Is the service known to be in a reliable state when running on k8s? Are there **actual important business reasons** to use Kubernetes for everything? Maybe. But I’d still be very hesitant.

In all other cases, consider **skipping Docker**, and going with a dedicated machine for each part of the DB cluster instead, so there is as little operational complexity as possible.

In Conclusion

Docker is great for running databases in a development environment! You can even use it for databases of small, non-critical projects which run on a single server. Just make sure to have regular backups (as you should in any case), and you’ll be fine.

For the love of all that’s good - don’t run dockerized databases in container orchestration/scheduling tools without making sure that they can handle stateful apps. Probably they don't do this by default, and you will see a lot of weird issues. Even if they can, you usually need to tune the app to be compatible with the tool in question.

Should you use Docker for production databases? No. Simply because there are better options, like the database services managed by your cloud provider. If you really have to self-host such services in a reliable fashion, you're in for a lot of work and learning. Set up dedicated machines and skip Docker. It's a great tool, but you probably don't need the upsides it can provide in this case.

See Also

- Is Docker-Compose Suited For Production?
- Using Docker At Your Own Pace
- What's Wrong With The Docker :latest Tag?
- Should Built Docker Images Be Used in a Development Environment?
- Should Dockerfiles Be Used in a Production Environment?

> Overriding One Single Value in Your Docker-Compose .env File