


EDB® WHITE PAPER

Deploying Postgres Databases in Containers



GETTING STARTED



CONTENTS

03

Introduction

04

What Is Driving Containers?

07

Why Containerized Databases Now?

12

Who Benefits From Using Containers?

14

Conclusion

Introduction

Today it is safe to say that containers and container orchestration have left the hype phase and are quickly becoming mainstream application development technologies.

The container growth trends are compelling:



- According to a [survey of 600+ IT decision makers](#) conducted by ClearPath Strategies for the Cloud Foundry Foundation, 30% of enterprises have already deployed containers and 42% are in the evaluation stage.
- From a [Sysdig report](#) we learn that the median number of containers per server now stands at 15, up 50% over the previous year.
- Containers are not only being experimented with in dev/test environments. A [Portworx survey](#) of more than 500 IT professionals revealed that in 2018, 83% of the enterprises running containers were using them in production, versus 67% in 2017.
- Diamanti's [Container Adoption Benchmark Survey](#) clearly shows that container use cases are becoming far more diverse, including new cloud-native applications (~55%), lightweight stateless apps (~39%), modernizing legacy apps (~31%), and databases (~30%).
- Last but not least, [451 Research reports](#) that revenues from container technologies in 2018 were \$1.53 billion, which is a little more than three times the revenues in 2015. By 2020, container technology revenues are expected to grow by another 75%, reaching \$2.7 billion.

But what is a container, exactly? A container is a resource-isolated software object that packages all the code, configurations, and dependencies needed to execute and run an application on any Linux or Windows platform, regardless of the environment (on-premises, public or private cloud, or even bare metal). Because multiple containers can share a single operating system (OS), they are significantly more lightweight than virtual machines (VMs). Container orchestration platforms, such as Kubernetes, automatically launch clusters of containers and manage their entire lifecycle, making it easier to deploy containers at scale.

In this white paper, we explore at some depth what is driving the growth of containers, as well as the opportunities and challenges of a specific container use case: databases.

What Is Driving Containers?

In his excellent article, [The Road to Abstraction](#), Stephen O’Grady places containers on a continuum of virtualization that started in the 1960s with COBOL, which buffered program developers from the bits and bytes of machine language. Another significant node on the virtualization continuum was Java Application Servers, which emerged at the turn of the millennium to provide an abstraction layer between Java apps and the underlying OS/hardware infrastructure.

In parallel, we witnessed the rise of VMs, with their ability to abstract and standardize infrastructure deployments on commodity servers. VMs are still widely deployed in data centers around the globe, and will most likely be with us for many years to come. However, there are numerous factors that have driven, and will continue to drive, the growth of container usage—whether alongside or instead of VMs. These drivers include:

- With more and more born-in-the-cloud apps, containers, along with microservices and serverless, are an integral part of the trend towards **cloud-native technology**.
- By abstracting apps completely from the OS and infrastructure layers, containers provide **unprecedented agility and flexibility** to support a DevOps approach of continuous integration/delivery/deployment. In addition, container images launch much faster than VMs, making them more suitable for today’s dynamic runtime environments, in which apps are expected to scale up and down on demand.



- Containers save costs throughout the application lifecycle, from dev/test to production. Some quantified examples of these savings (based on a [Forrester report](#)) are:
 - Reduced hypervisor licensing fees that can lower production environment costs by as much as 50%.
 - Process optimizations that not only streamline dev/test costs by as much as 70%, but also shorten time-to-market so that revenues and other business values are captured more quickly.
 - Thrifty runtime resource consumption as compared to VM deployments, since you don't need to run as many copies of the OS—up to 80% fewer servers.
- As opposed to proprietary VM platforms, containers have been primarily open source. Enterprise development teams using containers benefit from a vibrant open-source community. They also avoid vendor lock-in, with [83% of containers](#) today running on the royalty-free Docker Engine.
- From a developer and operator standpoint, container images run exactly the same in development, staging, and production, regardless of where they are deployed. This consistency eliminates the “it works on my machine” response that you often hear when problems arise due to a discrepancy between environments.



All of the benefits described above have undoubtedly contributed to the enthusiastic adoption of containers. However, we would argue that the single most important container value proposition is **application portability**, which is alluded to in the last bullet point. With increasingly diverse and complex hybrid and multi-cloud environments becoming the norm, it is a huge benefit to be able to operate the same software in the same way across multiple public and private clouds, and even in on-premises virtualized environments. All the app deployment and infrastructure requirements are packaged in the container image, which can be launched anywhere, with full confidence that it will perform as expected.

It is also important to acknowledge that containers could not have gone mainstream without the emergence of container orchestration frameworks that manage and deploy containers across clusters, on demand. The leading framework by far is Kubernetes, the open-source pioneer of container management that originally came out of Google. Kubernetes is also the underlying foundation for other frameworks, such as

OpenShift, GKE, Amazon EKS, and IBM Cloud Private, and is now being incorporated into frameworks that were not originally Kubernetes-based, such as Docker Enterprise and Cloud Foundry with PKS.

With orchestration, containers are automatically reconfigured, scaled, upgraded, updated, and migrated without disrupting applications and services. The orchestration framework monitors the health of the runtime container deployment and ensures high availability and continuity in failover or disaster scenarios. It can also monitor load and automatically scale services up and down in response to changes in demand. The container orchestrator facilitates networking among containers or to external environments. It handles data storage in general, and statefulness in particular, through the management of attached persistent or ephemeral volumes, which can be local or in the cloud.

Why Containerized Databases Now?

A containerized database is an encapsulation of its DBMS server software, with access to a physical database file residing somewhere within the network. Each DBMS is encased in its own container image. Containerizing a database, however, is not quite as straightforward as containerizing an application. Some of the database containerization challenges often cited are:

- Databases typically require high-throughput, low-latency networking. However, Docker containers do not natively provide the level of storage and network resource isolation that is necessary to achieve these requirements.

- The considerable disk space required to store large amounts of data in a containerized database makes it less agile and less relocatable.
- Databases are inherently stateful and durable, while containers are typically stateless and ephemeral. The workarounds put into place to handle persistent data storage and longer-than-usual container lifespans often detract from the key container benefit of reduced runtime resource usage.
- Databases typically have numerous tuning parameters, many of which are dynamic. Building a new immutable container image for every possible database configuration can quickly result in image sprawl. It should be noted, however, that this issue is even more of a challenge in VM deployments, since containers are considerably more lightweight than VMs.



An alternative to containerized databases is Database-as-a-Service (DBaaS), an API-based cloud service model whereby the service provider is responsible for the required database physical infrastructure and server-side DBMS resources, including performance configurations.

The Bottom Line

Ultimately, each project team must make its own decision about whether a containerized database or DBaaS is the best cloud database provisioning fit for the overall application architecture and its development process. The primary drivers for a DBaaS cloud deployment are typically productivity and simplified administration, while for a containerized database, they are usually portability and automation. In any case, when choosing the optimal pathway to a database in the cloud, the following containerized database characteristics should be taken into account:

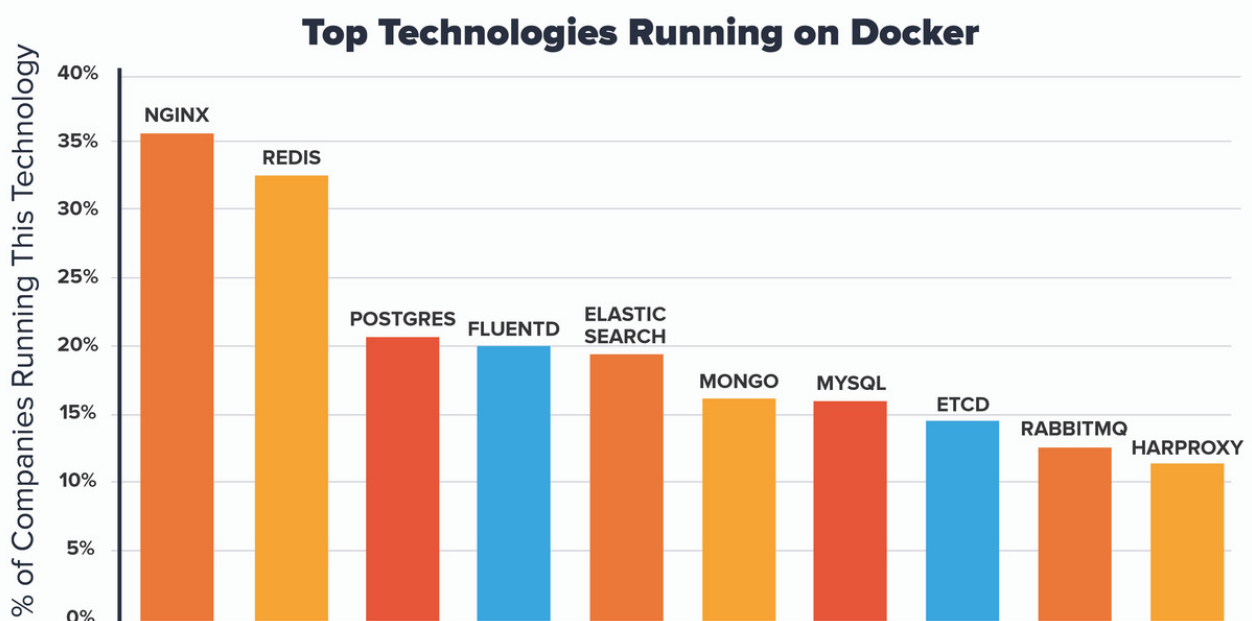
- Smaller containerized databases are well-suited to the microservices architecture that is now often favored over large monolithic applications. In contrast to a DBaaS database deployment, where the database is a central resource for multiple applications, a containerized database becomes, in essence, a component of the specific application that it serves.
- When the automated/scripted deployment of containerized databases is used in conjunction with an orchestration framework, the result works together with traditional clustering to ensure high availability for both stateful and stateless systems.





- The inherent elasticity of containerized databases supports more flexible, and hence, less wasteful upfront database capacity planning and provisioning. With containers, you can approach the database as an on-demand utility.
- Similarly, because containerized databases separate storage from compute, storage performance and capacity can be scaled independently of compute resources.
- A software-defined containerized database provides a crucial missing link in high-velocity DevOps cycles, allowing development and operations teams to collaborate seamlessly.

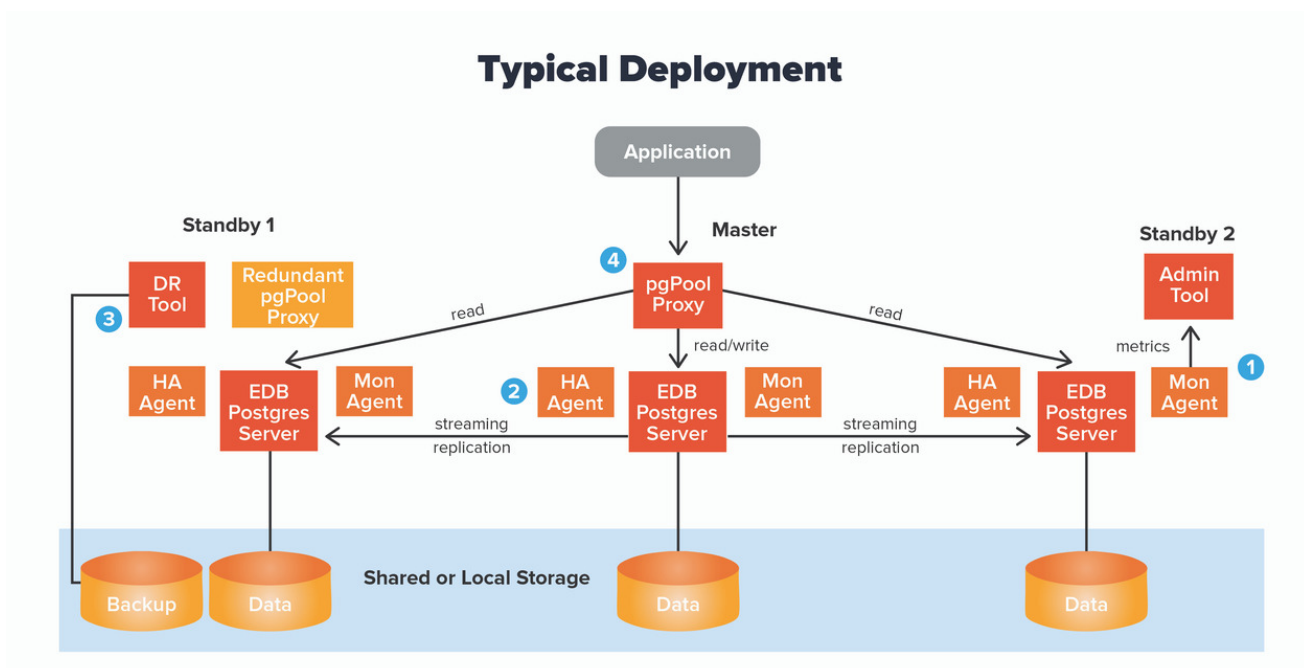
One indication that containerized databases are trending is that Postgres, a well-known open-source relational database, is currently the [third most popular technology being run on Docker](#).



Source: [Datadog](#)

How EDB Postgres Accelerates Containerization

Before we start laying out the architecture of a containerized Postgres deployment, it is important to understand what a standard Postgres environment configured for high availability looks like as illustrated below:



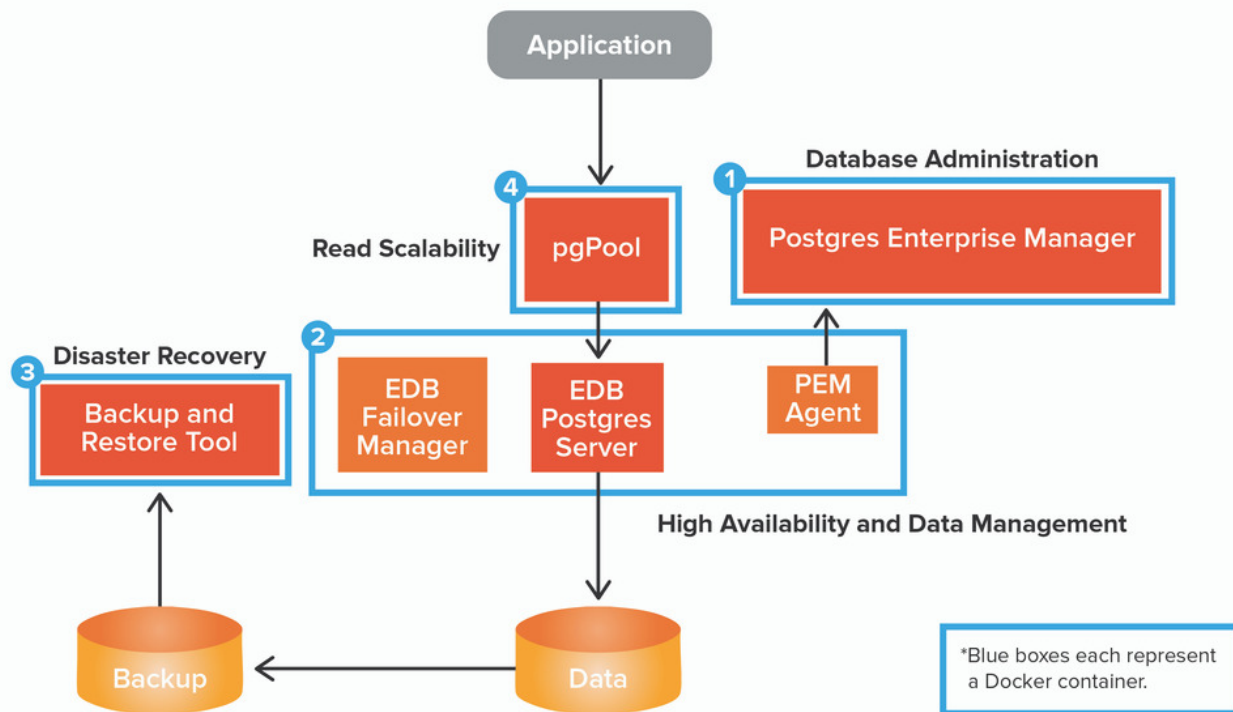
- 1. Monitoring and Administration:** Each Postgres server is monitored by an agent that reports its tracked data to an administration tool.
- 2. High Availability:** EDB Postgres streams incremental changes from the master to any number of read-only standby replicas. A high-availability agent watches for failures and automatically triggers failover protocols according to the nature of the detected failure.
- 3. Disaster Recovery:** EDB Postgres maintains a backup data store and automatically orchestrates enterprise-grade disaster recovery procedures as required.
- 4. Scalability:** EDB is one of the major contributors to the pgPool proxy, which helps applications scale by load balancing read transactions to the replicas while directing write requests to the master. There is also a redundant pgPool proxy to which the application is automatically connected by the high-availability agent in a failover situation.

In order to create a resilient Postgres architecture that can operate at scale, the following four capabilities must be included:

- Failover management for high availability
- Database monitoring and administration management
- Backup for disaster recovery
- Query routing and load balancing for scalability

EDB Postgres is delivered as four containers, all of which are managed by Kubernetes:

Recommended Containerized Deployment



1. Database Administration: Postgres Enterprise Manager® (PEM), packaged in this container, monitors the database, collecting performance and status data that's displayed within dashboards and analyzed for alert conditions. These alerts can be relayed to operators or to other enterprise-level management systems.

2. High Availability and Data Management: This container includes EDB Postgres Advanced Server along with EDB Postgres Failover Manager (EFM). Users can have Kubernetes spin up several replicas of a master container so if the master fails, one of the replicas can take over automatically.

3. Disaster Recovery: This container includes the EDB Postgres Backup and Recovery Tool (BART), which can back up databases in multiple different containers. This enables BART to oversee multiple deployments.

4. Read Scalability: This container delivers query routing and connection pooling using pgPool. The benefit of this container is scaling read activity where pgPool acts as a load balancer routing queries to replica databases. It sits in front of the other containers and can be scaled independently of the database container.

Who Benefits From Using Containers?

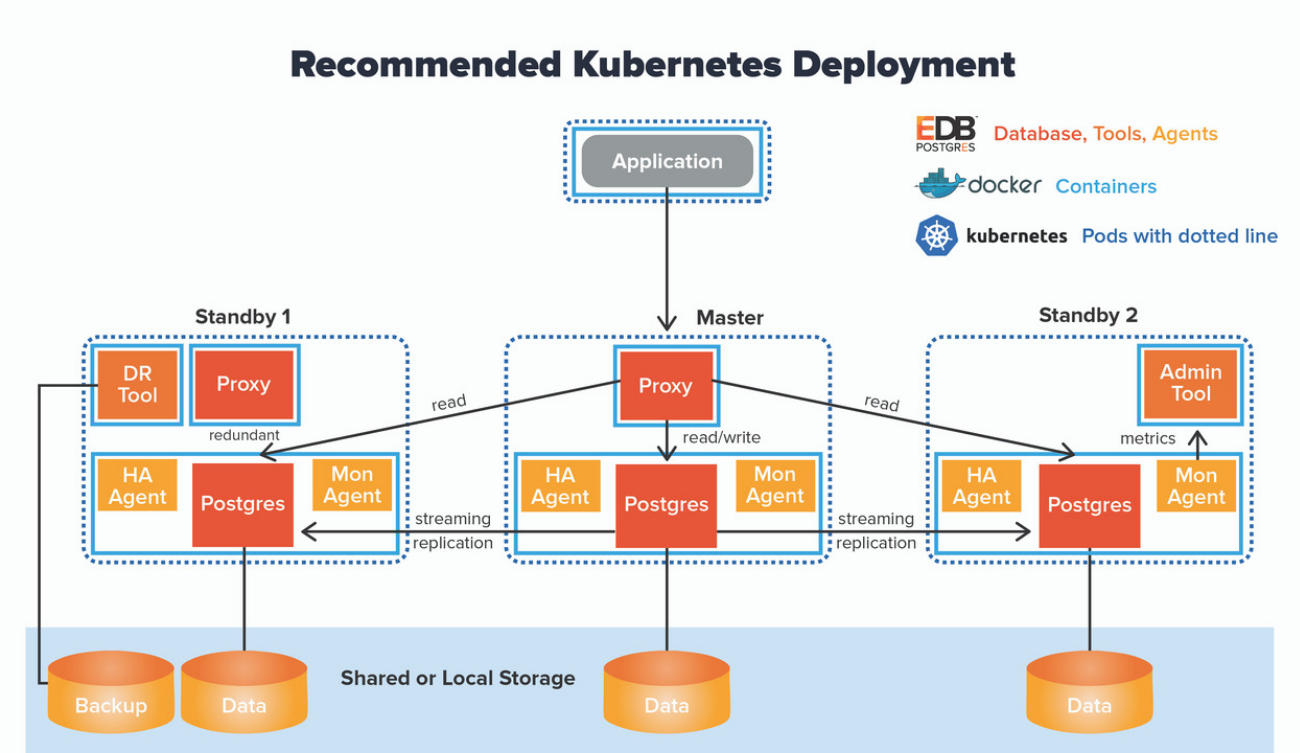
Anyone who wants to modernize their infrastructure.

EDB Postgres is platform agnostic and can be deployed on any Kubernetes platform such as Red Hat OpenShift, Google Kubernetes Engine, Pivotal Cloud Foundry, Docker, IBM Cloud, or Amazon Web Services (AWS).

The following schematic shows a typical configuration of EDB Postgres in a production environment, with a master and two replica pods for high availability and

failover. Replica pods are a way to manage a deployment so that redundant database servers can be on separate physical hardware.

Management and monitoring is done through PEM, with shared storage being the preferable data storage practice in production for the database itself, as well as for backup and recovery.



EDB Postgres makes it possible for EDB customers to get immediate out-of-the-box benefits from containerized Postgres databases. Typically, they start with simple applications and move to more complex systems over time. EDB works closely with its customers, sharing with them best practices for effectively leveraging containers in next-generation, microservice-based applications.

Through its [Architectural Roadmap and Solution Blueprint Service](#), EDB solution architects help their customers design a customized container roadmap for getting from where they are today (struggling with large and cumbersome databases), to a new world of small nimble deployments that are fully aligned with their strategic business objectives. Leveraging a reference library of proven blueprints, full-stack tool sets, and API integration designs, EDB customers can accelerate their digital transformation initiatives, model a robust open-source-based data architecture, and deploy rapidly across complex environments.



Conclusion

In summary, containers and container orchestration have matured to the point that they are now positioned at the very core of cloud-native initiatives. Like VMs, containers abstract the application layer from the compute/storage/network infrastructure layer. Going one step further, however, containers also abstract the operating system layer. The result is a lightweight, resource-optimized, self-contained application that performs consistently across a wide range of diverse platforms. Enterprises around the globe are using containers for their dev/test and production workloads to rapidly develop and deploy born-in-the-cloud products and services that scale effortlessly.

In addition to containerized applications, containerized databases have emerged as part of the paradigm shift from large monolithic applications to applications composed of microservices. Rather than a large centralized database that serves multiple applications, containerized databases have become an on-demand utility that is an integral part of the application itself. That being said,

containerized databases have raised a unique set of challenges in terms of high data availability, backup and recovery, and other critical database performance and compliance requirements.

EDB Postgres allows enterprises to benefit from containerized databases without forfeiting critical database administration and monitoring requirements. Using native Postgres Docker containers orchestrated with Kubernetes, EDB Postgres ensures high data availability, as well as seamless failover, scalability and load balancing, automated monitoring and tuning, and robust backup and recovery.



Please visit the EDB website to learn more and try EDB Postgres on Kubernetes—a containerized database management system orchestrated by Kubernetes.

 www.enterprisedb.com/containers



EnterpriseDB | www.enterprisedb.com

EnterpriseDB, EDB and EDB Postgres are trademarks of EnterpriseDB Corporation.
Other names may be trademarks of their respective owners. Copyright© 2019. All rights reserved. 20190405

EDB[™]
POSTGRES