

Containerization with dotCMS:

Everything You Need to Know



	Key Takeaways	04
	An Introduction to Containers	06
•	How Do Containers Work?	08
	Containers: The Perfect Home for Your Microservices	09
•	What are Microservices?	09
	How Containers Help With Microservice Management	10
•	01. Reduced Costs, Enhanced Processing Power	10
•	02. Freedom for Developers	10
•	03. More Flexibility, More Consistency	11
•	Docker Images and Reference Implementations from dotCMS	12
	Why Docker and Docker Swarm?	15
	Container Orchestration: Running dotCMS in Kubernetes	18
	Beyond Theory: How dotCMS Customers Can Leverage Containers	19
	The Future is Contained	20
	About dotCMS	21
	References	23



As big name brands like

Google, Uber, Netflix and Amazon move further away from monolithic architectures and embrace microservices instead, the discussion around containers is heating up.

However, the growth of the containerization market isn't restricted to the big names alone. In fact, the application container market is set to grow from \$762 million in 2016 to \$2.7 billion in 2020¹ – a sharp rise that's set to be facilitated by large and medium-sized companies alike.

dotCMS has invested significant developer time and resources into scaling dotCMS with containers via Docker, giving dotCMS customers the opportunity to build with a platform that they can independently scale and different dimensions of their ecosystem depending on the load they need to support.



Growth of the Containerization Market





Key Takeaways

dotCMS is a hybrid CMS that helps enterprises make content the centerpiece of their brand. **With dotCMS, organizations can build, manage, and scale all of their content in one centralized system.** We provide the delivery, integration, and APIs needed to meet your needs in an ever-changing digital world.

Many months prior to the release of dotCMS 5.0, the dotCMS team were running and testing dotCMS in Docker in order to keep dotCMS in line with the industry's latest pivot; to containerize everything. Now that dotCMS 5.0 is here — and now that we've moved several of our own properties to Docker-ized installations — **we're ready to guide dotCMS customers towards containerization.**

Key Takeaways

Here are the key takeaways from this whitepaper.

01. How does dotCMS Fit in a Containerized Application Architecture?

dotCMS provides a Docker Image for dotCMS itself, enabling users to run dotCMS in a Docker container. Further Docker Images are provided via dotCMS' Docker repositories², including images for ElasticSearch, Postgres, and HazelCast, empowering dotCMS users with the ability to containerize the different elements of their dotCMS-powered project, **making them easier to scale and manage independently** – while also allowing them to communicate and work in tandem.

02. Does dotCMS Provide any Reference Implementations?

dotCMS provides reference implementations for

- **docker-compose**
- **Docker Swarm**
- **Kubernetes**

03. Are There Any Prerequisites to Containerize dotCMS?

You'll need an instance of dotCMS 5.0 or above, as well as access to Docker. As for container orchestration, our containers are orchestrator agnostic, meaning they can be run in any Docker-compliant orchestrator such as Kubernetes, Docker Swarm, Docker Community Edition, Docker Enterprise Edition, Mesos, OpenShift, Rancher, and docker-compose.

Also, our Docker containers can be run in the custom cloud provider environments such as AWS (EKS, ECS, Fargate), Azure, Google Cloud Platform, and Digital Ocean.

04. Can dotCMS Assist our DevOps Team to Containerize dotCMS?

Absolutely, dotCMS has **extensive documentation** as well as **experienced DevOps engineers** to help make your transition to containers fast, smooth, and beneficial.



An Introduction to Containers

Available for both Linux-based and Microsoft-based applications, **a container virtualizes the operating system instead of hardware**, making a container a more portable and efficient alternative to virtual machines (VMs) in many cases. Containers also afford developers the ability to scale individual containers up or down in isolation, depending on their needs.

Here's how Docker defines a container:

"A container is a standard unit of software that packages up code and all its dependencies so the application runs quickly and reliably from one computing environment to another. A Docker container image is a lightweight, standalone, executable package of software that includes everything needed to run an application: code, runtime, system tools, system libraries and settings.

Container images become containers at runtime and in the case of Docker containers - images become containers when they run on Docker Engine. Available for both Linux and Windows-based applications, containerized software will always run the same, regardless of the infrastructure. Containers isolate software from its environment and ensure that it works uniformly despite differences for instance between development and staging³."

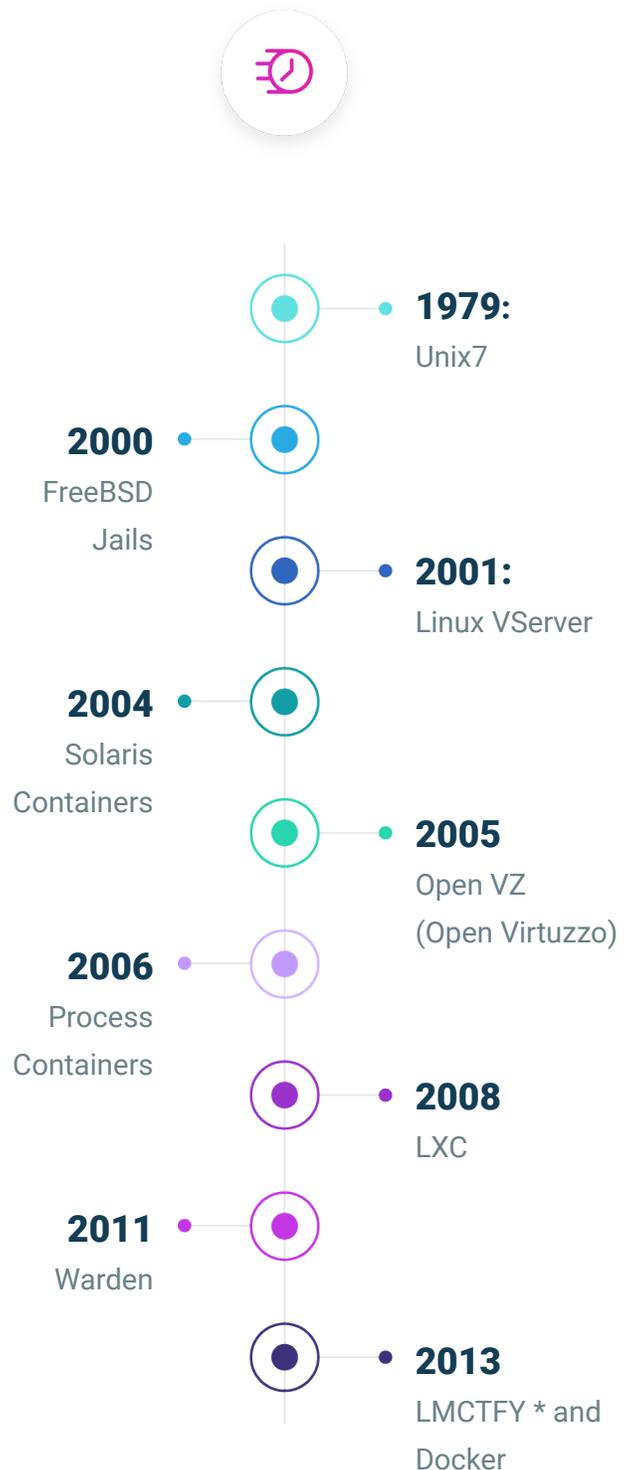
While virtual machines (VM) virtualize a machine, a container virtualizes an entire operating system so that multiple workloads can run on a single OS instance. **With VMs, the hardware is being virtualized to run multiple OS instances** – which, in some instances, **slows applications down, slows development down, and gradually increases the total cost of ownership as the ecosystem scales**. While VMs still have a place in the modern IT ecosystem, containers are becoming increasingly sought after, as they leverage just one OS for multiple applications, increasing speed and portability while also lowering costs.

Like virtual machines, containers provide an environment for microservices to be deployed, managed, and scaled independently – but in a more streamlined fashion as mentioned above.

As you'll discover in this white paper, containers aren't a new technology, they're a tried-and-tested solution that has stood the test of time, and will continue to do so as we enter a new era of microservice-based applications.

In fact, the first recorded instance of containerization occurred in 1979 with the release of UnixV7⁴. It brought with it the `chroot`⁵ system call, which allowed for modification of the root directory of a process and its children to a new location in the filesystem. In other words, enabled developers to segregate and isolate file access for each process.

The next major step towards modern containerization wasn't taken until the turn of the 21st century, but we have come a long way since then, as [this timeline illustrates](#):



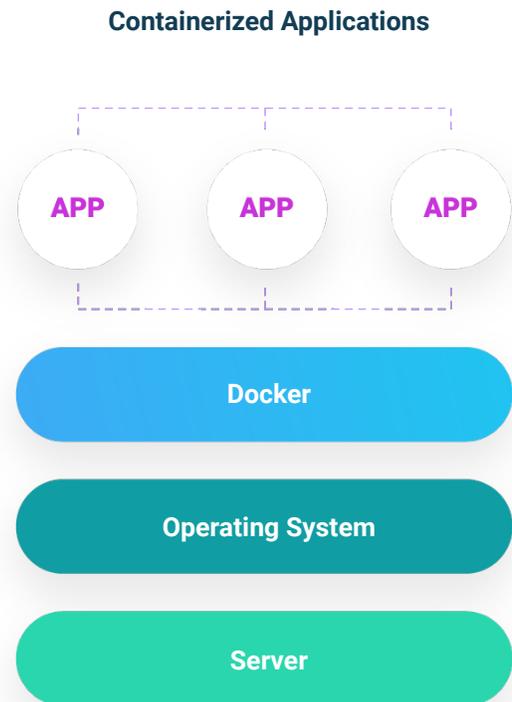
* LMCTFY: Let Me Contain That For You

How Do Containers Work?

Containers make it easy to package applications – including microservices – **for faster and easier development, deployment, and maintenance**. By isolating services, you remove the conflicts that can arise in monolithic architecture.

Containers, unlike VMs which provide hardware virtualization, **provide operating-system-level virtualization**. Even though containers are similar to VMs, since they have its own private space for processing, executing commands, mounting file systems, and having its own private network interface and IP address; **the significant difference between containers and VMs is that containers share the host's operating system with other containers**. Thus, making them more lightweight.

Multiple containers can run on a single OS, and since the OS is shared across all the containers, the components that need to be developed from scratch are the binaries and libraries – which can easily be added via Docker images (we'll explain Docker images, and how dotCMS provides them, later in this whitepaper).



Graphic Above: These containers sit on top of a Docker engine, which in turn, sits on top of the host operating system. The Docker engine utilizes a Linux or Windows Kernel which allows developers to easily create containers on top of the operating system.



Containers: The Perfect Home for Your Microservices

Containers serve as isolated digital capsules for microservices, making them easier to maintain, update, and scale independently.

A recent microservices survey⁶, which compiled the views of over 2,100 developers and IT specialists, found that larger enterprises see microservices as a path to modernization, with 36 percent of microservices efforts being related to modernizing legacy applications.



What are Microservices?

A microservice can be defined as an **isolated application that works in tandem with other services**. A microservice architecture is formed of multiple, isolated, yet communicating services – a contrast to monolithic architecture, where all the applications operate from a single, rigid stack.

In short, monolithic architecture becomes increasingly difficult to manage and maintain as the digital ecosystem scales. For instance, developers aren't able to update or modify a single component of the stack without system-wide implications. This often leads to downtime, friction between developers and engineers, testing issues, and ultimately a slower time-to-market coupled with a patchy customer experience post-launch.

A **microservice architecture addresses these issues by segmenting components into individual services that can be deployed, scaled, and modified in isolation** – and thus, without interfering with the other individual components. A key advantage of microservices is that they can be scaled individually based on its own resource requirements.



How Containers Help With Microservice Management

01. Reduced Costs, Enhanced Processing Power

In contrast to using virtual machines, with containerization **you can leverage a single operating system instance that supports multiple containers**, each running within its own, separate execution environment.

With multiple containers running on a solitary OS, you reduce your expenditure while simultaneously freeing up more processing power for your applications. Even if we consider efficiency alone, containers easily trump virtual machines when it comes to housing microservices.

02. Freedom for Developers

Because microservices are isolated inside their containers, **developers can use different tools and languages for different services**, increasing freedom and preventing technology lock-in. Yet, standardized packaging simplifies testing and development. This is achieved **by creating a Dockerfile** that defines the language, framework, and library for that particular microservice.

For example, you can create a Dockerfile to create a Docker image for a microservice that uses Java on the Spring framework. The container created for this Docker Image can easily be placed on a host next to another container created from a Docker image using Ruby and the Sinatra framework. Since the container execution environment isolates each container running on the host, there is no risk of the language, library or framework colliding with one another.

03. More Flexibility, More Consistency

Containers, and the microservices within, can be deployed in the cloud on on-premise, and can be orchestrated individually via container orchestration tools such as Docker Swarm and Kubernetes – both of which play nicely with dotCMS.

And yet, developers will find it easier to keep a level of consistency throughout their ecosystem.

Because containers organize microservices in the same containerized environment, collaboration between developers, testers, and administrators becomes easier, more seamless, and therefore more consistent across the delivery chain. In other words, containers help you build a healthy DevOps environment.

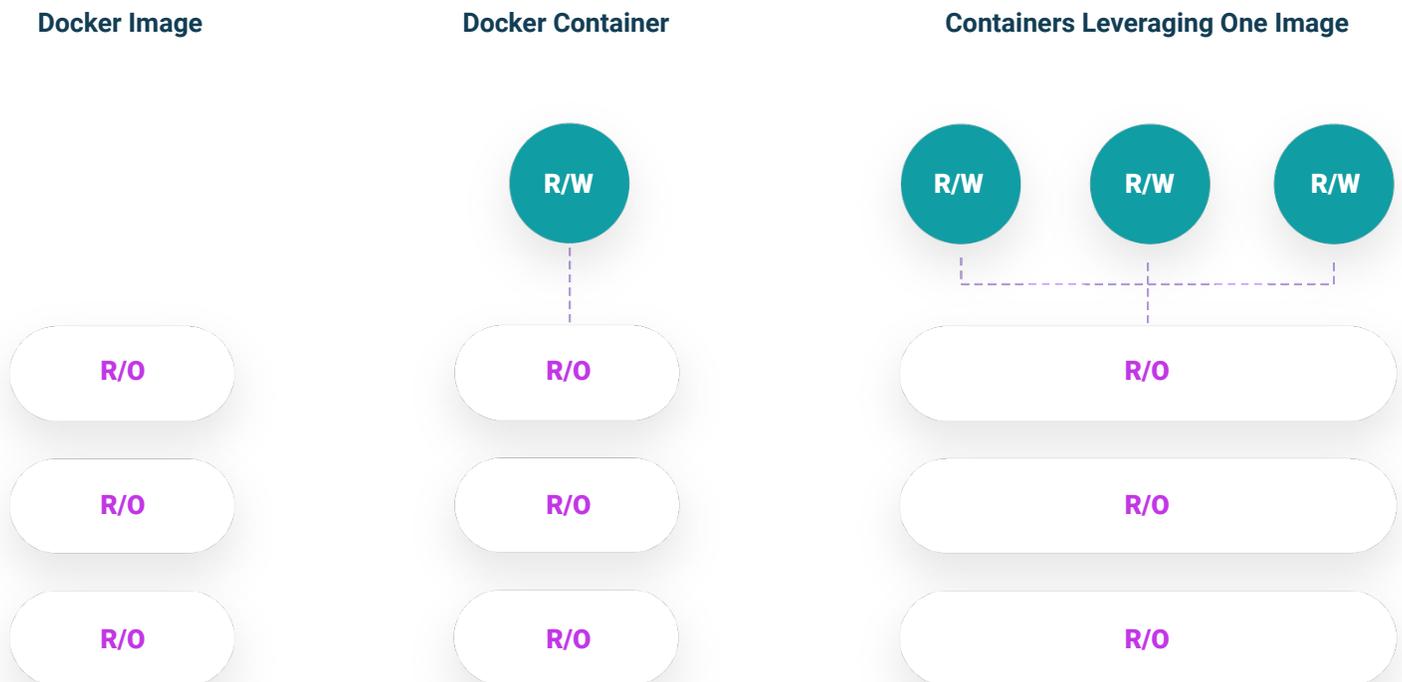


Docker Images and Reference Implementations from dotCMS

You could look at Docker images as the building blocks of your container. Made up of multiple, read-only layers, a Docker image is a static and executable version of an application or service.

An image becomes a container when a readable/writable layer is added on top of static layers. Those underlying static layers can be reused in different containers across your ecosystem. The base image contains all of the dependencies needed to execute code in a container.

The “Docker run” command takes the Docker image as a template and produces a container from it. Docker engine takes an image, adds the top writable layer, and then initializes various settings such as network ports, resource limits, and the container name.



R/O = Read Only Image Layers

R/W = Read and Write Layers

Currently, there are six different docker images made available by dotCMS:

01. dotCMS:

This is the default dotCMS image. It can be used by itself as a standalone demo which uses H2 or with a plugin it can be configured to use a production ready database. When used by itself without a plugin, it is essentially the same as pulling down the tar/zip file of the release, uncompressing release, and running it. It will run using H2 as its database and with embedded ElasticSearch and Caching.

02. Haproxy:

This is included for an example of a load balancer properly configured to work with dotCMS. If your infrastructure has a load balancer, you should consider using it for production use rather than this image. This image is useful for demo, development, and testing purposes.

03. Hazelcast:

This image provides Hazelcast 3.9.2 along with our configuration and discovery scripts that allows dotCMS to dynamically discover these nodes and to enable dynamic scaling of the Hazelcast caching layer independent of the number of dotCMS nodes.

04. Hazelcast Management Center

An image for running Hazelcast's Management Center in your containerized environment. Please note that if you scale your Hazelcast nodes to more than two, you will need to secure a license from Hazelcast for this tool to work properly.

05. ElasticSearch

This image provides ElasticSearch 6.1.3 along with our configuration and discovery scripts that allows dotCMS to dynamically discover these nodes and enable dynamic scaling of ElasticSearch layer independently of the number of dotCMS nodes.

06. Postgres

A very popular open source database engine. Even if you chose not to run your production database in a container, this image can be helpful for development and testing purposes.

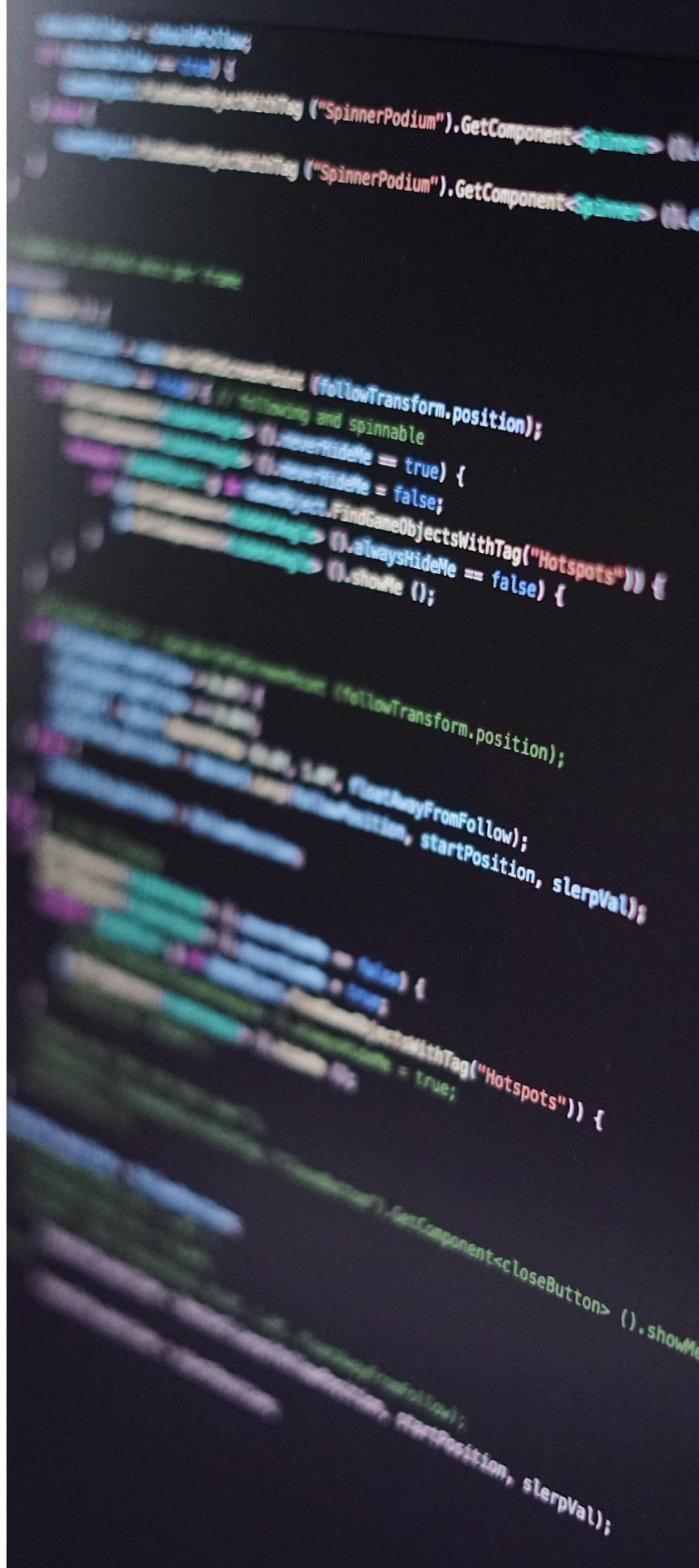
As for reference implementations, these are example configurations for container orchestrators that show how the containers interact with one another. [dotCMS provides reference implementations for docker-compose, Docker Swarm, and Kubernetes.](#)

06

Docker Images

Available by dotCMS

-  dotcms
-  HAProxy
-  hazelcast
-  hazelcast
Management Center
-  elasticsearch
-  PostgreSQL





Why Docker and Docker Swarm?

dotCMS is a best-of-breed solution. Our history is laden with examples of dotCMS customers integrating with leading technologies. This is down to our mission to give our customers access to the best third-party technologies via APIs and strategic partnerships so that they can serve every dimension of their digital presence with cutting-edge and industry leading technology.

In other words, dotCMS doesn't intend to be a "Jack of all trades", **we intend to be the best at content management – while allowing our customers to access and leverage the best solutions for other industries.** With this same mission in mind, we chose to provide our reference implementation using Docker Swarm, and we chose to provide containers

that can be run in any Linux Docker-compatible orchestrator.

There are several reasons for this choice, but most importantly of all, **the YAML file for Swarm is easy to understand and is an effective way of communicating the needed configuration for services, volumes, and networks.** The YAML file also works with docker-compose which makes it easy for simple POCs and for developers to use the entire stack locally on their development machines.

All our reference implementations can be ported into any orchestration environment other than Docker Swarm, as demonstrated by dotCMS' Senior DevOps Engineer Brent Griffin⁷.

Here are some additional reasons for our selection of Docker and Docker Swarm.

01. Reputation

Launched in 2013, Docker is known to be the leading containerization tool on the market. While Docker didn't invent the practice of containerization, it did package and popularize the idea. For the last five years, Docker has become the darling of the container market.

02. Built for the Microservices Boom

For granular control, lower expenditure, and to facilitate a DevOps environment, the world's largest brands have already transitioned from monolithic architecture to microservices – and the digital world is following in droves. Docker was built with this microservice boom in mind.

03. Portability

Containerization provides you with a way to standardize application environments. The Docker engine provides portability where you can operate the application across several different environments.



04. Fast, Efficient Deployment Cycles

Docker helps to reduce the cycle time between writing code, testing code, and deployment. The Docker ecosystem enables developers to write code locally and share their development stack with their colleagues. Once the code is ready, developers can push both their code and their stack to a testing environment to run the required tests. On successful completion of the testing stage, developers can then push the Docker Image into production and deployment.



05. Ease of Use

Docker has made it easier for developers, system admins, architects, and programmers to utilize containers to **quickly build and test portable applications**. Anyone can create and package an application on their laptop and run it on either public cloud, private cloud or even bare metal.

06. Modularity and Scalability

The Docker ecosystem enables you to break up your application's functionality into individual containers. For instance, you could have a database in one container and a Node.js app in another. The docker ecosystem allows you to link these containers together via API calls. And since these components are running independently, you can update and scale your component more efficiently.



Container Orchestration: Running dotCMS in Kubernetes

dotCMS provides a Kubernetes reference implementation **to make it easy for users to leverage the container orchestration software.**

When it comes to alternative tools, experienced engineers will have no problem running dotCMS in the container orchestration tool of their choice. As dotCMS' Senior DevOps Engineer Brent Griffin demonstrated on the dotCMS blog⁸, our Docker reference implementation can quickly be ported to Kubernetes – even without a Kubernetes-specific reference implementation – without issue. Similar steps can be taken to marry dotCMS Docker images and containers with third-party orchestrators.

Our current plans are to release a 'Docker Compose.

yaml' file along with our Docker images as a reference implementation that shows how each container needs to be configured and how the containers depend on each other.

We believe that the Docker-Compose file succinctly captures all of the relevant information needed for configuring the containers to work in any orchestration environment. Of course, the Docker-Compose file can be used to run directly in Docker Swarm and Docker Compose.

Reference implementation from a Docker-Compose file and ported it over to Kubernetes. While Kubernetes does not native dependency management, one of the reasons for our customized third-party containers is that we have added some custom discovery logic that ensures their dependencies are in place prior to the main process in the container starting up.



Beyond Theory: How dotCMS Customers Can Leverage Containers

Our support for Docker now enables dotCMS customers to easily jump on the containerization bandwagon. Along with [dotCMS' Docker image](#), [a Docker compose file is also on the horizon for dotCMS users](#), which enables the Docker image to work with the container management software of your choice.

In a recent webinar hosted by dotCMS, Brent Griffin, dotCMS' Senior DevOps Engineer, provided a demonstration of how dotCMS can be scaled using Docker⁹. In the demonstration, Griffin broke up dotCMS into seven different components, highlighting how one of the components, the Elastic Search functionality, normally comes under heavy load due to numerous content pull requests.

This results in the overall system running slow, not because of dotCMS getting taxed, but primarily down to the rising number of Elastic Search queries being backed up in a queue.

Griffin showed that, by containerizing the Elastic Search capability, [dotCMS users can compartmentalize that component, integrate it with the other components, and then scale it to allow it to work more efficiently without overloading the overall system.](#)

dotCMS Docker images have been designed as “Orchestrator Agnostic”, so brands can use Kubernetes, Swarm, or any other orchestrator. Additionally, dotCMS supports complete internal testing against both Swarm and Kubernetes based orchestrators. dotCMS uses can also externalize services, like an ElasticSearch layer for example, making them individually scalable.

In short, dotCMS helps its customers leverage containerization to run websites and applications more efficiently, and at a lower cost than ever before — all while delivering a competitive customer experience.



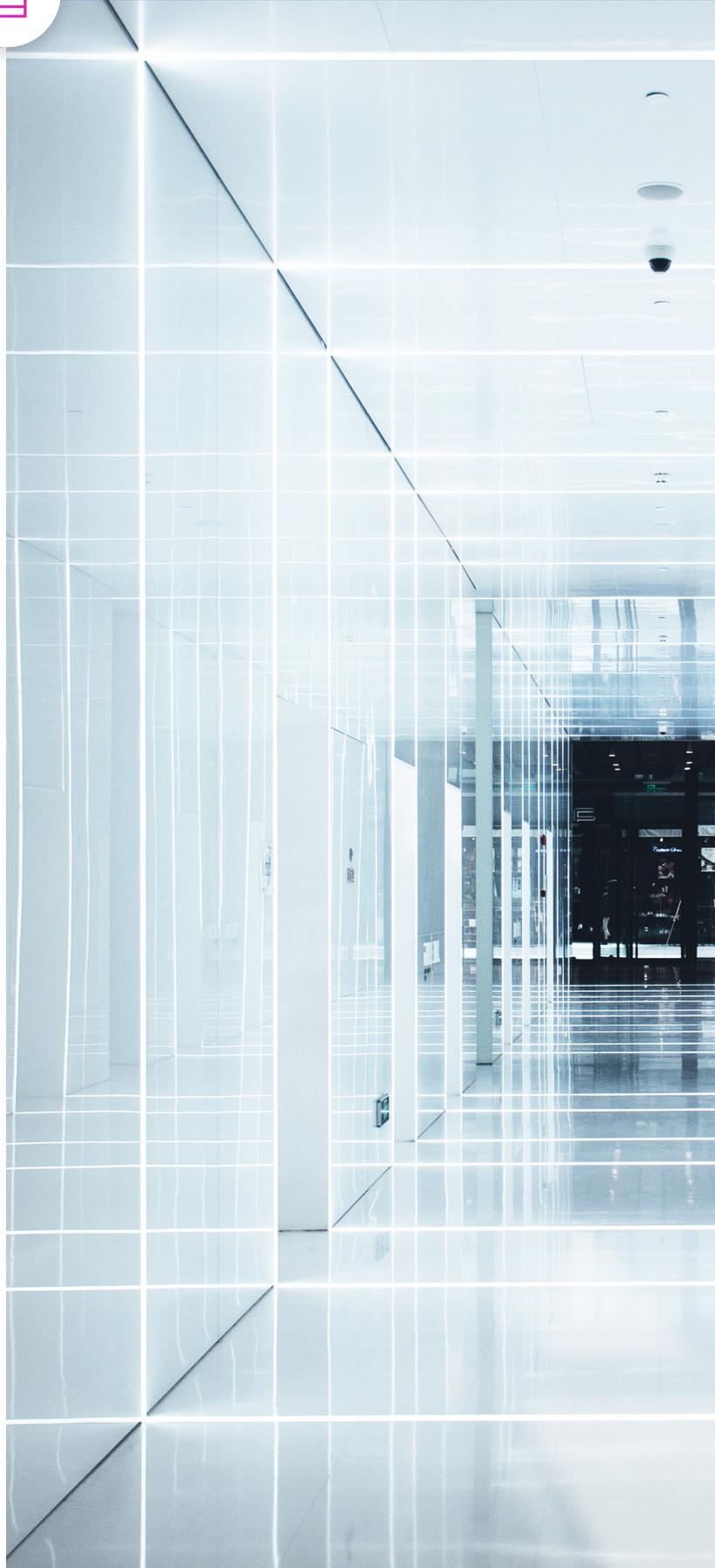
The Future is Contained

As well as facilitating headless content orchestration,

dotCMS has taken steps to ensure that containers and container orchestration can be done seamlessly – and with the markets leading tools.

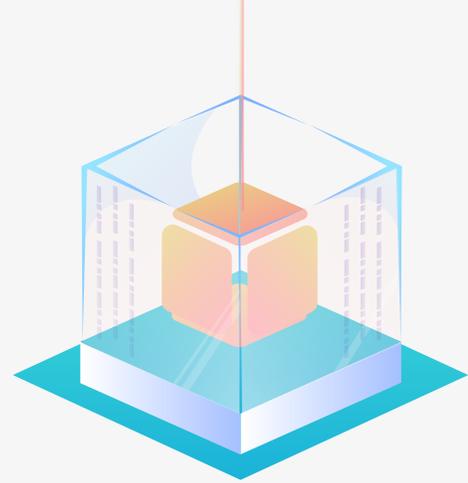
Containers serve as the perfect home for applications and microservices by facilitating DevOps environments, decreasing expenditure, and speeding up the time to market. Thus, you can expect to see more of the world's leading brands leaving virtual machines behind in favor of Docker-powered containers.

For more information on running dotCMS in containers, contact us.





About dotCMS

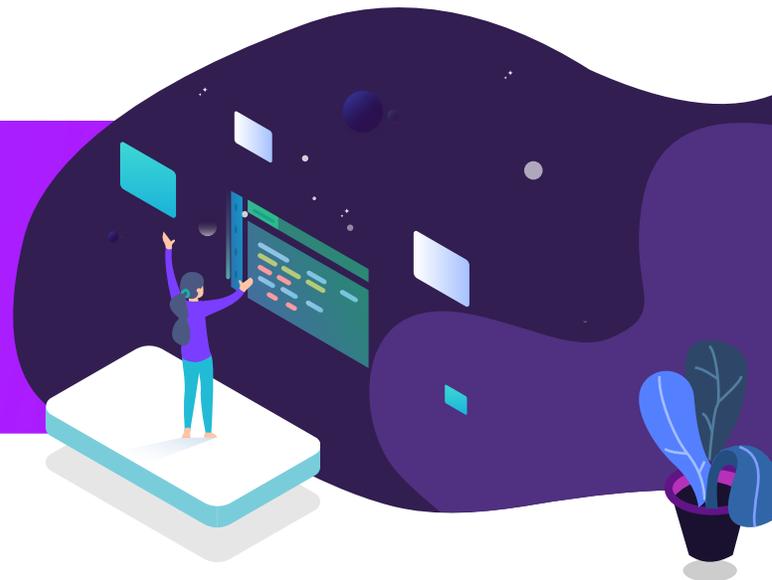


dotCMS is an open-source Java, customer experience orchestration hub for companies that want to drive business outcomes with their websites and other content-driven applications. dotCMS provides the technology to deliver connected and continuous customer experiences that business teams can orchestrate.

Extensible, scalable, and with headless content management capabilities, organizations can rapidly build their Digital Experience Platform and drive innovation while their marketing and business teams drive customer experiences for every touchpoint, in every customer journey, on any device – all from a single system

Founded in 2003, dotCMS is a privately owned U.S. company with offices in Miami (Florida), Boston (Massachusetts), and San Jose (Costa Rica). With a global network of certified implementation partners and an active open-source community, dotCMS has generated more than a half-million downloads and over 10,000 implementations and integration projects in over 70 countries. Notable dotCMS customers include: Telus, Standard & Poors, Hospital Corporation of America, Royal Bank of Canada, DirecTV, Nomura Bank, Thomson Reuters, China Mobile, Aon, DriveTest Ontario, and ICANN.

**SCHEDULE A
dotCMS demo at dotcms.com**





About dotCMS

contact

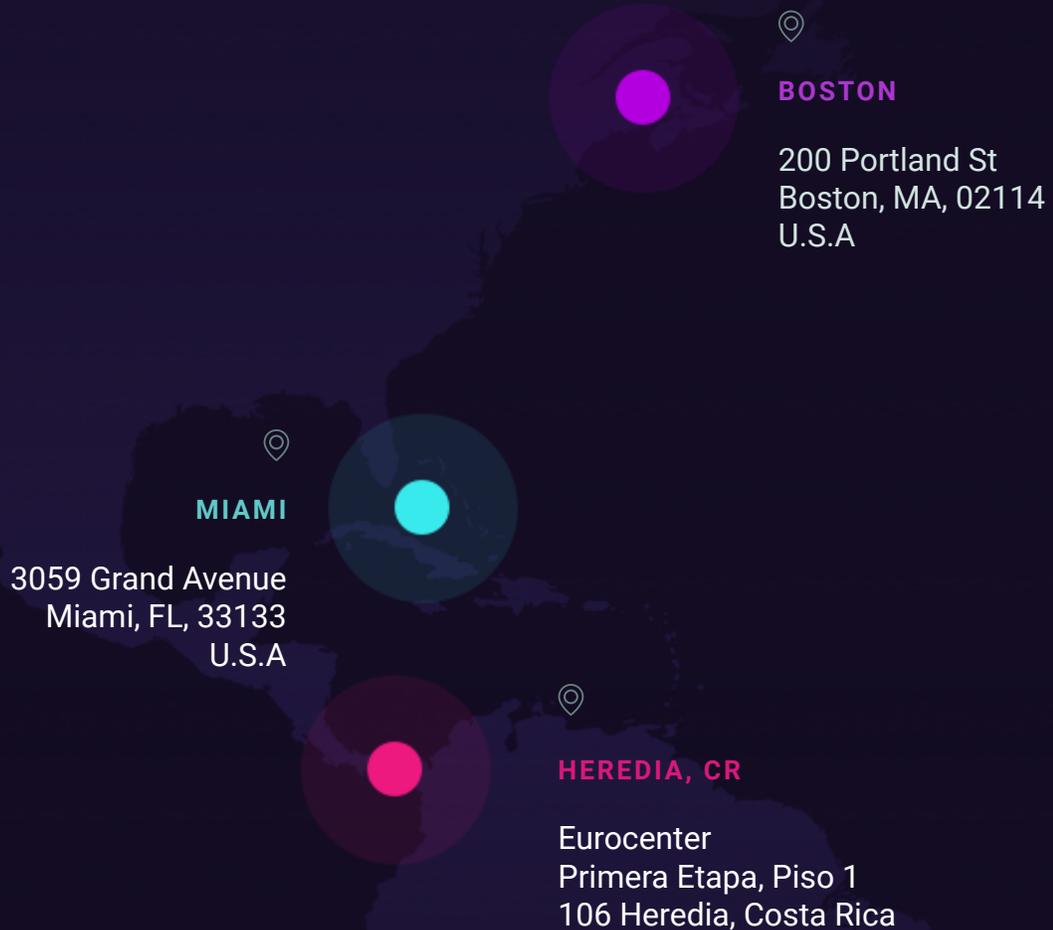
WEB: dotcms.com

PHONE: + 1 - 305 - 900 - 2001

EMAIL: info@dotcms.com



office locations





References

- 1** "7.72 Billion Function-as-a-Service Market 2017 ... - Business Wire." 27 Feb. 2017, <https://www.businesswire.com/news/home/20170227006262/en/7.72-Billion-Function-as-a-Service-Market-2017---Global>. Accessed 8 Aug. 2018.
- 2** dotCMS Docker Repositories <https://hub.docker.com/u/dotcms/>
- 3** Docker: What is a Container? <https://www.docker.com/resources/what-container>
- 4** A Brief History of Containers: <https://blog.aquasec.com/a-brief-history-of-containers-from-1970s-chroot-to-docker-2016>
- 5** Wikipedia: chroot: <https://en.wikipedia.org/wiki/Chroot>
- 6** Microservices Survey, Lightbend. <http://www.zdnet.com/article/survey-of-2000-java-and-scala-developers-reveals-shake-up-in-enterprise-application-architecture/>
- 7** Container Orchestration: Running dotCMS in Kubernetes <https://dotcms.com/blog/post/container-orchestration-running-dotcms-in-kubernetes>
- 8** Container Orchestration: Running dotCMS in Kubernetes <https://dotcms.com/blog/post/container-orchestration-running-dotcms-in-kubernetes>
- 9** Webinar: Why Containerization Makes Sense for Your CMS <https://dotcms.com/company/events/why-containerization-makes-sense-for-your-cms>