# Powering Microservices with Docker

To accelerate digital innovation, many IT organizations are exploring microservices architectures, but to get there they need to address transition complexity, keep a check on platform costs, minimize time to production and avoid service disruptions. The Docker platform is a strong contender that supports all these objectives.

## EXECUTIVE SUMMARY

Enterprise architects seek to elevate IT agility and scalability by breaking down business functional models into bounded contexts with explicit relationships. Microservices architecture[1] helps to achieve these goals by mapping bounded contexts with autonomous units of software (i.e., microservices), each focusing on a single business capability and equipped with well-defined interfaces.

Cohesive teams are organized by the microservices that they build and manage, with independent development velocities, implementation approaches, choice of technology stacks and tools. In support of innovation agility and velocity, IT organizations also often adopt DevOps[2] – a practice that emphasizes a systemic, iterative and collaborative software development approach, with a high degree of automation in continuous integration and continuous delivery[3] (CI/CD).

Apart from enabling an elastic and "always on" IT infrastructure, cloud computing has fueled the evolution of platforms and technologies that simultaneously support microservices and

When used in production, Docker images are deployed and run on a number of containers, usually spread across physical or virtual hosts for scalability, using the popular service instance per container pattern.

DevOps. Open source as well as proprietary platform-as-a-service (PaaS) offerings such as Cloud Foundry, IBM Bluemix, Microsoft Azure, RedHat OpenShift, etc. provide the programming models, delivery automation capabilities and management tools, abstracting the underlying virtualized or cloud infrastructure.

Other open source frameworks and libraries such as Netflix OSS[4] are available as foundational components on which organizations can build their own microservices platform, often relying on an infrastructure as a service (IaaS) layer hosted in the data center or public cloud.

Container technology is a lightweight virtualization capability that has been available in Linux and Unix operating systems for many years (and now in Microsoft Windows, as well). Containers are highly portable across environments and provide improved infrastructure utilization. Docker[5] has extended this technology by adding new capabilities that significantly impact the way software is developed, deployed and consumed.

This white paper explores what makes Docker a great vehicle to convert microservices architecture and DevOps from thoughtful constructs to meaningful realities.

## BUILDING A SOFTWARE DEFINED ENVIRONMENT

Docker started as an open source deployment automation project that introduced an innovative approach for packaging the application software, along with all run-time dependencies, in a container image. A container image is a file system and a set of parameters that fully define the behavior of any instance of the container in execution. Images can be layered such that each base image resolves dependencies of the image in the layer above (e.g., an application framework over platform libraries and run-time over the operating system). Docker images are stored and version-controlled in a registry — either the publicly hosted Docker Hub or the Docker Trusted Registry (DTR) hosted on-premises — to facilitate search and reuse.

Target images are automatically assembled from base container images pulled from the registry when required using a set of command-line instructions. The build automation system uses Dockerfile — a simple and human-readable text script, making it a self-documenting and declarative build automation mechanism. Integrated with a continuous integration tool such as Jenkins, Docker provides the capability for build and deployment automation across various software lifecycle stages. Distribution and provisioning can be automated with Docker-aware provisioning tools such as Vagrant or Ansible.

When used in production, Docker images are deployed and run on a number of containers, usually spread across physical or virtual hosts for scalability, using the popular service instance per container pattern. These containers can be clustered and centrally managed using the Docker Swarm/Universal Control Plane (UCP) or third-party open source alternatives such as Kubernetes or Mesos.

Docker containers provide an efficient and simple form of isolation: containers get their own memory space and network stack and also support resource limits or reservation.

Cloud providers have rapidly adopted Docker to build container-as-a-service (CaaS) offerings, where the container infrastructure, image registry, and automation tool-chain, as well as the orchestration and management capabilities, are delivered as a service. Public-cloud-hosted CaaS examples are Amazon EC2 Container Service (distinct from Docker-aware AWS Elastic Beanstalk), Google Container Engine and Azure Container Service. Docker Datacenter and Rancher are CaaS options that can be deployed on public as well as private cloud.

## AN ECOSYSTEM FOR MICROSERVICES

Docker's advantages significantly multiply when used along with popular open source tools such as Kubernetes and Jenkins or commercial, enterprise-grade build and release management automation products. With the capabilities described above, the Docker ecosystem offers several features that help implement microservices architecture and DevOps practices:

- **Container as the unit of deployment:** The cross-functional team that owns a given microservice continually develops and manages a container image as its deliverable software package. The Dockerfile used for its build automation is common across environments – development, testing, production or any other environment. In effect, developers define or extend (by extending base images defined by operations) and use a production-like environment throughout the development lifecycle. Docker images are portable, from a developer's laptop to the data center to the cloud. Defects arising from configuration differences between the development and production environments are automatically eliminated. By unifying the build and deploy processes for a given microservice across its lifecycle stages, the Docker ecosystem significantly reduces the amount of complexity and room for error.

- **Encapsulation and isolation:** Each container is a self-contained "full stack" environment for the application it hosts. Docker allows a wide range of technology platforms and tools to realize services as long as system resource prerequisites are met and a robust interface governance is observed. This enables a technical as well as functional encapsulation, hiding the implementation details. Docker containers provide an efficient and simple form of isolation: containers get their own memory space and network stack and also support resource limits or reservation. Not only does it provide an independent delivery lifecycle and velocity, it also localizes the impact of any defects or failures.

  Docker employs a very efficient image-packaging strategy that reduces the image-building and publishing time and resists infrastructure sprawl. Encapsulating deployment artifacts into Docker containers, rather than writing topology-aware automation scripts, improves the manageability and stability of the deployment process.

PaaS offerings (especially those that are vendor driven) and the Docker ecosystem are often compared, although they do not necessarily compete and can be beneficial if used together.

- **Dynamic service discovery, orchestration and scaling:** Docker container initialization overhead is very small, which allows near-instantaneous container spin-off and dynamic auto-scaling of infrastructure. Native tools within the Docker ecosystem (Swarm and Compose) as well as third-party tools (Kubernetes, Mesos) enable service discovery, orchestration and load balancing capability.

  In a composite application, services can be explicitly linked to resolve dependencies; however, they can be independently scaled and managed. Rapid innovation and fail-fast strategies such as blue-green deployment[6] or A/B testing[7] can be implemented with the help of automation scripts or commands issued from a single console by slicing the infrastructure in a more efficient and controlled way.

- **Coherent developer tools ecosystem:** Having evolved beyond Linux container technology, Docker is natively available for Windows 10 with Hyper-V and Mac OS X El Capitan 10.11 or newer. It is a first-class citizen in the target environments supported by many popular developer tools (especially open source). For example, Eclipse Neon provides a Dockerfile editor, Docker registry connect and image search/pull/push capability, container management, console access to a shell within a running container, etc. Eclipse Che (a workspace on the cloud) also supports Docker Compose. Microsoft has also released a preview version of Visual Studio tools for Docker. Consequently, enterprise developers moving from traditional programming environments to Docker-based polyglot microservices have a natural transition path over a period of time.

- **Reduced enterprise middleware complexity:** Traditional enterprise middleware offers the capability to address deployment concerns such as clustering, load balancing, state management, application monitoring, etc., which is beneficial for large monolithic applications. With the adoption of microservices architectural style and containerized deployment into cloud, this capability is redundant. In response, middleware vendors now offer smaller-footprint versions (e.g., IBM WebSphere Liberty profile, RedHat JBoss, WildFly) as container images, to address complexity reduction and commoditization of enterprise middleware to embrace microservices and containerization.

## CAAS VS. PAAS

PaaS offerings (especially those that are vendor driven) and the Docker ecosystem are often compared, although they do not necessarily compete and can be beneficial if used together. Many PaaS vendors have actively adopted the Docker container as their internal unit of deployment and utilize many of the same open source components as the Docker ecosystem. Docker-based CaaS can be effectively used as the foundation to build a custom PaaS platform for microservices implementation. That said, organizations looking to adopt microservices architecture need to identify and track the strengths of both these technologies. Figure 1 summarizes our observations at present.

## Integral Support for Microservices

This is the extent to which a platform enables microservices with built-in features and capabilities. PaaS platforms tend to bake in the microservices architecture elements and practices, whereas the Docker ecosystem takes a relatively neutral approach. For example, Pivotal Cloud Foundry PaaS provides an implementation of a circuit-breaker pattern used for building resilience against cascading failures across services, whereas a custom implementation (such as container adaptation of Hystrix from Netflix OSS) may be necessary in a Docker-based CaaS.

## CI/CD Automation

This dimension indicates how well a given platform supports automation in continuous integration and continuous delivery. It is critical for high velocity innovation and both PaaS and the Docker ecosystem tend to leverage best-of-breed open source CI/CD tools to implement it.

## Infrastructure Abstraction

Platforms typically hide the details of underlying infrastructure from developers by automating deployment and monitoring aspects. PaaS offerings tend to encapsulate hardware and virtualization layers, providing a rich API for developers to use in the application lifecycle. PaaS developers provision application run-times and available services from a catalog. On the other hand, the Docker ecosystem allows developers to construct their application run-time by pulling the required Docker images from a registry, building custom code and applying configurations – in addition to providing tools to automate container deployment, load balancing, etc.

## Production Ready

Production readiness is a function of technology maturity and stakeholder perception built from field experience. As adoption grows, additional

## Comparing, Contrasting CaaS and PaaS Approaches

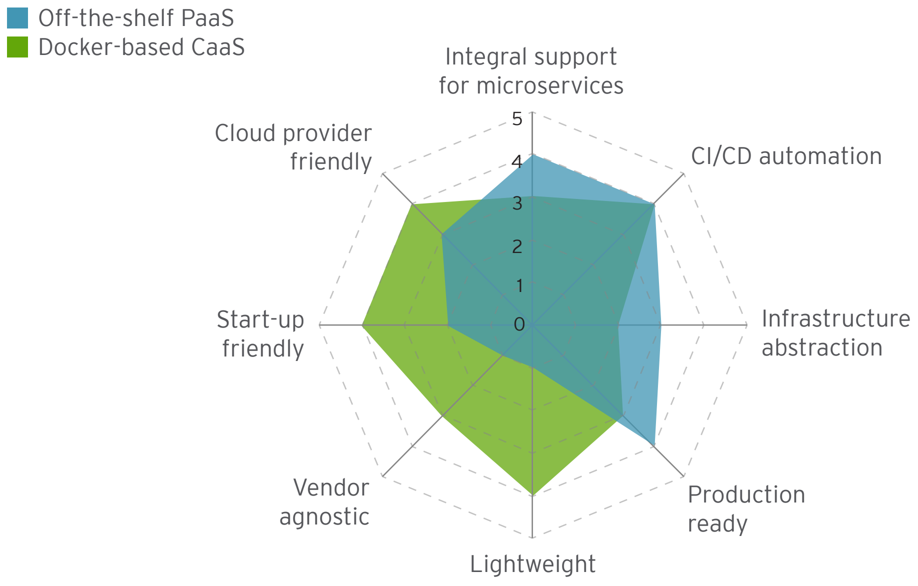

Figure 1

Docker has far less of an entry (or exit) barrier from a technology adoption perspective, compared to most PaaS offerings.

enterprises are likely to extend the continuous delivery automation into production environments. At the time of this writing, inter-container networking and security are among the main complexity areas in Docker container deployment.

### Lightweight

Although a relative term, lightweight in this context means a low overhead, nimble platform that does not warrant a large up-front infrastructure setup or change management effort. Docker has far less of an entry (or exit) barrier from a technology adoption perspective, compared to most PaaS offerings.

### Vendor Agnostic

Portability and avoidance of vendor lock-in are quite high on the agenda of most organizations going digital. With the penetration of open source components in each and every aspect of today's software lifecycle, platforms are far less proprietary than ever before. However, it is still necessary to trade off between out-of-the-box value from a platform versus flexibility, control and freedom of choice.

### Start-Up Friendly

This comparison parameter is significant since enabling rapid innovation is a key promise of microservices and DevOps, and start-ups have often been the birthplace of disruptive innovation. Start-up-friendly platforms are characterized by a low entry barrier, open source culture and a vibrant community that drives rapid and collaborative feature enhancements via experimental work.

### Cloud Provider Friendly

Not only start-ups, but even large businesses and enterprises are betting big on cloud-native systems, expecting the underlying infrastructure and services to be techno-commercially elastic. In turn, cloud providers look to include rapid innovation platform capabilities within their offerings and are becoming the main source of these capabilities. Naturally, the cloud-provider-friendly platforms will find broader and accelerated traction.

## ENTRY POINTS INTO THE ENTERPRISE IT LANDSCAPE

Docker is enormously popular; however, it has been finding its way into the enterprise IT landscape via one of three distinct entry points: as an instrument of lightweight virtualization, as the foundation of CaaS platform (public or on-premises) or as the built-in container governance construct of PaaS (also public or on-premises).

Organizations that are freshly implementing microservices, or have very few legacy monolithic systems, can leverage Docker in a CaaS environment. Start-ups can choose from the numerous IaaS cloud providers that fully support (or even promote) Docker, whereas others looking to host on premises can build their own CaaS fairly quickly with modest infrastructure updates or investments.

On the other hand, large organizations may take a gradual migration route due to the preponderance of monolithic legacy applications running in their production environments. Our current view, based on conversations and engagement experiences with our clients, indicates that a majority of such organizations have evaluated Docker and are using Docker container images as lightweight virtual instances in build/test automation, but aren't yet ready to deploy it in their production environments. The focus is on image standardization and reuse via DTR and just-in-time instantiation of containers (e.g., Jenkins slaves) with an aspiration to achieve blue-green deployment, A/B testing or canary releases.

A number of large IT organizations with whom we have spoken have made up-front investments in PaaS cloud (typically on premises) to achieve better infrastructure abstraction or even production readiness, and have embarked on vendor-supported monolith-to-microservices transformation projects. Docker containers are seen even in these environments, often natively built into the PaaS or in an experimental/labs setup.

## LOOKING AHEAD

Organizations adopting containerization can leverage Docker as a software-defined unit of application delivery, testing and deployment. Docker adds significant value into CI/CD automation due to its low-overhead virtualization and command-line programming interface for scripting.

Docker registry promotes discovery and reuse of self-contained software, whereas Docker UCP or open source alternatives enable service discovery, orchestration and load balancing of container-hosted microservices instances.

IT organizations looking at a Docker-based CaaS model as an option to build a microservices architecture need to consider the following points in order to validate techno-cultural fit:

- Does the existing IT infrastructure support Docker out of the box or with minor upgrades?
- Does the IT landscape leverage Docker-aware open source software?
- Is the IT team inclined to build and manage in-house applications provisioning and deployment?
- Are there custom nonfunctional components (e.g., cluster management, load balancing) or a CI/CD automation harness that can be reused in a containerized environment?
- Has containerization been achieved or planned in the production environment?

Affirmative responses indicate a natural fit for the Docker-based CaaS model.

## FOOTNOTES

1 Microservices architecture is an architectural style that promotes implementation of software applications as a set of independent, self-contained services, each focusing on one business concern; www.cognizant.com/InsightsWhitepapers/Overcoming-Ongoing-Digital-Transformational-Challenges-with-a-Microservices-Architecture-codex1598.pdf.

2 DevOps is a practice that integrates the activities of developers and IT operations to enable a more agile relationship. For a deeper dive, read www.cognizant.com/content/dam/Cognizant_Dotcom/whitepapers/patterns-for-success-lessons-learned-when-adopting-enterprise-devops-codex2393.pdf.

3 Continuous delivery refers to a software engineering approach where software is released in short, rapid cycles leveraging build, test and deployment automation; martinfowler.com/books/continuousDelivery.html.

4 Netflix OSS is a set of components useful for building a microservices platform. Netflix deploys these in production for its own services and has open sourced under Apache License; netflix.github.io/.

5 Docker is an open platform for developers and sysadmins to build, ship and run distributed applications, whether on laptops, data center virtual machines or the cloud. For more information, visit www.docker.com.

6 Blue-green deployment is a release-management technique where two identical environments called blue and green are simultaneously running; at any time, one is "live" and the other is used for fresh deployments and preproduction testing; martinfowler.com/bliki/BlueGreenDeployment.html.

7 A/B testing refers to a controlled experiment in production environment where two variants of the same software is simultaneously available to targeted sets of users; en.wikipedia.org/wiki/A/B_testing.

## ABOUT THE AUTHORS

### Milind Lele

**Chief Architect, Global Technology Office**

Milind Lele is a Chief Architect within Cognizant's Global Technology Office, where he leads numerous initiatives in software engineering and architecture labs. He has a post-graduate diploma in software technology and has over 22 years of IT experience spanning enterprise architecture, technology consulting, large green-field system integration projects and technology research, and innovation. Milind can be reached at Milind.Lele@cognizant.com.

### Saroj Pradhan

**Head of Software Engineering and Architecture Lab**

Saroj Pradhan heads the Software Engineering and Architecture Lab within Cognizant's Global Technology Office. He has over 20 years of experience in product engineering and innovation, building solutions and platforms in software engineering automation, DevOps and cloud computing. Saroj has a master's degree in computer applications from National Institute of Technology, Rourkela. He can be reached at Saroj.Pradhan@cognizant.com.

## ABOUT COGNIZANT

Cognizant®

**World Headquarters**
500 Frank W. Burr Blvd.
Teaneck, NJ 07666 USA
Phone: +1 201 801 0233
Fax: +1 201 801 0243
Toll Free: +1 888 937 3277

**European Headquarters**
1 Kingdom Street
Paddington Central
London W2 6BD England
Phone: +44 (0) 20 7297 7600
Fax: +44 (0) 20 7121 0102

**India Operations Headquarters**
#5/535 Old Mahabalipuram Road
Okkiyam Pettai, Thoraipakkam
Chennai, 600 096 India
Phone: +91 (0) 44 4209 6000
Fax: +91 (0) 44 4209 6060

TL Codex 2790