E-BOOK

BEST PRACTICES FOR MIGRATING TO CONTAINERIZED APPLICATIONS

BACKGROUND

When thinking about migrating applications into containers, there are three main high-level strategies: lift and shift, augment, and rewrite.[1]

No matter which method you choose, it is important to recognize that most software was designed and written before modern, image-based containers were invented.[2] Even if you choose the "lift and shift" method, where you might run a monolithic application inside a single container, it is likely that your application will need to be modified. To successfully move to containers, you will need a solid migration strategy that takes into account the needs of your applications and the nature of Linux® containers.

This document will outline specific, technical recommendations and guidelines for migrating software into containers, ranging from image build procedures to how they should run in production. The application requirements will dictate how the application should be migrated.

MINDSET

Using containers is as much of a business advantage as a technical one. When building and using containers, layering is crucial. You need to look at your application and think about each of the pieces and how they work together—similar to the way you can break up a program into a series of classes and functions. Containers are composed of packages and scripts that combine with other containers to build your application. So approach containers with the mindset that your application is made up of smaller units, and the packaging of those units into something easily consumable will make your containerized application easier to understand, deploy, and maintain.

PROPER LAYERING

The purpose of layering is to provide a thin level of abstraction above the previous layer to build something more complex. Layers are logical units where the contents are the same type of object or perform a similar task.

1 *"Containers for Grownups: Migrating Traditional & Existing Applications."* http://schd.ws/hosted_files/lcccna2016/91/Containers%20for%20Grownups_%20Migrating%20Traditional%20%26%20Existing%20Applications.pdf. *Accessed 24 Jul. 2017.*

2 *"Containers for Grownups: Migrating Traditional & Existing Applications."* http://schd.ws/hosted_files/lcccna2016/91/Containers%20for%20Grownups_%20Migrating%20Traditional%20%26%20Existing%20Applications.pdf. *Accessed 24 Jul. 2017.*

The right number of layers will make your container easy to consume. Too many layers will be too complex and too difficult to consume. The proper number of layers for an application should reflect the complexity of your application—the more complex the application, the more layers. For example, if a Hello World container prints to standard output (stdout) "Hello World," it requires no configuration, process management, or dependencies, so it needs a single layer. But, if we expand the Hello World application to say "hello" to the user, we will need a second layer to gather input.

START SCRIPTS

Start scripts are very effective in providing a thin layer of abstraction above the service layer that runs when a container starts. Start scripts deliver additional muscle on top of the base layer using a very simple application programming interface (API) for the operator to plug into. The most common tasks of start scripts are setting permissions, moving config files, changing ownership of files, cleaning directories, and starting the service.

REVOLVE AROUND THE RESTART

Every life-cycle action for an application will use a restart because it is a cheap and effective way to signal to the process that an operation is occurring. And life-cycle operations require a restart in order change how the process is running.

As an example, think about performing a reconfigure operation for MariaDB. Consider a JSON file, a simple remote procedure call (RPC), that maps config file locations and start information. This JSON file is interpreted by set_configs.py, which copies the config files to their locations when the container starts. The user reconfigures MariaDB by changing a configuration setting on the host and restarting the container.

EMPOWER ORCHESTRATION

Every layer of your application is not meant to be in the container, so do not add too many layers. If you do, you will quickly overlap with existing tools, other layers, and over complicate the container. We do not want to add additional work by building tools into our containers that can only deploy and manage our application. Instead, we want to make it easy for orchestration tools to manage our well-packaged containers.

APPLICATION REQUIREMENTS

Applications have specific requirements from an architectural, security, and performance perspective. Some of these requirements will have an impact on the level of effort required to migrate the application into a container or break it apart into multiple containers.[3]

Figure 1. Application requirements

ARCHITECTURE

From an architectural perspective, moving applications into containers is not unlike a Unix to Linux migration, or an operating system upgrade. Often, an application will have been running for years. The documentation will often be nonexistent or out of date. As with most migrations, the technical person performing it will have to do the work necessary to understand how the application works structurally. They will have to reverse engineer how it was set up. At a minimum, they must be able to answer questions like:

1. Where are the binaries for this application? Are they installed through an installer that puts them in a single place, or are they spread throughout the file system? Is there a single binary that is easy to start, or does it have a simple systemd unit file that can be used?
2. Where does the data for this application reside? Is it read-only or read-write? Can it safely be written to by two concurrent processes?
3. Where does all of the configuration data reside? Is it in a single directory, single file, or multiple places throughout the file system?
4. What kind of secret data does this application have? Can the location of secrets be configured in the application? Can they be moved into separate directories, or can they be accessed using some kind of key through an identity or certificate server?
5. What kind of network access does this application need? Is it simple HTTP? Is it name server, which will require user datagram protocol (UDP)? Or, is it a really complex application that needs point-to-point encryption between containers using something like internet protocol security (IPsec)?
6. Is the installer a shell script that can be reverse engineered for more information about application setup? Are the binaries installed through RPMs or some other kind of package manager?

3 *"Container Tidbits: When Should I Break My Application into Multiple ...." 16 Mar. 2016, http://rhelblog.redhat.com/2016/03/16/container-tidbits-when-should-i-break-my-application-into-multiple-containers/. Accessed 22 Mar. 2017.*

7. Does the application's licensing allow you to easily distribute the application inside of a container image? Sometimes licensing is very restrictive; other times you may have a site license.
8. Does the application restart easily? Apache can restart thousands of times without failing, but a database may corrupt the tables. Could this make it harder to orchestrate and recover?

Answering these questions determines if your application is even a good fit for container migration—if the difficulty level is too high, it will not provide a return on investment.[4]

TABLE 1. TYPICAL WORKLOADS SEEN IN THE DATACENTER

	EASY	MODERATE	DIFFICULT
Code	Completely isolated (single process)	Somewhat isolated (multiple processes)	Self-modifying (e.g. actor model)
Configuration	One file	Several files	Anywhere in file system
Data	Saved in single space	Saved in several places	Anywhere in file system
Secrets	Static files	Network	Dynamic generation of certifcations
Network	HTTP, HTTPs	TCP, UDP	IPSEC, highly isolated
Installation	Packages, source	Installer and understood configuration	Installers (install.sh)
Licensing	Open source	Proprietary	Restrictive and proprietary

SECURITY

In many ways, security decisions for containerized applications are no different from regular applications that run in processes.[5] You have to decide what level of isolation is enough for the given application. When evaluating a workload for migration, and deciding whether containers offer enough isolation, it is important to review how much isolation the workload has where it is currently running.

Going from left to right in Figure 2, each technology decision offers increasing isolation. For some applications, regular Linux processes offer enough isolation. It is common to run MySQL and a web server on the same Linux operating system instance. At the other end of the spectrum, there are times when two copies of an application need to live in two different datacenters, which are affected by different weather and earthquake patterns—this is common for disaster recovery.

As an example, high-performance computing (HPC) workloads are commonly run today in large clusters with only regular Linux process isolation. This is not perfect—researchers in a large cluster could attempt to hack each other's processes, but this is commonly determined to be an acceptable level of risk.

4 *"Architecting Containers Part 4: Workload Characteristics and ...." 21 Apr. 2016, http://rhelblog.redhat.com/2016/04/21/architecting-containers-part-4-workload-characteristics-and-candidates-for-containerization/. Accessed 24 Jul. 2017.*

5 *"Why containers are really just fancy files and fancy processes." 6 Mar. 2017, http://sdtimes.com/guest-view-containers-really-just-fancy-files-fancy-processes/. Accessed 22 Mar. 2017.*

As another example, even with virtual machines, it is common to have external-facing services, such as domain name server (DNS), HTTP, or virtual private network (VPN) services running in an external network in a completely different virtualization cluster from internal-facing services, such as Oracle databases or SAP instances. This configuration would be the equivalent of rack-level isolation. Very few security experts would be comfortable running these two types of workloads in the same virtualization cluster, and the same is true with container platforms—typically these types of services would be run in different clusters.

So, when you wonder if containers offer enough isolation, think about it in the context of the workload requirements.

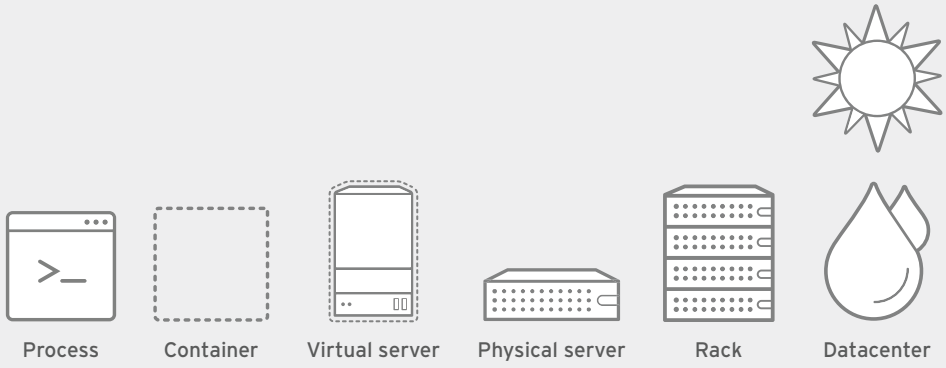

Figure 2. The tenancy scale

PERFORMANCE

Workloads must also be analyzed for performance. Containers and virtualization are additive technologies that can be combined with bare-metal hardware to provide features and capabilities. Table 2 offers a quick guide to help determine important capabilities that should be considered when migrating applications into containers.

Containers are Linux processes that use technologies, such as control groups (cgroups), Security-Enhanced Linux (SELinux), and namespaces to provide a higher level of isolation to applications. This allows them to run at native or near native speed. There is no layer of abstraction like virtualization, so all of the containerized applications in a cluster must be based on the same hardware architecture and operating system.

Using the same example, in an HPC environment, containers may be added to bare metal to provide a similar level of performance, while increasing the level of isolation. On the other hand, if the workload is a corporate application that requires components that live in Windows and Linux, a combination of containers and virtualization may be a better choice. Running containers inside of virtual machines offers the combination of hardware freedom, better isolation, and increased manageability using container images.[6]

6 *"When Containers and Virtualization Do - and Don't - Work Together …."* *https://summits.brighttalk.com/webinar/when-containers-and-virtualization-do-and-dont-work-together/. Accessed 22 Mar. 2017.*

Use Table 2 to better understand the trade-offs of combining different technologies:

TABLE 2. WORKLOAD PLATFORM COMPARISON

	BARE METAL	+CONTAINERS	+VIRTUALIZATION
CPU intensive	Fast	**Fast**	**Fast**
Memory intensive	**Fast**	**Fast**	**Fast**
Disk I/O latency	**Fast**	**Fast**	**Medium**
Disk I/O throughput	Fast	Fast	Fast
Network latency	Fast	Fast	Medium
Network throughput	Fast	Fast	Fast
Deployment speed	Slow	Fast	Medium
Uptime (live migration)	No	No	Yes
Alternative OS	**Yes**	**Some**	**Yes**

TECHNICAL CHECKLISTS

Best practices

- Layer your application.
- The number of layers should reflect the complexity of your application.
- Containers are a slightly higher level of abstraction than an RPM.
- Avoid solving every problem inside the container.
- Use the start script layer to provide a simple extraction from the process runtime.
- Build clear and concise operations into the container to be controlled by outside tools.
- Identify and separate code, configuration, and data.
- Code should live in the image layers.
- Configuration, data, and secrets should come from the environment.
- Containers are meant to be restarted.
- Do not recreate processes.
- Never build off the latest tag—it prevents builds from being reproducible over time.
- Use liveliness and readiness checks.

Architecture

Consider this checklist when creating containerized application images.[7] It is important to know the following information:

1. Identify the location of all binaries necessary for the application to run in a container.
   a. Use layers—think about core builds and application runtime layers.[8,9]
   b. Identify dependencies and determine if previous layers should contain the dependencies, especially if they can be shared or used by other applications.
   c. Identify how the binaries will be started: script, systemd, etc.
2. Identify files and directories that contain configuration for the application. These will need to be bind-mounted into the application at runtime. Configuration should come from the environment (dev/QA/production) and should not be embedded inside of the container image.
3. Identify the files and directories that will contain data for the application. These will need to be bind-mounted into the application at runtime. Application data should come from the environment—dev/QA/production—and should not be embedded inside the container image.
4. Determine what network protocols the application will need. This will determine if these services[10] are served internally to the cluster or external to customer.
5. Can the installer script be reverse engineered to better understand how it works?
   a. Determine configuration changes it makes—try to determine if scripting or configuration management could be used to replace the installer script. Could the configuration be passed to the container image at runtime to set the parameters dynamically?
   b. Determine where data is stored—try to determine if the data has schema set up during install and if this could be scripted at application runtime.
6. Determine if the service can be restarted easily. If the service is sensitive to restarts, use liveliness and readiness checks[11] to determine if operator intervention is necessary. It is critical to automate as much as possible in a container orchestration environment.
7. Determine where logging and output should go. Typically, RPM-installed applications will put logs in /var/log or other known locations. In a containerized environment, this could dump a lot of data into the read-write layer.[12]

---

7 *"GitHub - opencontainers/image-spec: OCI Image Format." https://github.com/opencontainers/image-spec. Accessed 22 Mar. 2017.*

8 *"Architecting Containers Part 5 - Red Hat Enterprise Linux Blog." 18 May. 2016, http://rhelblog.redhat.com/2016/05/18/architecting-containers-part-5-building-a-secure-and-manageable-container-software-supply-chain/. Accessed 22 Mar. 2017.*

9 *"GitHub - fatherlinux/container-supply-chain: Demo showing how to ...." https://github.com/fatherlinux/container-supply-chain. Accessed 22 Mar. 2017.*

10 *"Services - Kubernetes." http://kubernetes.io/docs/user-guide/services/. Accessed 22 Mar. 2017.*

11 *"Configuring Liveness and Readiness Probes - Kubernetes." http://kubernetes.io/docs/tasks/configure-pod-container/configure-liveness-readiness-probes/. Accessed 22 Mar. 2017.*

12 *"GitHub - fatherlinux/container-internals-lab: Container internals lab for ...." https://github.com/fatherlinux/container-internals-lab. Accessed 17 Jul. 2017.*

8. Determine if scaling individual processes is necessary. Break the application up into multiple containers if necessary.[13]

Security

Consider this checklist when creating containerized application images.[14] It is important to know the following information:

1. Always use policies, where possible. Use security context constraints and service accounts[15] to set these policies. Red Hat has found that profile-based controls generally work better and are typically more widely adopted and maintained than custom rules for each application.
2. Identify files that contain secrets[16] for the application. This includes things like certificates and passwords. These should never be embedded in a container image because anyone who downloads the image will have access to them. They should be protected and provided by the container environment.
3. Avoid running containers on low ports that require privilege. Use the built-in security context constraints[17] that block this.
4. Avoid running containers as root.[18] Even with the use of kernel user namespaces, there is the risk of escalation out of the container, which could compromise the underlying container host. Use the built-in security context constraints that block this.
5. Where possible, run containers with a read-only root file system. Use security context constraints to enforce this.
   a. For web content, run the service with a read-only volume mount as well.[19]
6. Determine the level of isolation needed. Use the principles of least privilege[20] and defense in depth.[21] In a Red Hat® OpenShift Container Platform environment, configure the following technologies with security context constraints and service accounts.

---

13 *"Container Tidbits: When Should I Break My Application into Multiple ...." 16 Mar. 2016, http://rhelblog.redhat.com/2016/03/16/container-tidbits-when-should-i-break-my-application-into-multiple-containers/. Accessed 24 Jul. 2017.*

14 *"GitHub - opencontainers/image-spec: OCI Image Format." https://github.com/opencontainers/image-spec. Accessed 22 Mar. 2017.*

15 *"Understanding Service Accounts and SCCs – OpenShift Blog." 15 Apr. 2016, https://blog.openshift.com/understanding-service-accounts-sccs/. Accessed 24 Jul. 2017.*

16 *"Secrets - Kubernetes." http://kubernetes.io/docs/user-guide/secrets/. Accessed 22 Mar. 2017.*

17 *"Authorization - Additional Concepts | Architecture | OpenShift ...." https://docs.openshift.com/container-platform/3.5/architecture/additional_concepts/authorization.html. Accessed 21 Jul. 2017.*

18 *"Can I haz non-privileged containers? by mhausenblas." http://canihaznonprivilegedcontainers.info/. Accessed 24 Jul. 2017.*

19 *"Volumes | Kubernetes." https://kubernetes.io/docs/concepts/storage/volumes/. Accessed 24 Jul. 2017.*

20 *"What is principle of least privilege (POLP)? - Definition from WhatIs.com." http://searchsecurity.techtarget.com/definition/principle-of-least-privilege-POLP. Accessed 21 Jul. 2017.*

21 *"GitHub - fatherlinux/container-defense-in-depth." https://github.com/fatherlinux/container-defense-in-depth. Accessed 21 Jul. 2017.*

a. SELinux—Red Hat Enterprise Linux systems come with a configurable profile that works out of the box.[22] Dynamic contexts are generated for each container using secure virtualization (sVirt).[23]

b. Secure computing (seccomp)[24]—Containers are run without a default seccomp profile. Users must determine and configure a profile for themselves.

c. Linux capabilities:[25]

   i. The default security context constraint in OpenShift is restricted, which drops the following capabilities: KILL, MKNOD, SYS_CHROOT, SETUID, SETGID.

   ii. Administrators may want to create custom constraints, which limit the following capabilities, especially for processes that will be granted root privilege (useful for administrators): AUDIT_CONTROL, BLOCK_SUSPEND, DAC_READ_SEARCH, IPC_LOCK, IPC_OWNER, LEASE, LINUX_IMMUTABLE, MAC_OVERRIDE, and MAC_ADMIN.

   iii. Administrators may want to create custom constraints, which grant the following capabilities, especially for processes which will be granted root privilege (useful for administrative tasks): NET_ADMIN, NET_BROADCAST, SYS_ADMIN, SYS_BOOT, SYS_MODULE, SYS_NICE, SYS_PTRACE, SYS_PACCT, SYS_RAWIO, SYS_RESOURCE, SYS_TIME, SYS_TTY_CONFIG, SYSLOG, and WAKE_ALARM.

d. Security context constraints are provided by Red Hat OpenShift Container Platform and provide good default rules.

Performance

Consider this checklist when creating containerized application images.[26] It is important to know the following information:

1. Determine if applications access or create temporary files. By default, these file systems are mounted-read-only. These can be mounted as volumes,[27] but be careful with read-write access from multiple processes.

22 *"Chapter 6. Docker SELinux Security Policy - Red Hat Customer Portal."* **https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux_atomic_host/7/html/container_security_guide/docker_selinux_security_policy**. *Accessed 21 Jul. 2017.*

23 *"Securing Docker Containers with sVirt and Trusted Sources - Crunch ...." 21 May, 2015,* **http://crunchtools.com/securing-docker-svirt/**. *Accessed 24 Jul. 2017.*

24 *"Authorization - Additional Concepts | Architecture | OpenShift Origin ...."* **https://docs.openshift.org/latest/architecture/additional_concepts/authorization.html**. *Accessed 24 Jul. 2017.*

25 *"Capabilities(7) - Linux manual page - man7.org."* **http://man7.org/linux/man-pages/man7/capabilities.7.html**. *Accessed 24 Jul. 2017.*

26 *"GitHub - opencontainers/image-spec: OCI Image Format."* **https://github.com/opencontainers/image-spec**. *Accessed 22 Mar. 2017.*

27 *"Volumes | Kubernetes."* **https://kubernetes.io/docs/concepts/storage/volumes/**. *Accessed 24 Jul. 2017.*

a. /sys

b. /proc:

i. A pseudo file system that provides an interface to kernel data structures.

ii. Mostly read-only, but some files allow kernel variables to be changed.

iii. Entries in /proc/[pid]ns represent the kernel namespaces instantiated for the containerized process id.[28]

iv. Examples include /proc/asound, /proc/bus, /proc/fs, /proc/irq, /proc/sys, and /proc/sysrq-trigger.

c. /dev:

i. Inside a container, the application can access limited device files, such as /dev/null and /dev/zero. Containerized applications cannot access host device files, such as /dev/sdX and /dev/ttySX, without using privileged mode.

d. /run:

i. After docker v1.10, user can pass–tmpfs option for docker run, then the /run is mounted as tmpfs inside a container.

ii. On Red Hat systems, /run/secrets is always mounted as tmpfs to provide a place to inject subscription information.

iii. On the Linux host, /run is mounted as tmpfs to save the process' temporary data (e.g. pid of daemon). This will be removed after a server reboot. Inside a container, only /run/secrets is mounted as tmpfs–/run itself is included in / (root) file system. Therefore, files under /run are not removed even if the container is restarted.

2. Determine if the application requires changes to kernel parameters (/proc/sys) or access to special hardware.

a. By default, kernel tuning variables under /proc/sys are read-only for a containerized process.

b. Running these types of applications may require configuring certain nodes with the correct kernel parameters or hardware and using node selectors to schedule the application on nodes with special configuration or resources.[29] Examples include /proc/sys/fs/mqueue, /proc/sys/kernel/{msgmax, msgmnb, msgmni, sem, shmall, shmmax, shmmni, and shm_rmid_forced}.

c. If the application needs to change kernel parameters itself (for example, via sysctl), it should be run in an isolated cluster with privileged containers.

---

28 *"namespaces(7) - Linux manual page - man7.org."* ***http://man7.org/linux/man-pages/man7/namespaces.7.html****. Accessed 24 Jul. 2017.*

29 *"Simple use of selectors to get pods to land on the desired nodes ...." 12 Aug. 2016,* ***https://blog.openshift.com/use-of-selectors-to-get-pods-on-desired-nodes/****. Accessed 24 Jul. 2017.*

3. Date, time, and locale:
   a. Determine if the application needs a different locale setting (for example, JST). Set this in the image at build time—rebuild image.[30]
   b. Determine if the application needs to change date.[31]
4. Determine if the application expects to use a fixed IP address.
   a. Avoid static IP configuration where possible.
   b. IP address of container network interface cannot be changed.
   c. Use hostnames where possible because the service network in Kubernetes will take care of networking.
   d. If the application has parameters or a configuration that includes an IP address, intercept the ENTRYPOINT to dynamically change the configuration file on start. Use tools like SED or Ansible® to do this.
5. Determine if the application requires the use of multiple network interfaces.
   a. Multiple virtual network interface controllers (NICs) are currently not supported with Kubernetes. Network redundant functions like bonding cannot be used inside containers; this should be done at the host level. Pods should be designed to fail and restart on a node with a working network. Configure the proper liveliness and readiness checks.
   b. Multiple network interfaces can be achieved with some third-party tools.[32]

CONCLUSION

Successfully migrating an existing application into a container, or containers, requires that you understand the application and develop a comprehensive plan. Almost any application can be containerized, but it is important to understand the amount of effort that will be required and ensure that the transition to containers preserves performance and maintains or improves security.

30 *"Changing The Time Zone In Linux (Command Line) – Linux Academy ...." 30 Jul. 2012, https://linuxacademy.com/blog/linux/changing-the-time-zone-in-linux-command-line/. Accessed 24 Jul. 2017.*

31 *"java - Is it possible change date in docker container? - Stack Overflow." 10 Apr. 2015, http://stackoverflow.com/questions/29556879/is-it-possible-change-date-in-docker-container. Accessed 22 Mar. 2017.*

32 *"Support multiple pod IP addresses · Issue #27398 · kubernetes ...." 14 Jun. 2016, https://github.com/kubernetes/kubernetes/issues/27398. Accessed 24 Jul. 2017.*

NORTH AMERICA
1 888 REDHAT1

EUROPE, MIDDLE EAST, AND AFRICA
00800 7334 2835
europe@redhat.com

ASIA PACIFIC
+65 6490 4200
apac@redhat.com

LATIN AMERICA
+54 11 4329 7300
info-latam@redhat.com

facebook.com/redhatinc
@redhatnews
linkedin.com/company/red-hat

redhat.com
#9195_01017