DEPLOY YOUR CONTAINERS WITH DELL EMC SCALEIO

ABSTRACT

This white paper describes the need for containers for the modern microservices based applications and explains how Dell EMC™ ScaleIO® with REX-Ray™ provides persistent storage for the containers.

October, 2016

TABLE OF CONTENTS

INTRODUCTION 4
AUDIENCE 4

PARADIGM SHIFT IN APPLICATION DEVELOPMENT 4
Traditional Applications 5
Modern Applications 5

INTRODUCTION TO CONTAINERS 6
Docker 7

HOW SCALEIO AND DOCKER ENABLE DEVOPS 8
Why use ScaleIO for DevOps 9
Docker Enables DevOps 10

DOCKER STORAGE 10
Limitations of Docker storage 12

SCALEIO VOLUME ORCHESTRATION USING REX-RAY FOR PERSISTENT DOCKER STORAGE 12
REX-Ray 12
REX-Ray Architecture 13
Setup and Configuration 14
Demonstration 16

CONTAINERS WITH PERSISTENT STORAGE SUPPORT THE COMPLETE APPLICATION LIFECYCLE 18

CONCLUSION 19

REFERENCES 20

LIST OF TABLES AND FIGURES 20

INTRODUCTION

Most traditional applications are based on a single tiered monolithic architecture in which all the code is combined into a single program running on a single host platform. These applications are self-contained, feature end-to-end software coding and are independent from other computing applications. These client applications are typically more difficult to deploy, harder to scale, and require additional operational overhead.

Most contemporary applications are evolving from a monolithic to a more microservices based architecture, which run as an independent service comprised of multiple code modules/segments that can communicate with each other using APIs. These 'modularized' applications are faster to test and deploy, easier to scale and can run on multiple platforms. However, microservices based applications pose new challenges for the operations team. Each microservice has its own software libraries and frameworks. Operations specialists have to maintain a different variety of hardware and software stacks to enable them. A new coding and quality assurance methodology called DevOps has arisen around microservices based applications. DevOps is based on a constant stream of code updates rather than periodic large software releases. DevOps has created its own set of unique infrastructure challenges.

Virtualization solutions have attempted to address this issue by abstracting the hardware from the operating system, although virtual machines do not enable easy deployment and scaling of the microservices based applications. Containers have their own isolated process space. They are a very good fit for acting as deployment units for services. However, containers have their own set of challenges. A container's data is not globally persistent and there are challenges with data persistency and data migration across different hosts.

Dell EMC ScaleIO, which is an enterprise grade high performance software defined storage solution is a perfect match for operations team to meet the storage needs of the developers on the fly.

REX-Ray is an open-source, vendor-agnostic storage orchestration tool which delivers persistent storage access for container runtimes, such as Docker and Mesos, and provides an easy interface for enabling advanced storage functionality across common storage, virtualization and cloud platforms.

This white paper describes the relationship between ScaleIO, REX-Ray and Docker. It also describes the solution to the container persistence problem using Redis (a key/value datastore used in modern applications) as an example.

AUDIENCE

The white paper is targeted for developers, system architects and storage administrators who are evaluating or deploying Software Defined Storage in a private cloud or containerized application environment.

It is assumed that the reader has an understanding of ScaleIO components and architecture. For better understanding of ScaleIO's architecture, please refer to the Dell EMC ScaleIO: Software Defined Server SAN – Architecture Matters white paper.

PARADIGM SHIFT IN APPLICATION DEVELOPMENT

Traditional applications tend to be more monolithic in their architecture, putting all of their functionality into a single process. Newly emerging modern applications are mostly loosely coupled, meaning they break down large applications into smaller independent applications and discrete code segments that communicate with each other over APIs.

This section describes this paradigm shift in application development and why there is a need for containers and a better DevOps strategy.

TRADITIONAL APPLICATIONS

A well-written software program should be able to abstract the complexity and present a simple consumable solution to a problem. Traditionally, an application was developed as a monolithic entity where all the services and their interactions were tied together using internal mechanisms and system calls. The application is responsible not just for a particular task but it needs to perform every step required to complete all the tasks for that application. Most monolithic applications run on a virtual machine infrastructure or client/server clusters and they form their own private data silos. Monolithic applications do not provide infrastructure modularity and are characterized by long development cycles and complex quality assurance.

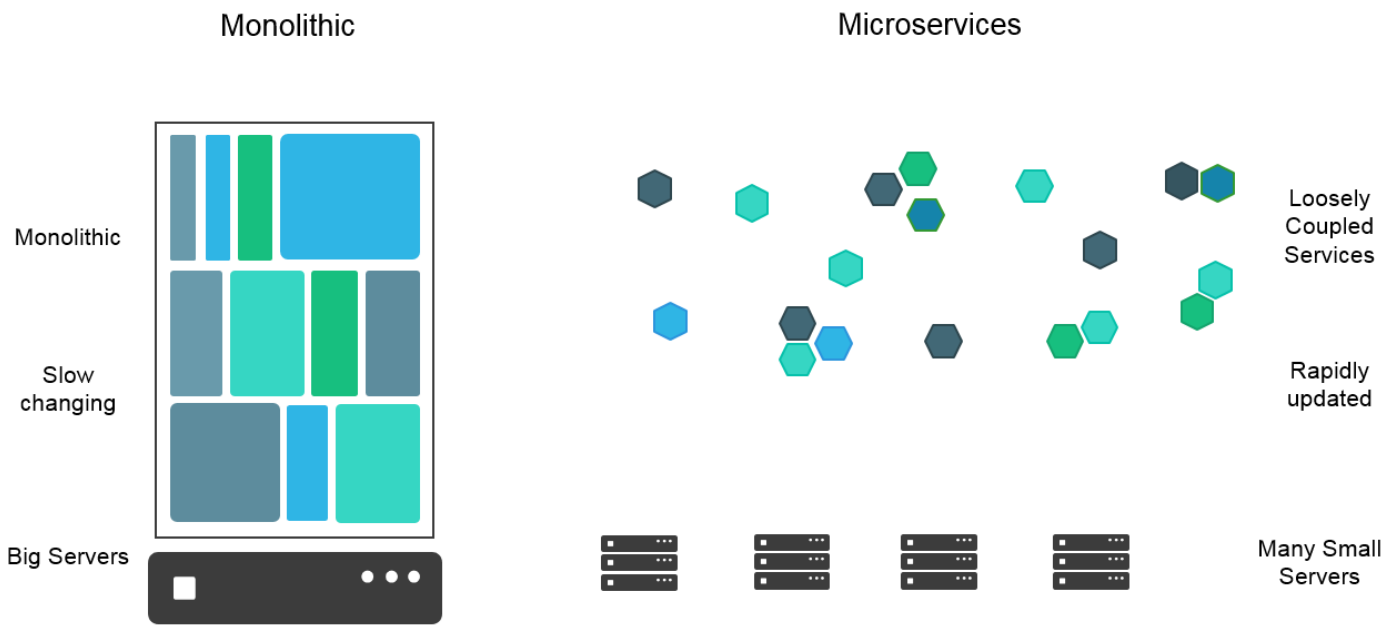
Some of the challenges of traditional monolithic applications are:

- As an application grows, the code increases with it. It can lead to longer development cycles and slows down the cadence of product releases.
- A monolithic application is fragile – it is either working as intended or non-functional, and usually without any way to recover without manual intervention or restarting.
- Since a monolithic application is packaged as a single entity, it needs to be deployed as a single unit all at once.
- Scaling of monolithic applications is an expensive and time consuming operation because the same packaged entity must be deployed on one or more servers. Each instance of the application uses the same amount of hardware resources which might not be required or even fully utilized.

MODERN APPLICATIONS

Modern day software development lifecycle has changed tremendously. Modern applications are based on microservices which break large software projects into smaller independent loosely coupled software modules and/or code segments. Individual modules are responsible for very specific tasks (or services) which communicate with other modules via universally accessible APIs. A popular term for this new type of modern application is “server-less computing.”

Figure 1) Paradigm shift in application development:



Advantages of microservices based applications:

- Large applications are not affected by a single failure of a component or host platform.

- It is easier to upgrade an application to a new technology because each discrete service is independent and the dependencies will be much fewer as compared to monolithic applications.
- Microservices based applications are easier to scale as each component can scale independently.

However, microservices based applications have their own set of challenges:

- This new generation of applications runs on a variety of software stacks and hardware infrastructure. Even a simple web application has its own application web server and a backend database server which can potentially be running on different hardware with its own specific operating system.
- Many of the modern applications have a scale-out architecture and they require separate infrastructure for load balancers, schedulers databases etc.
- Each service has its own software library and frameworks which are used for the frontend and backend development. It can be a big challenge for operations teams to align coded microservices to backend resources.
- The applications need to run on separate infrastructures including developer, test and production environment.
- The microservice application modules tend to be in a constant state of flux, and changes are quickly and constantly rolled into production systems. There are no monolithic releases or upgrade maintenance windows.

Due to the challenges listed above, an operations development team has to maintain many different frameworks, languages and databases. They also need to make sure that all the deployment targets such as individual deployment environments, pre-production, QA, staging etc. are always the same. Often developers run out of the storage or compute resources on which they need to deploy their applications. Operations teams often need to fulfill these unplanned demands which can result in product delays.

The following sections will describe how Docker containers running on ScaleIO solves these challenges and enables better coordination between developers and the operations team while reducing cost.

INTRODUCTION TO CONTAINERS

Linux container (LXC) is an operating system level virtualization method for running multiple isolated Linux processes (containers) on a single physical host using a single Linux kernel. Linux container allows running multiple isolated user space instances. The Linux kernel provides the `cgroups` functionality which is similar to `chroot`, which provides an isolation environment to host and create a virtual copy of the software system. In addition, `cgroups` allows limitation and prioritization of resources (CPU, memory, network etc.) without the need for any virtual machines. LXC compiles kernel's `cgroups` and support for isolated namespace to provide an isolated environment for applications.

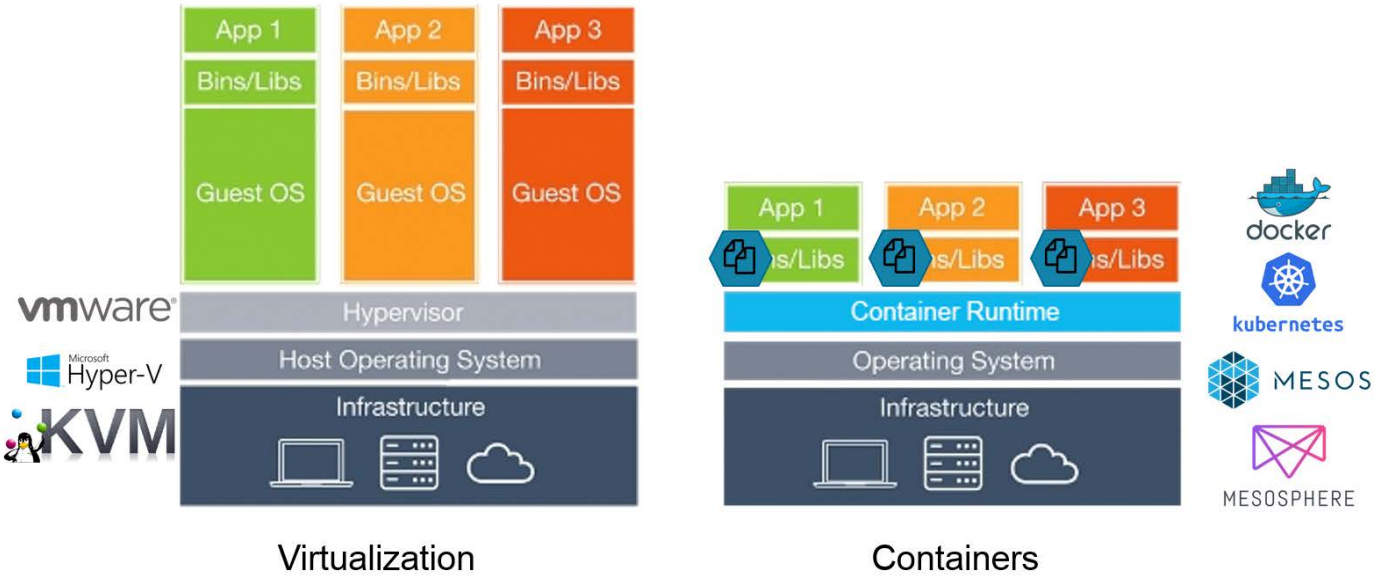
Linux containers are often thought of as super lightweight virtual machines since it provides the isolation based on `cgroups` and namespace within the Linux kernels. However, it is important to draw the differences between a container and a virtual machine.

Table 1) Virtual machines vs. containers:

	Virtual Machines	Containers
Hypervisor	Every virtual machine has its own guest operating system that has its own binaries, libraries, and dependencies with an application on top.	Containers don't need a hypervisor. They share the host's Linux kernel to provide isolation without the overhead of an entire operating system.
Virtualization Level	Virtual machines represent hardware level virtualization and hence requires a lot more resources to run.	Containers provide operating system virtualization and thus require a lot fewer resources as they run on a same operating system.
Provisioning	Virtual machines are heavy weight hence they are time consuming to provision.	Containers are considerably smaller, easier to download and faster to provision.
Runtime Scalability	Virtual machines are difficult to scale.	Containers have a real time scale out architecture.
Upgrades	Upgrades are difficult and time consuming as all the components of the operating system needs to be upgraded.	The time spent patching or updating a multitude of Guest Virtual Machines is reduced to the host level.
Security	Virtual machines are abstracted at the physical hardware level that translates to individual kernels. Vulnerabilities in particular OS versions can't be leveraged to compromise other VMs running on the same physical host.	Containers share the same kernel; storage admins need to apply special care to avoid security issues from adjacent containers.

Operationally, it makes a lot of sense to look at containers for cost saving measures.

Figure 2) Virtualization vs. Containers:



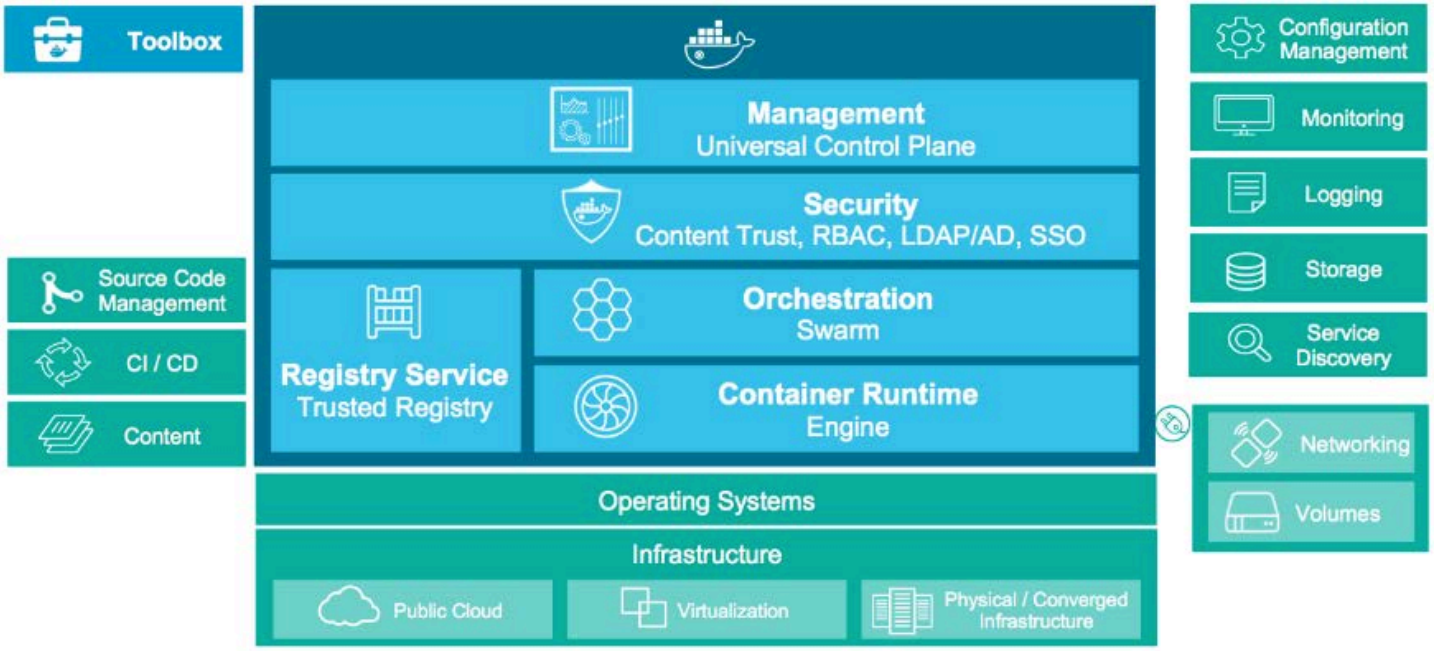
DOCKER

Docker is an open source platform for developers and sysadmins to build, ship and run distributed applications. It is a container orchestration engine that separates the application from the underlying operating system just like Linux containers (LXC). Docker is also viewed as an LXC. The current version of Docker uses its on default library `libcontainer` for running an application instead of using LXC as the default engine. By doing so, Docker has decoupled itself from the Linux dependency and thus it has the capability of running on any operating system.

Docker provides all the container benefits which are discussed above, but in addition, it provides the following on top of LXC:

- Docker is optimized for the deployment of applications as opposed to machines.
- Docker enables bundling up an application and all its dependencies into a single object that can be transferred to any Docker-enable machine. The bundled environment, also known as the Docker image, ensures that the exposed environment is always the same.
- Docker provides versioning capabilities with all the history being maintained. It provides an option to commit new versions or rollback to a previous version.
- Docker hosts a large number of container templates on the Docker Hub open community.[1]
- Containers isolate applications from one another and the underlying infrastructure, while providing an added layer of protection for the application.
- Docker containers are based on open standards, enabling containers to run on all major Linux distributions and on Microsoft Windows and on top of any infrastructure.

Figure 3) Docker infrastructure:



HOW SCALEIO AND DOCKER ENABLE DEVOPS

DevOps is a concept that emphasis collaboration and communication between developers and their operations team. The goal of the DevOps is to address the rapidly changing needs of the developers, be it compute or storage. A good DevOps strategy improves the quality and the speed of the software being shipped by an organization.

In this section, we will discuss, how ScaleIO and Docker solve the challenges described above.

[1] https://hub.docker.com/

WHY USE SCALEIO FOR DEVOPS

With so many complex components involved in modern applications, even a single bottleneck will slow down the speed of innovation. The core of these components is enterprise storage. For an operations team, it is very important that the storage infrastructure is reliable, simple to manage, high performing and elastically scalable.

ScaleIO is a software defined storage which runs on industry standard x86 servers. It discovers the local HDDs and SSDs in each server, abstracts them away from each individual server, pools them together as a shared resource, and automates the provisioning of storage performance and capacity out of that pool back to individual applications.

ScaleIO has the following advantages which makes it a perfect fit for microservices-style, containers-based application development:

- **Lightweight**: ScaleIO components i.e. (ScaleIO Data Server, ScaleIO Data Client and Meta-Data Manager) are very lightweight and use only a small amount of CPU and memory resources. This leaves a lot of resources for the application to use and run in parallel with ScaleIO (ScaleIO is commonly deployed as a hyperconverged system where the ScaleIO software runs on the same servers as customer applications).

- **Automation**: There are many tools released by the {code} by Dell EMC[2] team to automate the deployment and management of ScaleIO in a DevOps environment. For example, REX-Ray[3] orchestrates ScaleIO volume provisioning and delivers persistent storage access for container runtimes, such as Docker and Mesos. Puppet for ScaleIO[4] and Ansible for ScaleIO[5] automate installing and configuring ScaleIO components and are commonly used DevOps practices.

- **Grow as you need:** One of the biggest challenges to an operations team is to plan ahead for the consumption of storage resources. However, more often than not, an operations team either overestimates or underestimates the resources required for development purposes. Current SAN deployments cannot address this issue as the infrastructure may consist of one or more storage arrays and corresponding compute nodes, resulting in infrastructure silos. As the demand grows, more servers and compute nodes are added to the individual silos in an unbalanced manner, which leads to waste of resources.

  ScaleIO takes a de-centralized approach and makes better use of all the resources. ScaleIO scales on-the-fly based on the requirements of the DevOps team. The operations team can respond to requirements ad-hoc instead of planning ahead for the future, saving both time (to provision storage) and money. ScaleIO eliminates complex, time-consuming, and expensive data migrations. Since ScaleIO is abstracting, pooling, and automating the disks inside the servers, the process of rolling the storage is seamless. You never move all the data off one ScaleIO cluster and move it to a newer ScaleIO cluster.

- **Infrastructure agnostic:** ScaleIO is completely infrastructure hardware and software agnostic making it a true software-defined storage product. It can be used with mixed server brands, any storage media types (HDDs, SSDs and PCIe flash cards) and most importantly any operating systems i.e. Linux, Windows or any hypervisor. There are other software defined hyperconverged storage solutions, but they have very strict requirements. They are not infrastructure agnostic and cannot support heterogeneous operating systems or hypervisors. Most applications built on microservices with DevOps principles need to support a variety of operating systems.

  Using ScaleIO, an operations team can maintain a single ScaleIO software-defined storage system to host variety of applications running on heterogeneous operating systems.

- **Scalability and Performance**: ScaleIO can start on as few as 3 servers and scales to over 1,000 servers. ScaleIO is managed as a single service rather than islands of arrays. ScaleIO storage performance is very fast and predictable because both the capacity and performance inherently grow as more storage devices and compute resources are added to the pool. The data in ScaleIO is distributed evenly across all the compute nodes and underlying storage devices, which results in faster rebuild and rebalance times; with no hot spots or need for dedicated hot spare media devices. This leads to a high degree of I/O parallelism which minimized I/O latency.

---

[2] http://emccode.com/

[3] https://github.com/emccode/rexray

[4] https://forge.puppet.com/cloudscaling/scaleio

[5] https://github.com/sperreault/ansible-scaleio

- **Manageability**: Management and deployment of traditional storage systems is one of the biggest challenges for operations teams. It can take several weeks to procure and deploy new storage arrays before they can be consumed by developers. With ScaleIO, an entire datacenter can run on just a handful of standardized server components with no proprietary hardware anywhere. This makes it very easy to manage the growth of resources. Additionally, ScaleIO's built in management makes it very easy to discover, deploy and manage a large ScaleIO system.

- **Availability and Resiliency**: ScaleIO uses a mesh-mirror approach (a many-to-many de-clustered RAID technique) to protect the data from any single point of failure. When a node fails, all the other nodes within the same storage pool work together to rebuild the data in the minimum amount of time possible, with the lowest impact to running applications.

  ScaleIO along with REX-Ray provides high availability for the data by not only replaying the failed container on a new node (thus allowing ScaleIO container storage to be persistent rather than ephemeral) but it also protects against data loss, in case the compute node running the container also contributes to the ScaleIO software-define storage system.

DOCKER ENABLES DEVOPS

Aside from the core container functionality, Docker is also a technology which enables both the developers and the operations team to collaborate in a DevOps environment. Docker provides common toolsets, packaging models and deployment mechanisms which simplify the containerization and distribution of the applications. Docker applications can run anywhere that has containers support.

Docker Provides Developers

- The ability to own the runtime environment of an application. Developers are responsible for maintaining and upgrading the libraries and other dependencies during the development and deployment of an application.

- Isolated containers, so application dependencies won't interfere with each other. Applications can have their own isolated container, so developers can run various versions of libraries and other dependencies for each application.

- The ability to work seamlessly with QA engineers. Developers can write code locally and share their work with the QA engineers using Docker containers. QA engineers can use the Docker to push their applications into a test environment to execute automated and manual tests. Once the bugs are identified, developers can fix them in the development environment and redeploy them to the test environment for testing.

Docker Provides Operations

- Segregation of duties between operations team and developers. Operations teams are responsible for maintaining the infrastructure i.e. operating system, file system, kernel etc. whereas, developers are responsible for the application and their dependencies.

- Significant improvement in the speed and reliability of continuous deployment and continuous integration systems as compared with virtual machines. Docker can streamline the development lifecycle by allowing developers to work in standardized environments using local containers which provide applications and services.

- Because the containers are so lightweight, address significant performance, costs, deployment, and portability issues normally associated with virtual machines.

DOCKER STORAGE

Docker uses an overlay file system to implement a copy-on-write process that stores any updated information to the root file system of a container. These changes are lost if a container is deleted or migrated from the system. A container therefore doesn't have persistent storage; Docker filesystems are temporary by default. You can create, modify or delete files just like any operating system, but once a Docker container is deleted or migrated to a different host, its data will be inaccessible.

The following procedure illustrates the problem:

1. Run a Redis Docker image on node01, save Redis data to a local Docker volume.

2. Remove the Redis image on node01.
3. Re-run the Redis image on node01 to check if its data is accessible.
4. Run the Redis image on node02 to check if the original container's data is accessible.

```
[root@node01 ~]# docker run -d --name redis --volume redis_vol:/data redis
8a25c554ff425602c2d15645a3e9de33d13eb5b45223cc3876bd9fcb98665611
[root@node01 ~]# docker exec -it redis redis-cli
127.0.0.1:6379> set data locally_persistent
OK
127.0.0.1:6379> save
OK
127.0.0.1:6379> exit
[root@node01 ~]# docker stop redis
redis
[root@node01 ~]# docker rm redis
redis

[root@node01 ~]# docker run -d --name redis --volume redis_vol:/data redis
bbe07851de259655c1e588558c4b0a1d5f9140e74e2484405028d08fddce811d
[root@node01 ~]# docker exec -it redis redis-cli
127.0.0.1:6379> get data
"locally_persistent"
127.0.0.1:6379> exit
```

```
[root@node02 ~]# docker run -d --name redis --volume redis_vol:/data redis

55cbd8036240927f3be5cb8b5089ca5a4e5a8693949bf16ac9cdb4c9b627c701

[root@node02 ~]# docker exec -it redis redis-cli

127.0.0.1:6379> get data

(nil)

127.0.0.1:6379> exit

[root@node02 ~]# docker stop redis

redis

[root@node02 ~]# docker rm redis

redis
```

Docker successfully attaches to the existing Docker volume on the same host making the Docker data persistent locally.

Now, we will run a new Redis container on node02 using the same Docker volume.

In the example above, we started a Redis container with redis_vol on node02. The Redis image on node02 failed to get the volume from node01, instead it created a new redis_vol.

LIMITATIONS OF DOCKER STORAGE

Docker containerized storage by default has the following two limitations, which makes it harder to use Docker containers in production:

- **Lack of external storage support**: By default, Docker stores all the volumes in `/var/lib/docker` directory, which can become capacity and performance bottleneck. It is also vulnerable to data loss if the host fails.
- **Data persistency**: Docker data volumes are not globally persistent. If a Docker container is moved from one physical host to another or if the node running Docker container fails, the Docker volume is not persistent.

The challenges of Docker storage can be solved using EMC ScaleIO and {code} REX-Ray, which are discussed in the next section.

SCALEIO VOLUME ORCHESTRATION USING REX-RAY FOR PERSISTENT DOCKER STORAGE

This section describes how REX-Ray orchestrates volume provisions for persistent Docker storage.

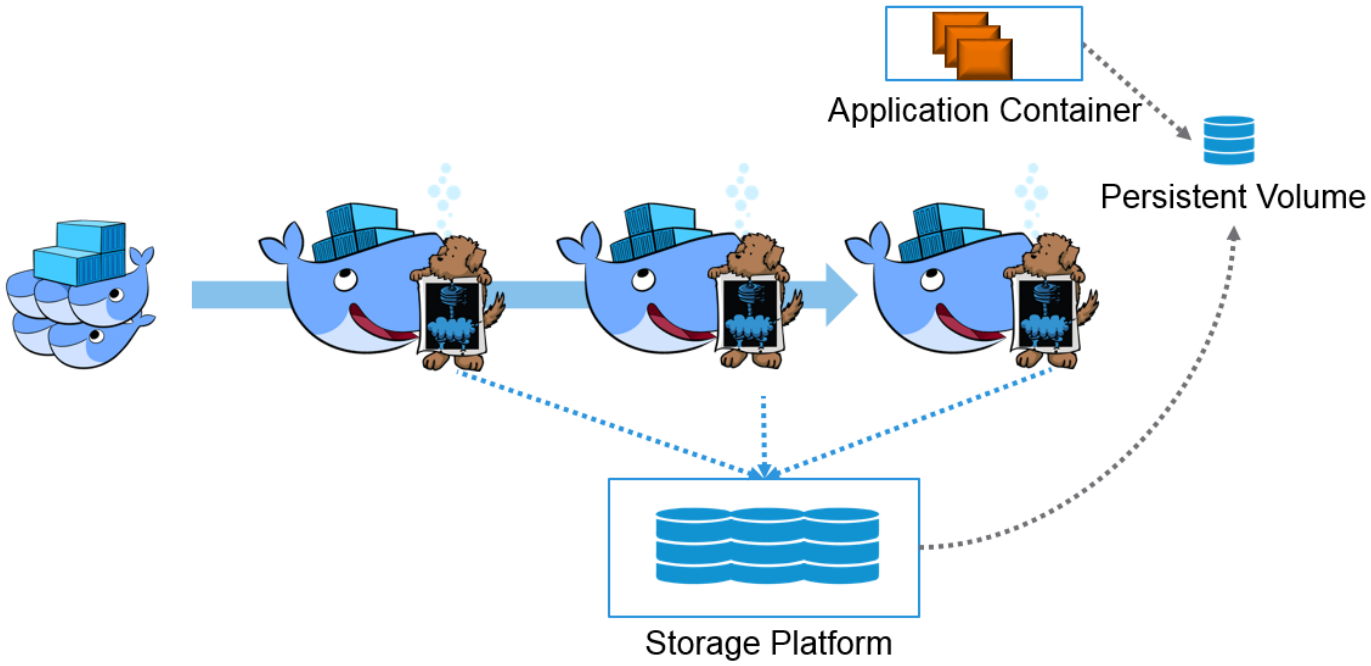
REX-RAY

REX-Ray is a storage orchestration tool that provides a set of common commands for managing multiple storage platforms. Built on top of the libStorage[6] framework, REX-Ray enables persistent storage for container runtimes such as Docker and Mesos.

REX-Ray also provides an easy interface for enabling advanced storage functionality across common storage, virtualization and cloud platforms.

[6] https://github.com/emccode/libstorage

Figure 4) Docker persistent storage using REX-Ray:



REX-Ray Architecture

REX-Ray uses a distributed of client-server model. The client abstracts local host processes (requests for volume attachment, discovery, format, and mounting of devices) while the server provides the necessary abstraction of the control plane for multiple storage platforms. Irrespective of platform, REX-Ray provides the following common functionality:

Cloud Platforms	Storage Platforms	Operating Systems	Container Platform Support

• AWS EC2 (EBS)
• Google Compute Engine
• OpenStack
  o Public Cloud (RackSpace, and others)
  o Private Cloud
	• EMC ScaleIO
	• EMC Isilon
	• Virtual Box
	• EMC XtremeIO
		• CentOS
		• CoreOS
		• Debian
		• Red Hat
		• Ubuntu
			• Docker
			• Mesos
			• Docker + Mesos

For further details, please follow the {code} by Dell EMC - REX-Ray official documentation.

SETUP AND CONFIGURATION

This section will describe the process of setting up ScaleIO and REX-Ray for application persistent storage running on Docker. By using ScaleIO and REX-Ray together, you are able to automatically provision and manage ScaleIO volumes to the container. Application containers accessing ScaleIO storage can move around the infrastructure, while REX-Ray will manage the data mapping from the original location to the destination without any need to provision ScaleIO volumes.

To demonstrate how REX-Ray and ScaleIO together deliver for persistent Docker storage, we will be using the following lab setup.

Table 2) Demo environment:

Operating System	CentOS
ScaleIO	ScaleIO Version: 2.0.1
Docker	Docker version - 1.12
REX-Ray	REX-Ray version – 0.6.0

To use REX-Ray with ScaleIO for persistent Docker storage, the following requirements should be met:

- The ScaleIO REST Gateway is required for the REX-Ray driver to function. To configure the ScaleIO-gateway, follow the steps below
    - Install `EMC-ScaleIO-gateway package`.
    - Edit the `/opt/emc/scaleio/gateway/webapps/ROOT/WEB-INF/classes/gatewayUser.properties` file and append the MDM IP addresses to the following `mdm.ip.addresses= parameter`.
    - By default, the password is the same as your administrative MDM password.
    - Start the gateway `service scaleio-gateway start`.
- The `libStorage` client or application that embeds the `libStorage` client must reside on a host that has the SDC client installed. The command `/opt/emc/scaleio/sdc/bin/drv_cfg --query_guid` should be executable and should return the local SDC GUID.
- The official Oracle Java Runtime Environment (JRE) is required. During testing, use of the Open Java Development Kit (JDK) resulted in unexpected errors.

Note: You can also install the ScaleIO Data Server (SDS) and ScaleIO Metadata Manager (MDM) on the same physical host.

To download the most recent stable version of Rex-Ray and install it to `/usr/bin/rexray` on Linux systems, execute the following command:

```
curl -sSL https://dl.bintray.com/emccode/rexray/install | sh
```

REX-Ray will be registered as either a SystemD or SystemV service depending upon the Operating System. For installing specific REX-Ray version, refer to http://rexray.readthedocs.io/en/stable/user-guide/installation/#installation.

Once you have installed the REX-Ray, it requires a configuration file for storing details used to communicate with storage providers. This can include authentication credentials and driver-specific configuration options. Refer to the libStorage Storage Providers documentation for sample configurations of all supported storage platforms. We will focus on the configuration file for ScaleIO.

Create a configuration file on the host at /etc/rexray/config.yml. Here is an example of a configuration file for REX-Ray version 0.6.0:

```
libstorage:
  service: scaleio
  integration:
    volume:
      operations:
        mount:
          preempt: true
scaleio:
  endpoint: https://gateway_ip/api
  insecure: true
  userName: admin
  password: <password>
  systemID: 0
  protectionDomainName: pd1
  storagePoolName: hdd
```

where,

- The `apiVersion` can optionally be set here to force certain API behavior. The default is to retrieve the endpoint API, and pass this version during calls.
- `insecure` should be set to true if you have not loaded the SSL certificates on the host.
- `protectionDomainName` tells REX-Ray the name of the ScaleIO Protection Domain. This parameter can be replaced by `protectionDomainID` which takes priority over `protectionDomainName`.
- `storagePoolName` tells REX-Ray the name of the ScaleIO Storage Pool to be used to create a volume. This parameter can be replaced by `storagePoolID` which takes priority over `storagePoolName`.
- `thinOrThick` determines whether to provision the ScaleIO volume as the default `ThinProvisioned`, or `ThickProvisioned`.

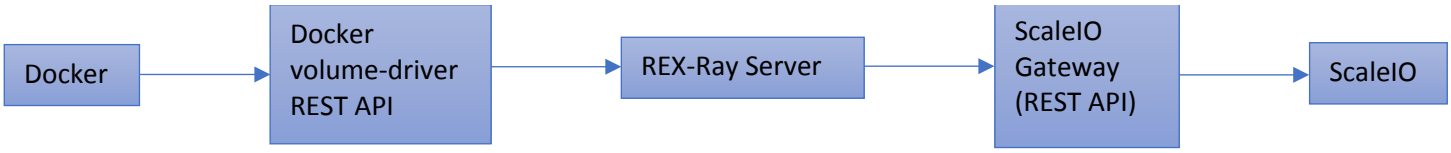
Note: Make sure the ScaleIO driver is activated. The driver enabled `libStorage` to manage storage on the underlying operating system. If the `libStorage` is not activates, REX-Ray will not work as expected.

To start REX-Ray as a service, execute the following command:

- `rexray service start`

To use globally persistent storage, run the Docker application using `--volume-driver=rexray`. Docker VolumeDriver REST API will call the REX-Ray server. The REX-Ray server then check its configuration file to communicate with the ScaleIO Gateway; the ScaleIO Gateway in turn calls the ScaleIO REST API to manage ScaleIO storage.

Figure 5) Docker, REX-Ray and ScaleIO API calls:



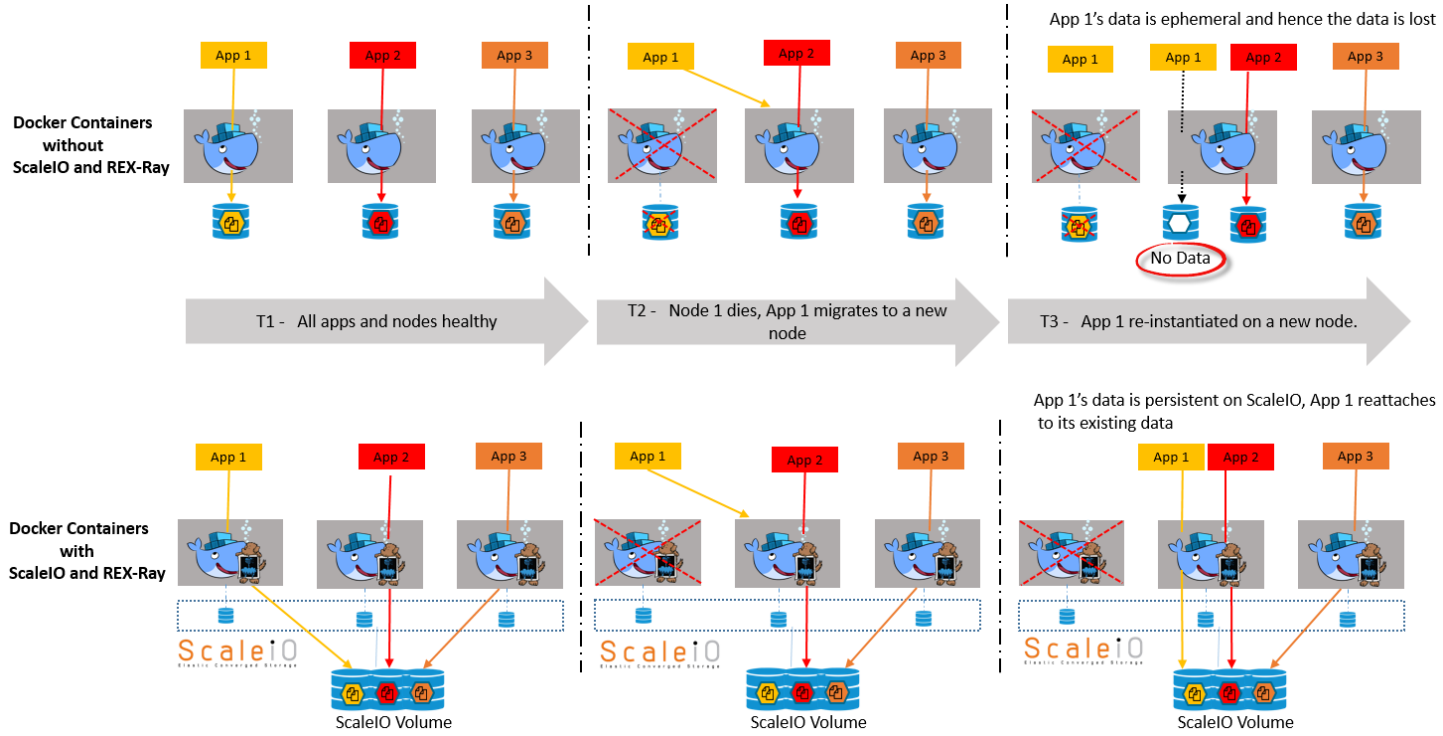
For more details, please follow REX-Ray User Guide.[7]

DEMONSTRATION

This section will demonstrate how REX-Ray automatically creates and provisions ScaleIO volumes to the containers for the persistent storage. ScaleIO and REX-Ray makes the Docker data globally persistent by storing it on the ScaleIO volumes. Docker data is accessible even if a host fails or if a Docker container is migrated to a different host.

Figure 6 below illustrates how Docker data is stored natively versus Docker data with ScaleIO and REX-Ray.

Figure 6) Docker containers with and without ScaleIO and REX-Ray:



We will perform the following steps to test the persistent Docker storage and orchestration.

1. Start a Redis image on node01 using –volume-driver=rexray
2. Save Redis data in a volume named scaleio_redis_vol
3. Remove the Redis image from node01.

[7] http://libstorage.readthedocs.io/en/stable/user-guide/storage-providers/#scaleio

4. Re-run the Redis image on node02 using the existing volume - scaleio_redis_vol.

```
[root@node01 ~]# docker run -d --name redis --volume-driver rexray --volume
scaleio_redis_vol:/data redis

2b148a355eb8b3799fb4e1d33c310853b8bcebf7451b4a5608d9d6fb167a8b06

[root@node01 ~]# docker exec -it redis redis-cli

127.0.0.1:6379> set data globally_persistent

OK

127.0.0.1:6379> save

OK

127.0.0.1:6379> exit

[root@node01 ~]# docker stop redis

redis

[root@node01 ~]# docker rm redis

redis
```

In the example above, REX-Ray performs the following two operations in the background:

- Create a new ScaleIO volume called `scaleio_redis_vol`

```
ScaleIO-10-108-161-84:~ # scli --query_all_volumes

Query-all-volumes returned 1 volumes

Protection Domain ee7c9f7300000000 Name: pd1

Storage Pool f2521f6900000000 Name: sp1

Volume ID: dd99649700000000 Name: scaleio_redis_vol Size: 16.0 GB (81920 MB) Mapped 1 SDC Thick-
provisioned
```

- Map the ScaleIO volume to the host node01.

```
[root@node02 ~]# docker run -d --name redis --volume-driver rexray --volume
scaleio_redis_vol:/data redis

f1aeee4e2bef9259e4bb0b1b2c9198065276b4d8ef5d30576ec90b6e63972632

[root@node02 ~]# docker exec -it redis redis-cli

127.0.0.1:6379> get data

"globally_persistent"

127.0.0.1:6379> exit
```

Now, we will start the Redis image on node02 using the existing `scaleio_redis_vol` volume

In the example above, we started a Redis container with `scaleio_redis_vol` on node02. The Redis image on node02 successfully attaches to the existing volume, making Docker data persistent across the nodes.

As we can see in the example above,

5. We created an Redis Docker image on host node01 using `–volume-driver=rexray`
6. Created a Docker volume - `scaleio_redis_vol`
7. Exit the Redis container
8. Created a new Redis container on host node02 using the same REX-Ray volume.
9. The data is persistent across containers and the Redis data created on node01 has successfully attaches to the existing volume on node02

Table 3 summarizes the advantages of ScaleIO and REX-Ray with Docker containers.

Table 3) Advantages of ScaleIO and REX-Ray with Docker containers

	Docker Containers without ScaleIO and REX-Ray	**Docker Containers with ScaleIO and REX-Ray**
Persistent Docker data	Container data is only locally persistent. Container's data is stored in /var/lib/docker directory of each host. If the host fails, the data is inaccessible.	Container data is stored on ScaleIO volumes, making it globally persistent.
Containers migration	Docker container lose access to existing data when started a new host,	Docker containers can migrate from one host to another without losing access to data.
Failover support	If a Docker container failover to a different host, the Docker data is inaccessible.	Container data is accessible if a Docker container failover to a different host.

Containers with Persistent Storage Support the Complete Application Lifecycle

As we have seen in the demonstration above, the container data is not globally persistent. If a host fails or if a Docker container migrates to a different host, the data is inaccessible. Developers and QA engineers use containers without persistent storage to edit or test their code. Since data is not persistent, every time engineers deploy a new container, they have to copy the code and the database to either edit or test it. Copying the data is not only time consuming but it also prevents QA engineers from testing the code for scalability and performance as they are limited by the constraints of a single compute node and its local storage.

Also, operations teams need to provide an infrastructure with persistent external storage to deploy applications for production use. This mandates additional testing in the production environment, which further delays delivery of the software.

ScaleIO along with REX-Ray provides a solution to the problems described above:

- Using ScaleIO and REX-Ray provides persistent external storage for Docker containers. Developers can deploy a new container, map to existing data and start to edit or test code. They spend more time editing and testing code rather than copying or reloading the data.

- QA engineers are not limited to the storage capacity or performance of a single compute node and they can test the application for scalability and performance seamlessly with ScaleIO. Also, instead of creating a new copy, QA engineers can use ScaleIO snapshots to create a writable copy of the existing production database to test the application.

- ScaleIO provides a single unified platform for developers and QA engineers to edit/test their code and also to deploy it in production (with quality of service controls). This allows the dev/test team to perform pre-production testing of their code again in a production-scale environment, which saves time and helps find bugs related to performance and scalability before the code enters production.

CONCLUSION

Applications are evolving from a rigid monolithic architecture to a more agile microservices based architecture. This new model of development brings new challenges to the operations team. They have to meet the needs of the developers by meeting growing infrastructure requirements, be it servers or storage.

Containers have emerged as a solution to provide microservices based infrastructure. However, the workloads running inside the containers are stateless and ephemeral. With the progression of container platforms from Mesos and Docker, stateful applications are run inside of containers, but these solutions have their own set of storage challenges.

ScaleIO, which is an enterprise grade software-defined storage solution, enables the operations team to rapidly meet the storage needs of a modern DevOps team, and overcomes the shortcomings of traditional SAN infrastructure.

REX-Ray is a storage orchestration tool that delivers persistent storage access for container runtimes, such as Docker and Mesos, and provides an easy interface for enabling advanced storage functionality across common storage, virtualization and cloud platforms.

Together, ScaleIO and REX-Ray enable DevOps by providing persistent external storage for containers which quickly and easily scales to meet unplanned storage needs on the fly. To see Docker, REX-Ray and ScaleIO in action, login to portal.emcdemo.com and launch the ‘Docker, Mesos, and ScaleIO for your Persistent Applications’ virtual lab.

REFERENCES

ScaleIO User Guide: ScaleIO User Guide

{code} by Dell EMC: {code} by Dell EMC

{code} Slack Channel: https://codecommunity.slack.com/

Container: Container

Docker: Docker

CloudScaling: http://cloudscaling.com/blog/cloud-computing/will-containers-replace-hypervisors-almost-certainly/

List of tables and figures

Table 1) Virtual machines vs. containers 7

Table 2) Demo environment 14

Table 3) Advantages of ScaleIO and REX-Ray with Docker containers 18

Figure 1) Paradigm shift in application development 5

Figure 2) Virtual machines vs. containers 7

Figure 3) Docker infrastructure 8

Figure 4) Docker persistent storage using REX-Ray 13

Figure 5) Docker, REX-Ray and ScaleIO API calls 16

Figure 6) Docker containers with and without ScaleIO and REX-Ray 16