DELL EMC

DEPLOY APACHE MESOS WITH SCALEIO AND REX-RAY FOR STATEFUL APPLICATIONS

ABSTRACT

This white paper describes the need for Apache Mesos in the modern data centers and explains how to deploy Mesos with Dell EMC™ ScaleIO® with REX-Ray™ for stateful applications.

November, 2016

TABLE OF CONTENTS

INTRODUCTION

The standard architecture and deployment strategy of contemporary applications is steadily becoming one based on microservices that run as an independent services comprised of multiple code modules and 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. As organizations are moving towards microservices and cloud-based infrastructure, they are trying to encapsulate the tasks associated with deploying and managing this architecture in higher- and higher-level abstractions.

Legacy IT systems are still running consolidated virtual machines (VMs) that suffer from inefficient resource utilization. Commonly, servers are either under- or over-utilized. VMs, being heavyweight, do not enable easy deployment and scaling of microservices-based applications. Containers, by contrast, offer an isolated process space, with managed deployments and automatic scaling.

Apache Mesos is an open source project which provides an abstraction layer for compute resources (CPU, RAM, Disk, Ports, etc.). This allows applications and developers to request arbitrary units of compute power without an IT provider having to worry about how this translates to bare-metal or VMs. Mesos uses Linux's control groups (cgroups) and containers to ensure that processes are isolated and are only allowed to consume a set amount of resources. However, a container's data is by default not globally persistent, and so extra work is needed for data persistence and data migration across different hosts inside a cluster.

This white paper describes how REX-Ray and ScaleIO provide persistent storage to Mesos applications using Redis (a key/value datastore used in modern applications) as an example.

ScaleIO, an enterprise-grade, scalable, hardware agnostic and high-performance software-defined storage solution, is a perfect match to enable an operations team to meet the storage needs of developers on the fly.

REX-Ray is a 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.

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 architecture, please refer to Dell EMC ScaleIO: Software Defined Server SAN – Architecture Matters white paper.
- Relationship between Docker containers, REX-Ray and ScaleIO. For better understanding of how REX-Ray and ScaleIO provides persistent storage for Docker containers, please refer to white paper Deploy your container with ScaleIO.

INTRODUCTION TO MESOS

Apache Mesos is an open source project, which abstracts CPU, memory, storage, and other compute resources away from machines (physical or virtual), enabling fault-tolerant and elastic distributed systems to easily be built and run effectively. Mesos is built using the same principles as the Linux kernel, only at a different level of abstraction. The Mesos kernel runs on every machine and provides applications (e.g., Hadoop, Spark, Kafka, Elastic Search) with APIs for resource management and scheduling across entire data center and cloud environments.

Mesos is a cluster manager, handling workloads in a distributed environment. Mesos introduces a distributed two-level scheduling mechanism called *resource offers*. Mesos decides how many resources to offer each framework, while frameworks decide which resources to accept and which computations to run on them. All the resources that exist on the machines in the cluster will be put in a single pool.

Mesos is a thin resource sharing layer that enables fine-grained sharing across diverse cluster computing frameworks, by giving frameworks a common interface for accessing cluster resources. The idea is to deploy multiple distributed systems to a shared pool of nodes in order to increase resource utilization.

REASONS TO USE APACHE MESOS IN A DATA CENTER

The goal of Apache Mesos is to abstract the compute resources (CPU, Memory, Disks etc.) from individual servers and present them as a single entity. This enables developers to request arbitrary units of compute resources without the need for a detailed or time-consuming provisioning process. A key design criteria of Apache Mesos, which differentiates it from other schedulers, is its two-level scheduler architecture. With a dual-level architecture, Mesos handles low level infrastructure scheduling operations while another layer (Frameworks) on top of Mesos handles all the application specific operations and logic. The two-layer architecture of Mesos has the following advantages over other schedulers:

- **Multiple workload support:** Mesos can support different types of distributed workloads such as Mesos and Docker containers, analytics (spark), big-data technologies (Kafka, Cassandra) etc. running on the same basic resource pool.
- **Scalability**: By having Mesos handle low-level infrastructure logic and delegating the application-specific logic to the framework, Mesos enables both horizontal and vertical scaling of the tasks and has ability to scale up to 10,000s of nodes.
- **Support for other distributed technologies**: Mesos supports multiple frameworks such as Kubernetes, which can run on top of Mesos. As new distributed technologies are being introduced, organizations can adapt to any such applications or frameworks and run it with Mesos. This makes Mesos future-proof platform for distributed technologies.

Some of the other benefits of having Mesos in a data center are:

- **Resiliency**: If a task is lost due to a failure or a crash of a node, Mesos informs the framework of the inconsistent state and the framework re-instantiate the task on some other node.
- **Utilization**: Mesos is a thin resource sharing layer that enables fine-grained sharing across diverse cluster computing frameworks, by giving frameworks a common, fine-grained interface for accessing cluster resources. This results in better utilization of the compute resources.
- **Testability**: Once an application is developed, the next task is to test it in the production environment. Testing at scale is time consuming as an application needs to be deployed on large number of servers. Since, Mesos presents application as a framework which enables automated scaling, testing an application as a whole becomes easier.
- **Policy based resource utilization**: Mesos enables developers with an option of policy based resource utilization. By using Mesos authorization and roles, it is possible to control how resources will be utilized.
- **Support for containers**: Mesos supports existing container technologies like Docker and it also provides its own container technology. Mesos implements the following containerizers:
    - Composing (note: this feature allows multiple container technologies to play together)
    - Docker
    - Mesos (default)

APACHE MESOS ARCHITECTURE

As described above, Apache Mesos has a two-layer architecture where Mesos consists of a master process that manages slave daemons running on each cluster node, and framework that runs tasks on these slaves. It has the following components:

- **Mesos Master**: Mesos master is a process which runs on a node in the cluster and orchestrates the running of tasks on slaves by receiving resource offers from slaves and offering those resources to registered frameworks, such as Mesosphere Marathon. The masters work to coordinate and manage the slave daemons. It's the master that decides how many resources it has to offer to the framework. A master knows how much resources it can offer by tracking the amount that the slaves report to have freely available. A master is a singular entity that coordinates the work, there are however multiple masters in a high availability set up. Only one master should be leader at any time, with Zookeeper is being used for leader elections.

- **Mesos Slave**: Mesos slave is a process which runs on a node in the cluster and offers up resources available on that node to the Mesos Master. The slave also takes schedule requests from the master and invokes an executor to launch a task. It is expected that only one slave is running on a node.

- **Frameworks:** Framework is responsible for running the tasks on the Mesos slaves. Mesos master decides how many resources need to be allocated to each framework. Resource allocation is based on policies such as fair sharing, priorities, etc. Each framework consists of two components:

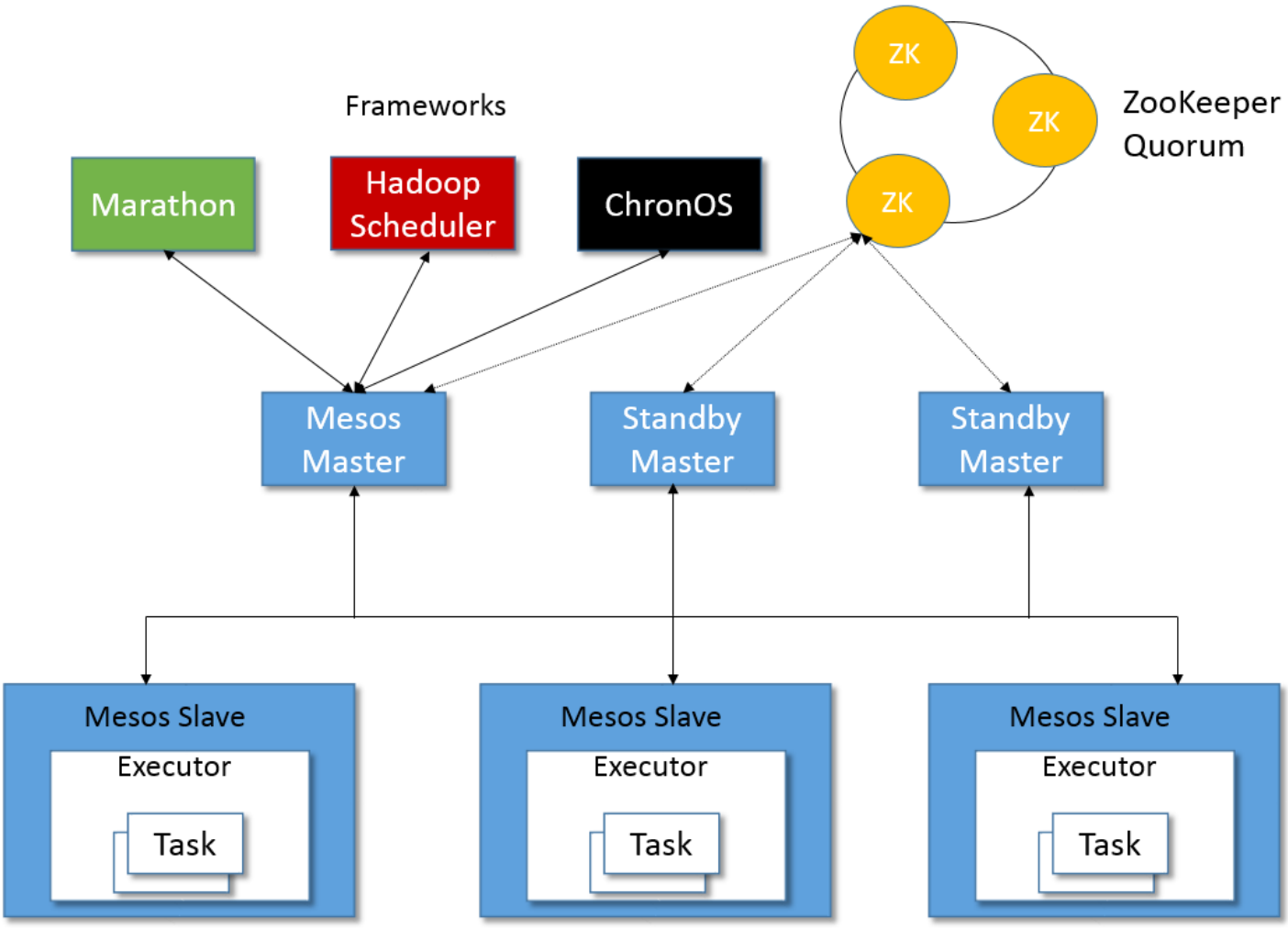
    - Scheduler: A scheduler registers with Mesos master to be offered resources

    - Executor: Executor is responsible for launching jobs on Mesos slave nodes.

  There are number of frameworks which can run on top of Apache Mesos such as Marathon (long running services), Chronos (batch scheduling), Hadoop (big data processing), Cassandra (distributed database) etc.

- **Zookeeper**: Zookeeper is a centralized service for maintaining configuration information, naming, providing distributed synchronization, and providing group services.

Figure 1) Apache Mesos architecture

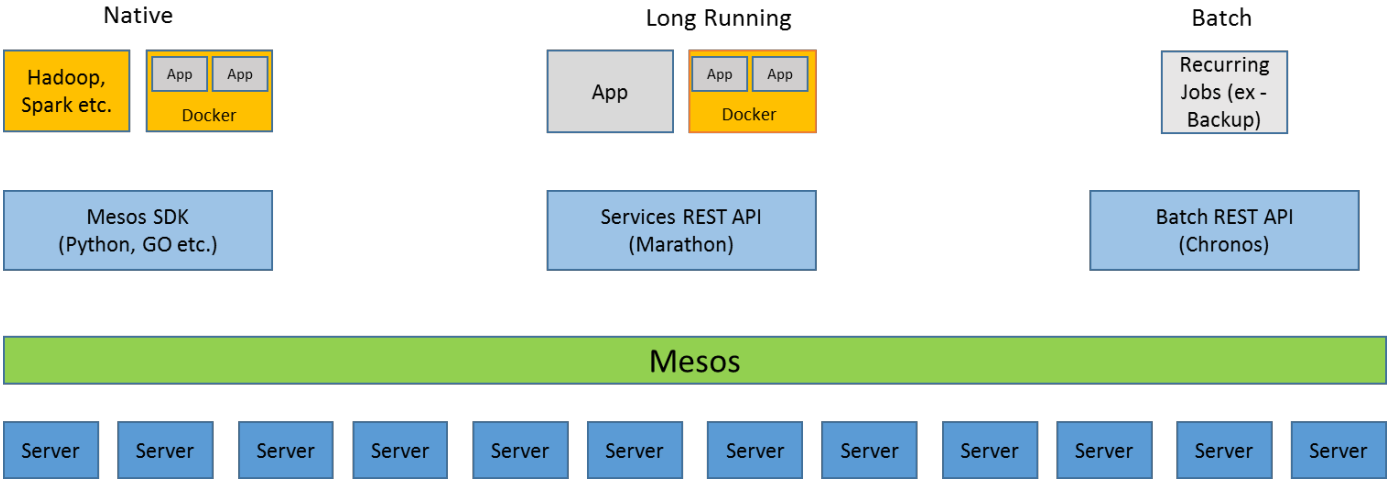


In the architecture diagram, the following occurs:

- Framework (such as Marathon, Hadoop and Chronos) wants to run a task on the cluster.
- Framework will send a request to the Master which in turn will get the resource information from the slaves.
- The master will then send an offer to framework and allow it to run a task on the available slaves.
- Framework may accept or reject the offer. If framework accepts the offer, it sends a list of tasks it wants to run on the Mesos cluster.
- Mesos slaves will execute the tasks in the list received from the master.

Mesos supports existing container technologies like Docker and it also provides its own container technology. Figure 3 below illustrates a typical data center with a Mesos deployment.

Figure 1) Data center with Mesos deployment



WHY USE SCALEIO AND REX-RAY WITH MESOS

Containers use the root file system of a container to implement a copy-on-write process that stores any updated information. These changes are lost if the container is deleted or migrated from the system. Therefore, containers don't have persistent storage. You can create, modify or delete files just like any operating system, but once the container is deleted or migrated to a different host, data residing only in the container will be inaccessible.

Container storage by default has two limitations likely to limit its usefulness in production:

- **Lack of external storage support**. By default, containers store all the volumes on the local storage of the node, which can become a capacity and performance bottleneck. It is also vulnerable to data loss if the node fails.
- **Data persistency**. Container data is not globally persistent. If a container is moved from one physical host to another or if the node running container fails, the container data is not persistent.

PERSISTENT EXTERNAL STORAGE FOR CONTAINERS USING REX-RAY

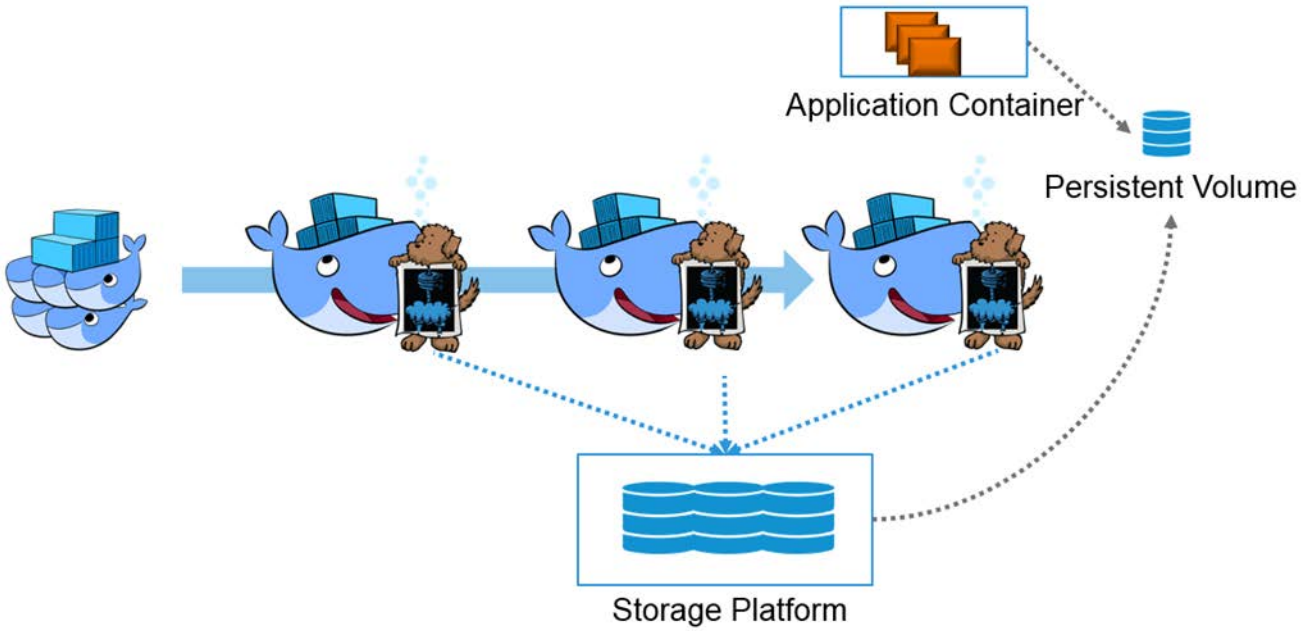
REX-Ray is a storage orchestration tool that provides a set of common commands for managing multiple storage platforms. Built on top of the libStorage[1] framework, REX-Ray enables persistent storage for container runtimes such as Docker and Mesos.

REX-Ray 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.

REX-Ray

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

Figure 2) REX-Ray architecture



REX-Ray uses a distributed 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	• Dell EMC ScaleIO • Dell EMC XtremeIO® • Dell EMC Isilon® • Virtual Box	• CentOS • CoreOS • Debian • RedHat • Ubuntu	• Docker • Mesos • Docker + Mesos

For further details, please follow the {code} by Dell EMC REX-Ray official documentation. For more understanding on how REX-Ray works, consult the white paper Deploy your containers with ScaleIO.

REASONS TO USE SCALEIO WITH APACHE MESOS

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

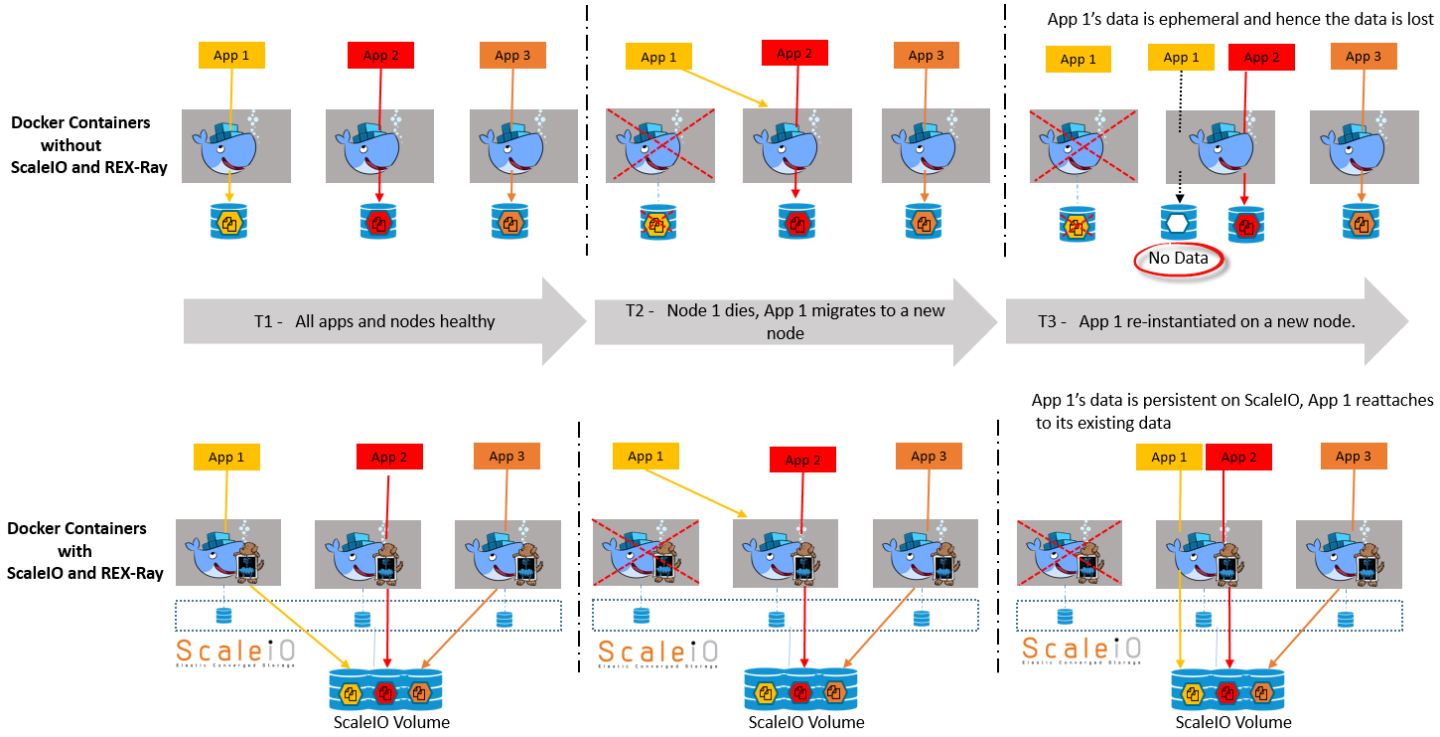
Also, operations teams need to provide an infrastructure with persistent external storage to deploy applications for production use. This mandates additional testing in the production environment, which may further extend software delivery dates.

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

- ScaleIO and REX-Ray provides persistent external storage for Docker containers. Developers can deploy a new container, map to the 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 an 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 an 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.

The diagram below illustrates how container data, using Docker containers as an example, is being stored with and without ScaleIO and REX-Ray.

Figure 3) Docker container with and without ScaleIO and REX-Ray



ScaleIO also have the following advantages which makes it a perfect fit for a microservices style containers based application development:

- **Lightweight**: ScaleIO components i.e. (ScaleIO Data Server, ScaleIO Data Client and Metadata Manager) are very lightweight and use only a small amount of CPU and memory resources. This leaves plenty of resources for an application to use and run in parallel with ScaleIO.

- **Automation**: There are many tools released by {code} by Dell EMC[2] team to automate the deployment and management of ScaleIO, enabling its use in a DevOps environment. ScaleIO has integration with REX-Ray[3], a service to orchestrate ScaleIO volume provisioning and delivery of persistent storage access for container runtimes, such as Docker and Mesos. There are tools for installing and configuring ScaleIO components such as Puppet for ScaleIO[4] and Ansible for ScaleIO[5].

- **Grow as you need:** One of the biggest challenges for the operations team is to plan ahead for the storage resources. However, more often than not, the operations team either overestimates or underestimates the resources required for the development purposes. Current SAN deployments cannot address this issue as the infrastructure may consist of one or more storage arrays and set of compute nodes resulting in infrastructure silos. As 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 can scale on the fly to any size in increments of one node. This lessens the burden of planning on the operations team and saves both time (to provision storage) and money (to acquire siloed and inaccessible storage). 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. You simply remove and replace individual servers over time as they need to be replaced/upgraded and ScaleIO takes care of the rest.

- **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 hyper-converged infrastructure (HCI) storage solutions, but they have very strict requirements. They are not infrastructure agnostic and cannot support heterogeneous operating systems or hypervisors. Most of the applications and more so the recent Platform 3 applications need to support a variety of operating systems.

  Using ScaleIO, the operations team can maintain a single ScaleIO cluster to host variety of applications running on heterogeneous operating systems.

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

- **Manageability**: Management and deployment of the storage servers is one of the biggest challenges for the operations teams. It can take several weeks to deploy new storage servers before they can be assigned to the developers. With ScaleIO, an entire data center 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, with ScaleIO Advanced Management Server (AMS), it is very easy to discover, deploy and manage a large ScaleIO system.

- **Availability and Resiliency**: ScaleIO uses mesh-mirror approach which is 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.

  ScaleIO along with REX-Ray provides high availability for the data by not only replaying the failed container on a new node but it also protects against the data loss, in case the node running the container also contributes to the ScaleIO storage.

---

[2] http://emccode.com/

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

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

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

DEMONSTRATION

In this section, we will deploy a Docker container on Mesos using Marathon framework. We will use ScaleIO and REX-Ray to provide persistent storage for the Docker containers.

To demonstrate how REX-Ray and ScaleIO together delivers persistent storage for the containers deployed via Mesos, we will be using the following setup:

Table 1) Demo environment

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

Table 2) Mesos services

Nodes / **Services**	**Node – 1 (62AC-1)**	**Node – 2 (62AC-2)**	**Node – 3 (62AC-3)**	**Node – 4 (62AC-4)**
Mesos Master	Yes	Yes	Yes	No
Mesos Slave	Yes	Yes	Yes	Yes
Zookeeper	Yes	Yes	Yes	No

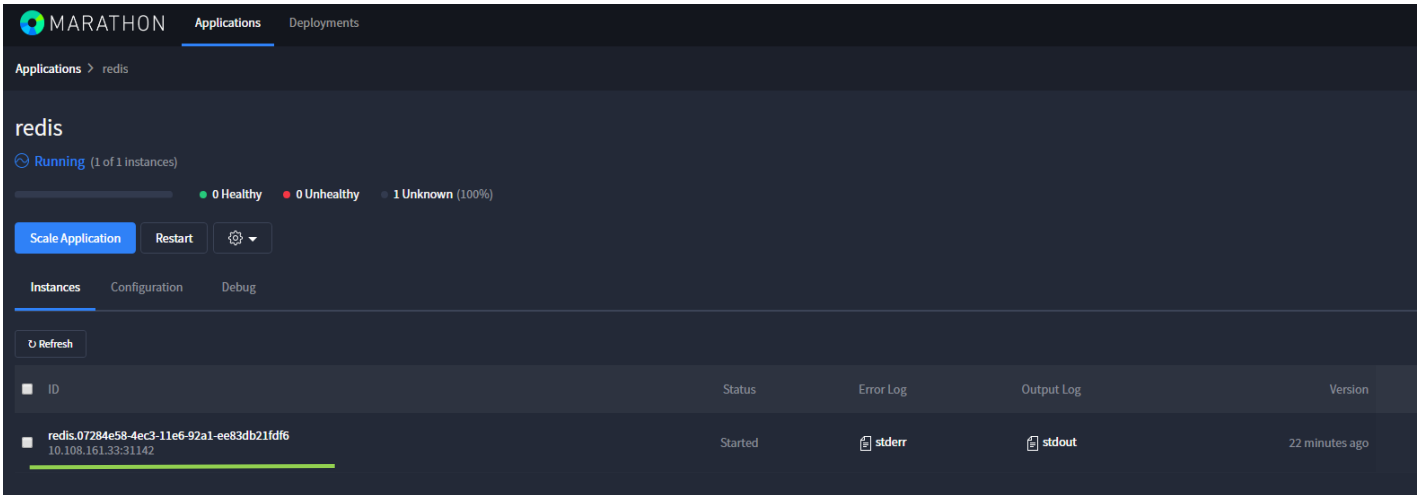
First, we will deploy a Docker container on Marathon using REX-Ray volume driver.

1. The name of this container is called Redis and it's using the official Redis image from Docker Hub.
2. Redis uses port 6379 for communication and it's being exposed out of the host via TCP. In addition, there are new parameters for storing data.
3. The volume-driver being used is REX-Ray
4. The volume value is showing that the persisted volume named redis-data will be mapped to the container's data folder.

The following configuration file is used to deploy Redis image on Marathon:

```
{
    "id": "redis",
    "container": {
      "docker": {
        "image": "redis",
        "network": "BRIDGE",
        "portMappings": [
          { "containerPort": 6379, "hostPort": 0, "protocol": "tcp"}
        ],
        "parameters": [
            { "key": "volume-driver", "value": "REX-Ray" },
            { "key": "volume", "value": "redis-data:/data" }
        ]
      }
    },
    "args": ["redis-server", "--appendonly", "yes"],
    "cpus": 0.2,
    "mem": 32.0,
    "instances": 1
}
```

To deploy the image on Marathon, we are going to invoke a POST request to the Marathon API. Once the image is deployed, click on the Redis image application from the dashboard to see the detailed view.



The image is deployed on node `10.108.161.33`. This will create a Docker volume called redis-data with REX-Ray as volume driver.

```
root@62AC-5:~# docker ps

CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS
PORTS                     NAMES

37a0e2272433        redis               "docker-entrypoint.sh"   8 minutes ago       Up 8
minutes        0.0.0.0:31142->6379/tcp   mesos-23f49112-39ab-4310-9117-125631763354-
S0.44aa3600-93c2-4d01-8f0d-9267b5c97617

e1097612f63d        ubuntu              "/bin/bash"              11 days ago         Up 11
days                                    container02

root@62AC-5:~# docker volume ls

DRIVER              VOLUME NAME

REX-Ray              docker_vol

REX-Ray              persistency_test

REX-Ray              redis-data
```

```
root@62AC-1:~# scli --query_all_volumes

Query-all-volumes returned 3 volumes

Protection Domain b9885bb700000000 Name: pd1

Storage Pool 048ab3c100000000 Name: hdd

 Volume ID: 03d464b900000000 Name: docker_vol Size: 16.0 GB (16384 MB) Mapped to 1 SDC Thin-provisioned

 Volume ID: 03d464ba00000001 Name: persistency_test Size: 16.0 GB (16384 MB) Unmapped Thin-provisioned

 Volume ID: 03d464bb00000002 Name: redis-data Size: 16.0 GB (16384 MB) Mapped to 1 SDC Thin-provisioned
```

This will also create a ScaleIO volume and map it to a host.

```
root@62AC-5:~# docker run -ti --rm prologic/redis-cli -h 10.108.161.33 -p 31142

10.108.161.33:31142> set data1 ScaleIO

OK

10.108.161.33:31142> set data2 REX-Ray

OK

10.108.161.33:31142> set data3 Docker

OK

10.108.161.33:31142> save

OK

10.108.161.33:31142> exit
```

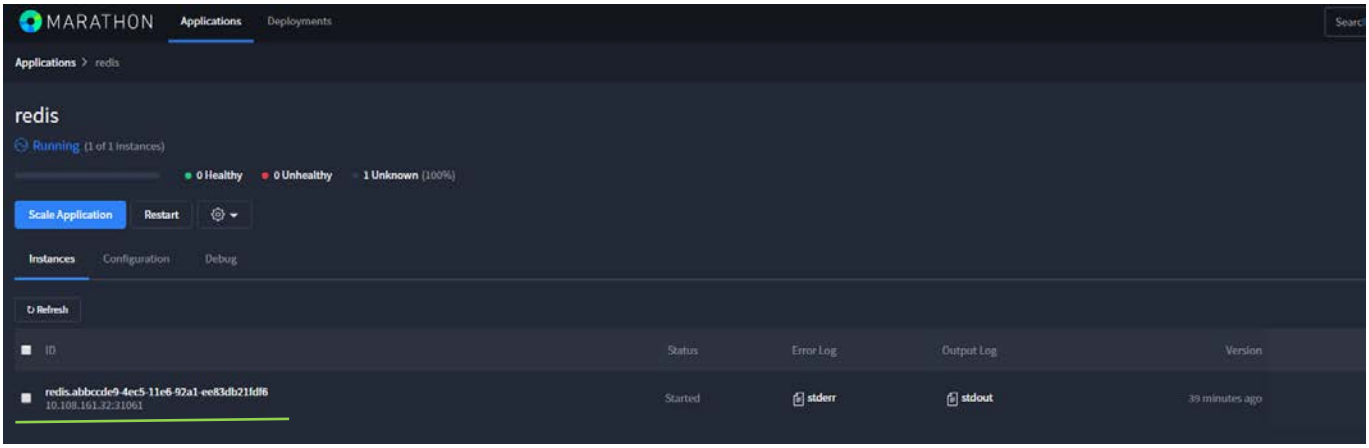
Connect to Redis using `redis-cli` container and create a test database.

To test the data persistency, we will stop the redis container on the host where the Redis image is running. This will force Marathon to deploy the Redis application on some other host.

```
root@62AC-5:~# docker stop 37a0e2272433

root@62AC-5:~#
```

Marathon, re-deployed the `redis` application on `10.108.161.32`



To test the data persistency, we will launch the host (10.108.161.32) on which Mesos re-deployed the Redis Docker image. As shown

```
root@62AC-4:~# docker ps

CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS
PORTS                     NAMES

41dcfbcc69ed        redis               "docker-entrypoint.sh"   16 minutes ago      Up 16
minutes       0.0.0.0:31061->6379/tcp   mesos-23f49112-39ab-4310-9117-125631763354-S3.aedafb68-
1a3d-4c7a-a9ff-7f67abfd78e0

root@62AC-4:~# docker run -ti --rm prologic/redis-cli -h 10.108.161.32 -p 31061

10.108.161.32:31061> get data1

"ScaleIO"

10.108.161.32:31061> get data2

"REX-Ray"

10.108.161.32:31061> get data3

"Docker"

10.108.161.32:31061>
```

in the figure 5, Container's data stored on ScaleIO using REX-Ray volume driver, should be able to map to the existing volume.

As we can see in the demonstration above, the data is persistent across Docker containers running on Mesos. The Redis image is deployed on a new server which successfully attaches to the existing volume, making the Redis application stateful.

Conclusion

In many cases, existing IT infrastructure is not able to address challenges faced by an operations team. These teams must meet the needs of the developers by responding to growing infrastructure requirements, whether in compute, storage, or network.

With the help of container executors (Docker, Mesos etc.) for Mesos, Mesos can run and manage containers in conjunction with Chronos and Marathon frameworks. Containers provide a consistent, compact and flexible means of packaging application builds. 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, an enterprise-grade software defined storage solution, enables the operations team to meet the storage needs on the fly and overcomes the shortcomings of the SAN infrastructure. ScaleIO is lightweight, infrastructure agnostic, and uses a de-centralized approach of provisioning storage resources which leads to more efficient, more granular, and more balanced use of those resources.

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 solution enable DevOps by providing persistent external storage for containers which can also fulfill the unplanned storage needs of the fly. To see Docker, REX-Ray and ScaleIO in action, login to portal.emcdemo.com and launch '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