VMware, Inc
World Headquarters
3401 Hillview Ave, Palo Alto, CA 9430
Tel: 1-877-486-927
http://www.vmware.com

Containerized SRAs:

Amendments to the SRA 2.0 specification

VMware Confidential

Author: Vladimir Penev

Last Updated Date: 9/25/2019

Revision history 3
Introduction 4
Requirements of containerized SRA 5
Managing SRA Configuration 7
Running an SRA Container 9
Detailed Running Procedure 11

Revision history

Date	Change description
10/15/2018	Initial draft
03/08/2019	Initial version
03/18/2019	Modified “Managing SRA Configuration” about read/write permissions of /srm/sra/conf
05/03/2019	Modified “Managing SRA Configuration” adding couple remarks about /srm/sra/conf
05/20/2019	Modified “Requirements of containerized SRA” adding the requirements about file format of SRA package
09/20/2019	Modified “Requirements of containerized SRA” adding sample commands for SRA Dockerfile to provide read/write access to /srm/sra/conf in the container

Introduction

This document contains the amendments to the Storage Replication Adapter (SRA) 2.0 interface specification which allow VMware Site Recovery Manager (SRM) to isolate the environment of running vendor specific SRA from its environment of run. It has to unify the way of the communication between SRM and the software in the container which implements the adapter.

When SRA is containerized it gives a freedom of implementation – used platform, tools and their versions.

Requirements of containerized SRA

The Site Recovery Manager virtual appliance uses docker images for the implementation of SRA functionality. The installation files must meet specific packaging requirements - they must be either a docker image archive in a `.tar` file format, or a compressed `.tar.gz` file for better/faster upload/load operations.

For SRM to use an SRA docker image, the image must meet a set of requirements which fall broadly into 2 categories:

a) image metadata requirements;
b) SRA implementation requirements.

The metadata requirement for an SRA image is to have a specific label with the key `com.vmware.srm.sra` embedded in it (see https://docs.docker.com/config/labels-custom-metadata/). The value of the key is not important as long as the key is present. SRM will use this key to filter the SRA images present in the SRM VA. Having this key is an easy way to identify the SRA images in case there are other unrelated images in the VA.

The SRA implementation requirements can be summarized as follows:

1. Container entry point requirements.
   Each docker container has a leading process which gets PID 1 inside it and which keeps the container alive. When the process exits, the container is stopped and *docker* will not allow the creation of any new process. The leading process is the entry point of the container. It can be specified from the command line when the container is run or with the `ENTRYPOINT` and/or `CMD` instructions of the *Dockerfile* used to build the image (see https://docs.docker.com/engine/reference/builder/). The SRM requirements for an SRA container's entry point are:
   - SRM will not specify the entry point of SRA containers from the command line. This means SRM will not expect SRA images to have a specific executable to serve as the entry point in a predefined location. It is left to the vendors to specify the entry point in their *Dockerfile* with the `ENTRYPOINT` and `CMD` instructions.
   - The entry point must not exit until the container is stopped from the outside which will result in the entry point process receiving first `SIGTERM` and `SIGKILL` about 10 seconds after that. The consequence of this is that the entry point is not allowed to daemonize since *docker* will decide it has exited and will stop the container.
   - The container entry point can be the same as the SRA entry point, but this is not required. SRM will run containers with `-di` flags passed to docker run command, which means detached mode with attached `stdin`. The easiest way to implement the requirements of the entry point is to provide an executable which simply reads from `stdin` which will block indefinitely.

2. SRA entry point requirements.
   The SRA entry point is the executable inside the container which SRM will call when it needs to execute an SRA command. This must not be confused with the *SRA container entry point* which coincidentally might be the same. The requirements for the SRA entry point are:
   - It must be located at a predefined location inside the container, namely the `/srm/sra/command`. In contrast to non-containerized SRAs, here the `.pl` extension is dropped and the requirement for the entry point to be implemented in *Perl is* relaxed. Vendors have the freedom to choose whatever language they like for the

`/srm/sra/command` entry point. It can be a native executable, a script in an interpreted language, even a bash script as long as it can be executed.

- SRM will run the `/srm/sra/command` inside the container with the `uid` and `gid` of the SRM server, which means the SRA entry point must have the necessary permissions for this. Since the files in docker images are usually owned by `root:root` and SRM's `uid` and `gid` are not known in advance, the `/srm/sra/command` must be executable by the `others` group.
- If the `/srm/sra/command` relies on services running inside the container which need time to fully start, the `/srm/sra/command` must not start until they are ready and only then service the request. SRM will not make any guarantees about the time gap between the starting of the container and the execution of the first SRA command. In reality, it is almost certain that SRM will execute the `queryInfo` SRA command immediately after starting the SRA container.

Managing SRA Configuration

Sometimes end users need to tweak the defaults of an SRA image configuration, following the advice of their SRA vendor. To meet this need, the SRM Configuration Service allows the following operations related to an SRA configuration:

- downloading the SRA configuration as a `.tar.gz` archive;
- uploading a modified SRA configuration as a `.tar.gz` archive;
- copying the SRA configuration between different versions of the SRA image;
- resetting an SRA image configuration to its factory settings.

These operations are available through the UI of the SRM Configuration Service.

SRM does not make any assumptions about the format of the SRA configuration or the number of configuration files. The only requirement imposed on SRA images is that they must put their configuration, if it must be user editable, in a dedicated directory inside the image: `/srm/sra/conf/`. There's also no requirement regarding the contents of this directory and its structure. It can be organized in sub-directories as the vendor sees fit.

To prevent end users from overwriting each other's changes to an SRA configuration in the case of concurrent requests, we employ the following procedure:

- When downloading an SRA configuration, we attach a special file to the configuration archive (`sra-configuration-version.txt`) which contains the SHA1 digest of the contents of `/srm/sra/conf/` at the time of download.
- When uploading an SRA configuration, we compare the SHA1 digest stored in `sra-configuration-version.txt` with the digest of the current configuration at the time of upload:
    - If the two hashes match, the current configuration is updated with the uploaded files.
    - If the two hashes don't match, this indicates that the configuration currently being uploaded is based on a snapshot of the files which is no longer current. Updating the configuration with the uploaded files carries a high risk of reverting already applied changes and is rejected.

In some cases the SRA provider might need to store additional configuration information during run-time. To support these cases, the folder `/srm/sra/conf/` has read/write permissions and the configuration files from an old version of the SRA can be copied in the new SRA version.

Using the option to write in /srm/sra/conf/ has some limitations:

- The size of the volume is limited up to the size of the main disk volumes. Writing in the configuration folder must be carefully controlled by the SRA implementation.
- Locking files in /srm/sra/conf/ can cause unpredictable blocks of the working SRA (docker) functionalities. Using shared configuration files in safe manner in parallel command executions depends on the SRA implementation.

Note: The srm-server service must provide correct owner and file permissions in the docker image (SRA) to make writing possible when SRA commands are executed. The srm-server service is running on Site Recovery Manager Virtual Appliance with the user srm (with group srm). The folders and files in /srm/sra/conf must be owned by srm user. The srm uid and gid are fixed to 660 on the Site Recovery Manager Virtual Appliance. The same values must be set in the docker image. You can use the following sample Dockerfile commands to provide write access to /srm/sra/conf:

```
RUN addgroup -gid 660 srm && \
  adduser \
  --disabled-password \
  --home /srm/sra \
  --ingroup srm \
  --uid 660 \
  srm
RUN mkdir -p /srm/sra
RUN chown -R srm:srm /srm
USER srm
```

Running an SRA Container

When SRM runs an SRA, it passes 4 pieces of information to it:

- the command to execute;
- the output file;
- the status file;
- the SRA log directory.

The command to execute is serialized as an XML and passed through `stdin`. The locations of the `output` and `status` files are also passed as part of the XML input via `stdin` as well as the location of the SRA log directory.

The `output` file is where the SRA writes the result of executing the `command` and is in XML format. The `status` file is where the SRA writes progress updates in the form of percentages wrapped in XML tags. The file is monitored and read by SRM to propagate the progress updates.

The `output` and `status` files are created in the `tmp` directory (as specified by SRM's configuration and/or the OS itself) and are deleted upon the SRA completing execution.

The SRA log directory is where the SRA writes its logs. Its location is determined by a combination of SRM config settings and/or OS specific paths. The contents of the directory become part of the support bundle when SRM logs are collected.

When the SRA is run inside a container, all of the above holds with the following exception – the files and directories must be accessible both outside and inside the container. To achieve this, SRM will use `docker` bind mounts (see https://docs.docker.com/storage/bind-mounts/) as follows:

- the `tmp` will be mounted at `/tmp` inside the container;
- the SRA log directory will be mounted at `/srm/sra/log` inside the container.

Although the tmp and SRA log directories will always be mounted at the same points inside the container, their locations will continue to be specified as part of the XML input passed to the SRA via `stdin` for compatibility reasons.

Bind mounts are specified when a container is created from a container image through `docker run` and are immutable: they cannot be changed afterwards and new bind mounts cannot be added. Since the locations of the tmp and SRA log directories outside the container depend on both SRM and OS settings, they can change between SRA runs and they can cause an SRA execution to fail if the outside directories do not match the bind mounts specified when the container was created.

To avoid this, SRM must have a way of matching a request to execute an SRA command to the container where the actual execution will take place. We will introduce the concept of an `execution id` which is the `sha1` hash of the concatenation of the following pieces in the specified order:

- the full image id of the SRA (as taken from `docker inspect`);
- the location of the `tmp` directory outside of the container;
- the location of the SRA log directory outside of the container.

The resulting `sha1` hash (40 byte hex string) will be stored as a label of the container (see https://docs.docker.com/config/labels-custom-metadata/) with the key `srm.sra.eid` (`eid` standing for *execution id*). When an execution request is received, SRM will hash the concatenation of the

image id, `tmp` dir and `log` dir and will try to find a running container with a matching `srm.sra.eid` label. If none is found, SRM will spawn a new container from the given image with bind mounts matching the `tmp` and `log` directories.

Detailed Running Procedure

The following instructions present technical details of how the SRM services work. They are included for a better understanding of the design of containerized SRAs. The SRA providers are not required to follow the instructions.

Use the following detailed procedure to manually run a container with an SRA image:

You must obtain the full image id of the SRA which will be executed:

```
docker inspect --format "{{ .Id }}" <imageid>
```

This will provide the full `sha256` id of the SRA image. Then you must concatenate `imageid` with the `tmpDir` and `logDir`, and then hash the resulting string with `sha1` to receive the `eid` (execution id). The `eid` is used to find a running container whose `srm.sra.eid` label has a matching value:

```
docker container ls -f "label=srm.sra.eid=<eid>" -q
```

If this command does not return a `containerid`, you must check if there is a matching container which has been stopped. To do this you use the same command with an added `-a` flag (meaning *list all*, both running and stopped containers):

```
docker container ls -a -f "label=srm.sra.eid=<eid>" -q
```

If it matches the `containerid`, you just have to start the container before executing the SRA:

```
docker start <containerid>
```

If no `containerid` is returned from the last `docker container ls` command, then there is no container (running or stopped) which matches the triple (`imageid`, `tmpDir`, `logDir`) of the current execution request. You must create one:

```
docker run -di -v "<tmpDir>:/tmp" -v "<logDir>:/srm/sra/log" -l
"srm.sra.eid=<eid>" <imageid>
```

By running this command, you will give the `containerid` of a new container which has the correct bind mounts for the current `tmpDir` and `logDir` (`-v` flags), and a label with the current `eid` which can be used to find the container later (`-l` flag).

By finding the `containerid` in one of the above ways, you can proceed to execute the SRA command:

```
docker exec -i -u <uid>:<gid> [-e "<name>=<value>" ...] <containerid>
/srm/sra/command
```

The `-i` flag is used to attach to the `stdin` of the SRA command to pass the XML input. The `-u` flag is used to avoid complications with permissions for the `output` and `status` files as well as the `logDir` and ensure that the command which runs inside the container with specific `uid` and `gid` passes with the SRA. The `-e` flags are used if the execution request has specified an optional set of environment variables for the SRA command inside the container. You must use a separate flag for each environment variable.