Executor Introduction - Overview

In DC/OS tasks are run (executed) by a Mesos Executor which is identified to run by the scheduler when it launches the task. Both the scheduler and executor(s) are referred to as a ‘framework’. The main concepts here are ‘schedulers’, ‘executors’, and ‘tasks’. If you were to stack them: tasks are run from executors, which in turn are activated/run from schedulers. For example, Marathon kicks off a Mesos Executor to run a shell script or Docker Container.

Mesos includes built-in executors that are in turn available to all schedulers, but schedulers can also use their own executors. Two built-in executors are: “Command Executor”, and “Default Executor”

A few things to know about **Executors**:

- Agents (agent nodes) run tasks that are executed by the “Mesos Executor”
- These tasks, Mesos tasks, are defined by their schedulers to be run by a specific executor, or by the default executor
- Each and every executor runs in it’s own container

A few things to know about **Schedulers**:

- Schedulers include software sub-components such as: Marathon; Cassandra; Kafka; and the default scheduler
- Schedulers must register with Mesos as a framework
- Each schedulers receives resource offerings describing CPU, RAM, Disk, etc and then in turn allocates them for tasks that will be run / launched by Mesos Agents

What about **Frameworks**:

- The concept of a framework in DC/OS includes a scheduler, tasks, and sometimes custom executors
- Framework and scheduler can sometimes be one-and-the-same. Within DC/OS, scheduler is the preferred terminology
-

Finally, what to consider about **Built-in Schedulers**:
DC/OS has two built-in schedulers that can be commonly used to launch tasks on DC/OS. These two built-in schedulers include:

- **Marathon**: A popular scheduler which provides services in the forms of apps and pods which will run continuously and in parallel
- **Metronome**: This scheduler provides jobs which can be run either immediately or on a defined schedule (e.g. like cron in Unix)

Motivation of Performing “Executor Shutdown”

Customer DC/OS environments have tasks that run for long durations. In several of their clusters it is not unusual to see tasks running for more than a few months in duration. In the event of upgrades from 1.7.4, to 1.8.8, tasks may be lost when agent nodes are upgraded. This has been a serious issue within the environment where thousands of tasks are known to be lost and required restart thus losing all application state and connectivity. This patch circumvents task lost state- although it is a best effort, there are inevitably tasks that will be lost but the key objective here is to lose as few tasks as possible.

The main motivation is to not lose tasks, especially long running tasks that require time intensive restarts. In an average Customer cluster, there can be >2500 tasks running across 100s of agent nodes. In the last production cluster upgrade, we had a total node count of 594 agent nodes running +2900 tasks. At the end of the upgrade window, another 450 tasks had been launched on the cluster- this alone underscores the importance of not losing tasks. Restarts are very time consuming, especially at scale. In the event of a total task lost state, restarts take several hours to complete (averaging 8-10 hours).

Last patch received by Customer, via local Solutions Architect (ajohnson@mesosphere), August 15, 2017. At the time of this writing, VZ, Mesosphere Support/Services/Engineering have provided another, much more robust version of the patch which is included here.

Motivation from customer perspective:
From customer: "reason for using the executor patch: we’ve used it for upgrades or patches that have nothing to do with dcos upgrading. For our patching we did not want the containers <= 5 days to restart since we can’t fix that bug until a later dcos upgrade."

Procedure for Executor Shutdown

The procedure for Executor Shutdown is a multi-step process and must be done prior to beginning the actual upgrade of agent nodes. Once the Executor Shutdown procedure has been done/completed the team will have five days to perform the upgrade before needing to re-run the Executor Shutdown. This clearly is a two-part caveat, the Executor Shutdown script saves tasks that are older than five days, and not completing the upgrade within this time window will end in tasks being lost (that are five days or older). How can this be an issue? How can an upgrade take more than five days? Historically, upgrades at Customer are considered week long events with the last major upgrade consuming more than 2 weeks for ~600 nodes.

The procedure for Executor Shutdown is fairly straightforward, the scripts do the repetitive work and the entire process is run at scale with Ansible, running 10 Executor Shutdown scripts in parallel across 10 nodes. In a cluster of approximately 600 nodes the entire process takes 1 to 1.5 days to complete. This is usually the first foray into the overall upgrade.

Running from Ansible, this is what happens for the Executor Shutdown:

1. Deploy Executor Shutdown container to the ER (Enterprise Registry)
2. Deploy the upgradepublicmesos_fix_1023.sh script to run the entire process
3. Verify that tasks were existing prior to the fix
4. Run the Executor Shutdown
5. Verify end state of tasks
6. Proceed to other agent nodes or remedy tasks missing

In earlier versions of the script there were two issues:

1. *Issue one*: Executor Shutdown script did not correctly grep for task names that contained spaces. ($FD=" "). In this scenario the script would not finish and required to be restarted a second, sometimes a third time in order to treat all tasks to proper shutdown. It eventually finished when there were no tasks that had " <white space> " in their naming.
2. *Issue two*: (this is an interesting predicament). On some nodes the Executor shutdown script called to DC/OS binaries that were already removed and/or upgraded. The work around for this predicament is to access the image of the binary in the hosts memory so that the script could call the correct version of the binary for the shutdown procedure.
    a. This ended up being a problem where the executor was started under one version of DC/OS, the agent was upgraded, but the binary that the executor was started with was removed so GDB could not attach to the process. We updated the script we gave Customer to instead load the binary from memory which fixed it.

Scripts for this Procedure

Upgradepublicmesos_fix_1023.sh

Capture start and end state and to execute the Shutdown script.

```
#!/bin/bash

# Nodelist - list of nodes that this will run on
#
NODELIST="$1"

# Name of gdb container
#
GDBNAME="baygeldin/gdb-examples"

# Check nodelist and script usage
#
if [ ! -f "$NODELIST" ]; then
    echo "Nodelist file not found. Create, retry"
    echo "Script Usage: ./scriptname 'nodelist' (where nodelist is file with a
list of nodes)"
    exit
fi

# Is the node up/down?
#
for i in `cat $NODELIST`; do
    ping -c 2 $i
    if [ $? != 0 ]; then
        echo "Couldn't ping $i, cancelling entire run..."
        exit 1
        echo ""
    fi
done

# Connect to the host, gather some pre-exec-patch info
#
for i in `cat $NODELIST`; do
    if [ ! -f $i.log ]; then
    touch $i.log
    fi
    echo "Working on gathering data for $i"
    echo `date` >>$i.log
    printf "$i mesos-slave version:\n " >>$i.log
```

```
    ssh $i ' /opt/mesosphere/bin/mesos-slave --version' >>$i.log
    printf "===\n" >>$i.log
    printf "Capturing NUMBER OF mesos-exec|mesos-doc:\n" >>$i.log
    ssh $i ps -aux |egrep 'mesos-exec|mesos-doc' |wc -l >>$i.log
    printf "===\n" >>$i.log
    printf "Capturing Docker processes before the upgrade:\n" >>$i.log
    ssh $i docker ps >> $i.log
    printf "===\n" >>$i.log
done
echo "Done gathering data for nodes"
echo ""

# Connect to the host, copy the exec patch, load, run, etc.
#
echo "Connecting to the nodes and loading the Executor patch"
for i in `cat $NODELIST`; do
    if [ ! -f $i.docps.before.txt ]; then
    touch $i.netstat.before.txt
    fi
    echo "Working on loading patch on $i"
    scp Customer-upgrade-13-oct-2017.tgz $i:/tmp
    ssh $i 'cd /tmp; tar xvf Customer-upgrade-13-oct-2017.tgz; cd
/tmp/Customer-upgrade-fix-13-Oct-2017; sudo docker load < gdb-examples.tar'
    echo `date` >>$i.netstat.before.txt
    printf "Checking netstat for mesos-docker|mesos-exec\n\n"
>>$i.netstat.before.txt
    ssh $i sudo netstat -nap|grep 5051 |egrep "mesos-docker|mesos-exec" >>
$i.netstat.before.txt
    # Run executor Socket Shutdown and check exit
    #
        ssh -tt $i 'cd /tmp/Customer-upgrade-fix-13-Oct-2017; sudo
./shutdown_executor_sockets.sh > save_exec.out'
        scp $i:/tmp/Customer-upgrade-fix-13-Oct-2017/save_exec.out
save_exec.out.$i
done

# Check after run
#
if [ -f $i.netstat.aft.txt ]
    then touch $i.netstat.aft.txt
fi

for i in `cat $NODELIST`; do
    ssh $i sudo netstat -nap|grep 5051 | egrep "mesos-docker|mesos-exec" >>
$i.netstat.aft.txt
    # Check output before restarting the mesos-slave - error if there is more
than 0 running...
```

```
        ssh $i 'if [ $(sudo netstat -nap |grep 5051 | egrep
'mesos-docker|mesos-exec' |wc -l) -gt 0 ]; then echo more than ZERO
mesos-docker on port 5051 found, something didn't run right on $i, exiting;
fi; exit 1
    ssh $i restart systemctl restart dcos-mesos-slave
    ssh $i sudo netstat -nap |grep 5051 | egrep "mesos-docker|mesos-exec"
>$i.netstat.aft.2.txt
done

# Stop RexRay
for i in `cat $NODELIST`; do
    ssh $i 'sudo systemctl stop rexray; sudo systemctl disable rexray';
done

# End
```

Shutdown_Executor_Sockets

Using GDB performs a shutdown for executor sockets so they are not trashed during an upgrade.

```
README:
#####
The following should be ran on each agent node BEFORE attempting an upgrade

1) Load the provided docker image file into your local registry
$ docker load < gdb-examples.tar

2) Run the `shutdown_executor_sockets.sh` script
$ ./shutdown_executor_sockets.sh
Found 2 running executors
..
Success!

3) Verify that the operation was successful by doing
$ sudo netstat -nap|grep 5051 | egrep 'mesos-docker|mesos-exec' # <--- The
output should be empty.
$ ps aux | egrep 'mesos-docker|mesos-exec' # <--- You should see executor
processes

4) From the time this script completes you only have 15 minutes to restart
your
agent before all executors commit suicide! Go ahead and do it now.
$ sudo systemctl restart dcos-mesos-slave
```

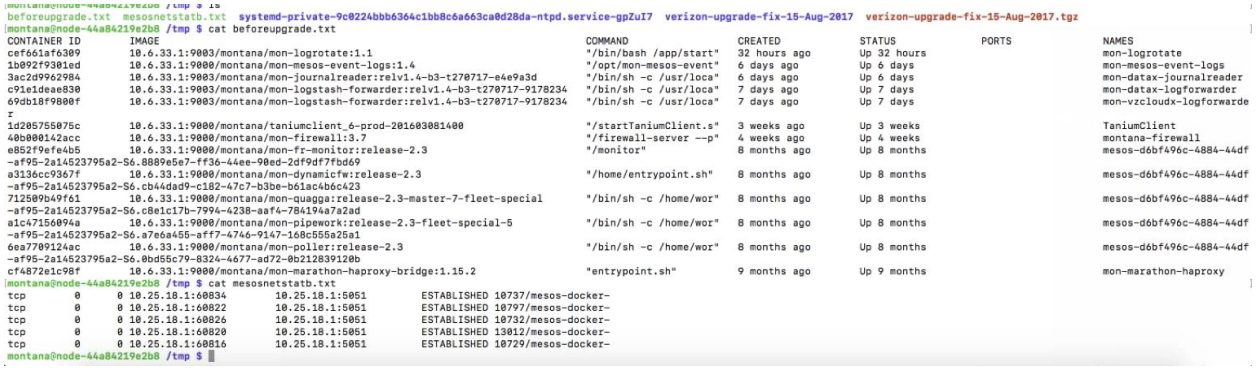
shutdown_executor_sockets.sh

```
#! /usr/bin/env bash

AGENT_PORT=5051
GDB_FILES_DIR=gdb_files

create_gdb_script() {
    PID=$1
    FD=$2
    GDB_FILE_NAME=$GDB_FILES_DIR/gdb_${PID}_${FD}

    echo "
set width 0
set height 0
set verbose off
set logging off
set solib-absolute-prefix /host
call shutdown($FD, 0)
quit
" > $GDB_FILE_NAME

   echo -n $GDB_FILE_NAME
}

mkdir -p $GDB_FILES_DIR

EXECUTOR_PIDS=$(ps aux | egrep 'mesos-executor|mesos-docker-executor' | grep
-v grep | awk '{print $2}')
#EXECUTOR_PIDS=$(ps aux | egrep 'ps aux' | grep -v grep | awk '{print $2}')
NUM=$(echo $EXECUTOR_PIDS | wc -w)

echo "Found $NUM running executors"

for PID in $EXECUTOR_PIDS; do
   FD=$(sudo lsof -P -p $PID | grep $AGENT_PORT | awk '{print $4}' | rev | cut
-c 2- | rev)
   [ "$FD" = "" ] && continue

   GDB_FILE=$(create_gdb_script $PID $FD)

   docker run \
     -it \
     --privileged \
     --ipc=host \
     --net=host \
     --pid=host \
```

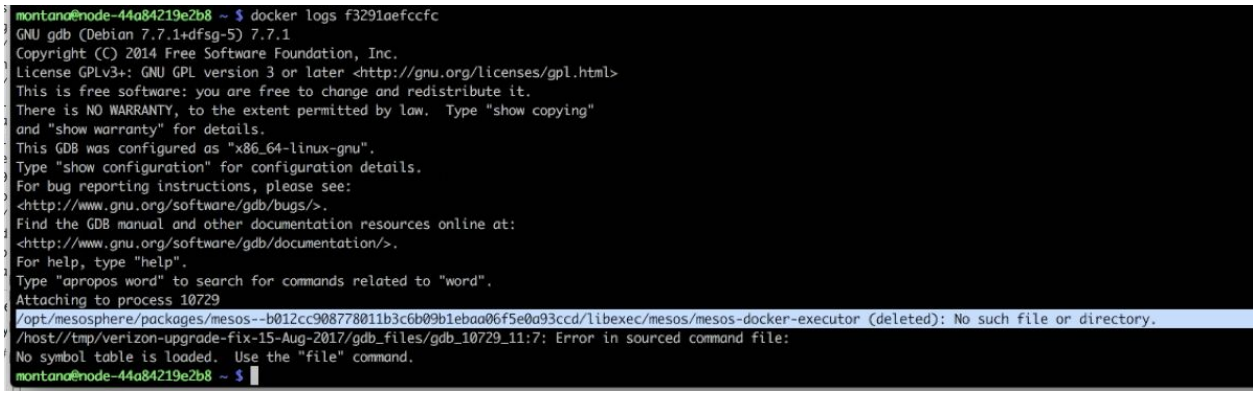
```
      -v /opt:/opt \
      -v /:/host \
      baygeldin/gdb-examples gdb \
      --command=/host/$PWD/$GDB_FILE /proc/$PID/exe $PID 2>&1 >/dev/null
    echo -n .
done

echo
echo "Success!"
```

Relevant Log Data

Logs - before and after:



Docker logs:



Manifest.json

```
Mesospheres-MBP:examples-tar arthurjohnson$ cat manifest.json |jq
[
  {
    "Config":
"85883b58eebfdd4dd7724a7b67e0a51e1507900479546dec2dd32c734b1cf72b.json",
    "RepoTags": [
      "baygeldin/gdb-examples:latest"
    ],
    "Layers": [

"4b029cd17078142030f714cc073f5c76c0e6eb7a57e962665ea5bef50f1376b3/layer.tar",

"a60c8b1fcf85307b761a8455d29211949cdc63d221c610a37b2a1582ce440f30/layer.tar",

"727315a913b9334eed5f878ef94bba37dd146874b7a03f1f70c78b52a440095a/layer.tar",

"cfd307dd8a9f64eb6a9d8dcf946c7e937c49027dc0ddca82e2f74167a40a89ec/layer.tar",

"cfad793b7844283ae27592a1386d4da878fecd631e393bb414b790a7367ebf35/layer.tar",

"90ddd708dc9f309cf57f3e266a9371d09209c4aa5989b306f779b52249def9b4/layer.tar",

"6b0ea045acc875aab2eb242118d1fec2d675d02faf273dc56318f47bfc2c68f3/layer.tar",

"c189bdc147c1bfb6e9ac8f3c005dfeef4dda3aee5339c13a4cba05068c3c248f/layer.tar",

"cdd18de849c929a9742bd85b3dd548dd1581684588c2247b5c4a164981feeb66/layer.tar"
    ]
  }
]
```

Output of journalctl -fu

```
Oct 12 16:43:29 node-44a84216d222 mesos-slave[35489]: I1012 16:43:29.363692
35513 slave.cpp:194] Flags at startup:
--appc_simple_discovery_uri_prefix="http://"
--appc_store_dir="/tmp/mesos/store/appc"
--attributes="MAC:44A84216D222;RACK:25;SLOT:11;TYPE:COMPUTE;SUBTYPE:MESOSFE;ENV
:PROD;REGION:ARLINGTON;ITRACKTITLE:AW82;ITNAME:ARLKR25S11;NODEMODEL:DELL_R630;N
ODECLASS:FE;EXTNETNAME:RAN;STORAGESERVICE:NONE;STORAGEROLE:NONE;STORAGEVENDOR:N
ONE;PUBLIC_IFCNAME:ENO2;PUBLIC_IP:169.254.2.11/18" --authenticatee="crammd5"
--cgroups_cpu_enable_pids_and_tids_count="false" --cgroups_enable_cfs="true"
--cgroups_hierarchy="/sys/fs/cgroup" --cgroups_limit_swap="false"
--cgroups_root="mesos" --container_disk_watch_interval="15secs"
--container_logger="org_apache_mesos_LogrotateContainerLogger"
--containerizers="docker,mesos" --default_role="slave_public"
--disk_watch_interval="1mins" --docker="docker" --docker_kill_orphans="true"
```

```
--docker_registry="https://registry-1.docker.io" --docker_remove_delay="1hrs"
--docker_socket="/var/run/docker.sock" --docker_stop_timeout="20secs"
--docker_store_dir="/tmp/mesos/store/docker"
--docker_volume_checkpoint_dir="/var/lib/mesos/isolators/docker/volume"
--enforce_container_disk_quota="false"
--executor_environment_variables="{"LD_LIBRARY_PATH":"\/opt\/mesosphere\/lib","
PATH":"\/usr\/bin:\/bin","SASL_PATH":"\/opt\/mesosphere\/lib\/sasl2","SHELL":"\
/usr\/bin\/bash"}" --executor_registration_timeout="10mins"
--executor_shutdown_grace_period="5secs" --fetcher_cache_dir="/tmp/mesos/fetch"
--fetcher_cache_size="2GB" --frameworks_home="" --gc_delay="2days"
--gc_disk_headroom="0.1" --hadoop_home="" --help="false"
--hooks="com_mesosphere_StatsEnvHook" --hostname_lookup="false"
--image_provisioner_backend="copy" --initialize_driver_logging="true"
--ip_discovery_command="/opt/mesosphere/bin/detect_ip"
--isolation="cgroups/cpu,cgroups/mem,posix/disk,filesystem/linux,docker/volume,
com_mesosphere_StatsIsolatorModule"
--launcher_dir="/opt/mesosphere/packages/mesos--6bfac93420e14cebec27701ed547548
fc4fba7bb/libexec/mesos" --log_dir="/var/log/mesos" --logbufsecs="0"
```

Conclusion

While this procedure deals with older versions of DC/OS and large numbers of tasks, it is possible to see related events in new releases of DC/OS and environments that may be hosting more than 100s of tasks. The effort of upgrades are to be rolling in nature, and therefore, as least disruptive as possible; caution should always be taken to make sure that the environment can survive changes to the underlying DC/OS code without impact to running tasks. As such, events such as upgrades should be done in a phased approach where changes are made in a small and controlled manner, checking to see the state of the system along the way before proceeding.