Manage app execution hooks

Astra Control Center

NetApp
October 11, 2022

This PDF was generated from https://docs.netapp.com/us-en/astra-control-center/use/execution-hook-examples.html on October 11, 2022. Always check docs.netapp.com for the latest.

Table of Contents

Manage app execution hooks

An execution hook is a custom action that you can configure to run in conjunction with a data protection operation of a managed app. For example, if you have a database app, you can use execution hooks to pause all database transactions before a snapshot, and resume transactions after the snapshot is complete. This ensures application-consistent snapshots.

Types of execution hooks

Astra Control supports the following types of execution hooks, based on when they can be run:

- Pre-snapshot
- Post-snapshot
- Pre-backup
- Post-backup
- Post-restore

Important notes about custom execution hooks

Consider the following when planning execution hooks for your apps.

- An execution hook must use a script to perform actions. Many execution hooks can reference the same script.
- Astra Control requires the scripts that execution hooks use to be written in the format of executable shell scripts.
- Script size is limited to 96KB.
- Astra Control uses execution hook settings and any matching criteria to determine which hooks are applicable to a snapshot, backup, or restore operation.
- All execution hook failures are soft failures; other hooks and the data protection operation are still attempted even if a hook fails. However, when a hook fails, a warning event is recorded in the **Activity** page event log.
- To create, edit, or delete execution hooks, you must be a user with Owner, Admin, or Member permissions.
- If an execution hook takes longer than 25 minutes to run, the hook will fail, creating an event log entry with a return code of "N/A". Any affected snapshot will time out and be marked as failed, with a resulting event log entry noting the timeout.
- For adhoc data protection operations, all hook events are generated and saved in the **Activity** page event log. However, for scheduled data protection operations, only hook failure events are recorded in the event log (events generated by the scheduled data protection operations themselves are still recorded).

Since execution hooks often reduce or completely disable the functionality of the application they are running against, you should always try to minimize the time your custom execution hooks take to run.
If you start a backup or snapshot operation with associated execution hooks but then cancel it, the hooks are still allowed to run if the backup or snapshot operation has already begun. This means that a post-backup execution hook cannot assume that the backup was completed.

Order of execution

When a data protection operation is run, execution hook events take place in the following order:

1. Any applicable custom pre-operation execution hooks are run on the appropriate containers. You can create and run as many custom pre-operation hooks as you need, but the order of execution of these hooks before the operation is neither guaranteed nor configurable.
2. The data protection operation is performed.
3. Any applicable custom post-operation execution hooks are run on the appropriate containers. You can create and run as many custom post-operation hooks as you need, but the order of execution of these hooks after the operation is neither guaranteed nor configurable.

If you create multiple execution hooks of the same type (for example, pre-snapshot), the order of execution of those hooks is not guaranteed. However, the order of execution of hooks of different types is guaranteed. For example, the order of execution of a configuration that has all five different types of hooks would look like this:

1. Pre-backup hooks executed
2. Pre-snapshot hooks executed
3. Post-snapshot hooks executed
4. Post-backup hooks executed
5. Post-restore hooks executed

You can see an example of this configuration in scenario number 2 from the table in Determine whether a hook will run.

You should always test your execution hook scripts before enabling them in a production environment. You can use the 'kubectl exec' command to conveniently test the scripts. After you enable the execution hooks in a production environment, test the resulting snapshots and backups to ensure they are consistent. You can do this by cloning the app to a temporary namespace, restoring the snapshot or backup, and then testing the app.

Determine whether a hook will run

Use the following table to help determine if a custom execution hook will run for your app.

Note that all high-level app operations consist of running one of the basic operations of snapshot, backup, or restore. Depending on the scenario, a clone operation can consists of various combinations of these operations, so what execution hooks a clone operation runs will vary.

In-place restore operations require an existing snapshot or backup, so these operations don’t run snapshot or backup hooks.

If you start but then cancel a backup that includes a snapshot and there are associated execution hooks, some hooks might run, and others might not. This means that a post-backup execution hook cannot assume that the backup was completed. Keep in mind the following points for cancelled backups with associated execution hooks:

- The pre-backup and post-backup hooks are always run.
- If the backup includes a new snapshot and the snapshot has started, the pre-snapshot and post-snapshot hooks are run.
- If the backup is cancelled prior to the snapshot starting, the pre-snapshot and post-snapshot hooks are not run.

Scenario	Operation	Existing snapshot	Existing backup	Namespace	Cluster	Snapshot hooks run	Backup hooks run	Restore hooks run
1	Clone	N	N	New	Same	Y	N	Y
2	Clone	N	N	New	Different	Y	Y	Y
3	Clone or restore	Y	N	New	Same	N	N	Y
4	Clone or restore	N	Y	New	Same	N	N	Y
5	Clone or restore	Y	N	New	Different	N	Y	Y
6	Clone or restore	N	Y	New	Different	N	N	Y
7	Restore	Y	N	Existing	Same	N	N	Y
8	Restore	N	Y	Existing	Same	N	N	Y
9	Snapshot	N/A	N/A	N/A	N/A	Y	N/A	N/A
10	Backup	N	N/A	N/A	N/A	Y	Y	N/A
11	Backup	Y	N/A	N/A	N/A	N	Y	N/A

View existing execution hooks

You can view existing custom execution hooks for an app.

Steps

1. Go to **Applications** and then select the name of a managed app.
2. Select the **Execution hooks** tab.

   You can view all enabled or disabled execution hooks in the resulting list. You can see a hook's status, source, and when it runs (pre- or post-operation). To view event logs surrounding execution hooks, go to the **Activity** page in the left-side navigation area.

View existing scripts

You can view the existing uploaded scripts. You can also see which scripts are in use, and what hooks are using them, on this page.

Steps

1. Go to **Account**.
2. Select the **Scripts** tab.

   You can see a list of existing uploaded scripts on this page. The **Used by** column shows which execution hooks are using each script.

Add a script

You can add one or more scripts that execution hooks can reference. Many execution hooks can reference the same script; this enables you to update many execution hooks by only changing one script.

Steps

1. Go to **Account**.
2. Select the **Scripts** tab.
3. Select **Add**.
4. Do one of the following:
   - Upload a custom script.
     a. Select the **Upload file** option.
     b. Browse to a file and upload it.
     c. Give the script a unique name.
     d. (Optional) Enter any notes other administrators should know about the script.
     e. Select **Save script**.
   - Paste in a custom script from the clipboard.
     a. Select the **Paste or type** option.
     b. Select the text field and paste the script text into the field.
     c. Give the script a unique name.
     d. (Optional) Enter any notes other administrators should know about the script.
5. Select **Save script**.

Result

The new script appears in the list on the **Scripts** tab.

Delete a script

You can remove a script from the system if it is no longer needed and not used by any execution hooks.

Steps

1. Go to **Account**.

2. Select the **Scripts** tab.
3. Choose a script you want to remove, and select the menu in the **Actions** column.
4. Select **Delete**.

> If the script is associated with one or more execution hooks, the **Delete** action is unavailable. To delete the script, first edit the associated execution hooks and associate them with a different script.

Create a custom execution hook

You can create a custom execution hook for an app. See Execution hook examples for hook examples. You need to have Owner, Admin, or Member permissions to create execution hooks.

> When you create a custom shell script to use as an execution hook, remember to specify the appropriate shell at the beginning of the file, unless you are running specific commands or providing the full path to an executable.

Steps

1. Select **Applications** and then select the name of a managed app.
2. Select the **Execution hooks** tab.
3. Select **Add**.
4. In the **Hook Details** area, determine when the hook should run by selecting an operation type from the **Operation** drop-down menu.
5. Enter a unique name for the hook.
6. (Optional) Enter any arguments to pass to the hook during execution, pressing the Enter key after each argument you enter to record each one.
7. In the **Container Images** area, if the hook should run against all container images contained within the application, enable the **Apply to all container images** check box. If instead the hook should act only on one or more specified container images, enter the container image names in the **Container image names to match** field.
8. In the **Script** area, do one of the following:
    - Add a new script.
        a. Select **Add**.
        b. Do one of the following:
            - Upload a custom script.
                i. Select the **Upload file** option.
                ii. Browse to a file and upload it.
                iii. Give the script a unique name.
                iv. (Optional) Enter any notes other administrators should know about the script.
                v. Select **Save script**.
            - Paste in a custom script from the clipboard.
                i. Select the **Paste or type** option.

ii. Select the text field and paste the script text into the field.

iii. Give the script a unique name.

iv. (Optional) Enter any notes other administrators should know about the script.

- Select an existing script from the list.

This instructs the execution hook to use this script.

9. Select **Add hook**.

Check the state of an execution hook

After a snapshot, backup, or restore operation finishes running, you can check the state of execution hooks that ran as part of the operation. You can use this status information to determine if you want to keep the execution hook, modify it, or delete it.

Steps

1. Select **Applications** and then select the name of a managed app.
2. Select the **Data protection** tab.
3. Select **Snapshots** to see running snapshots, or **Backups** to see running backups.

   The **Hook state** shows the status of the execution hook run after the operation is complete. You can hover over the state for more details. For example, if there are execution hook failures during a snapshot, hovering over the hook state for that snapshot gives a list of failed execution hooks. To see reasons for each failure, you can check the **Activity** page in the left-side navigation area.

View script usage

You can see which execution hooks use a particular script in the Astra Control web UI.

Steps

1. Select **Account**.
2. Select the **Scripts** tab.

   The **Used by** column in the list of scripts contains details on which hooks are using each script in the list.

3. Select the information in the **Used by** column for a script you are interested in.

   A more detailed list appears, with the names of hooks that are using the script and the type of operation they are configured to run with.

Disable an execution hook

You can disable an execution hook if you want to temporarily prevent it from running before or after a snapshot of an app. You need to have Owner, Admin, or Member permissions to disable execution hooks.

Steps

1. Select **Applications** and then select the name of a managed app.
2. Select the **Execution hooks** tab.

3. Select the Options menu in the **Actions** column for a hook that you wish to disable.
4. Select **Disable**.

Delete an execution hook

You can remove an execution hook entirely if you no longer need it. You need to have Owner, Admin, or Member permissions to delete execution hooks.

Steps

1. Select **Applications** and then select the name of a managed app.
2. Select the **Execution hooks** tab.
3. Select the Options menu in the **Actions** column for a hook that you wish to delete.
4. Select **Delete**.

Execution hook examples

Use the following examples to get an idea of how to structure your execution hooks. You can use these hooks as templates, or as test scripts.

Simple success example

This is an example of a simple hook that succeeds and writes a message to standard output and standard error.

```
#!/bin/sh

# success_sample.sh
#
# A simple noop success hook script for testing purposes.
#
# args: None
#



#
# Writes the given message to standard output
#
# $* - The message to write
#
msg() {
    echo "$*"
}



#
# Writes the given information message to standard output
```

```
#
# $* - The message to write
#
info() {
    msg "INFO: $*"
}

#
# Writes the given error message to standard error
#
# $* - The message to write
#
error() {
    msg "ERROR: $*" 1>&2
}


#
# main
#

# log something to stdout
info "running success_sample.sh"

# exit with 0 to indicate success
info "exit 0"
exit 0
```

Simple success example (bash version)

This is an example of a simple hook that succeeds and writes a message to standard output and standard error, written for bash.

```
#!/bin/bash

# success_sample.bash
#
# A simple noop success hook script for testing purposes.
#
# args: None


#
# Writes the given message to standard output
#
# $* - The message to write
```

```
#
msg() {
    echo "$*"
}


#
# Writes the given information message to standard output
#
# $* - The message to write
#
info() {
    msg "INFO: $*"
}

#
# Writes the given error message to standard error
#
# $* - The message to write
#
error() {
    msg "ERROR: $*" 1>&2
}


#
# main
#

# log something to stdout
info "running success_sample.bash"

# exit with 0 to indicate success
info "exit 0"
exit 0
```

Simple success example (zsh version)

This is an example of a simple hook that succeeds and writes a message to standard output and standard error, written for Z shell.

```
#!/bin/zsh

# success_sample.zsh
#
# A simple noop success hook script for testing purposes.
```

```
#
# args: None
#


#
# Writes the given message to standard output
#
# $* - The message to write
#
msg() {
    echo "$*"
}


#
# Writes the given information message to standard output
#
# $* - The message to write
#
info() {
    msg "INFO: $*"
}

#
# Writes the given error message to standard error
#
# $* - The message to write
#
error() {
    msg "ERROR: $*" 1>&2
}

#
# main
#

# log something to stdout
info "running success_sample.zsh"

# exit with 0 to indicate success
info "exit 0"
exit 0
```

Success with arguments example

The following example demonstrates how you can use args in a hook.

```
#!/bin/sh

# success_sample_args.sh
#
# A simple success hook script with args for testing purposes.
#
# args: Up to two optional args that are echoed to stdout

#
# Writes the given message to standard output
#
# $* - The message to write
#
msg() {
    echo "$*"
}


#
# Writes the given information message to standard output
#
# $* - The message to write
#
info() {
    msg "INFO: $*"
}

#
# Writes the given error message to standard error
#
# $* - The message to write
#
error() {
    msg "ERROR: $*" 1>&2
}


#
# main
#

# log something to stdout
```

```
info "running success_sample_args.sh"


# collect args
arg1=$1
arg2=$2

# output args and arg count to stdout
info "number of args: $#"
info "arg1 ${arg1}"
info "arg2 ${arg2}"

# exit with 0 to indicate success
info "exit 0"
exit 0
```

Pre-snapshot / post-snapshot hook example

The following example demonstrates how the same script can be used for both a pre-snapshot and a post-snapshot hook.

```
#!/bin/sh

# success_sample_pre_post.sh
#
# A simple success hook script example with an arg for testing purposes
# to demonstrate how the same script can be used for both a prehook and
posthook
#
# args: [pre|post]

# unique error codes for every error case
ebase=100
eusage=$((ebase+1))
ebadstage=$((ebase+2))
epre=$((ebase+3))
epost=$((ebase+4))


#
# Writes the given message to standard output
#
# $* - The message to write
#
msg() {
    echo "$*"
```

```
}


#
# Writes the given information message to standard output
#
# $* - The message to write
#
info() {
    msg "INFO: $*"
}

#
# Writes the given error message to standard error
#
# $* - The message to write
#
error() {
    msg "ERROR: $*" 1>&2
}


#
# Would run prehook steps here
#
prehook() {
    info "Running noop prehook"
    return 0
}

#
# Would run posthook steps here
#
posthook() {
    info "Running noop posthook"
    return 0
}


#
# main
#

# check arg
stage=$1
if [ -z "${stage}" ]; then
```

```
    echo "Usage: $0 <pre|post>"
    exit ${eusage}
fi

if [ "${stage}" != "pre" ] && [ "${stage}" != "post" ]; then
    echo "Invalid arg: ${stage}"
    exit ${ebadstage}
fi

# log something to stdout
info "running success_sample_pre_post.sh"

if [ "${stage}" = "pre" ]; then
    prehook
    rc=$?
    if [ ${rc} -ne 0 ]; then
        error "Error during prehook"
    fi
fi

if [ "${stage}" = "post" ]; then
    posthook
    rc=$?
    if [ ${rc} -ne 0 ]; then
        error "Error during posthook"
    fi
fi

exit ${rc}
```

Failure example

The following example demonstrates how you can handle failures in a hook.

```
#!/bin/sh

# failure_sample_arg_exit_code.sh
#
# A simple failure hook script for testing purposes.
#
# args: [the exit code to return]
#


#
# Writes the given message to standard output
```

```
#
# $* - The message to write
#
msg() {
    echo "$*"
}


#
# Writes the given information message to standard output
#
# $* - The message to write
#
info() {
    msg "INFO: $*"
}

#
# Writes the given error message to standard error
#
# $* - The message to write
#
error() {
    msg "ERROR: $*" 1>&2
}


#
# main
#

# log something to stdout
info "running failure_sample_arg_exit_code.sh"

argexitcode=$1

# log to stderr
error "script failed, returning exit code ${argexitcode}"

# exit with specified exit code
exit ${argexitcode}
```

Verbose failure example

The following example demonstrates how you can handle failures in a hook, with more verbose logging.

```
#!/bin/sh

# failure_sample_verbose.sh
#
# A simple failure hook script with args for testing purposes.
#
# args: [The number of lines to output to stdout]


#
# Writes the given message to standard output
#
# $* - The message to write
#
msg() {
    echo "$*"
}


#
# Writes the given information message to standard output
#
# $* - The message to write
#
info() {
    msg "INFO: $*"
}

#
# Writes the given error message to standard error
#
# $* - The message to write
#
error() {
    msg "ERROR: $*" 1>&2
}


#
# main
#

# log something to stdout
info "running failure_sample_verbose.sh"


# output arg value to stdout
```

```
linecount=$1
info "line count ${linecount}"

# write out a line to stdout based on line count arg
i=1
while [ "$i" -le ${linecount} ]; do
    info "This is line ${i} from failure_sample_verbose.sh"
    i=$(( i + 1 ))
done

error "exiting with error code 8"
exit 8
```

Failure with an exit code example

The following example demonstrates a hook failing with an exit code.

```
#!/bin/sh

# failure_sample_arg_exit_code.sh
#
# A simple failure hook script for testing purposes.
#
# args: [the exit code to return]
#


#
# Writes the given message to standard output
#
# $* - The message to write
#
msg() {
    echo "$*"
}


#
# Writes the given information message to standard output
#
# $* - The message to write
#
info() {
    msg "INFO: $*"
}
```

```
#
# Writes the given error message to standard error
#
# $* - The message to write
#
error() {
    msg "ERROR: $*" 1>&2
}


#
# main
#

# log something to stdout
info "running failure_sample_arg_exit_code.sh"

argexitcode=$1

# log to stderr
error "script failed, returning exit code ${argexitcode}"

# exit with specified exit code
exit ${argexitcode}
```

Success after failure example

The following example demonstrates a hook failing the first time it is run, but succeeding after the second run.

```
#!/bin/sh

# failure_then_success_sample.sh
#
# A hook script that fails on initial run but succeeds on second run for
testing purposes.
#
# Helpful for testing retry logic for post hooks.
#
# args: None
#

#
# Writes the given message to standard output
#
# $* - The message to write
#
```

```
msg() {
    echo "$*"
}


#
# Writes the given information message to standard output
#
# $* - The message to write
#
info() {
    msg "INFO: $*"
}

#
# Writes the given error message to standard error
#
# $* - The message to write
#
error() {
    msg "ERROR: $*" 1>&2
}


#
# main
#

# log something to stdout
info "running failure_success sample.sh"


if [ -e /tmp/hook-test.junk ] ; then
    info "File does exist.  Removing /tmp/hook-test.junk"
    rm /tmp/hook-test.junk
    info "Second run so returning exit code 0"
    exit 0
else
    info "File does not exist.  Creating /tmp/hook-test.junk"
    echo "test" > /tmp/hook-test.junk
    error "Failed first run, returning exit code 5"
    exit 5
fi
```

Copyright Information

Copyright © 2022 NetApp, Inc. All rights reserved. Printed in the U.S. No part of this document covered by copyright may be reproduced in any form or by any means-graphic, electronic, or mechanical, including photocopying, recording, taping, or storage in an electronic retrieval system-without prior written permission of the copyright owner.

Software derived from copyrighted NetApp material is subject to the following license and disclaimer:

THIS SOFTWARE IS PROVIDED BY NETAPP “AS IS” AND WITHOUT ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE, WHICH ARE HEREBY DISCLAIMED. IN NO EVENT SHALL NETAPP BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

NetApp reserves the right to change any products described herein at any time, and without notice. NetApp assumes no responsibility or liability arising from the use of products described herein, except as expressly agreed to in writing by NetApp. The use or purchase of this product does not convey a license under any patent rights, trademark rights, or any other intellectual property rights of NetApp.

The product described in this manual may be protected by one or more U.S. patents, foreign patents, or pending applications.

RESTRICTED RIGHTS LEGEND: Use, duplication, or disclosure by the government is subject to restrictions as set forth in subparagraph (c)(1)(ii) of the Rights in Technical Data and Computer Software clause at DFARS 252.277-7103 (October 1988) and FAR 52-227-19 (June 1987).

Trademark Information

NETAPP, the NETAPP logo, and the marks listed at http://www.netapp.com/TM are trademarks of NetApp, Inc. Other company and product names may be trademarks of their respective owners.