# Gérer les crochets d'exécution de l'application

Astra Control Center

NetApp
November 21, 2023

This PDF was generated from https://docs.netapp.com/fr-fr/astra-control-center-2112/use/execution-hook-examples.html on November 21, 2023. Always check docs.netapp.com for the latest.

# Sommaire

Gérer les crochets d’exécution de l’application . . . . . 1
Crochets d’exécution par défaut et expressions régulières . . . . . 1
Remarques importantes sur les crochets d’exécution personnalisés . . . . . 1
Afficher les crochets d’exécution existants . . . . . 2
Créer un crochet d’exécution personnalisé . . . . . 2
Désactivez un crochet d’exécution . . . . . 3
Supprimer un crochet d’exécution . . . . . 4
Exemples de crochet d’exécution . . . . . 4

# Gérer les crochets d’exécution de l’application

Un crochet d’exécution est un script personnalisé que vous pouvez exécuter avant ou après un instantané d’une application gérée. Par exemple, si vous disposez d’une application de base de données, vous pouvez utiliser des crochets d’exécution pour interrompre toutes les transactions de base de données avant un instantané et reprendre les transactions une fois l’instantané terminé. Les snapshots sont ainsi cohérents au niveau des applications.

## Crochets d’exécution par défaut et expressions régulières

Pour certaines applications, Astra Control est doté de crochets d’exécution par défaut fournis par NetApp qui gèrent les opérations de gel et de dégel avant et après les snapshots. Astra Control utilise des expressions régulières pour faire correspondre l’image de conteneur d’une application à ces applications :

- MariaDB
  - Expression régulière correspondante : \bmariadb\b
- MySQL
  - Expression régulière correspondante : \bmysql\b
- PostgreSQL
  - Expression régulière correspondante : \bpostgresql\b

S’il y a une correspondance, les crochets d’exécution par défaut fournis par NetApp pour cette application apparaissent dans la liste des crochets d’exécution actifs de l’application, et ces crochets s’exécutent automatiquement lorsque des instantanés de cette application sont effectués. Si l’une de vos applications personnalisées possède un nom d’image similaire qui correspond à l’une des expressions régulières (et que vous ne souhaitez pas utiliser les crochets d’exécution par défaut), vous pouvez modifier le nom de l’image, ou désactivez le crochet d’exécution par défaut pour cette application et utilisez plutôt un crochet personnalisé.

Vous ne pouvez pas supprimer ou modifier les crochets d’exécution par défaut.

## Remarques importantes sur les crochets d’exécution personnalisés

Lors de la planification de crochets d’exécution pour vos applications, tenez compte des points suivants.

- Astra Control nécessite que les crochets d’exécution soient écrits au format de scripts shell exécutables.
- La taille du script est limitée à 128 Ko.
- Astra Control utilise les paramètres de crochet d’exécution et tout critère de correspondance pour déterminer quels crochets s’appliquent à un instantané.
- Toutes les défaillances de crochet d’exécution sont des défaillances logicielles ; d’autres crochets et l’instantané sont toujours tentées même en cas de défaillance d’un crochet. Cependant, lorsqu’un crochet échoue, un événement d’avertissement est enregistré dans le journal des événements de la page **activité**.
- Pour créer, modifier ou supprimer des crochets d’exécution, vous devez être un utilisateur disposant des autorisations propriétaire, administrateur ou membre.
- Si l’exécution d’un crochet d’exécution prend plus de 25 minutes, le crochet échoue, créant une entrée de

journal d'événements avec un code retour « N/A ». Tout instantané affecté expire et sera marqué comme ayant échoué, avec une entrée du journal des événements qui en résulte indiquant le délai d'attente.

> Puisque les crochets d'exécution réduisent souvent ou désactivent complètement la fonctionnalité de l'application contre laquelle ils sont en cours d'exécution, vous devez toujours essayer de réduire le temps d'exécution de vos crochets d'exécution personnalisés.

Lors de l'exécution d'un instantané, les événements de hook d'exécution ont lieu dans l'ordre suivant :

1. Tous les crochets d'exécution par défaut fournis par NetApp s'exécutent sur les conteneurs appropriés.
2. Tous les crochets d'exécution pré-snapshot personnalisés applicables sont exécutés sur les conteneurs appropriés. Vous pouvez créer et exécuter autant de crochets pré-snapshot personnalisés que vous le souhaitez, mais l'ordre d'exécution de ces crochets avant que le snapshot ne soit ni garanti ni configurable.
3. Le snapshot est effectué.
4. Tous les crochets d'exécution post-snapshot personnalisés applicables sont exécutés sur les conteneurs appropriés. Vous pouvez créer et exécuter autant de crochets post-snapshot personnalisés que vous le souhaitez, mais l'ordre d'exécution de ces crochets après le snapshot n'est ni garanti ni configurable.
5. Tous les crochets d'exécution post-snapshot par défaut fournis par NetApp s'exécutent sur les conteneurs appropriés.

> Vous devez toujours tester vos scripts d'exécution avant de les activer dans un environnement de production. Vous pouvez utiliser la commande 'kubectl exec' pour tester aisément les scripts. Une fois que vous avez activé les crochets d'exécution dans un environnement de production, testez les snapshots obtenus pour vous assurer qu'ils sont cohérents. Pour ce faire, vous pouvez cloner l'application dans un espace de noms temporaire, restaurer le snapshot, puis tester l'application.

# Afficher les crochets d'exécution existants

Vous pouvez afficher les crochets d'exécution d'une application personnalisés ou par défaut fournis par NetApp.

**Étapes**

1. Accédez à **applications**, puis sélectionnez le nom d'une application gérée.
2. Sélectionnez l'onglet **crochets d'exécution**.

   Vous pouvez afficher tous les crochets d'exécution activés ou désactivés dans la liste résultante. Vous pouvez voir l'état d'un crochet, sa source et le moment où il est exécuté (pré ou post-instantané). Pour afficher les journaux d'événements entourant les crochets d'exécution, accédez à la page **activité** dans la zone de navigation de gauche.

# Créer un crochet d'exécution personnalisé

Vous pouvez créer un crochet d'exécution personnalisé pour une application. Voir "Exemples de crochet d'exécution" pour des exemples de crochet. Vous devez disposer d'autorisations propriétaire, administrateur ou membre pour créer des crochets d'exécution.

> Lorsque vous créez un script de shell personnalisé à utiliser comme crochet d'exécution, n'oubliez pas de spécifier le shell approprié au début du fichier, sauf si vous exécutez des commandes linux ou fournissez le chemin complet à un exécutable.

**Étapes**

1. Sélectionnez **applications**, puis le nom d'une application gérée.
2. Sélectionnez l'onglet **crochets d'exécution**.
3. Sélectionnez **Ajouter un nouveau crochet**.
4. Dans la zone **Détails du crochet**, selon le moment où le crochet doit être exécuté, choisissez **Préinstantané** ou **Post-instantané**.
5. Saisissez un nom unique pour le crochet.
6. (Facultatif) saisissez les arguments à transmettre au crochet pendant l'exécution, en appuyant sur la touche entrée après chaque argument que vous entrez pour enregistrer chacun.
7. Dans la zone **Images conteneur**, si le crochet doit être exécuté sur toutes les images de conteneur contenues dans l'application, activez la case à cocher **appliquer à toutes les images de conteneur**. Si, à la place, le crochet ne doit agir que sur une ou plusieurs images de conteneur spécifiées, entrez les noms d'image de conteneur dans le champ **noms d'image de conteneur à associer**.
8. Dans la zone **script**, effectuez l'une des opérations suivantes :
   - Charger un script personnalisé.
     i. Sélectionnez l'option **Télécharger le fichier**.
     ii. Accédez à un fichier et téléchargez-le.
     iii. Donnez un nom unique au script.
     iv. (Facultatif) Entrez toutes les notes que les autres administrateurs doivent connaître au sujet du script.
   - Coller dans un script personnalisé à partir du presse-papiers.
     i. Sélectionnez l'option **Coller dans le presse-papiers**.
     ii. Sélectionnez le champ de texte et collez le texte du script dans le champ.
     iii. Donnez un nom unique au script.
     iv. (Facultatif) Entrez toutes les notes que les autres administrateurs doivent connaître au sujet du script.
9. Sélectionnez **Ajouter crochet**.

# Désactivez un crochet d'exécution

Vous pouvez désactiver un crochet d'exécution si vous souhaitez l'empêcher temporairement de s'exécuter avant ou après un instantané d'une application. Vous devez disposer d'autorisations propriétaire, administrateur ou membre pour désactiver les crochets d'exécution.

**Étapes**

1. Sélectionnez **applications**, puis le nom d'une application gérée.
2. Sélectionnez l'onglet **crochets d'exécution**.
3. Sélectionnez la liste déroulante **actions** pour un crochet que vous souhaitez désactiver.
4. Sélectionnez **Désactiver**.

# Supprimer un crochet d’exécution

Vous pouvez supprimer entièrement un crochet d’exécution si vous n’en avez plus besoin. Vous devez disposer d’autorisations propriétaire, administrateur ou membre pour supprimer les crochets d’exécution.

**Étapes**

1. Sélectionnez **applications**, puis le nom d’une application gérée.
2. Sélectionnez l’onglet **crochets d’exécution**.
3. Sélectionnez la liste déroulante **actions** pour un crochet que vous souhaitez supprimer.
4. Sélectionnez **Supprimer**.

# Exemples de crochet d’exécution

Utilisez les exemples suivants pour avoir une idée de la structure de vos crochets d’exécution. Vous pouvez utiliser ces crochets comme modèles ou comme scripts de test.

## Exemple de réussite simple

Voici un exemple de crochet simple qui réussit et écrit un message sur une sortie standard et une erreur standard.

```
#!/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
```

## Exemple de réussite simple (version bash)

Voici un exemple de crochet simple qui réussit et écrit un message sur une sortie standard et une erreur standard, écrit pour 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
```

## Exemple de réussite simple (version zsh)

Voici un exemple de crochet simple qui réussit et écrit un message sur une sortie standard et une erreur standard, écrite pour le shell Z.

```
#!/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
```

## Exemple de réussite avec les arguments

L’exemple suivant montre comment utiliser des args dans un crochet.

```
#!/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
```

## Exemple de crochet pré-instantané/post-instantané

L'exemple suivant montre comment le même script peut être utilisé à la fois pour un pré-snapshot et un crochet post-snapshot.

```
#!/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}
```

## Exemple de panne

L'exemple suivant montre comment vous pouvez gérer les défaillances d'un crochet.

```
#!/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}
```

## Exemple détaillé d’échec

L’exemple suivant montre comment gérer les défaillances d’un crochet, avec une consignation plus détaillée.

```
#!/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
```

## Échec avec un exemple de code de sortie

L'exemple suivant illustre l'échec d'un crochet avec un code de sortie.

```
#!/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}
```

## Exemple de succès après échec

L'exemple suivant illustre l'échec d'un crochet lors de sa première exécution, mais la réussite après la seconde course.

```
#!/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
```

**Informations sur le copyright**

Copyright © 2023 NetApp, Inc. Tous droits réservés. Imprimé aux États-Unis. Aucune partie de ce document protégé par copyright ne peut être reproduite sous quelque forme que ce soit ou selon quelque méthode que ce soit (graphique, électronique ou mécanique, notamment par photocopie, enregistrement ou stockage dans un système de récupération électronique) sans l'autorisation écrite préalable du détenteur du droit de copyright.

Les logiciels dérivés des éléments NetApp protégés par copyright sont soumis à la licence et à l'avis de non-responsabilité suivants :

CE LOGICIEL EST FOURNI PAR NETAPP « EN L'ÉTAT » ET SANS GARANTIES EXPRESSES OU TACITES, Y COMPRIS LES GARANTIES TACITES DE QUALITÉ MARCHANDE ET D'ADÉQUATION À UN USAGE PARTICULIER, QUI SONT EXCLUES PAR LES PRÉSENTES. EN AUCUN CAS NETAPP NE SERA TENU POUR RESPONSABLE DE DOMMAGES DIRECTS, INDIRECTS, ACCESSOIRES, PARTICULIERS OU EXEMPLAIRES (Y COMPRIS L'ACHAT DE BIENS ET DE SERVICES DE SUBSTITUTION, LA PERTE DE JOUISSANCE, DE DONNÉES OU DE PROFITS, OU L'INTERRUPTION D'ACTIVITÉ), QUELLES QU'EN SOIENT LA CAUSE ET LA DOCTRINE DE RESPONSABILITÉ, QU'IL S'AGISSE DE RESPONSABILITÉ CONTRACTUELLE, STRICTE OU DÉLICTUELLE (Y COMPRIS LA NÉGLIGENCE OU AUTRE) DÉCOULANT DE L'UTILISATION DE CE LOGICIEL, MÊME SI LA SOCIÉTÉ A ÉTÉ INFORMÉE DE LA POSSIBILITÉ DE TELS DOMMAGES.

NetApp se réserve le droit de modifier les produits décrits dans le présent document à tout moment et sans préavis. NetApp décline toute responsabilité découlant de l'utilisation des produits décrits dans le présent document, sauf accord explicite écrit de NetApp. L'utilisation ou l'achat de ce produit ne concède pas de licence dans le cadre de droits de brevet, de droits de marque commerciale ou de tout autre droit de propriété intellectuelle de NetApp.

Le produit décrit dans ce manuel peut être protégé par un ou plusieurs brevets américains, étrangers ou par une demande en attente.

LÉGENDE DE RESTRICTION DES DROITS : L'utilisation, la duplication ou la divulgation par le gouvernement sont sujettes aux restrictions énoncées dans le sous-paragraphe (b)(3) de la clause Rights in Technical Data-Noncommercial Items du DFARS 252.227-7013 (février 2014) et du FAR 52.227-19 (décembre 2007).

Les données contenues dans les présentes se rapportent à un produit et/ou service commercial (tel que défini par la clause FAR 2.101). Il s'agit de données propriétaires de NetApp, Inc. Toutes les données techniques et tous les logiciels fournis par NetApp en vertu du présent Accord sont à caractère commercial et ont été exclusivement développés à l'aide de fonds privés. Le gouvernement des États-Unis dispose d'une licence limitée irrévocable, non exclusive, non cessible, non transférable et mondiale. Cette licence lui permet d'utiliser uniquement les données relatives au contrat du gouvernement des États-Unis d'après lequel les données lui ont été fournies ou celles qui sont nécessaires à son exécution. Sauf dispositions contraires énoncées dans les présentes, l'utilisation, la divulgation, la reproduction, la modification, l'exécution, l'affichage des données sont interdits sans avoir obtenu le consentement écrit préalable de NetApp, Inc. Les droits de licences du Département de la Défense du gouvernement des États-Unis se limitent aux droits identifiés par la clause 252.227-7015(b) du DFARS (février 2014).

**Informations sur les marques commerciales**

NETAPP, le logo NETAPP et les marques citées sur le site http://www.netapp.com/TM sont des marques déposées ou des marques commerciales de NetApp, Inc. Les autres noms de marques et de produits sont des marques commerciales de leurs propriétaires respectifs.