アプリケーション実行フックを管理します

Astra Control Center

NetApp
November 21, 2023

目次

アプリケーション実行フックを管理します

実行フックは、管理対象アプリケーションのスナップショットの前または後に実行できるカスタムスクリプトです。たとえば、データベースアプリケーションがある場合、実行フックを使用して、スナップショットの前にすべてのデータベーストランザクションを一時停止し、スナップショットの完了後にトランザクションを再開できます。これにより、アプリケーションと整合性のある Snapshot を作成できます。

デフォルトの実行フックと正規表現

一部のアプリケーションでは、ネットアップが提供するデフォルトの実行フックが Astra Control に付属しており、スナップショットの前後にフリーズや再開の操作を処理します。Astra Control では、正規表現を使用して、アプリケーションのコンテナイメージを次のアプリケーションに照合します。

- MariaDB
  - 正規表現 \bmariadb\b に一致しています
- MySQL
  - 正規表現 \bmysqql\b に一致しています
- PostgreSQL
  - 正規表現 \bpostgresql\b と一致します

一致した場合は、そのアプリケーションのデフォルトの実行フックがアプリケーションのアクティブな実行フックのリストに表示され、そのアプリケーションのスナップショットが作成されると、それらのフックが自動的に実行されます。カスタムアプリケーションの 1 つに、正規表現の 1 つと一致するように表示されるイメージ名が似ている場合（デフォルトの実行フックを使用しない場合）、イメージ名を変更することができます。 または、そのアプリケーションのデフォルト実行フックを無効にして、代わりにカスタムフックを使用します。

デフォルトの実行フックを削除または変更することはできません。

カスタム実行フックに関する重要な注意事項

アプリケーションの実行フックを計画するときは、次の点を考慮してください。

- Astra Control では、実行フックを実行可能なシェルスクリプトの形式で記述する必要があります。
- スクリプトサイズは 128KB に制限されています。
- Astra Control は、実行フックの設定と一致条件を使用して、スナップショットに適用できるフックを決定します。
- 実行フックの障害はすべてソフトな障害です他のフックとスナップショットは ' フックが失敗しても試行されますただし、フックが失敗すると、 * アクティビティ * ページイベントログに警告イベントが記録されます。
- 実行フックを作成、編集、または削除するには、 Owner 、 Admin 、または Member 権限を持つユーザーである必要があります。
- 実行フックの実行に 25 分以上かかる場合 ' フックは失敗し ' 戻りコードが N/A のイベント・ログ・エント

リが作成されます該当する Snapshot はタイムアウトして失敗とマークされ、タイムアウトを通知するイベントログエントリが生成されます。

> 実行フックは、実行中のアプリケーションの機能を低下させるか、完全に無効にすることが多いため、カスタム実行フックの実行時間を最小限に抑えるようにしてください。

スナップショットが実行されると、実行フックイベントが次の順序で実行されます。

1. ネットアップが提供するデフォルトの Snapshot 前実行フックは、該当するコンテナで実行されます。
2. 適用可能なカスタムスナップショット前実行フックは、適切なコンテナで実行されます。必要な数のカスタムスナップショット前フックを作成して実行できますが ' スナップショットの実行順序は保証も構成もされていません
3. スナップショットが実行されます。
4. 適用可能なカスタムスナップショット後実行フックは、適切なコンテナで実行されます。必要な数のカスタムスナップショット後フックを作成して実行できますが ' スナップショット後のこれらのフックの実行順序は保証されておらず ' 構成もできません
5. ネットアップが提供するデフォルトのポスト Snapshot 実行フックは、該当するコンテナで実行されます。

> 本番環境で実行スクリプトを有効にする前に、必ず実行フックスクリプトをテストしてください。'kubectl exec' コマンドを使用すると、スクリプトを簡単にテストできます。本番環境で実行フックを有効にしたら、作成されたスナップショットの整合性をテストします。これを行うには、アプリケーションを一時ネームスペースにクローニングし、スナップショットをリストアしてから、アプリケーションをテストします。

既存の実行フックを表示します

既存のカスタム実行フックまたはネットアップが提供するアプリケーションのデフォルト実行フックを表示できます。

手順

1. 「 * アプリケーション」に移動し、管理アプリの名前を選択します。
2. [実行フック *] タブを選択します。

   有効または無効になっているすべての実行フックを結果リストに表示できます。フックのステータス ' ソース ' および実行時（スナップショット前またはスナップショット後）を表示できます実行フックに関連するイベントログを表示するには、左側のナビゲーション領域の * アクティビティ * ページに移動します。

カスタム実行フックを作成します

アプリケーションのカスタム実行フックを作成できます。を参照してください "実行フックの例" フックの例を参照してください。実行フックを作成するには、 Owner 、 Admin 、または Member のいずれかの権限が必要です。

> 実行フックとして使用するカスタムシェルスクリプトを作成する場合は、 Linux コマンドを実行しているか、実行可能ファイルへの完全パスを提供している場合を除き、ファイルの先頭に適切なシェルを指定するようにしてください。

手順

1. 「 * アプリケーション」を選択し、管理アプリの名前を選択します。
2. [実行フック *] タブを選択します。
3. [* 新しいフックを追加 *] を選択します。
4. フックの詳細 * 領域で、フックを実行するタイミングに応じて、「 * Pre-Snapshot * 」または「 * Post-Snapshot * 」を選択します。
5. フックの一意の名前を入力します。
6. （オプション）実行中にフックに渡す引数を入力し、各引数を入力した後で Enter キーを押して、それぞれを記録します。
7. [* Container Images * （コンテナイメージ * ）] 領域で、アプリケーションに含まれるすべてのコンテナイメージに対してフックを実行する必要がある場合は、 [* Apply to all container images * （すべてのコンテナイメージに適用 * ）] チェックボックスを有効にします。代わりに、フックが 1 つ以上の指定されたコンテナイメージに対してのみ機能する場合は、 * Container image names to match * フィールドにコンテナイメージ名を入力します。
8. [* スクリプト * （ * Script * ）] 領域で、次のいずれかを実行します。
   - カスタムスクリプトをアップロードする。
     i. [ファイルのアップロード（ Upload file ）] オプションを選択します。
     ii. ファイルを参照してアップロードします。
     iii. スクリプトに一意の名前を付けます。
     iv. （オプション）他の管理者がスクリプトについて知っておく必要があるメモを入力します。
   - クリップボードからカスタムスクリプトを貼り付けます。
     i. クリップボードから貼り付け * オプションを選択します。
     ii. テキストフィールドを選択し、スクリプトテキストをフィールドに貼り付けます。
     iii. スクリプトに一意の名前を付けます。
     iv. （オプション）他の管理者がスクリプトについて知っておく必要があるメモを入力します。
9. [* フックを追加 *] を選択します。

実行フックを無効にします

アプリケーションのスナップショットの前または後に実行を一時的に禁止する場合は、実行フックを無効にできます。実行フックを無効にするには、 Owner 、 Admin 、または Member のいずれかの権限が必要です。

手順

1. 「 * アプリケーション」を選択し、管理アプリの名前を選択します。
2. [実行フック *] タブを選択します。
3. 無効にするフックの * アクション * 列のオプションメニューを選択します。

4. [**Disable**] を選択します。

実行フックを削除します

不要になった実行フックは完全に削除できます。実行フックを削除するには、 Owner 、 Admin 、または Member のいずれかの権限が必要です。

手順

1. 「 * アプリケーション」を選択し、管理アプリの名前を選択します。
2. [実行フック *] タブを選択します。
3. 削除するフックの * アクション * 列のオプションメニューを選択します。
4. 「 * 削除」を選択します。

実行フックの例

次の例を使用して、実行フックの構造を確認してください。これらのフックは、テンプレートまたはテストスクリプトとして使用できます。

シンプルな成功例

次に、成功し、標準出力および標準エラーにメッセージを書き込む単純フックの例を示します。

```
#!/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
```

シンプルな成功の例（ **bash** バージョン）

次に、 bash 用に書かれた標準出力と標準エラーにメッセージを書き込む単純なフックの例を示します。

```
#!/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
```

単純な成功例（ **zsh** バージョン）

これは、成功した単純なフックの例であり、標準出力と標準エラーに Z シェル用に記述されたメッセージを書き込みます。

```
#!/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
```

引数を指定した成功の例

次の例は、フックで args を使用する方法を示しています。

```
#!/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
```

スナップショット前 / スナップショット後のフックの例

次の例は、 Snapshot 前フックと Snapshot 後フックの両方に同じスクリプトを使用する方法を示しています。

```
#!/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}
```

失敗の例

次の例は、フックで障害を処理する方法を示しています。

```
#!/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}
```

詳細なエラーの例

次の例では ' フックの失敗をより詳細なロギングで処理する方法を示します

```
#!/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
```

終了コード例を使用した失敗

次の例は、終了コードを使用したフックの失敗を示しています。

```
#!/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}
```

失敗後の成功の例

次の例では、最初の実行時にフックが失敗していますが、 2 回目の実行後に成功しています。

```
#!/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
```