管理应用程序执行挂钩

Astra Control Center

NetApp
June 06, 2024

目录

管理应用程序执行挂钩

执行钩是一种自定义脚本，您可以在托管应用程序快照之前或之后运行该脚本。例如，如果您有一个数据库应用程序，则可以使用执行挂钩在快照之前暂停所有数据库事务，并在快照完成后恢复事务。这样可以确保应用程序一致的快照。

默认执行挂钩和正则表达式

对于某些应用程序， Astra Control 附带了 NetApp 提供的默认执行挂钩，用于处理快照前后的冻结和解冻操作。Astra Control 使用正则表达式将应用程序的容器映像与以下应用程序匹配：

- MariaDB
  - 匹配正则表达式： \bmariadb\b
- MySQL
  - 匹配正则表达式： \bmysql\b
- PostgreSQL
  - 匹配正则表达式： \bpostgresql\b

如果存在匹配项，则 NetApp 为该应用程序提供的默认执行挂钩将显示在该应用程序的活动执行挂钩列表中，这些挂钩将在该应用程序创建快照时自动运行。如果某个自定义应用程序的映像名称类似，恰好与其中一个正则表达式匹配（并且您不想使用默认执行挂钩），则可以更改映像名称， 或者禁用该应用程序的默认执行连接，而改用自定义连接。

您不能删除或修改默认执行挂钩。

有关自定义执行挂钩的重要注意事项

在为应用程序规划执行挂钩时，请考虑以下几点。

- Astra Control 要求以可执行 Shell 脚本的格式编写执行挂钩。
- 脚本大小限制为 128 KB 。
- Astra Control 使用执行挂钩设置和任何匹配条件来确定哪些挂钩适用于快照。
- 所有执行挂机故障均为软故障；即使某个挂机发生故障，仍会尝试使用其他挂机和快照。但是，如果挂机发生故障，则会在 * 活动 * 页面事件日志中记录一个警告事件。
- 要创建，编辑或删除执行挂钩，您必须是具有所有者，管理员或成员权限的用户。
- 如果执行挂机运行时间超过 25 分钟，则此挂机将失败，从而创建返回代码为不适用的事件日志条目。任何受影响的快照都将超时并标记为失败，并会生成一个事件日志条目，用于记录超时情况。

> 由于执行挂钩通常会减少或完全禁用其所运行的应用程序的功能，因此您应始终尽量缩短自定义执行挂钩运行所需的时间。

运行快照时，执行钩事件按以下顺序发生：

1. NetApp 提供的任何适用的默认快照前执行挂钩都会在相应的容器上运行。

2. 任何适用的自定义快照前执行挂钩都会在相应的容器上运行。您可以根据需要创建和运行任意数量的自定义预快照挂钩，但在创建快照之前执行这些挂钩的顺序既不能保证也不可配置。
3. 执行快照。
4. 任何适用的自定义快照后执行挂钩都会在相应的容器上运行。您可以根据需要创建和运行任意数量的自定义快照后挂钩，但这些挂钩在快照后的执行顺序既不能保证也不可配置。
5. NetApp 提供的任何适用的默认快照后执行挂钩都会在相应的容器上运行。

> 在生产环境中启用执行钩脚本之前，应始终对其进行测试。您可以使用 "kubectl exec" 命令方便地测试脚本。在生产环境中启用执行挂钩后，测试生成的快照以确保其一致。为此，您可以将应用程序克隆到临时命名空间，还原快照，然后测试应用程序。

查看现有执行挂钩

您可以查看应用程序的现有自定义或 NetApp 提供的默认执行挂钩。

步骤

1. 转到 * 应用程序 * ，然后选择受管应用程序的名称。
2. 选择 * 执行挂钩 * 选项卡。

   您可以在显示的列表中查看所有已启用或已禁用的执行挂钩。您可以查看挂机的状态，源以及运行时间（快照前或快照后）。要查看与执行挂钩相关的事件日志，请转到左侧导航区域中的 * 活动 * 页面。

创建自定义执行挂钩

您可以为应用程序创建自定义执行挂钩。请参见 "执行钩示例" 有关挂机示例。要创建执行挂钩，您需要拥有所有者，管理员或成员权限。

> 创建用作执行挂钩的自定义 Shell 脚本时，请务必在文件开头指定适当的 shell ，除非您正在运行 Linux 命令或提供可执行文件的完整路径。

步骤

1. 选择 * 应用程序 * ，然后选择受管应用程序的名称。
2. 选择 * 执行挂钩 * 选项卡。
3. 选择 * 添加新挂钩 * 。
4. 在 * 挂机详细信息 * 区域中，根据挂机应运行的时间，选择 * 预 Snapshot * 或 * 后 Snapshot * 。
5. 输入此挂钩的唯一名称。
6. （可选）输入执行期间传递到挂机的任何参数，在输入的每个参数之后按 Enter 键以记录每个参数。
7. 在 * 容器映像 * 区域中，如果此挂钩应针对应用程序中包含的所有容器映像运行，请启用 * 应用于所有容器映像 * 复选框。如果该挂钩只能作用于一个或多个指定的容器映像，请在 * 要匹配的容器映像名称 * 字段中输入容器映像名称。
8. 在 * 脚本 * 区域中，执行以下操作之一：
   - 上传自定义脚本。

    i. 选择 * 上传文件 * 选项。
    ii. 浏览到文件并上传。
    iii. 为脚本指定一个唯一名称。
    iv. （可选）输入其他管理员应了解的有关该脚本的任何注释。
  - 从剪贴板粘贴到自定义脚本中。
    i. 选择 * 从剪贴板粘贴 * 选项。
    ii. 选择文本字段并将脚本文本粘贴到字段中。
    iii. 为脚本指定一个唯一名称。
    iv. （可选）输入其他管理员应了解的有关该脚本的任何注释。
9. 选择 * 添加挂钩 * 。

禁用执行挂钩

如果要暂时阻止执行挂钩在应用程序快照之前或之后运行，可以禁用执行挂钩。要禁用执行挂钩，您需要拥有所有者，管理员或成员权限。

步骤

1. 选择 * 应用程序 * ，然后选择受管应用程序的名称。
2. 选择 * 执行挂钩 * 选项卡。
3. 在 * 操作 * 列中选择要禁用的挂机的选项菜单。
4. 选择 * 禁用 * 。

删除执行挂钩

如果您不再需要执行挂钩，则可以将其完全移除。要删除执行挂钩，您需要拥有所有者，管理员或成员权限。

步骤

1. 选择 * 应用程序 * ，然后选择受管应用程序的名称。
2. 选择 * 执行挂钩 * 选项卡。
3. 在 * 操作 * 列中选择要删除的挂机的选项菜单。
4. 选择 * 删除 * 。

执行钩示例

使用以下示例了解如何构建执行挂钩。您可以将这些挂钩用作模板或测试脚本。

简单的成功示例

这是一个简单的钩子示例，它成功地将消息写入标准输出和标准错误。

```
#!/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
```

简单成功示例（ bash 版本）

这是一个简单的钩子示例，该钩子成功地将消息写入标准输出和标准错误，并写入 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
```

简单成功示例（ **zsh** 版本）

这是一个简单的钩子示例，该钩子成功地将消息写入标准输出和标准错误，并写入 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
```

成功使用参数示例

以下示例演示了如何在挂机中使用 args 。

```
#!/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
```

快照前 / 快照后挂钩示例

以下示例演示了如何对快照前和快照后挂钩使用同一脚本。

```
#!/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}
```

故障示例

以下示例演示了如何处理挂机故障。

```
#!/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}
```

详细故障示例

以下示例演示了如何处理挂机故障，并提供更详细的日志记录。

```
#!/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
```

退出代码示例失败

以下示例显示了一个连接失败并显示退出代码。

```
#!/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}
```

失败后成功示例

以下示例显示了首次运行时发生故障的挂钩，但在第二次运行后仍会成功。

```
#!/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
```