



Entwickeln Sie ein Plug-in für Ihre Applikation

SnapCenter Software 4.5

NetApp
January 18, 2024

This PDF was generated from https://docs.netapp.com/de-de/snapcenter-45/protect-scc/concept_develop_a_plug_in_for_your_application.html on January 18, 2024. Always check docs.netapp.com for the latest.

Inhalt

- Entwickeln Sie ein Plug-in für Ihre Applikation 1
 - Überblick 1
 - PERL-basierte Entwicklung 3
 - NATIVER Stil 12
 - Java-Stil 15
 - Benutzerdefiniertes Plug-in in SnapCenter 23

Entwickeln Sie ein Plug-in für Ihre Applikation

Überblick

Der SnapCenter Server ermöglicht die Implementierung und das Management von Applikationen als Plug-ins für SnapCenter. Die Applikationen Ihrer Wahl können zur Datensicherung und zu Managementfunktionen in den SnapCenter Server integriert werden.

Mit SnapCenter können Sie benutzerdefinierte Plug-ins mit unterschiedlichen Programmiersprachen entwickeln. Sie können ein benutzerdefiniertes Plug-in über Perl, Java, BATCH oder andere Skriptsprachen entwickeln.

Um benutzerdefinierte Plug-ins in SnapCenter zu verwenden, müssen Sie die folgenden Aufgaben ausführen:

- Erstellen Sie ein Plug-in für Ihre Anwendung mithilfe der Anweisungen in diesem Handbuch
- Erstellen Sie eine Beschreibungsdatei
- Exportieren Sie das benutzerdefinierte Plug-in, um es auf dem SnapCenter-Host zu installieren
- Laden Sie die Plug-in-ZIP-Datei auf den SnapCenter-Server hoch

Allgemeine Plug-in-Bearbeitung bei allen API-Aufrufen

Verwenden Sie für jeden API-Aufruf die folgenden Informationen:

- Plug-in-Parameter
- Exit-Codes
- Fehlermeldungen protokollieren
- Datenkonsistenz

Verwenden Sie Plug-in-Parameter

Bei jedem API-Aufruf wird eine Reihe von Parametern an das Plug-in übergeben. In der folgenden Tabelle sind die spezifischen Informationen für die Parameter aufgeführt.

Parameter	Zweck
AKTION	Legt den Workflow-Namen fest. Entdecken Sie zum Beispiel, sichern, fileOrVolRestore oder cloneVolAndRun
RESSOURCEN	Listet Ressourcen auf, die geschützt werden sollen. Eine Ressource wird durch UID und Typ identifiziert. Die Liste wird dem Plug-in im folgenden Format angezeigt: „<UID>,<TYPE>;<UID>,<TYPE>“. Beispiel: „Instance1,instance;Instance2\\DB1,Database“

Parameter	Zweck
APP_NAME	Bestimmt, welches Plug-in verwendet wird. Zum Beispiel DB2, MYSQL. SnapCenter Server bietet integrierte Unterstützung für die aufgeführten Applikationen. Bei diesem Parameter wird die Groß-/Kleinschreibung beachtet.
APP_IGNORE_ERROR	(J oder N) Dies führt dazu, dass SnapCenter beendet wird oder nicht beendet wird, wenn ein Anwendungsfehler auftritt. Dies ist nützlich, wenn Sie Backups mehrerer Datenbanken durchführen und nicht möchten, dass der Backup-Vorgang durch einen einzelnen Fehler unterbrochen wird.
<RESOURCE_NAME>__APP_INSTANCE_USERNAME	SnapCenter-Anmeldeinformationen für die Ressource festgelegt.
<RESOURCE_NAME>_APP_INSTANCE_PASSWORD	SnapCenter-Anmeldeinformationen für die Ressource festgelegt.
<RESOURCE_NAME>_<CUSTOM_PARAM>	Jeder benutzerdefinierte Schlüsselwert auf Ressourcenebene steht für Plug-ins zur Verfügung, die mit „<RESOURCE_NAME>_“ vordefiniert sind. Wenn ein benutzerdefinierter Schlüssel beispielsweise „MASTER_SLAVE“ für eine Ressource namens „MySQLDB“ ist, wird er als MySQLDB_MASTER_SLAVE verfügbar sein

Exit-Codes verwenden

Das Plug-in gibt den Status des Vorgangs über Exit-Codes zurück an den Host. Jeder Code hat eine bestimmte Bedeutung, und das Plug-in verwendet den richtigen Exit-Code, um die gleiche anzuzeigen.

Die folgende Tabelle zeigt Fehlercodes und deren Bedeutung.

Beenden Sie den Code	Zweck
0	Erfolgreicher Betrieb.
99	Der angeforderte Vorgang wird nicht unterstützt oder implementiert.
100	Fehlgeschlagener Vorgang, unquiesce überspringen und beenden. Deaktivieren ist standardmäßig.
101	Fehlgeschlagener Vorgang; fahren Sie mit dem Backup-Vorgang fort.

Beenden Sie den Code	Zweck
Andere	Vorgang fehlgeschlagen, Ausführung unquiesce und Beenden.

Fehlermeldungen protokollieren

Die Fehlermeldungen werden vom Plug-in an den SnapCenter-Server übergeben. Die Meldung enthält die Meldung, die Protokollebene und den Zeitstempel.

In der folgenden Tabelle sind die Ebenen und ihre Zwecke aufgeführt.

Parameter	Zweck
INFO	Informationsmeldung
WARNEN	Warnmeldung
FEHLER	Fehlermeldung
DEBUGGEN	Debug-Nachricht
VERFOLGEN	Trace-Nachricht

Wahrung der Datenkonsistenz

Benutzerdefinierte Plug-ins bewahren Daten zwischen Operationen derselben Workflow-Ausführung auf. Ein Plug-in kann zum Beispiel Daten am Ende der Stilllegung speichern, die auch während des Betriebs ausgeführt werden können.

Die zu haltenden Daten werden als Teil des Ergebnisobjekts durch Plug-in festgelegt. Es folgt einem bestimmten Format und wird im Detail unter jedem Stil der Plug-in-Entwicklung beschrieben.

PERL-basierte Entwicklung

Bei der Entwicklung des Plug-ins mittels PERL müssen Sie bestimmte Konventionen beachten.

- Inhalte müssen lesbar sein
- Obligatorische Operationen setenv, quiesce und unquiesce implementieren müssen
- Sie müssen eine bestimmte Syntax verwenden, um die Ergebnisse an den Agenten weiterzuleiten
- Der Inhalt sollte als <PLUGIN_NAME>.pm Datei gespeichert werden

Verfügbare Operationen sind

- Setenv
- Version

- Stilllegen
- Unquiesce
- Clone_Pre, Clone_Post
- Restore_Pre, Restore
- Bereinigung

Allgemeine Plug-in-Handhabung

Verwenden des Ergebnisobjekts

Jeder benutzerdefinierte Plug-in-Vorgang muss das Ergebnisobjekt definieren. Dieses Objekt sendet Nachrichten, Exit Code, stdout und stderr zurück an den Host-Agent.

Ergebnisobjekt:

```
my $result = {
```

```
    exit_code => 0,
    stdout => "",
    stderr => "",
};
```

Ergebnisobjekt wird zurückgegeben:

```
return $result;
```

Wahrung der Datenkonsistenz

Es ist möglich, Daten zwischen den Operationen (außer Bereinigung) als Teil der gleichen Workflow-Ausführung zu erhalten. Dies erfolgt mit Schlüsselwert-Paaren. Die Daten werden als Teil des Ergebnisobjekts festgelegt und sind in den nachfolgenden Operationen desselben Workflows erhalten und verfügbar.

Im folgenden Codebeispiel werden die zu haltenden Daten festgelegt:

```
my $result = {
    exit_code => 0,
    stdout => "",
    stderr => "",
};
$result->{env}->{'key1'} = 'value1';
$result->{env}->{'key2'} = 'value2';
...
return $result
```

Der obige Code setzt zwei Schlüssel-Wert-Paare, die als Eingabe in der nachfolgenden Operation zur Verfügung stehen. Auf die beiden Schlüsselwertware kann über folgenden Code zugegriffen werden:

```
sub setENV {  
    my ($self, $config) = @_;  
    my $first_value = $config->{'key1'};  
    my $second_value = $config->{'key2'};  
    ...  
}
```

=== Logging error messages

Jeder Vorgang kann Nachrichten an den Host-Agent senden, der den Inhalt anzeigt und speichert. Eine Nachricht enthält die Nachrichtenebene, einen Zeitstempel und einen Nachrichtentext. Mehrzeilare Nachrichten werden unterstützt.

Load the SnapCreator::Event Class:

```
my $msgObj = new SnapCreator::Event();  
my @message_a = ();
```

Verwenden Sie msgObj, um eine Nachricht mithilfe der Erfassungsmethode zu erfassen.


```
$msgObj->collect(\@message_a, INFO, "My INFO Message");  
$msgObj->collect(\@message_a, WARN, "My WARN Message");  
$msgObj->collect(\@message_a, ERROR, "My ERROR Message");  
$msgObj->collect(\@message_a, DEBUG, "My DEBUG Message");  
$msgObj->collect(\@message_a, TRACE, "My TRACE Message");
```


Meldungen auf das Ergebnisobjekt anwenden:

```
$result->{message} = \@message_a;
```

Verwendung von Plug-in-Stiften

Benutzerdefinierte Plug-ins müssen Plug-in-Stiche enthalten. Dies sind Methoden, die der SnapCenter-Server auf Grundlage eines Workflows aufruft.

Steckschraube	Optional/Erforderlich	Zweck
Setenv	Erforderlich	<p>Dieser Stub legt die Umgebung und das Konfigurationsobjekt fest.</p> <p>Hier sollte eine Analyse oder Handhabung der Umgebung durchgeführt werden. Jedes Mal, wenn ein Stub genannt wird, wird die setenv stub kurz zuvor aufgerufen. Es ist nur für PERL-Plug-ins erforderlich.</p>
Version	Optional	Diese Stub wird verwendet, um die Anwendungsversion zu erhalten.
Ermitteln	Optional	<p>Mit diesem Stub werden Anwendungsobjekte wie beispielsweise die auf dem Agenten oder Host gehostete Instanz oder Datenbank ermittelt.</p> <p>Das Plug-in wird voraussichtlich im Rahmen der Antwort erkannte Applikationsobjekte in einem bestimmten Format zurückgeben. Dieser Stub wird nur verwendet, wenn die Anwendung in SnapDrive für Unix integriert ist.</p> <div>  <p>Linux-Dateisystem (Linux-Varianten) wird unterstützt. AIX/Solaris (Unix-Varianten) werden nicht unterstützt.</p> </div>

Steckschraube	Optional/Erforderlich	Zweck
Discovery_complete	Optional	<p>Mit diesem Stub werden Anwendungsobjekte wie beispielsweise die auf dem Agenten oder Host gehostete Instanz oder Datenbank ermittelt.</p> <p>Das Plug-in wird voraussichtlich im Rahmen der Antwort erkannte Applikationsobjekte in einem bestimmten Format zurückgeben. Dieser Stub wird nur verwendet, wenn die Anwendung in SnapDrive für Unix integriert ist.</p> <div>  <p>Linux-Dateisystem (Linux-Varianten) wird unterstützt. AIX und Solaris (Unix-Varianten) werden nicht unterstützt.</p> </div>
Stilllegen	Erforderlich	<p>Diese Stub ist für das Ausführen eines Quiesce zuständig, das bedeutet, die Applikation in einen Zustand zu legen, in dem Sie eine Snapshot Kopie erstellen können. Dies wird vor dem Vorgang der Snapshot Kopie aufgerufen. Die zu behaltenden Metadaten der Applikation sollten im Rahmen der Antwort festgelegt werden, die bei einem nachfolgenden Klon oder bei Wiederherstellungen auf der entsprechenden Storage Snapshot Kopie in Form von Konfigurationsparametern zurückgegeben werden müssen.</p>
Nicht Stilllegen	Erforderlich	<p>Diese Stub ist verantwortlich für die Durchführung eines Unquiesce, das bedeutet, Anwendung in einen normalen Zustand. Dies wird aufgerufen, nachdem Sie eine Snapshot Kopie erstellt haben.</p>

Steckschraube	Optional/Erforderlich	Zweck
Clone_Pre	Optional	Diese Stub ist für das Durchführen von Preclone-Aufgaben zuständig. Voraussetzung dafür ist, dass Sie die integrierte SnapCenter Server Klonchnittstelle verwenden und beim Ausführen eines Klonvorgangs ausgelöst wird.
Clone_Post	Optional	Diese Stub ist für das Durchführen von Aufgaben nach dem Klonen verantwortlich. Hierbei wird vorausgesetzt, dass Sie die integrierte SnapCenter Server Klonchnittstelle verwenden und nur beim Ausführen eines Klonvorgangs ausgelöst wird.
Wiederherstellen_Pre	Optional	Diese Stub ist für die Durchführung von Vorratstore-Aufgaben zuständig. Hierbei wird vorausgesetzt, dass Sie die integrierte SnapCenter Server Restore-Schnittstelle verwenden und während der Wiederherstellung ausgelöst werden.
Wiederherstellen	Optional	Diese Stub ist für die Durchführung von Aufgaben zur Wiederherstellung von Anwendungen verantwortlich. Hierbei wird vorausgesetzt, dass Sie die integrierte SnapCenter Server-Wiederherstellungsschnittstelle verwenden und nur bei der Durchführung einer Wiederherstellung ausgelöst wird.

Steckschraube	Optional/Erforderlich	Zweck
Bereinigung	Optional	Diese Stub ist für die Durchführung der Bereinigung nach Backup-, Wiederherstellungs- oder Klonvorgängen verantwortlich. Die Bereinigung kann während der normalen Workflow-Ausführung oder bei einem Workflow-Ausfall erfolgen. Sie können den Workflow-Namen infilern, unter dem die Bereinigung aufgerufen wird, indem Sie auf die Konfiguration Parameter AKTION, die Backup, KlonVolAndLun oder fileOrVolRestore sein kann. Der Konfigurationsparameter ERROR_MESSAGE gibt an, ob beim Ausführen des Workflows Fehler aufgetreten sind. Wenn ERROR_MESSAGE definiert ist und NICHT Null, wird die Bereinigung während der Ausführung des Workflow-Fehlers aufgerufen.
App_Version	Optional	Diese Stub wird von SnapCenter verwendet, um Anwendungsversionsdetails zu erhalten, die vom Plug-in verwaltet werden.

Informationen zum Plug-in-Paket

Jedes Plug-in muss folgende Informationen haben:

```

package MOCK;
our @ISA = qw(SnapCreator::Mod);
=head1 NAME
MOCK - class which represents a MOCK module.
=cut
=head1 DESCRIPTION
MOCK implements methods which only log requests.
=cut
use strict;
use warnings;
use diagnostics;
use SnapCreator::Util::Generic qw ( trim isEmpty );
use SnapCreator::Util::OS qw ( isWindows isUnix getUid
createTmpFile );
use SnapCreator::Event qw ( INFO ERROR WARN DEBUG COMMENT ASUP
CMD DUMP );
my $msgObj = new SnapCreator::Event();
my %config_h = ();

```

Betrieb

Sie können verschiedene Vorgänge wie Setenv, Version, Quiesce und Unquiesce codieren, die von den benutzerdefinierten Plug-ins unterstützt werden.

Vorgang setenv

Für Plug-ins, die mit PERL erstellt wurden, ist die setenv-Operation erforderlich. Sie können die ENV einstellen und problemlos auf Plug-in-Parameter zugreifen.

```

sub setENV {
    my ($self, $obj) = @_;
    %config_h = %{$obj};
    my $result = {
        exit_code => 0,
        stdout => "",
        stderr => "",
    };
    return $result;
}

```

Versionsbetrieb

Der Versionsvorgang gibt die Versionsinformationen der Anwendung zurück.

```

sub version {
    my $version_result = {
        major => 1,
        minor => 2,
        patch => 1,
        build => 0
    };
    my @message_a = ();
    $msgObj->collect(\@message_a, INFO, "VOLUMES
$config_h{'VOLUMES'}");
    $msgObj->collect(\@message_a, INFO,
"$config_h{'APP_NAME'}::quiesce");
    $version_result->{message} = \@message_a;
    return $version_result;
}

```

Betrieb stilllegen

Der Quiesce-Vorgang führt einen Quiesce-Vorgang der Anwendung für Ressourcen durch, die im PARAMETER RESSOURCEN aufgeführt sind.

```

sub quiesce {
    my $result = {
        exit_code => 0,
        stdout => "",
        stderr => "",
    };
    my @message_a = ();
    $msgObj->collect(\@message_a, INFO, "VOLUMES
$config_h{'VOLUMES'}");
    $msgObj->collect(\@message_a, INFO,
"$config_h{'APP_NAME'}::quiesce");
    $result->{message} = \@message_a;
    return $result;
}

```

Vorgang nicht stilllegen

Um die Anwendung stillzulegen, muss der Vorgang nicht stillgelegt werden. Die Liste der Ressourcen ist im PARAMETER RESSOURCEN verfügbar.

```

sub unquiesce {
    my $result = {
        exit_code => 0,
        stdout => "",
        stderr => "",
    };
    my @message_a = ();
    $msgObj->collect(\@message_a, INFO, "VOLUMES
$config_h{'VOLUMES'}");
    $msgObj->collect(\@message_a, INFO,
"$config_h{'APP_NAME'}::unquiesce");
    $result->{message} = \@message_a;
    return $result;
}

```

NATIVER Stil

SnapCenter unterstützt zur Erstellung von Plug-ins nicht-PERL-Programmierung oder Skriptsprachen. Dies wird als NATIVE Stil-Programmierung bekannt, die Skript- oder BATCH-Datei sein kann.

Die Plug-ins IM NATIVEN Stil müssen den folgenden Konventionen entsprechen:

Das Plug-in muss ausführbar sein

- Bei Unix-Systemen muss der Benutzer, der den Agenten ausführt, über Ausführungsberechtigungen auf dem Plug-in verfügen
- Bei Windows-Systemen müssen PowerShell-Plug-ins das Suffix .ps1 aufweisen, andere Windows-Skripte müssen entweder über ein Suffix .cmd oder .bat verfügen und müssen vom Benutzer ausführbar sein
- Die Plug-ins müssen auf Befehlszeilenargumente wie „-quiesce“, „-unquiesce“ reagieren
- Die Plug-ins müssen Exit Code 99 zurückgeben, falls keine Operation oder Funktion implementiert ist
- Die Plug-ins müssen eine bestimmte Syntax verwenden, um Ergebnisse an den Server weiterzuleiten

Allgemeine Plug-in-Handhabung

Protokollieren von Fehlermeldungen

Jeder Vorgang kann Nachrichten an den Server senden, der den Inhalt anzeigt und speichert. Eine Nachricht enthält die Nachrichtenebene, einen Zeitstempel und einen Nachrichtentext. Mehrzeilare Nachrichten werden unterstützt.

Format:

```

SC_MSG#<level>#<timestamp>#<message>
SC_MESSAGE#<level>#<timestamp>#<message>

```

Verwendung von Plug-in-Stiften

SnapCenter-Plug-ins müssen Plug-in-Stiche implementieren. Hierbei handelt es sich um Methoden, die der SnapCenter-Server basierend auf einem bestimmten Workflow aufruft.

Steckschraube	Optional/Erforderlich	Zweck
Stilllegen	Erforderlich	Diese Stub ist für die Durchführung eines Stillgeleges verantwortlich. Dadurch wird die Applikation in einen Zustand versetzt, in dem wir eine Snapshot Kopie erstellen können. Dies wird vor dem Vorgang der Snapshot-Kopie des Storage aufgerufen.
Unquiesce	Erforderlich	Diese Stub ist für die Durchführung eines Unstillzulegen verantwortlich. Die Anwendung wird in den normalen Zustand versetzt. Dies wird nach dem Vorgang der Snapshot Kopie des Storage aufgerufen.
Clone_Pre	Optional	Diese Stub ist für die Durchführung von Aufgaben vor dem Klonen verantwortlich. Angenommen, Sie verwenden die integrierte SnapCenter Klonschnittstelle und wird auch nur ausgelöst, wenn Sie die Aktion „Clone_vol oder Clone_lun“ ausführen.
Clone_Post	Optional	Diese Stub ist für das Durchführen von Aufgaben nach dem Klonen verantwortlich. Angenommen, Sie verwenden die integrierte SnapCenter Klonschnittstelle und wird auch nur ausgelöst, wenn Sie „Clone_vol oder Clone_lun“ Vorgänge ausführen.
Wiederherstellen_Pre	Optional	Diese Stub ist für die Durchführung von Aufgaben vor der Wiederherstellung verantwortlich. Angenommen, Sie verwenden die integrierte SnapCenter Restore-Schnittstelle und wird nur während des Restore-Vorgangs ausgelöst.

Steckschraube	Optional/Erforderlich	Zweck
Wiederherstellen	Optional	Diese Stub ist für alle Wiederherstellungsmaßnahmen verantwortlich. Dies setzt voraus, dass Sie keine integrierte Wiederherstellungsschnittstelle verwenden. Sie wird während des Wiederherstellungsvorgangs ausgelöst.

Beispiele

Windows PowerShell

Überprüfen Sie, ob das Skript auf Ihrem System ausgeführt werden kann. Wenn Sie das Skript nicht ausführen können, setzen Sie Set-ExecutionPolicy Bypass für das Skript ein, und wiederholen Sie den Vorgang.


```

if ($args.length -ne 1) {
    write-warning "You must specify a method";
    break;
}
function log ($level, $message) {
    $d = get-date
    echo "SC_MSG#$level#$d#$message"
}
function quiesce {
    $app_name = (get-item env:APP_NAME).value
    log "INFO" "Quiescing application using script $app_name";
    log "INFO" "Quiescing application finished successfully"
}
function unquiesce {
    $app_name = (get-item env:APP_NAME).value
    log "INFO" "Unquiescing application using script $app_name";
    log "INFO" "Unquiescing application finished successfully"
}
switch ($args[0]) {
    "-quiesce" {
        quiesce;
    }
    "-unquiesce" {
        unquiesce;
    }
    default {
        write-error "Function $args[0] is not implemented";
        exit 99;
    }
}
exit 0;

```

Java-Stil

Ein benutzerdefiniertes Java Plug-in interagiert direkt mit einer Applikation wie Datenbank, Instanz usw.

Einschränkungen

Es gibt bestimmte Einschränkungen, die Sie beim entwickeln eines Plug-ins mit Java-Programmiersprache beachten sollten.

Steckkennlinie	Java Plug-in
Komplexität	Niedrig bis mittel

Steckkennlinie	Java Plug-in
Platzbedarf für Arbeitsspeicher	Bis zu 10-20 MB
Abhängigkeiten von anderen Bibliotheken	Bibliotheken für die Anwendungskommunikation
Anzahl der Threads	1
Thread-Laufzeit	Weniger als eine Stunde

Grund für Java-Einschränkungen

Ziel des SnapCenter-Agenten ist es, eine kontinuierliche, sichere und robuste Anwendungsintegration zu gewährleisten. Durch die Unterstützung von Java-Plug-ins ist es möglich, dass Plug-ins Speicherlecks und andere unerwünschte Probleme einführen. Diese Probleme sind schwer zu bewältigen, vor allem, wenn das Ziel ist, die Dinge einfach zu bedienen. Wenn die Komplexität eines Plug-ins nicht zu komplex ist, ist es viel seltener wahrscheinlich, dass die Entwickler die Fehler verursacht hätten. Die Gefahr eines Java-Plug-ins besteht darin, dass sie in derselben JVM laufen wie der SnapCenter-Agent selbst. Wenn das Plug-in abstürzt oder Speicher leckt, kann es auch negative Auswirkungen auf den Agent haben.

Unterstützte Methoden

Methode	Erforderlich	Beschreibung	Wann und von wem gerufen?
Version	Ja.	Muss die Version des Plug-ins zurückgeben.	Vom SnapCenter-Server oder -Agent, um die Version des Plug-ins anzufordern.
Stilllegen	Ja.	Muss die Applikation stilllegen. In den meisten Fällen bedeutet dies, die Applikation in einen Zustand zu versetzen, in dem der SnapCenter Server ein Backup erstellen kann (z. B. eine Snapshot Kopie).	Bevor der SnapCenter-Server eine Snapshot(s)-Kopie erstellt oder ein Backup im Allgemeinen durchführt.
Nicht Stilllegen	Ja.	Muss die Applikation stilllegen. In den meisten Fällen bedeutet dies, dass die Applikation wieder in einen normalen Betriebszustand zurückversetzt wird.	Nachdem der SnapCenter Server eine Snapshot Kopie erstellt hat oder ein Backup im Allgemeinen ausgeführt hat.

Methode	Erforderlich	Beschreibung	Wann und von wem gerufen?
Bereinigung	Nein	Verantwortlich für die Reinigung von allen, die das Plug-in braucht, um zu reinigen.	Wenn ein Workflow auf dem SnapCenter-Server beendet ist (erfolgreich oder mit einem Fehler).
KlonVorr	Nein	Sollte Aktionen durchführen, die vor der Durchführung eines Klonvorgangs ausgeführt werden müssen.	Wenn ein Benutzer eine Aktion „KlonVol“ oder „cloneRun“ auslöst und den integrierten Klon-Assistenten (GUI/CLI) verwendet.
KlonPost	Nein	Sollte Aktionen durchführen, die nach der Durchführung eines Klonvorgangs ausgeführt werden müssen.	Wenn ein Benutzer eine Aktion „KlonVol“ oder „cloneRun“ auslöst und den integrierten Klon-Assistenten (GUI/CLI) verwendet.
WiederherstellungPre	Nein	Sollte Aktionen durchführen, die vor dem Wiederherstellungsvorgang ausgeführt werden müssen.	Wenn ein Benutzer einen Wiederherstellungsvorgang auslöst.
Wiederherstellen	Nein	Verantwortlich für die Durchführung einer Wiederherstellung/Wiederherstellung der Anwendung.	Wenn ein Benutzer einen Wiederherstellungsvorgang auslöst.
AppVersion	Nein	Zum Abrufen der vom Plug-in verwalteten Anwendungsversion.	Im Rahmen von ASUP Datenerfassung in jedem Workflow, wie beispielsweise Backup/Restore/Klonen,

Lernprogramm

In diesem Abschnitt wird beschrieben, wie Sie ein benutzerdefiniertes Plug-in mithilfe der Java-Programmiersprache erstellen.

Sonnenfinsternis einrichten

1. Erstellen Sie ein neues Java-Projekt "TutorialPlugin" in Eclipse
2. Klicken Sie Auf **Fertig Stellen**
3. Klicken Sie mit der rechten Maustaste auf das **neue Projekt** → **Eigenschaften** → **Java Build Path** →

Bibliotheken → Externe Jars hinzufügen

4. Navigieren Sie zum Ordner `../lib/` des Host Agent, und wählen Sie Jars `scAgent-5.0-core.jar` und `common-5.0.jar` aus
5. Wählen Sie das Projekt aus und klicken Sie mit der rechten Maustaste auf den Ordner **src** → **Neu** → **Paket** und erstellen Sie ein neues Paket mit dem Namen `com.netapp.snapcreator.agent.plugin.TutorialPlugin`
6. Klicken Sie mit der rechten Maustaste auf das neue Paket, und wählen Sie **Neu** → **Java-Klasse**.
 - a. Geben Sie den Namen als `TutorialPlugin` ein.
 - b. Klicken Sie auf die Schaltfläche zum Durchsuchen von Superclass und suchen Sie nach `"*AbstractPlugin"`. Es sollte nur ein Ergebnis angezeigt werden:

```
"AbstractPlugin - com.netapp.snapcreator.agent.nextgen.plugin".  
.. Klicken Sie Auf *Fertig Stellen*.  
.. Java-Klasse:
```

```

package com.netapp.snapcreator.agent.plugin.TutorialPlugin;
import
com.netapp.snapcreator.agent.nextgen.common.result.Describe
Result;
import
com.netapp.snapcreator.agent.nextgen.common.result.Result;
import
com.netapp.snapcreator.agent.nextgen.common.result.VersionR
esult;
import
com.netapp.snapcreator.agent.nextgen.context.Context;
import
com.netapp.snapcreator.agent.nextgen.plugin.AbstractPlugin;
public class TutorialPlugin extends AbstractPlugin {
    @Override
    public DescribeResult describe(Context context) {
        // TODO Auto-generated method stub
        return null;
    }
    @Override
    public Result quiesce(Context context) {
        // TODO Auto-generated method stub
        return null;
    }
    @Override
    public Result unquiesce(Context context) {
        // TODO Auto-generated method stub
        return null;
    }
    @Override
    public VersionResult version() {
        // TODO Auto-generated method stub
        return null;
    }
}

```

Umsetzung der erforderlichen Methoden

Quiesce, unquiesce und Version sind obligatorische Methoden, die jedes benutzerdefinierte Java Plug-in implementieren muss.

Die folgende Versionsmethode gibt die Version des Plug-ins zurück.

```

@Override
public VersionResult version() {
    VersionResult versionResult = VersionResult.builder()
                                                .withMajor(1)
                                                .withMinor(0)
                                                .withPatch(0)
                                                .withBuild(0)
                                                .build();

    return versionResult;
}

```

Below is the implementation of quiesce and unquiesce method. These will be interacting with the application, which is being protected by SnapCenter Server. As this is just a tutorial, the application part is not explained, and the focus is more on the functionality that SnapCenter Agent provides the following to the plug-in developers:

```

@Override
public Result quiesce(Context context) {
    final Logger logger = context.getLogger();
    /*
     * TODO: Add application interaction here
     */
}

```

```

logger.error("Something bad happened.");
logger.info("Successfully handled application");

```

```

Result result = Result.builder()
                      .withExitCode(0)
                      .withMessages(logger.getMessages())
                      .build();

return result;
}

```

Die Methode wird in einem Kontextobjekt übergeben. Dazu gehören mehrere Helfer, zum Beispiel ein Logger und ein Context Store, sowie die Informationen über den aktuellen Vorgang (Workflow-ID, Job-ID). Wir können den Logger erhalten, indem wir den endgültigen Logger `Logger = context.getLogger();` anrufen. Das Logger-Objekt bietet ähnliche Methoden, die von anderen Protokollierungs-Frameworks bekannt sind, z. B. Logback. Im Ergebnisobjekt können Sie auch den Exit-Code angeben. In diesem Beispiel wird Null zurückgegeben, da kein Problem aufgetreten ist. Andere Exit-Codes können verschiedenen Fehlerszenarien zugeordnet werden.

Ergebnisobjekt wird verwendet

Das Ergebnisobjekt enthält die folgenden Parameter:

Parameter	Standard	Beschreibung
Konfigurations	Leere Konfiguration	Mit diesem Parameter können Konfigurationsparameter zurück an den Server gesendet werden. Es kann Parameter sein, die das Plug-in aktualisieren möchte. Ob diese Änderung sich tatsächlich in der Konfiguration auf dem SnapCenter-Server wiederfindet, hängt vom PARAMETER APP_CONF_PERSISTENZ=Y oder N in der Konfiguration ab.
Code-Code	0	Zeigt den Status des Vorgangs an. Ein „0“ bedeutet, dass der Vorgang erfolgreich ausgeführt wurde. Andere Werte weisen auf Fehler oder Warnungen hin.
Stdout	Leere Liste	Damit können stdout-Nachrichten an den SnapCenter-Server zurückgesendet werden.
Stderr	Leere Liste	Damit können stderr-Nachrichten an den SnapCenter-Server zurückgesendet werden.
Nachrichten	Leere Liste	Diese Liste enthält alle Meldungen, die ein Plug-in zum Server zurückkehren möchte. Der SnapCenter-Server zeigt diese Meldungen in der CLI oder GUI an.

Der SnapCenter Agent stellt Builders zur Verfügung ("[Baumuster](#)") Für alle seine Ergebnistypen. Daher ist es sehr einfach, sie zu verwenden:

```
Result result = Result.builder()
    .withExitCode(0)
    .withStdout(stdout)
    .withStderr(stderr)
    .withConfig(config)
    .withMessages(logger.getMessages())
    .build()
```

Setzen Sie beispielsweise den Exit-Code auf 0, legen Sie Listen für stdout und stderr fest, legen Sie die Konfigurationsparameter fest und fügen Sie die Protokollmeldungen an, die an den Server zurückgesendet werden. Wenn Sie nicht alle Parameter benötigen, senden Sie nur die erforderlichen Parameter. Da jeder Parameter einen Standardwert hat, ist das Ergebnis unberührt, wenn Sie `.withExitCode(0)` aus dem unten stehenden Code entfernen:

```
Result result = Result.builder()
    .withExitCode(0)
    .withMessages(logger.getMessages())
    .build();
```

VersionResult

Der `VersionResult` informiert den SnapCenter-Server über die Plug-in-Version. Da es auch vom Ergebnis erbt, enthält es die Parameter `config`, `exitCode`, `stdout`, `stderr` und Nachrichten.

Parameter	Standard	Beschreibung
Major	0	Hauptversionsfeld des Plug-ins.
Gering	0	Kleines Versionsfeld des Plug-ins.
Patch	0	Feld für die Patch-Version des Plug-ins.
Entwickeln	0	Build-Versionsfeld des Plug-ins.

Beispiel:

```
VersionResult result = VersionResult.builder()
    .withMajor(1)
    .withMinor(0)
    .withPatch(0)
    .withBuild(0)
    .build();
```

Verwenden des Kontextobjekts

Das Kontextobjekt bietet folgende Methoden:

Kontextsethode	Zweck
<code>String getWorkflowId();</code>	Gibt die Workflow-id zurück, die vom SnapCenter-Server für den aktuellen Workflow verwendet wird.

Kontextsethode	Zweck
Config getConfig();	Gibt die Konfiguration zurück, die vom SnapCenter-Server an den Agenten gesendet wird.

Workflow-ID

Die Workflow-ID ist die id, die der SnapCenter-Server verwendet, um auf einen bestimmten laufenden Workflow zu verweisen.

Konfigurations

Dieses Objekt enthält (die meisten) Parameter, die ein Benutzer in der Konfiguration auf dem SnapCenter-Server festlegen kann. Aus Sicherheitsgründen können jedoch einige dieser Parameter auf Server-Seite gefiltert werden. Nachfolgend ein Beispiel für den Zugriff auf die Konfiguration und den Abruf eines Parameters:

```
final Config config = context.getConfig();
String myParameter =
config.getParameter("PLUGIN_MANDATORY_PARAMETER");
```

""// myParameter" enthält jetzt den Parameter, der aus der Konfiguration auf dem SnapCenter-Server gelesen wird. Wenn ein Parameterschlüssel nicht vorhanden ist, wird ein leerer String (") zurückgegeben.

Das Plug-in wird exportiert

Sie müssen das Plug-in exportieren, um es auf dem SnapCenter-Host zu installieren.

Führen Sie in Eclipse die folgenden Aufgaben aus:

1. Klicken Sie mit der rechten Maustaste auf das Basispaket des Plug-ins (in unserem Beispiel `com.netapp.snapcreator.agent.plugin.TutorialPlugin`).
2. Wählen Sie **Export** → **Java** → **Jar-Datei**
3. Klicken Sie Auf **Weiter**.
4. Geben Sie im folgenden Fenster den Ziel-JAR-Dateipfad an: `tutorial_plugin.jar` die Basisklasse des Plug-ins heißt `TutorialPlugin.class`, das Plug-in muss einem Ordner mit dem gleichen Namen hinzugefügt werden.

Wenn Ihr Plug-in von zusätzlichen Bibliotheken abhängt, können Sie den folgenden Ordner erstellen: `Lib/`

Sie können JAR-Dateien hinzufügen, von denen das Plug-in abhängig ist (z. B. ein Datenbanktreiber). Wenn SnapCenter das Plug-in lädt, ordnet es automatisch alle JAR-Dateien in diesem Ordner zu und fügt sie dem Klassenpfad hinzu.

Benutzerdefiniertes Plug-in in SnapCenter

Benutzerdefiniertes Plug-in in SnapCenter

Das benutzerdefinierte Plug-in, das mit Java, PERL oder NATIVEM Stil erstellt wurde, kann über SnapCenter

Server auf dem Host installiert werden, um die Datensicherung Ihrer Anwendung zu ermöglichen. Sie müssen das Plug-in exportiert haben, um es auf dem SnapCenter-Host zu installieren. Verwenden Sie dazu das in diesem Tutorial beschriebene Verfahren.

Erstellen einer Plug-in-Beschreibungsdatei

Für jedes erstellte Plug-in müssen Sie eine Beschreibungsdatei haben. Die Beschreibungsdatei beschreibt die Details des Plug-ins. Der Name der Datei muss Plugin_descriptor.XML sein.

Verwenden der Attribute der Plug-in-Deskriptordatei und ihrer Bedeutung

Attribut	Beschreibung
Name	<p>Name des Plug-ins. Alphanumerische Zeichen sind zulässig. Zum Beispiel DB2, MYSQL, MongoDB</p> <p>Stellen Sie bei Plug-ins, die im NATIVEN Stil erstellt wurden, sicher, dass Sie nicht die Erweiterung der Datei bereitstellen. Wenn der Plug-in-Name beispielsweise MongoDB.sh lautet, geben Sie den Namen als MongoDB an.</p>
Version	Plug-in-Version: Kann sowohl die Haupt- als auch die Nebenversion enthalten. Beispiel: 1.0, 1.1, 2.0, 2.1
DisplayName	Der Plug-in-Name, der im SnapCenter-Server angezeigt werden soll. Wenn mehrere Versionen desselben Plug-ins geschrieben werden, stellen Sie sicher, dass der Anzeigename bei allen Versionen gleich ist.
PluginType	Sprache zum Erstellen des Plug-ins. Unterstützte Werte sind Perl, Java und Native. Nativer Plug-in-Typ umfasst Unix/Linux Shell-Skripte, Windows-Skripte, Python oder andere Skriptsprache.
OSName	Der Host-OS-Name, auf dem das Plug-in installiert ist. Gültige Werte sind Windows und Linux. Es ist möglich, dass ein einzelnes Plug-in für die Implementierung auf mehreren Betriebssystemtypen, wie BEISPIELSWEISE PERL-Plug-in, zur Verfügung steht.
OSVersion	Die Host-Betriebssystemversion, bei der das Plug-in installiert ist.
Einfindungsname	Name des Ressourcentyps, den das Plug-in unterstützen kann. Zum Beispiel Datenbank, Instanz, Sammlungen.

Attribut	Beschreibung
Übergeordnet	<p>Wenn der ResourceName hierarchisch von einem anderen Ressourcentyp abhängig ist, bestimmt Parent den übergeordneten ResourceType.</p> <p>Zum Beispiel, DB2-Plug-in, die ResourceName „Datenbank“ hat eine übergeordnete „Instanz“.</p>
RequireFileSystemPlugin	Ja oder Nein Legt fest, ob die Registerkarte Wiederherstellung im Wiederherstellungsassistenten angezeigt wird.
ResourceRequiresAuthentifizierung	Ja oder Nein Legt fest, ob die Ressourcen, die automatisch erkannt werden oder nicht automatisch erkannt wurden, Anmeldeinformationen benötigen, um die Datensicherungsvorgänge nach der Ermittlung des Storage auszuführen.
RequireFileSystemClone	Ja oder Nein Stellt fest, ob das Plug-in eine Filesystem-Plug-in-Integration für Klon-Workflow erforderlich ist.

Ein Beispiel für die Datei Plugin_descriptor.xml für benutzerdefinierte Plug-in DB2 ist wie folgt:

```

<Plugin>
<SMSServer></SMSServer>
<Name>DB2</Name>
<Version>1.0</Version>
<PluginType>Perl</PluginType>
<DisplayName>Custom DB2 Plugin</DisplayName>
<SupportedOS>
<OS>
<OSName>windows</OSName>
<OSVersion>2012</OSVersion>
</OS>
<OS>
<OSName>Linux</OSName>
<OSVersion>7</OSVersion>
</OS>
</SupportedOS>
<ResourceTypes>
<ResourceType>
<ResourceName>Database</ResourceName>
<Parent>Instance</Parent>
</ResourceType>
<ResourceType>
<ResourceName>Instance</ResourceName>
</ResourceType>
</ResourceTypes>
<RequireFileSystemPlugin>no</RequireFileSystemPlugin>
<ResourceRequiresAuthentication>yes</ResourceRequiresAuthentication>
<SupportsApplicationRecovery>yes</SupportsApplicationRecovery>
</Plugin>

```

ZIP-Datei wird erstellt

Nachdem ein Plug-in entwickelt wurde und eine Deskriptordatei erstellt wurde, müssen Sie die Plug-in-Dateien und die Datei `Plugin_descriptor.xml` in einen Ordner einfügen und entpacken.

Vor dem Erstellen einer ZIP-Datei müssen Sie Folgendes berücksichtigen:

- Der Skriptname muss mit dem Plug-in-Namen übereinstimmen.
- Für PERL-Plug-in muss der ZIP-Ordner einen Ordner mit der Skriptdatei enthalten und die Deskriptordatei außerhalb dieses Ordners liegen. Der Ordnername muss mit dem Plug-in-Namen identisch sein.
- Bei anderen Plug-ins als DEM PERL Plug-in muss der ZIP-Ordner den Deskriptor und die Skriptdateien enthalten.
- Die Betriebssystemversion muss eine Zahl sein.

Beispiele:

- DB2-Plugin: Fügen Sie DB2.pm und Plugin_descriptor.xml Datei zu "DB2.zip".
- Plug-in entwickelt mit Java: Fügen Sie JAR-Dateien, abhängige JAR-Dateien und Plugin_descriptor.xml-Datei in einen Ordner und ZIP es.

Hochladen der ZIP-Datei für das Plug-in

Sie müssen die Plug-in-ZIP-Datei auf den SnapCenter-Server hochladen, damit das Plug-in für die Bereitstellung auf dem gewünschten Host zur Verfügung steht.

Sie können das Plug-in über die UI oder Cmdlets hochladen.

UI:

- Laden Sie die Plug-in-ZIP-Datei als Teil des **Add-** oder **Modify Host**-Workflow-Assistenten hoch
- Klicken Sie auf * „Wählen Sie, um benutzerdefinierte Plug-in hochzuladen“*

PowerShell:

- Cmdlet "Upload-SmPluginPackage"

Beispiel: PS> Upload-SmPluginPackage -AbsolutePath c:\DB2_1.zip

Ausführliche Informationen zu PowerShell Cmdlets finden Sie in der Hilfe zu SnapCenter Cmdlet oder in den Cmdlet Referenzinformationen.

["SnapCenter Software Cmdlet Referenzhandbuch"](#).

Bereitstellen benutzerdefinierter Plug-ins

Das hochgeladene benutzerdefinierte Plug-in steht nun als Teil des Workflows **Add** und **Modify Host** für die Bereitstellung auf dem gewünschten Host zur Verfügung. Sie können mehrere Plug-ins-Versionen auf den SnapCenter-Server hochladen lassen, und Sie können die gewünschte Version für die Bereitstellung auf einem bestimmten Host auswählen.

Weitere Informationen zum Hochladen des Plug-ins finden Sie unter: ["Fügen Sie Hosts hinzu und installieren Sie Plug-in-Pakete auf Remote-Hosts"](#)

Copyright-Informationen

Copyright © 2024 NetApp. Alle Rechte vorbehalten. Gedruckt in den USA. Dieses urheberrechtlich geschützte Dokument darf ohne die vorherige schriftliche Genehmigung des Urheberrechtsinhabers in keiner Form und durch keine Mittel – weder grafische noch elektronische oder mechanische, einschließlich Fotokopieren, Aufnehmen oder Speichern in einem elektronischen Abrufsystem – auch nicht in Teilen, vervielfältigt werden.

Software, die von urheberrechtlich geschütztem NetApp Material abgeleitet wird, unterliegt der folgenden Lizenz und dem folgenden Haftungsausschluss:

DIE VORLIEGENDE SOFTWARE WIRD IN DER VORLIEGENDEN FORM VON NETAPP ZUR VERFÜGUNG GESTELLT, D. H. OHNE JEGLICHE EXPLIZITE ODER IMPLIZITE GEWÄHRLEISTUNG, EINSCHLIESSLICH, JEDOCH NICHT BESCHRÄNKT AUF DIE STILLSCHWEIGENDE GEWÄHRLEISTUNG DER MARKTGÄNGIGKEIT UND EIGNUNG FÜR EINEN BESTIMMTEN ZWECK, DIE HIERMIT AUSGESCHLOSSEN WERDEN. NETAPP ÜBERNIMMT KEINERLEI HAFTUNG FÜR DIREKTE, INDIREKTE, ZUFÄLLIGE, BESONDERE, BEISPIELHAFTE SCHÄDEN ODER FOLGESCHÄDEN (EINSCHLIESSLICH, JEDOCH NICHT BESCHRÄNKT AUF DIE BESCHAFFUNG VON ERSATZWAREN ODER -DIENSTLEISTUNGEN, NUTZUNGS-, DATEN- ODER GEWINNVERLUSTE ODER UNTERBRECHUNG DES GESCHÄFTSBETRIEBS), UNABHÄNGIG DAVON, WIE SIE VERURSACHT WURDEN UND AUF WELCHER HAFTUNGSTHEORIE SIE BERUHEN, OB AUS VERTRAGLICH FESTGELEGTER HAFTUNG, VERSCHULDENSUNABHÄNGIGER HAFTUNG ODER DELIKTSHAFTUNG (EINSCHLIESSLICH FAHRLÄSSIGKEIT ODER AUF ANDEREM WEGE), DIE IN IRGEND EINER WEISE AUS DER NUTZUNG DIESER SOFTWARE RESULTIEREN, SELBST WENN AUF DIE MÖGLICHKEIT DERARTIGER SCHÄDEN HINGEWIESEN WURDE.

NetApp behält sich das Recht vor, die hierin beschriebenen Produkte jederzeit und ohne Vorankündigung zu ändern. NetApp übernimmt keine Verantwortung oder Haftung, die sich aus der Verwendung der hier beschriebenen Produkte ergibt, es sei denn, NetApp hat dem ausdrücklich in schriftlicher Form zugestimmt. Die Verwendung oder der Erwerb dieses Produkts stellt keine Lizenzierung im Rahmen eines Patentrechts, Markenrechts oder eines anderen Rechts an geistigem Eigentum von NetApp dar.

Das in diesem Dokument beschriebene Produkt kann durch ein oder mehrere US-amerikanische Patente, ausländische Patente oder anhängige Patentanmeldungen geschützt sein.

ERLÄUTERUNG ZU „RESTRICTED RIGHTS“: Nutzung, Vervielfältigung oder Offenlegung durch die US-Regierung unterliegt den Einschränkungen gemäß Unterabschnitt (b)(3) der Klausel „Rights in Technical Data – Noncommercial Items“ in DFARS 252.227-7013 (Februar 2014) und FAR 52.227-19 (Dezember 2007).

Die hierin enthaltenen Daten beziehen sich auf ein kommerzielles Produkt und/oder einen kommerziellen Service (wie in FAR 2.101 definiert) und sind Eigentum von NetApp, Inc. Alle technischen Daten und die Computersoftware von NetApp, die unter diesem Vertrag bereitgestellt werden, sind gewerblicher Natur und wurden ausschließlich unter Verwendung privater Mittel entwickelt. Die US-Regierung besitzt eine nicht ausschließliche, nicht übertragbare, nicht unterlizenzierbare, weltweite, limitierte unwiderrufliche Lizenz zur Nutzung der Daten nur in Verbindung mit und zur Unterstützung des Vertrags der US-Regierung, unter dem die Daten bereitgestellt wurden. Sofern in den vorliegenden Bedingungen nicht anders angegeben, dürfen die Daten ohne vorherige schriftliche Genehmigung von NetApp, Inc. nicht verwendet, offengelegt, vervielfältigt, geändert, aufgeführt oder angezeigt werden. Die Lizenzrechte der US-Regierung für das US-Verteidigungsministerium sind auf die in DFARS-Klausel 252.227-7015(b) (Februar 2014) genannten Rechte beschränkt.

Markeninformationen

NETAPP, das NETAPP Logo und die unter <http://www.netapp.com/TM> aufgeführten Marken sind Marken von NetApp, Inc. Andere Firmen und Produktnamen können Marken der jeweiligen Eigentümer sein.