



# アプリケーション用のプラグインを開発します SnapCenter Software 4.9

NetApp  
March 20, 2024

# 目次

アプリケーション用のプラグインを開発します .....	1
概要 .....	1
Perl ベースの開発 .....	3
ネイティブ形式 .....	11
Java スタイル .....	14
SnapCenter のカスタムプラグイン .....	22

# アプリケーション用のプラグインを開発します

## 概要

SnapCenter サーバを使用すると、SnapCenter へのプラグインとしてアプリケーションを導入および管理できます。任意のアプリケーションをSnapCenterサーバに接続して、データを保護できます 管理機能：

SnapCenter では、さまざまなプログラミング言語を使用してカスタムプラグインを開発できます。可能です Perl、Java、バッチ、またはその他のスクリプト言語を使用してカスタムプラグインを開発します。

SnapCenter でカスタムプラグインを使用するには、次のタスクを実行する必要があります。

- このガイドの手順に従って、使用するアプリケーション用のプラグインを作成します
- 概要ファイルを作成します
- カスタムプラグインをエクスポートして SnapCenter ホストにインストールする
- プラグインの zip ファイルを SnapCenter サーバにアップロードします

## すべての API 呼び出しでの汎用プラグインの処理

すべての API 呼び出しについて、次の情報を使用します。

- プラグインパラメータ
- 終了コード
- エラーメッセージを記録します
- データの整合性

## プラグインパラメータを使用します

一連のパラメータは、作成されたすべての API 呼び出しの一環としてプラグインに渡されます。次の表に、パラメータの具体的な情報を示します。

パラメータ	目的
アクション	ワークフロー名を指定します。たとえば、discover、backup、fileOrVolRestore、などです cloneVolAndLunの略
リソース	保護対象のリソースが表示されます。リソースは UID とタイプで識別されます。次の形式でプラグインに表示されます。  「<UID>、<type>; <UID>、<type>」のように入力します。例：インスタンス1、インスタンス、インスタンス2\DB1、データベース

パラメータ	目的
APP_NAME を使用している	使用するプラグインを指定します。たとえば、db2、mysql のように指定します。SnapCenter サーバには、リストされているアプリケーションに対するサポートが組み込まれています。このパラメータでは大文字と小文字が区別されます。
APP_IGNORE_ERROR	(Y または N) これにより、アプリケーションエラーが発生した場合、SnapCenter が終了するか、終了しません。これは、複数のデータベースをバックアップし、単一の障害が発生しないようにする場合に便利です。バックアップ処理を停止します。
<resource_name> ____APP_INSTANY_USERNAME	SnapCenter クレデンシャルは、リソースに対して設定されます。
<resource_name> _APP_INSTANY_PASSWORD	SnapCenter クレデンシャルは、リソースに対して設定されます。
<resource_name> _<custom_param> です	各リソースレベルのカスタムキー値は、プレフィックスが付いたプラグインで使用できます。「<RESOURCE_NAME>_」を参照してください。たとえば、を指定します。カスタムキーは、リソースの「MASTER_SLAVE」です。「MySQLDB」という名前前の場合は、として使用できます。MySQLDB_MASTER_SLAVE

#### 終了コードを使用します

プラグインは、終了コードを使用して処理のステータスをホストに戻します。各コードには特定の意味があり、プラグインは正しい終了コードを使用して同じことを示します。

次の表に、エラーコードとその意味を示します。

終了コード	目的
0	処理に成功しました。
99ドル	要求された処理はサポートされていないか実装されて
100です	処理に失敗しました。休止解除をスキップして終了します。デフォルトでは休止解除が実行されます。
101です	処理に失敗しました。バックアップ処理を続行してください。

終了コード	目的
その他	処理に失敗しました。休止解除を実行して終了します。

エラーメッセージを記録します

エラーメッセージは、プラグインから SnapCenter サーバーに渡されます。メッセージです メッセージ、ログレベル、およびタイムスタンプが含まれます。

次の表に、レベルとその目的を示します。

パラメータ	目的
情報	情報メッセージ
警告	警告メッセージ
エラー	エラーメッセージです
デバッグ	デバッグメッセージ
トレース	メッセージをトレースします

データの整合性を維持

カスタムプラグインでは、同じワークフローの実行操作間でデータが保持されます。の場合 たとえば、休止の終了時にプラグインにデータを格納し、休止解除時にこのデータを使用できます 操作。

保持するデータはプラグインによって Result オブジェクトの一部として設定されます。それが特定のフォーマットに従います については、プラグイン開発の各スタイルで詳しく説明しています。

## Perl ベースの開発

Perl を使用してプラグインを開発するには、特定の規則に従う必要があります。

- 内容は読み取り可能である必要があります
- setenv、quiesce、および unquiesce の必須処理を実装する必要があります
- 結果をエージェントに戻すには、特定の構文を使用する必要があります
- 内容は <plugin\_name>.pm ファイルとして保存してください

使用可能な処理はです

- setenv
- バージョン

- 休止
- 休止解除
- clone\_pre 、 clone\_post
- restore\_pre 、 restore を実行します
- クリーンアップ

## 一般的なプラグイン処理

結果オブジェクトを使用する

カスタムプラグイン処理では、必ず結果オブジェクトを定義する必要があります。このオブジェクトは、メッセージ、終了コード、 stdout 、 stderr をホストエージェントに送信します。

結果オブジェクト：

```
my $result = {
```

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

結果オブジェクトを返します。

```
return $result;
```

データの整合性を維持します

同じワークフローの実行の一部として、処理間でデータを保持（クリーンアップを除く）できます。この設定には、キーと値のペアを使用します。キーと値のデータペアは結果オブジェクトの一部として設定され、保持され、同じワークフローの後続の操作で使用できます。

次のコードサンプルは、保持するデータを設定します。

```

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

```

上記のコードでは、2つのキーと値のペアを設定します。これらのペアは、後続の操作で入力として使用できます。2つのキーと値のペアには、次のコードを使用してアクセスできます。

```

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

```

=== Logging error messages

各処理では、メッセージをホストエージェントに送信して戻すことができます。エージェントは、コンテンツを表示して保存します。メッセージには、メッセージレベル、タイムスタンプ、およびメッセージテキストが含まれます。複数行のメッセージがサポートされます。

```

Load the SnapCreator::Event Class:
my $msgObj = new SnapCreator::Event();
my @message_a = ();

```

`msgObj` を使用して、`Collect` メソッドを使用してメッセージをキャプチャします。

```

$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");

```

結果オブジェクトにメッセージを適用します。

```

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

```

## プラグインスタブを使用する

カスタムプラグインでは、プラグインのスタブを公開する必要があります。これらは、SnapCenter サーバがワークフローに基づいて呼び出すメソッドです。

プラグインスタブ	オプション / 必須	目的
setenv	必須	<p>このスタブは、環境と構成オブジェクトを設定します。</p> <p>ここでは、環境の解析または処理を行う必要があります。スタブが呼び出されるたびに、setenv スタブが直前に呼び出されます。これは Perl 形式のプラグインの場合にのみ必要です。</p>
バージョン	任意。	<p>このスタブは、アプリケーションのバージョンを取得するために使用されます。</p>
調査	任意。	<p>このスタブは、エージェントまたはホストでホストされているインスタンスまたはデータベースなどのアプリケーションオブジェクトを検出するために使用されます。</p> <p>このプラグインは、検出されたアプリケーションオブジェクトを応答の一部として特定の形式で返す必要があります。このスタブは、アプリケーションが SnapDrive for Unix に統合されている場合にのみ使用されます。</p> <div data-bbox="1078 1325 1474 1577"><p>Linux ファイルシステム（Linux フレーバ）がサポートされています。AIX/Solaris（UNIX 版）はサポートされていません。</p></div>



プラグインスタブ	オプション / 必須	目的
Discovery_complete の手順を実行します	任意。	<p>このスタブは、エージェントまたはホストでホストされているインスタンスまたはデータベースなどのアプリケーションオブジェクトを検出するために使用されます。</p> <p>このプラグインは、検出されたアプリケーションオブジェクトを応答の一部として特定の形式で返す必要があります。このスタブは、アプリケーションが SnapDrive for Unix に統合されている場合にのみ使用されます。</p> <div style="border: 1px solid gray; padding: 5px; margin-top: 10px;">  <p>Linux ファイルシステム（Linux フレーバ）がサポートされています。AIX および Solaris（UNIX 版）はサポートされていません。</p> </div>
休止	必須	<p>このスタブは、アプリケーションを Snapshot コピーの作成が可能な状態にする休止を実行します。これは、Snapshot コピー処理の前に呼び出されます。保持するアプリケーションのメタデータは応答の一部として設定する必要があります。これは、対応するストレージ Snapshot コピーの以降のクローニング処理またはリストア処理中に、構成パラメータの形式で返されます。</p>
休止解除	必須	<p>このスタブは、アプリケーションを通常の状態に戻すことを意味し、休止解除を実行します。この呼び出しは、Snapshot コピーの作成後に行われます。</p>
clone_pre	任意。	<p>このスタブは、クローニング前タスクを実行する役割を果たします。このパラメータは、組み込みの SnapCenter サーバクローニングインターフェイスを使用していることを前提としており、クローニング処理の実行時にトリガーされます。</p>

プラグインスタブ	オプション / 必須	目的
clone_post をクリックしてください	任意。	この STUB は、クローニング後のタスクの実行を担当します。このパラメータは、組み込みの SnapCenter サーバクローニングインターフェイスを使用していることを前提としており、クローニング処理の実行時にのみトリガーされます。
restore_pre	任意。	このスタブは、リストア前のタスクの実行を担当します。これは、組み込みの SnapCenter Server リストアインターフェイスを使用しており、リストア処理中にトリガーされることを前提としています。
リストア	任意。	このスタブは、アプリケーションのリストアタスクを実行する役割を果たします。この要件は、組み込みの SnapCenter Server リストアインターフェイスを使用していることを前提としており、リストア処理の実行時にのみトリガーされます。
クリーンアップ	任意。	この STUB は、バックアップ、リストア、またはクローン処理後にクリーンアップを実行する場合の説明です。クリーンアップは、通常のワークフローの実行中またはワークフローの失敗時に実行できます。このワークフロー名では、バックアップ、cloneVolAndLun、または fileOrVolRestore などの設定パラメータアクションを参照して、クリーンアップを呼び出すことができます。設定パラメータ ERROR_MESSAGE は 'ワークフローの実行中にエラーが発生したかどうかを示します。ERROR_MESSAGE が定義されていて NULL ではない場合 'ワークフロー失敗の実行中にクリーンアップが呼び出されます'。
APP_VERSION	任意。	このスタブは、SnapCenterがアプリケーションを取得するために使用します。プラグインで管理されるバージョンの詳細。

## プラグインパッケージの情報

すべてのプラグインについて、次の情報が必要です。

```
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 = ();
```

## 処理

ブート時、バージョン、休止、休止解除など、カスタムプラグインでサポートされるさまざまな処理をコード化できます。

### setENV 動作

Perl を使用して作成されたプラグインに対して、setENV 操作が必要です。ENV を設定すると、プラグインパラメータに簡単にアクセスできます。

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

## バージョン処理

バージョン処理は、アプリケーションのバージョン情報を返します。

```
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;
}
```

## 休止処理

休止処理を実行すると、resources パラメータにリストされているリソースに対してアプリケーション休止処理が実行されます。

```
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;
}
```

## 休止解除処理

アプリケーションの休止解除には休止解除処理が必要です。リソースのリストは、resources パラメータで指定できます。

```

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;
}

```

## ネイティブ形式

SnapCenter では、Perl 以外のプログラミング言語やスクリプト言語を使用してプラグインを作成できます。これは、スクリプトまたはバッチファイルとして使用できるネイティブスタイルプログラミングと呼ばれます。

ネイティブ形式のプラグインは、次に示す特定の表記規則に従う必要があります。

プラグインが実行可能である必要があります

- UNIX システムの場合、エージェントを実行するユーザにはプラグインに対する実行権限が必要です
- Windows システムの場合、PowerShell プラグインにはサフィックス.ps1、その他の Windows が必要です スクリプトは、.cmd または .bat サフィックスを持ち、ユーザが実行可能である必要があります
- プラグインは、「-quiesce」、「-unquiesce」などのコマンドライン引数に対応する必要があります。
- 操作または関数が実装されていない場合、プラグインは終了コード 99 を返す必要があります
- プラグインは、特定の構文を使用して結果をサーバに渡す必要があります

## 一般的なプラグイン処理

### エラーメッセージのロギング

各オペレーションは 'サーバにメッセージを送信することができますサーバは' コンテンツを表示して保存しますメッセージには、メッセージレベル、タイムスタンプ、およびメッセージテキストが含まれます。複数行のメッセージがサポートされます。

の形式で入力し

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

## プラグインスタブを使用する

SnapCenter プラグインはプラグインスタブを実装する必要があります。SnapCenter サーバが呼び出すメソッドは、特定のワークフローに基づいています。

プラグインスタブ	オプション / 必須	目的
休止	必須	このスタブは休止を実行します。が配置されます Snapshot コピーを作成できる状態に適用します。これは、ストレージ Snapshot コピー処理の前に呼び出されます。
休止解除	必須	休止解除を実行する場合は、このスタブを指定します。配置されず 通常の状態のアプリケーション。これは保管後に呼び出されず Snapshot コピー処理
clone_pre	任意。	このスタブは、クローニング前のタスクを実行する役割を果たします。この場合、組み込みの SnapCenter クローニングインターフェイスを使用しており、「clone_vol または clone_lun」操作の実行時にのみトリガーされることを前提としています。
clone_post をクリックしてください	任意。	この STUB は、クローニング後のタスクの実行を担当します。このパラメータは、組み込みの SnapCenter クローニングインターフェイスを使用していること、および「clone_vol」または「clone_lun」処理を実行するときのみトリガーされることを前提としています。
restore_pre	任意。	このスタブはリストア前のタスクを実行するためのものです。この処理は、組み込みの SnapCenter リストアインターフェイスを使用していることを前提としており、リストア処理の実行中にのみ実行されます。

プラグインスタブ	オプション / 必須	目的
リストア	任意。	このスタブは、すべてのリストアアクションを実行するためのものです。これ組み込みのリストアインターフェイスを使用していないことを前提としています。このコマンドはリストア処理の実行中にトリガーされます。

例

#### Windows PowerShell の場合

スクリプトをシステムで実行できるかどうかを確認します。スクリプトを実行できない場合は、スクリプトに対して Set-ExecutionPolicy bypass を設定して、操作を再試行します。

```

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 スタイル

Java カスタムプラグインは、データベースやインスタンスなどのアプリケーションと直接対話します。

### 制限

Java プログラミング言語を使用してプラグインを開発する場合は、一定の制限事項に注意する必要があります。



プラグインの特性	Java プラグイン
複雑さ	低～中
メモリフットプリント	最大 10～20 MB
他のライブラリとの依存関係	アプリケーション通信用ライブラリ
スレッド数	1.
スレッドランタイム	1 時間未満

### Java の制限の理由

SnapCenter エージェントの目標は、継続的、安全、堅牢なアプリケーション統合を実現することです。Java プラグインをサポートすることで、プラグインがメモリリークなどの不要な問題をもたらす可能性があります。これらの課題に取り組むことは困難です。特に、使いやすいものを維持することが目的である場合には困難です。プラグインの複雑さがそれほど複雑でない場合は、開発者がエラーを発生させてしまう可能性ははるかに低くなります。Java プラグインの危険性は、プラグインがあることです。SnapCenter エージェント自体と同じ JVM 内で実行されています。プラグインがクラッシュしたりメモリがリークしたりすると、Agent に悪影響を与える可能性もあります。

### サポートされている方法

メソッド	必須	説明	いつ、誰が電話をかけましたか？
バージョン	はい。	プラグインのバージョンを返す必要があります。	のバージョンを要求する SnapCenter サーバまたはエージェント プラグイン。
休止	はい。	アプリケーションで休止を実行する必要があります。ほとんどの場合、この状態になると、SnapCenter サーバでバックアップ（Snapshot コピーなど）を作成できるようになります。	SnapCenter サーバが Snapshot コピーまたはを作成する前に一般的なバックアップを実行します。
休止解除	はい。	アプリケーションに対して休止解除を実行する必要があります。ほとんどの場合、これアプリケーションを通常の動作状態に戻すことを意味します。	SnapCenter サーバが Snapshot コピーまたはを作成したあと一般的にバックアップを実行しました。

メソッド	必須	説明	いつ、誰が電話をかけましたか？
クリーンアップ	いいえ	プラグインがクリーンアップする必要があるすべての項目をクリーンアップする責任があります。	SnapCenter サーバでワークフローが完了したとき（正常終了したとき、または障害が発生したとき）。
clonePre-	いいえ	クローニング処理を実行する前に、必要な処理を実行する必要があります。	ユーザが「cloneVol」または「cloneLun」アクションをトリガーし、組み込みのクローニングウィザード（GUI / CLI）を使用する場合。
clonePost を実行します	いいえ	クローニング処理の実行後に必要な処理を実行する必要があります。	ユーザが「cloneVol」または「cloneLun」アクションをトリガーし、組み込みのクローニングウィザード（GUI / CLI）を使用する場合。
restorePre	いいえ	は、リストア処理が呼び出される前に実行する必要がある操作を実行します。	ユーザがリストア処理をトリガーした場合。
リストア	いいえ	アプリケーションのリストア / リカバリを実行します。	ユーザがリストア処理をトリガーした場合。
AppVersion（アプリバージョン）	いいえ	プラグインによって管理されているアプリケーションバージョンを取得する。	バックアップ / リストア / クローンなど、すべてのワークフローで ASUP データ収集の一部として実行

## チュートリアル

このセクションでは、Java プログラミング言語を使用してカスタムプラグインを作成する方法について説明します。

### Eclipse のセットアップ

1. Eclipse で新しい Java プロジェクト「TutorialPlugin」を作成します
2. [完了] をクリックします。
3. 新しいプロジェクト \* → \* プロパティ \* → \* Java ビルドパス \* → \* ライブラリ \* → \* 外部 JAR の追加 \* を右クリックします

4. ホスト・エージェントの `.lib/folder` に移動し `jar scAgent-5.0-core.jar` と `common-5.0.jar` を選択します
5. プロジェクトを選択し、`* src フォルダー *` → `* New *` → `* Package *` を右クリックして、`com.netapp.snapcreator.agent.plugin.TutorialPlugin` という名前で新しいパッケージを作成します
6. 新しいパッケージを右クリックし '新規作成 > Java クラス' を選択します
  - a. `TutorialPlugin` という名前を入力してください。
  - b. スーパークラスの参照ボタンをクリックし、「`* AbstractPlugin`」を検索します。表示される結果は1つだけです。

```
"AbstractPlugin - com.netapp.snapcreator.agent.nextgen.plugin".  
.. [ 完了 ] をクリックします。  
.. Java クラス :
```

```

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;
    }
}

```

必要なメソッドを実装する

カスタム Java プラグインを実装するには、休止、休止解除、およびバージョンの各必須メソッドが必要です。

以下は、プラグインのバージョンを返すバージョンメソッドです。

```

@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 plugin 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;
}

```

メソッドは `Context` オブジェクトで渡されます。これには、ロガーとコンテキストストアなどの複数のヘルパーと、現在の操作に関する情報（ワークフロー ID、ジョブ ID）が含まれます。ロガーは、`context.getLogger();` を呼び出すことで取得できます。`logger` オブジェクトは、`logback` などの他のロギングフレームワークで知られている同様のメソッドを提供します。結果オブジェクトでは、終了コードを指定することもできます。この例では、問題が存在しないため 0 が返されます。その他の終了コードは、さまざまな障害シナリオに対応する場合があります。

結果オブジェクトを使用します

result オブジェクトには、次のパラメータが含まれます。

パラメータ	デフォルト	説明
構成	空です 構成	このパラメータを使用すると、設定パラメータをサーバに返送できます。それは、プラグインで更新するパラメータを指定できます。この変化がそうであるかどうか SnapCenterサーバの設定に実際に反映されるのは、に依存します CONFIG の APP_CONF_PERSISTENCY = Y または N パラメータ。
イキシコード	0	処理のステータスを示します。「0」は、操作が行われたことを示しますが実行されました。その他の値は、エラーまたは警告を示します。
標準出力	空です リスト	これは、stdoutメッセージをSnapCenterに送信するために使用できます サーバ：
stderr	空です リスト	これは、stderrメッセージをSnapCenterに送信するために使用できます サーバ：
メッセージ	空です リスト	このリストには、プラグインが返すすべてのメッセージが含まれています サーバこれらのメッセージは、SnapCenterサーバの CLI または GUI に表示されます。

SnapCenter エージェントはビルダーを提供します ("ビルダパターン")すべてのために 結果タイプ。これにより、これらの機能を非常に簡単に使用できます。

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

たとえば、終了コードを 0 に設定し、stdout と stderr のリストを設定し、config パラメータを設定して、サ

サーバに送信されるログメッセージを追加します。すべてのパラメータが不要な場合は、必要なパラメータのみを送信します。各パラメータにはデフォルト値が設定されているため、以下のコードから `.withExitCode(0)` を削除しても、結果は影響を受けません。

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

## VersionResult

VersionResult は、SnapCenter サーバにプラグインのバージョンを通知します。それはまた継承します 結果から、config、exitCode、stdout、stderr、およびmessagesパラメータが含まれます。

パラメータ	デフォルト	説明
メジャー (Major)	0	プラグインのメジャーバージョンフィールド。
マイナー	0	プラグインのマイナーバージョンフィールド。
パッチ	0	プラグインの PATCH version フィールド。
構築	0	プラグインのビルドバージョンフィールド。

例：

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

## コンテキストオブジェクトの使用

コンテキストオブジェクトには、次のメソッドがあります。

コンテキストメソッド	目的
文字列 <code>getWorkflowId()</code> ;	にSnapCenterサーバで使用されているワークフローIDを返します 現在のワークフロー。

コンテキストメソッド	目的
Config getConfig () ;	SnapCenterサーバから送信中の設定を返します 捜査官

## ワークフロー ID

ワークフローIDは、SnapCenterサーバが特定の実行を参照するために使用するIDです ワークフロー：

## 構成

このオブジェクトには、の設定でユーザが設定できるパラメータのほとんどが含まれています SnapCenterサーバ。ただし、セキュリティ上の理由により、これらのパラメータの一部が取得される可能性があります サーバー側でフィルタリングされます。次に、Config and Retrieveにアクセスする例を示します パラメータ：

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

"//MyParameter"には、SnapCenterサーバの設定から読み取ったパラメータが含まれるようになりました 設定パラメータキーが存在しない場合は、空の文字列("")を返します。

## プラグインのエクスポート

SnapCenter ホストにインストールするには、プラグインをエクスポートする必要があります。

Eclipse では、次のタスクを実行します。

1. プラグインの基本パッケージを右クリックします(この例では com.netapp.snapcreator.agent.plugin.TutorialPluginを参照)。
2. 「\* Export \* → \* Java \* → \* JAR File \*」を選択します
3. 「\* 次へ \*」をクリックします。
4. 次のウィンドウで、jarファイルの保存先パスtutorial\_plugin.jarを指定します プラグインの基本クラスはTutorialPlugin.classという名前で、プラグインをフォルダに追加する必要があります 同じ名前で

プラグインが追加のライブラリに依存している場合は、lib/ というフォルダを作成できます

jar ファイルを追加できます。このプラグインは従属ファイルに依存します (たとえば、データベース・ドライバ)。いつ SnapCenterはプラグインをロードし、このフォルダ内のすべてのjarファイルをとに自動的に関連付けます クラスパスに追加します。

# SnapCenter のカスタムプラグイン

## SnapCenter のカスタムプラグイン

Java、Perl、またはネイティブ形式を使用して作成したカスタムプラグインを、SnapCenter サーバを使用してホストにインストールし、アプリケーションのデータを保護することができます。このチュートリアルで



提供されている手順を使用して SnapCenter ホストにインストールするには、プラグインをエクスポートしておく必要があります。

プラグイン概要ファイルを作成しています

プラグインを作成するたびに、概要ファイルが必要になります。概要ファイルには、プラグインの詳細が記述されています。ファイルの名前は、プラグイン記述子 .xml である必要があります。

プラグイン記述子ファイルの属性とその重要度を使用する

属性	説明
名前	プラグインの名前。英数字を使用できます。たとえば、DB2、MySQL、MongoDB などです  ネイティブ形式で作成したプラグインの場合は、ファイルの拡張子を指定しないでください。たとえば、プラグインの名前が MongoDB である場合は、MongoDB という名前を指定します。
バージョン	プラグインのバージョン。メジャーバージョンとマイナーバージョンの両方を含めることができます。たとえば、1.0、1.1、2.0、2.1 のようになります
表示名	SnapCenterサーバーに表示するプラグイン名。同じプラグインの複数のバージョンが書き込まれている場合は、表示名がすべてのバージョンで同じであることを確認してください。
プラグインタイプ ( PluginType )	プラグインの作成に使用する言語。サポートされている値は Perl、Java、および Native です。標準のプラグインタイプには、Unix/Linux シェルスクリプト、Windows スクリプト、Python、またはその他のスクリプト言語が含まれています。
osname のように指定し	プラグインがインストールされているホスト OS の名前。有効な値は Windows と Linux です。Linux：1つのプラグインを、Perl タイププラグインなど、複数の OS タイプに導入できます。
osVersion をクリックします	プラグインがインストールされているホスト OS のバージョン。
ResourceName の略	プラグインでサポート可能なリソースタイプの名前。例：database、instance、コレクション。

属性	説明
親 ( Parent )	<p>ResourceNameが別のリソースタイプに階層的に依存している場合は 親(Parent)-親リソースタイプを決定します</p> <p>たとえば、DB2 プラグインの場合、ResourceName 「 Database 」には親の 「 Instance 」があります。</p>
FileSystemPlugin が必要です	はいまたはいいえリカバリタブがであるかどうかを指定します リストアウィザードに表示されます。
ResourceRequiresAuthentication の略	はいまたはいいえリソースが自動検出されたかどうかを指定します 自動検出のあとにデータ保護処理を実行するにはクレデンシャルが必要です ストレージを検出しています。
FileSystemClone が必要です	はいまたはいいえプラグインでクローン用にファイルシステムプラグインの統合が必要かどうかを指定します ワークフロー：

カスタムプラグイン DB2 の Plugin\_descriptor.xml ファイルの例は次のとおりです。

```
<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 ファイルを作成しています

プラグインが開発され、記述子ファイルが作成されたら、プラグインファイルを追加する必要があります。Plugin\_descriptor.xml ファイルをフォルダに移動してzip圧縮します。

ZIP ファイルを作成する前に、次の点を考慮してください。

- スクリプト名はプラグイン名と同じである必要があります。
- Perlプラグインの場合、ZIPフォルダにスクリプトファイルと格納されたフォルダが含まれている必要があります。ディスクリプタファイルはこのフォルダの外にある必要があります。フォルダ名は同じである必要があります。プラグイン名。
- Perlプラグイン以外のプラグインの場合は、ZIPフォルダに記述子とが含まれている必要があります。スクリプトファイル。
- OS のバージョンは番号である必要があります。

## 例

- DB2 プラグイン：DB2.pm と Plugin\_descriptor.xml ファイルを「DB2.zip」に追加します。
- Javaを使用して開発されたプラグイン：jarファイル、依存するjarファイル、およびを追加します Plugin\_descriptor.xmlファイルをフォルダに保存してzip圧縮します。

プラグインの **ZIP** ファイルをアップロードしています

プラグインをで使用できるように、プラグインのZIPファイルをSnapCenterサーバにアップロードする必要があります 目的のホストへの導入

UI またはコマンドレットを使用して、プラグインをアップロードできます。

- UI : \*
- プラグインの ZIP ファイルを \* Add \* または \* Modify Host \* ワークフローウィザードの一部としてアップロードします
- [ 選択 ] をクリックしてカスタムプラグインをアップロードします。 \*
- PowerShell : \*
- uploadSmPluginPackage コマンドレット

たとえば、PS> Upload-SmPluginPackage-AbsolutePath c : \DB2\_1.zip のように入力します

PowerShellコマンドレットの詳細については、SnapCenterコマンドレットのヘルプまたはを使用してください コマンドレットのリファレンス情報を参照してください。

["SnapCenter ソフトウェアコマンドレットリファレンスガイド"](#)。

### カスタムプラグインの導入

アップロードしたカスタムプラグインが、の一部として目的のホストに導入できるようになります \*[ホストの追加]および[ホストの変更]ワークフロー。には複数のバージョンのプラグインをアップロードできます SnapCenterサーバを選択すると、特定のホストに導入するバージョンを選択できます。

プラグインのアップロード方法の詳細については、を参照してください。 ["ホストを追加し、プラグインパッケージをリモートホストにインストールする"](#)

## 著作権に関する情報

Copyright © 2024 NetApp, Inc. All Rights Reserved. Printed in the U.S.このドキュメントは著作権によって保護されています。著作権所有者の書面による事前承諾がある場合を除き、画像媒体、電子媒体、および写真複写、記録媒体、テープ媒体、電子検索システムへの組み込みを含む機械媒体など、いかなる形式および方法による複製も禁止します。

ネットアップの著作物から派生したソフトウェアは、次に示す使用許諾条項および免責条項の対象となります。

このソフトウェアは、ネットアップによって「現状のまま」提供されています。ネットアップは明示的な保証、または商品性および特定目的に対する適合性の暗示的保証を含み、かつこれに限定されないいかなる暗示的な保証も行いません。ネットアップは、代替品または代替サービスの調達、使用不能、データ損失、利益損失、業務中断を含み、かつこれに限定されない、このソフトウェアの使用により生じたすべての直接的損害、間接的損害、偶発的損害、特別損害、懲罰的損害、必然的損害の発生に対して、損失の発生の可能性が通知されていたとしても、その発生理由、根拠とする責任論、契約の有無、厳格責任、不法行為（過失またはそうでない場合を含む）にかかわらず、一切の責任を負いません。

ネットアップは、ここに記載されているすべての製品に対する変更を随時、予告なく行う権利を保有します。ネットアップによる明示的な書面による合意がある場合を除き、ここに記載されている製品の使用により生じる責任および義務に対して、ネットアップは責任を負いません。この製品の使用または購入は、ネットアップの特許権、商標権、または他の知的所有権に基づくライセンスの供与とはみなされません。

このマニュアルに記載されている製品は、1つ以上の米国特許、その他の国の特許、および出願中の特許によって保護されている場合があります。

権利の制限について：政府による使用、複製、開示は、DFARS 252.227-7013（2014年2月）およびFAR 5252.227-19（2007年12月）のRights in Technical Data -Noncommercial Items（技術データ - 非商用品目に関する諸権利）条項の(b)(3)項、に規定された制限が適用されます。

本書に含まれるデータは商用製品および/または商用サービス（FAR 2.101の定義に基づく）に関係し、データの所有権はNetApp, Inc.にあります。本契約に基づき提供されるすべてのネットアップの技術データおよびコンピュータソフトウェアは、商用目的であり、私費のみで開発されたものです。米国政府は本データに対し、非独占的かつ移転およびサブライセンス不可で、全世界を対象とする取り消し不能の制限付き使用权を有し、本データの提供の根拠となった米国政府契約に関連し、当該契約の裏付けとする場合にのみ本データを使用できます。前述の場合を除き、NetApp, Inc.の書面による許可を事前に得ることなく、本データを使用、開示、転載、改変するほか、上演または展示することはできません。国防総省にかかる米国政府のデータ使用权については、DFARS 252.227-7015(b)項（2014年2月）で定められた権利のみが認められます。

## 商標に関する情報

NetApp、NetAppのロゴ、<http://www.netapp.com/TM>に記載されているマークは、NetApp, Inc.の商標です。その他の会社名と製品名は、それを所有する各社の商標である場合があります。