アプリケーション用のプラグインを開発します

SnapCenter Software 4.6

NetApp
September 29, 2025

目次

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。データ保護機能と管理機能を備えた SnapCenter サーバに、お好みのアプリケーションを接続できます。

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。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できます。

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 、または cloneVolAndLun などです
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：「 Instance1 、 Instance ； Instance2\\DB1 、 Database 」

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する場合に、単一障害でバックアップ処理を停止しないようにする場合に便利です。
<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	すべてのリソースレベルのカスタムキー値は、先頭に「 <resource_name>_ 」を付けたプラグインで使用できます。たとえば、カスタムキーが「 MySQLDB 」という名前のリソースの「 MASTER_SLAVE 」である場合、このキーは MySQLDB_MASTER_SLAVE として使用できます

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	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	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	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。メッセージには、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。たとえば、プラグインは休止の終了時にデータを格納でき、休止解除処理に使用できます。

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。具体的な形式で記述され、プラグイン開発の各形式で詳しく説明されています。

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	このスタブは、環境と構成オブジェクトを設定します。 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。
バージョン	任意。	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。
調査	任意。	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。 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。 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。

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。	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。 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。 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。
休止	必須	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。
休止解除	必須	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。
clone_pre	任意。	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。

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 がプラグインによって管理されるアプリケーションバージョンの詳細を取得するために使用されます。

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 プラグインのサフィックスは .ps1 に、その他の Windows スクリプトのサフィックスは .cmd または .bat にする必要があり、ユーザによって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。Snapshot コピーを作成できる状態にアプリケーションを配置します。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。アプリケーションは通常の状態になります。ストレージ Snapshot コピー処理のあとに呼び出されます。
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に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。この要件は、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、 SnapCenter エージェント自体と同じ JVM で実行されていることです。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。	SnapCenter サーバまたはエージェントがプラグインのバージョンを要求します。
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 コピーを作成する前、または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、これは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 サーバで Snapshot コピーが作成されるか、または一般的にバックアップが実行されます。

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 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;
}
```

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、最終ロガーロガー = 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	構成が空です	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。プラグインで更新するパラメータを指定できます。この変更が SnapCenter サーバの構成に実際に反映されるかどうかは、設定の APP_CONF_PERSISTENCE = Y または N パラメータに依存します。
イキシコード	0	処理のステータスを示します。「 0 」は、操作が正常に実行されたことを示します。その他の値は、エラーまたは警告を示します。
標準出力	リストが空です	これは、 stdout メッセージを SnapCenter サーバに返送するために使用できます。
stderr	リストが空です	このオプションを使用すると、 stderr メッセージを SnapCenter サーバに返送できます。
メッセージ	リストが空です	このリストには、プラグインがサーバーに返すすべてのメッセージが含まれています。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"）をクリックします。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。また、 result から継承されるため、 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 getWorkflowId();	現在のワークフローで SnapCenter サーバによって使用されているワークフロー ID を返します。

コンテキストメソッド	目的
Config GetConfig （）；	SnapCenter サーバからエージェントに送信されている設定を返します。

ワークフロー **ID**

ワークフロー ID は、実行中の特定のワークフローを SnapCenter サーバが参照するために使用する ID です。

構成

このオブジェクトには、ユーザが SnapCenter サーバの設定で設定できるパラメータのほとんどが含まれます。ただし、セキュリティ上の理由から、これらのパラメータの一部はサーバ側でフィルタリングされる場合があります。次に、 Config にアクセスしてパラメータを取得する例を示します。

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

"//MyParameter" に、設定パラメータキーが存在しない場合に SnapCenter サーバの設定から読み取られたパラメータが含まれるようになりました。空の文字列 ("") が返されます。

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. プラグインのベース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 ）。
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、インストール先の jar ファイルのパスを指定します。 tutorial_plugin.jar プラグインのベースクラスは TutorialPlugin.class という名前で、同じ名前のフォルダにプラグインを追加する必要があります。

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）。SnapCenter は、プラグインをロードすると、このフォルダ内のすべての jar ファイルを自動的に関連付けて、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 サーバに表示されるプラグインの名前。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。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。たとえば、データベース、インスタンス、コレクションなどです。

属性	説明
親（ Parent ）	場合、 ResourceName は階層的に別のリソースタイプに依存し、 Parent は親のリソースタイプを決定します。 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。
FileSystemPlugin が必要です	はいまたはいいえリストアウィザードにリカバリタブを表示するかどうかを指定します。
ResourceRequiresAuthentication の略	はいまたはいいえ自動で検出されたリソース、または自動で検出されなかったリソースに、ストレージの検出後にデータ保護処理を実行するためのクレデンシャルが必要かどうかを指定します。
FileSystemClone が必要です	はいまたはいいえクローンワークフローにファイルシステムプラグインを統合する必要があるかどうかを指定します。

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て記述子ファイルが作成されたら、プラグインファイルと Plugin_descriptor.xml ファイルをフォルダに追加して zip する必要があります。

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が格納されているフォルダと、記述ファイルがこのフォルダの外部にある必要があります。フォルダ名はプラグイン名と同じである必要があります。
- Perl プラグイン以外のプラグインを使用する場合は、 ZIP フォルダに記述子とスクリプトファイルが含まれている必要があります。
- 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、 Plugin_descriptor.xml ファイルをフォルダに追加して zip ファイルを保存します。

プラグインの **ZIP** ファイルをアップロードしています

プラグインを目的のホストに導入できるように、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。

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 のコマンドレットのヘルプを使用するか、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を、 * Add * および * Modify Host * ワークフローの一環として、目的のホストに導入できるようになりました。SnapCenter サーバに複数のバージョンのプラグインをアップロードして、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

商標に関する情報