Desenvolva um plug-in para sua aplicação

SnapCenter Software 6.0

NetApp
December 19, 2024

Índice

Desenvolva um plug-in para sua aplicação

Visão geral

O servidor SnapCenter permite que você implante e gerencie seus aplicativos como plug-ins para o SnapCenter. Os aplicativos de sua escolha podem ser conetados ao servidor SnapCenter para recursos de proteção e gerenciamento de dados.

O SnapCenter permite que você desenvolva plug-ins personalizados usando diferentes linguagens de programação. Você pode desenvolver um plug-in personalizado usando Perl, Java, BATCH ou outras linguagens de script.

Para usar plug-ins personalizados no SnapCenter, você deve executar as seguintes tarefas:

- Crie um plug-in para sua aplicação usando as instruções deste guia
- Crie um arquivo de descrição
- Exporte o plug-in personalizado para instalá-lo no host SnapCenter
- Carregue o ficheiro zip plug-in para o servidor SnapCenter

Tratamento genérico de plug-in em todas as chamadas de API

Para cada chamada de API, use as seguintes informações:

- Parâmetros do plug-in
- Códigos de saída
- Registar mensagens de erro
- Consistência de dados

Use parâmetros Plug-in

Um conjunto de parâmetros é passado para o plug-in como parte de cada chamada de API feita. A tabela a seguir lista as informações específicas para os parâmetros.

Parâmetro	Finalidade
AÇÃO	Determina o nome do fluxo de trabalho. Por exemplo, descobrir, fazer backup, fileOrVolRestore ou cloneVolAndLun
RECURSOS	Lista os recursos a serem protegidos. Um recurso é identificado por UID e tipo. A lista é apresentada ao plug-in no seguinte formato: "<UID>, <TYPE>; <UID>, <TYPE>". Por exemplo, "Instance1,Instance;Instance2' DB1,Database"

Parâmetro	Finalidade
NOME_APP	Determina qual plug-in está sendo usado. Por exemplo, DB2, MySQL. O servidor SnapCenter tem suporte interno para os aplicativos listados. Este parâmetro é sensível a maiúsculas e minúsculas.
APP_IGNORE_ERROR	(Y ou N) isso faz com que o SnapCenter saia ou não saia quando um erro de aplicativo for encontrado. Isso é útil quando você está fazendo backup de vários bancos de dados e não quer que uma única falha pare a operação de backup.
<RESOURCE_NAME>__APP_INSTANCE_USERNAME	A credencial SnapCenter está definida para o recurso.
<RESOURCE_NAME>_APP_INSTANCE_PASSWORD	A credencial SnapCenter está definida para o recurso.
<RESOURCE_NAME>_<CUSTOM_PARAM>	Cada valor de chave personalizada no nível de recurso está disponível para plug-ins pré-fixados com "<RESOURCE_NAME>_". Por exemplo, se uma chave personalizada for "MASTER_SLAVE" para um recurso chamado "MySQLDB", ela estará disponível como MySQLDB_MASTER_SLAVE

Utilize códigos de saída

O plug-in retorna o status da operação de volta ao host por meio de códigos de saída. Cada código tem um significado específico e o plug-in usa o código de saída direito para indicar o mesmo.

A tabela a seguir mostra os códigos de erro e seu significado.

Código de saída	Finalidade
0	Operação bem-sucedida.
99	A operação solicitada não é suportada ou implementada.
100	Falha na operação, ignore unquiesce e saia. Unquiesce é por padrão.
101	Falha na operação, continue com a operação de backup.
outros	Falha na operação, execute unquiesce e saia.

Registar mensagens de erro

As mensagens de erro são passadas do plug-in para o servidor SnapCenter. A mensagem inclui a mensagem, o nível do log e o carimbo de hora.

A tabela a seguir lista os níveis e seus propósitos.

Parâmetro	Finalidade
INFORMAÇÕES	mensagem informativa
AVISAR	mensagem de aviso
ERRO	mensagem de erro
DEPURAR	mensagem de depuração
TRAÇADO	mensagem de rastreio

Preservar a consistência de dados

Plug-ins personalizados preservam dados entre operações da mesma execução de fluxo de trabalho. Por exemplo, um plug-in pode armazenar dados no final do quiesce, que pode ser usado durante a operação de unquiesce.

Os dados a serem preservados são definidos como parte do objeto resultado por plug-in, seguindo um formato específico e descrito em detalhes sob cada estilo de desenvolvimento de plug-in.

Desenvolvimento baseado EM PERL

Você deve seguir certas convenções ao desenvolver o plug-in usando PERL.

- O conteúdo deve ser legível
- Deve implementar operações obrigatórias setenv, quiesce e unquiesce
- Deve usar uma sintaxe específica para passar os resultados de volta ao agente
- O conteúdo deve ser salvo como arquivo <PLUGIN_NAME>.pm

As operações disponíveis são

- Setenv
- versão
- quiesce
- unquiesce
- clone_pre, clone_post
- restore_pre, restaurar
- limpeza

Manuseamento geral do plug-in

Usando o objeto resultados

Cada operação de plug-in personalizada deve definir o objeto resultados. Esse objeto envia mensagens, código de saída, stdout e stderr de volta ao agente host.

Objeto resultados:

```
my $result = {
```

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

Retornando o objeto resultados:

```
return $result;
```

Preservar a consistência dos dados

É possível preservar dados entre operações (exceto limpeza) como parte da mesma execução do fluxo de trabalho. Isso é feito usando pares chave-valor. Os pares de dados de valor-chave são definidos como parte do objeto de resultado e são retidos e disponíveis nas operações subsequentes do mesmo fluxo de trabalho.

A amostra de código a seguir define os dados a serem preservados:

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

O código acima define dois pares de chave-valor, que estão disponíveis como entrada na operação subsequente. Os dois pares de chave-valor são acessíveis usando o seguinte código:

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

```
 === Logging error messages
Cada operação pode enviar mensagens de volta ao agente host, que exibe e
armazena o conteúdo. Uma mensagem contém o nível da mensagem, um carimbo
de data/hora e um texto da mensagem. As mensagens multilinha são
suportadas.
```

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

Use o msgObj para capturar uma mensagem usando o método coletar.

```
$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");
```

Aplicar mensagens ao objeto resultados:

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

Usando stubs plug-in

Plug-ins personalizados devem expor stubs de plug-in. Estes são métodos que o servidor SnapCenter chama, com base em um fluxo de trabalho.

Encaixe de encaixe	Opcional/obrigatório	Finalidade
Setenv	obrigatório	Este stub define o ambiente e o objeto de configuração. Qualquer análise ou manipulação de ambiente deve ser feita aqui. Cada vez que um stub é chamado, o stub setenv é chamado pouco antes. É necessário apenas para plug-ins estilo PERL.
Versão	Opcional	Este esboço é usado para obter a versão do aplicativo.
Descubra	Opcional	Este stub é usado para descobrir objetos de aplicativos como instância ou banco de dados hospedado no agente ou host. Espera-se que o plug-in retorne objetos de aplicativo descobertos em formato específico como parte da resposta. Este stub é usado apenas no caso de a aplicação ser integrada com o SnapDrive para Unix. O sistema de arquivos Linux (Linux Flavors) é suportado. AIX/Solaris (Unix flavors) não são suportados.

Encaixe de encaixe	Opcional/obrigatório	Finalidade
discovery_complete	Opcional	Este stub é usado para descobrir objetos de aplicativos como instância ou banco de dados hospedado no agente ou host. Espera-se que o plug-in retorne objetos de aplicativo descobertos em formato específico como parte da resposta. Este stub é usado apenas no caso de a aplicação ser integrada com o SnapDrive para Unix. O sistema de arquivos Linux (Linux Flavors) é suportado. AIX e Solaris (versões Unix) não são suportados.
Quiesce	obrigatório	Este esboço é responsável por executar um quiesce, o que significa colocar o aplicativo em um estado em que você pode criar um Snapshot. Isso é chamado antes da operação Snapshot. Os metadados do aplicativo a serem retidos devem ser definidos como parte da resposta, que devem ser retornados durante operações subsequentes de clone ou restauração no Snapshot de storage correspondente na forma de parâmetros de configuração.
Unquiesce	obrigatório	Este esboço é responsável por executar um unquiesce, o que significa colocar a aplicação em um estado normal. Isso é chamado depois que você cria uma captura Instantânea.
clone_pre	opcional	Este esboço é responsável por executar tarefas de pré-clone. Isso pressupõe que você esteja usando a interface de clonagem do servidor SnapCenter integrada e é acionada ao executar uma operação de clone.

Encaixe de encaixe	Opcional/obrigatório	Finalidade
clone_post	opcional	Este esboço é responsável por executar tarefas pós-clone. Isso pressupõe que você esteja usando a interface de clonagem do servidor SnapCenter integrada e é acionada somente quando executar operação de clone.
restore_pre	opcional	Este esboço é responsável por executar tarefas de pré-restauração. Isso pressupõe que você esteja usando a interface de restauração interna do servidor SnapCenter e é acionado durante a execução da operação de restauração.
Restaurar	opcional	Este esboço é responsável por executar tarefas de restauração de aplicativos. Isso pressupõe que você esteja usando a interface de restauração interna do servidor SnapCenter e só é acionado ao executar a operação de restauração.
Limpeza	opcional	Este stub é responsável por executar a limpeza após operações de backup, restauração ou clone. A limpeza pode ocorrer durante a execução normal do fluxo de trabalho ou no caso de uma falha do fluxo de trabalho. Você pode inferir o nome do fluxo de trabalho sob o qual a limpeza é chamada consultando a AÇÃO do parâmetro de configuração, que pode ser backup, cloneVolAndLun ou fileOrVolRestore. O parâmetro de configuração ERROR_MESSAGE indica se houve algum erro durante a execução do fluxo de trabalho. Se ERROR_MESSAGE for definido e NÃO NULL, então a limpeza é chamada durante a execução de falha do fluxo de trabalho.

Encaixe de encaixe	Opcional/obrigatório	Finalidade
app_version	Opcional	Este esboço é usado pelo SnapCenter para obter detalhes da versão do aplicativo gerenciados pelo plug-in.

Informações sobre o pacote de plug-in

Cada plug-in deve ter as seguintes informações:

```
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 = ();
```

Operações

Você pode codificar várias operações como setenv, Version, quiesce e Unquiesce, que são suportadas pelos plug-ins personalizados.

Operação setenv

A operação setenv é necessária para plug-ins criados usando PERL. Pode definir o ENV e aceder facilmente aos parâmetros do plug-in.

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

Operação da versão

A operação versão retorna as informações da versão do aplicativo.

```
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;
}
```

Operações de quiesce

A operação do quiesce executa a operação do quiesce do aplicativo nos recursos listados no parâmetro RECURSOS.

```
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;
}
```

Anular a operação

A operação Unquiesce é necessária para desbloquear a aplicação. A lista de recursos está disponível no parâmetro RECURSOS.

```
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;
}
```

Estilo de NATIVE

O SnapCenter suporta linguagens de programação ou script não-PERL para criar plug-ins. Isso é conhecido como programação de estilo NATIVO, que pode ser script ou arquivo EM LOTE.

Os plug-ins de estilo NATIVO devem seguir certas convenções fornecidas abaixo:

O plug-in deve ser executável

- Para sistemas Unix, o usuário que executa o agente deve ter executado o Privileges no plug-in
- Para sistemas Windows, os plug-ins do PowerShell devem ter o sufixo .ps1, outros scripts do Windows devem ter o sufixo .cmd ou .bat e devem ser executáveis pelo usuário
- Os plug-ins devem reagir ao argumento de linha de comando como "-quiesce", "-unquiesce"
- Os plug-ins devem retornar o código de saída 99 caso uma operação ou função não seja implementada
- Os plug-ins devem usar uma sintaxe específica para passar os resultados de volta ao servidor

Manuseamento geral do plug-in

Registar mensagens de erro

Cada operação pode enviar mensagens de volta para o servidor, que exibe e armazena o conteúdo. Uma mensagem contém o nível da mensagem, um carimbo de data/hora e um texto da mensagem. As mensagens multilinha são suportadas.

Formato:

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

Usando stubs plug-in

Os plug-ins do SnapCenter devem implementar stubs de plug-in. Estes são métodos que o servidor SnapCenter chama com base em um fluxo de trabalho específico.

Encaixe de encaixe	Opcional/obrigatório	Finalidade
quiesce	obrigatório	Este esboço é responsável por realizar um quiesce. Ele coloca o aplicativo em um estado onde podemos criar um Snapshot. Isso é chamado antes da operação Snapshot do armazenamento.
unquiesce	obrigatório	Este esboço é responsável por realizar um esboço. Ele coloca a aplicação em um estado normal. Isso é chamado após a operação Snapshot do armazenamento.
clone_pre	opcional	Este esboço é responsável por executar tarefas de pré-clone. Isso pressupõe que você está usando a interface de clonagem SnapCenter integrada e também é acionado apenas durante a execução da ação "clone_vol ou clone_lun".

Encaixe de encaixe	Opcional/obrigatório	Finalidade
clone_post	Opcional	Este esboço é responsável por executar tarefas pós-clone. Isso pressupõe que você esteja usando a interface de clonagem SnapCenter integrada e também é acionado apenas durante a execução de operações "clone_vol ou clone_lun".
restore_pre	Opcional	Este esboço é responsável por executar tarefas de pré-restauração. Isso pressupõe que você esteja usando a interface de restauração do SnapCenter integrada e só é acionado durante a operação de restauração.
restaurar	opcional	Este esboço é responsável por executar todas as ações de restauração. Isso pressupõe que você não está usando a interface de restauração interna. Ele é acionado durante a execução da operação de restauração.

Exemplos

Windows PowerShell

Verifique se o script pode ser executado em seu sistema. Se você não puder executar o script, defina o desvio Set-ExecutionPolicy para o script e tente novamente a operação.

```
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;
```

Estilo Java

Um plug-in personalizado Java interage diretamente com um aplicativo como banco de dados, instância e assim por diante.

Limitações

Há certas limitações que você deve estar ciente ao desenvolver um plug-in usando linguagem de programação Java.

Característica de encaixe	Plug-in Java
Complexidade	Baixo a médio

Característica de encaixe	Plug-in Java
Espaço físico da memória	Até 10-20 MB
Dependências em outras bibliotecas	Bibliotecas para comunicação de aplicativos
Número de threads	1
Tempo de execução da thread	Menos de uma hora

Razão para limitações do Java

O objetivo do agente SnapCenter é garantir uma integração de aplicativos contínua, segura e robusta. Ao suportar plug-ins Java, é possível que os plug-ins introduzam vazamentos de memória e outros problemas indesejados. Essas questões são difíceis de resolver, especialmente quando o objetivo é manter as coisas simples de usar. Se a complexidade de um plug-in não for muito complexa, é muito menos provável que os desenvolvedores tenham introduzido os erros. O perigo do plug-in Java é que eles estão sendo executados na mesma JVM que o próprio agente SnapCenter. Quando o plug-in trava ou vaza memória, ele também pode afetar negativamente o agente.

Métodos suportados

Método	Obrigatório	Descrição	Chamado quando e por quem?
Versão	Sim	Precisa retornar a versão do plug-in.	Pelo servidor SnapCenter ou agente para solicitar a versão do plug-in.
Quiesce	Sim	Precisa executar um quiesce no aplicativo. Na maioria dos casos, isso significa colocar o aplicativo em um estado em que o servidor SnapCenter pode criar um backup (por exemplo, um instantâneo).	Antes que o servidor SnapCenter crie uma cópia Snapshot(s) ou execute um backup em geral.
Unquiesce	Sim	Precisa executar um unquiesce no aplicativo. Na maioria dos casos, isso significa colocar o aplicativo de volta em um estado de operação normal.	Depois que o servidor SnapCenter tiver criado uma captura Instantânea ou tiver executado uma cópia de segurança em geral.

Método	Obrigatório	Descrição	Chamado quando e por quem?
Limpeza	Não	Responsável pela limpeza de qualquer coisa que o plug-in precise limpar.	Quando um fluxo de trabalho no servidor SnapCenter terminar (com êxito ou com uma falha).
ClonePre	Não	Deve executar ações que precisam acontecer antes de uma operação de clone ser executada.	Quando um usuário aciona uma ação "cloneVol" ou "cloneLun" e usa o assistente de clonagem integrado (GUI/CLI).
ClonePost	Não	Deve executar ações que precisam acontecer depois que uma operação de clone foi executada.	Quando um usuário aciona uma ação "cloneVol" ou "cloneLun" e usa o assistente de clonagem integrado (GUI/CLI).
RestauroPre	Não	Deve executar ações que precisam acontecer antes da operação de restauração ser chamada.	Quando um usuário aciona uma operação de restauração.
Restaurar	Não	Responsável por executar uma restauração/recuperação do aplicativo.	Quando um usuário aciona uma operação de restauração.
AppVersion	Não	Para recuperar a versão do aplicativo gerenciada pelo plug-in.	Como parte da coleta de dados ASUP em todos os fluxos de trabalho, como Backup/Restore/Clone.

Tutorial

Esta seção descreve como criar um plug-in personalizado usando a linguagem de programação Java.

Configurando o eclipse

1. Crie um novo Projeto Java "TutorialPlugin" no Eclipse
2. Clique em **Finish**
3. Clique com o botão direito do rato em **New project → Properties → Java Build Path → Libraries → Add External JARs**
4. Navegue até a pasta ../lib/ do host Agent e selecione JARs scAgent-5.0-core.jar e common-5.0.jar

5. Selecione o projeto e clique com o botão direito na pasta **src** → **novo** → **Pacote** e crie um novo pacote com o nome com.NetApp.snapcreator.agent.plugin.TutorialPlugin
6. Clique com o botão direito do Mouse no novo pacote e selecione Nova → Classe Java.
    a. Digite o nome como TutorialPlugin.
    b. Clique no botão de navegação da superclasse e procure "*AbstractPlugin". Apenas um resultado deve aparecer:

```
 "AbstractPlugin - com.netapp.snapcreator.agent.nextgen.plugin".
.. Clique em *Finish*.
.. Classe Java:
```

```
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;
  }
}
```

Implementar os métodos necessários

Quiesce, unquiesce e versão são métodos obrigatórios que cada plug-in Java personalizado deve implementar.

O seguinte é um método de versão para retornar a versão do plug-in.

```
@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;
}
```

O método é passado em um objeto de contexto. Isso contém vários ajudantes, por exemplo, um Logger e um armazenamento de contexto, e também as informações sobre a operação atual (Workflow-ID, job-ID). Nós podemos obter o logger chamando o logger logger logger final context.getLogger();. O objeto logger fornece métodos semelhantes conhecidos de outros frameworks de log, por exemplo, logback. No objeto resultado, você também pode especificar o código de saída. Neste exemplo, zero é retornado, uma vez que não houve problema. Outros códigos de saída podem ser mapeados para diferentes cenários de falha.

Usando objeto resultado

O objeto resultado contém os seguintes parâmetros:

Parâmetro	Padrão	Descrição
Config	Configuração vazia	Este parâmetro pode ser usado para enviar parâmetros de configuração de volta para o servidor. Pode ser parâmetros que o plug-in deseja atualizar. Se essa alteração é realmente refletida na configuração no servidor SnapCenter depende do parâmetro APP_conf_PERSISTENCY_Y ou N na configuração.
ExitCode	0	Indica o estado da operação. Um "0" significa que a operação foi executada com sucesso. Outros valores indicam erros ou avisos.
Stdout	Lista vazia	Isso pode ser usado para transmitir mensagens stdout de volta para o servidor SnapCenter.
Stderr	Lista vazia	Isso pode ser usado para transmitir mensagens stderr de volta para o servidor SnapCenter.
Mensagens	Lista vazia	Esta lista contém todas as mensagens que um plug-in deseja retornar ao servidor. O servidor SnapCenter exibe essas mensagens na CLI ou GUI.

O Agente SnapCenter fornece construtores ("Padrão do construtor") para todos os seus tipos de resultados. Isso torna o uso deles muito simples:

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

Por exemplo, defina o código de saída como 0, defina listas para stdout e stderr, defina parâmetros de configuração e também anexe as mensagens de log que serão enviadas de volta ao servidor. Se você não

precisa de todos os parâmetros, envie apenas os que são necessários. Como cada parâmetro tem um valor padrão, se você remover .withExitCode(0) do código abaixo, o resultado não será afetado:

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

Versão atual

A VersionResult informa ao servidor SnapCenter a versão do plug-in. Como ele também herda de result, ele contém os parâmetros config, exitCode, stdout, stderr e messages.

Parâmetro	Padrão	Descrição
Maior	0	Campo de versão principal do plug-in.
Menor	0	Campo de versão menor do plug-in.
Patch	0	Campo versão patch do plug-in.
Construir	0	Criar campo versão do plug-in.

Por exemplo:

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

Usando o Objeto de contexto

O objeto de contexto fornece os seguintes métodos:

Método de contexto	Finalidade
String getWorkflowId();	Retorna o ID do fluxo de trabalho que está sendo usado pelo servidor SnapCenter para o fluxo de trabalho atual.
Config getConfig();	Retorna a configuração que está sendo enviada do servidor SnapCenter para o Agente.

ID do fluxo de trabalho

O ID do fluxo de trabalho é o ID que o servidor SnapCenter usa para se referir a um fluxo de trabalho em execução específico.

Config

Este objeto contém (a maioria) dos parâmetros que um usuário pode definir na configuração no servidor SnapCenter. No entanto, devido a razões de segurança, alguns desses parâmetros podem ser filtrados no lado do servidor. A seguir está um exemplo de como acessar o Config e recuperar um parâmetro:

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

""// myParameter" agora contém o parâmetro lido a partir da configuração no servidor SnapCenter se uma chave de parâmetro de configuração não existir, ele retornará uma String vazia ("").

Exportar o plug-in

Você deve exportar o plug-in para instalá-lo no host SnapCenter.

No Eclipse execute as seguintes tarefas:

1. Clique com o botão direito no pacote base do plug-in (no nosso exemplo com.NetApp.snapcreator.agent.plugin.TutorialPlugin).
2. Selecione **Export → Java → jar File**
3. Clique em **seguinte**.
4. Na janela a seguir, especifique o caminho do arquivo jar de destino: tutorial_plugin.jar a classe base do plug-in é chamada TutorialPlugin.class, o plug-in deve ser adicionado a uma pasta com o mesmo nome.

Se o plug-in depender de bibliotecas adicionais, você pode criar a seguinte pasta: Lib/

Você pode adicionar arquivos jar, nos quais o plug-in depende (por exemplo, um driver de banco de dados). Quando o SnapCenter carrega o plug-in, ele associa automaticamente todos os arquivos jar nesta pasta e os adiciona ao classpath.

Plug-in personalizado no SnapCenter

Plug-in personalizado no SnapCenter

O plug-in personalizado criado usando Java, PERL ou estilo NATIVO pode ser instalado no host usando o servidor SnapCenter para habilitar a proteção de dados do seu aplicativo. Você deve ter exportado o plug-in para instalá-lo no host SnapCenter usando o procedimento fornecido neste tutorial.

Criando um arquivo de descrição do plug-in

Para cada plug-in criado, você deve ter um arquivo de descrição. O arquivo de descrição descreve os detalhes do plug-in. O nome do arquivo deve ser Plugin_descritor.xml.

Usando atributos de arquivo do descritor de plug-in e seu significado

Atributo	Descrição
Nome	Nome do plug-in. São permitidos carateres alfanuméricos. Por exemplo, DB2, MySQL, MongoDB Para plug-ins criados no estilo NATIVO, certifique-se de que não forneça a extensão do arquivo. Por exemplo, se o nome do plug-in for MongoDB.sh, especifique o nome como MongoDB.
Versão	Versão de plug-in. Pode incluir tanto a versão maior como a menor. Por exemplo, 1,0, 1,1, 2,0, 2,1
Nome de exibição	O nome do plug-in a ser exibido no servidor SnapCenter. Se várias versões do mesmo plug-in forem escritas, verifique se o nome de exibição é o mesmo em todas as versões.
PluginType	Linguagem usada para criar o plug-in. Os valores suportados são Perl, Java e Native. O tipo de plug-in nativo inclui scripts de shell Unix/Linux, scripts Windows, Python ou qualquer outra linguagem de script.
Nome do sistema operacional	O nome do sistema operacional do host onde o plug-in está instalado. Valores válidos são Windows e Linux. É possível que um único plug-in esteja disponível para implantação em vários tipos de SO, como o plug-in do tipo PERL.
Versão do sistema operacional	A versão do sistema operacional do host onde o plug-in está instalado.
ResourceName	Nome do tipo de recurso que o plug-in pode suportar. Por exemplo, banco de dados, instância, coleções.
Pai	No caso, o ResourceName é hierarquicamente dependente de outro tipo de recurso e, em seguida, o pai determina o ResourceType pai. Por exemplo, o plug-in DB2, o ResourceName "Database" tem uma "Instância" pai.
RequireFileSystemPlugin	Sim ou não determina se a guia recuperação é exibida no assistente de restauração.

Atributo	Descrição
ResourceRequiresAuthentication	Sim ou não determina se os recursos, que são detetados automaticamente ou não foram detetados automaticamente, precisam de credenciais para executar as operações de proteção de dados após a descoberta do armazenamento.
RequireFileSystemClone	Sim ou não determina se o plug-in requer integração de plug-in do sistema de arquivos para o fluxo de trabalho clone.

Um exemplo do arquivo Plugin_descriptor.xml para o plug-in personalizado DB2 é o seguinte:

```
<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>
```

Criando um arquivo ZIP

Depois que um plug-in é desenvolvido e um arquivo descritor é criado, você deve adicionar os arquivos plug-in e o arquivo Plugin_descriptor.xml a uma pasta e zip-lo.

Você deve considerar o seguinte antes de criar um arquivo ZIP:

- O nome do script deve ser igual ao nome do plug-in.
- Para o plug-in PERL, a pasta ZIP deve conter uma pasta com o arquivo de script e o arquivo de descritor deve estar fora dessa pasta. O nome da pasta deve ser o mesmo que o nome do plug-in.
- Para plug-ins diferentes do plug-in PERL, a pasta ZIP deve conter o descritor e os arquivos de script.
- A versão do SO deve ser um número.

Exemplos:

- DB2 plug-in: Adicione DB2.pm e Plugin_descriptor.xml arquivo para "DB2.zip".
- Plug-in desenvolvido usando Java: Adicione arquivos jar, arquivos jar dependentes e arquivo Plugin_descriptor.xml para uma pasta e zip-lo.

Carregar o ficheiro ZIP do plug-in

Você deve carregar o arquivo ZIP do plug-in para o servidor SnapCenter para que o plug-in esteja disponível para implantação no host desejado.

Você pode fazer o upload do plug-in usando a IU ou cmdlets.

UI:

- Carregue o arquivo ZIP do plug-in como parte do assistente de fluxo de trabalho **Add** ou **Modify Host**
- Clique em **"Selecionar para carregar plug-in personalizado"**
- PowerShell:*
- Cmdlet Upload-SmPluginPackage

  Por exemplo, PS> Upload-SmPluginPackage -AbsolutePath c: DB2_1.zip

  Para obter informações detalhadas sobre cmdlets do PowerShell, use a ajuda do cmdlet SnapCenter ou consulte as informações de referência do cmdlet.

"Guia de referência de cmdlet do software SnapCenter".

Implantando os plug-ins personalizados

O plug-in personalizado carregado agora está disponível para implantação no host desejado como parte do fluxo de trabalho **Add** e **Modify Host**. Você pode ter várias versões de plug-ins carregados para o servidor SnapCenter e pode selecionar a versão desejada para implantar em um host específico.

Para obter mais informações sobre como carregar o plug-in, consulte, "Adicione hosts e instale pacotes plug-in em hosts remotos"

Informações sobre direitos autorais

Informações sobre marcas comerciais