Nexus アプリケーション開発：SDK

Cisco SDK について

Cisco SDK は、Yocto 2.0 に基づく開発キットです。Cisco NX-OS リリース 9.2(1) が動作する Cisco Nexus スイッチで実行するアプリケーションをビルドするためのすべてのツールが含まれています。基本コンポーネントは、多くのアプリケーションで一般的に使用される C クロスコンパイラ、リンカ、ライブラリ、およびヘッダーファイルです。リストは網羅的ではないため、特定のアプリケーションに必要な依存関係をダウンロードしてビルドすることが必要となる場合があります。一部のアプリケーションは、Cisco devhub Web サイトからダウンロードしてすぐに使用できるようになっていて、ビルドの必要はありません。SDK は、スイッチに直接インストールできる RPM パッケージをビルドするために使用できます。

SDK のインストール

以下にシステム要件を示します。

- SDK は、ほとんどの最新の 64 ビット x86_64 Linux システムで実行できます。CentOS 7 および Ubuntu 14.04 で検証済みです。Bash シェルで SDK をインストールして実行します。
- SDK には、32 ビット アーキテクチャと 64 ビット アーキテクチャの両方のバイナリが含まれているため、32 ビット ライブラリもインストールされている x86_64 Linux システムで実行する必要があります。

手順

32 ビットライブラリがインストールされているかどうかを確認します。

例：

```
bash$ ls /lib/ld-linux.so.2
```

このファイルが存在する場合は、32 ビットライブラリがすでにインストールされています。それ以外の場合は、次のように 32 ビット ライブラリをインストールします。

- CentOS 7 の場合：

  ```
  bash$ sudo dnf install glibc.i686
  ```

- Ubuntu 14.04 の場合：

  ```
  bash$ sudo apt-get install gcc-multilib
  ```

インストールと環境の初期化の手順

SDK は https://devhub.cisco.com/artifactory/open-nxos/10.0.1/nx-linux-x86_64-nxos-rootfs-n9k-sup-toolchain-1.1.0.sh からダウンロードできます。

このファイルは自己解凍アーカイブで、SDK を任意のディレクトリにインストールできます。SDK のインストール ディレクトリへのパスの入力が求められます。

```
bash$ ./wrlinux-8.0.0.25-glibc-x86_64-n9000-nxos-image-rpm-sdk-sdk.sh
Wind River Linux SDK installer version 8.0-n9000
================================================
Enter target directory for SDK (default: /opt/windriver/wrlinux/8.0-n9000):
You are about to install the SDK to "/opt/windriver/wrlinux/8.0-n9000". Proceed[Y/n]? Y
Extracting
SDK.....................................................................................done
Setting it up...done
SDK has been successfully set up and is ready to be used.

. environment-setup-corei7-64-nxos-linux
. environment-setup-corei7-32-nxosmllib32-linux

source environment-setup-corei7-64-nxos-linux
source environment-setup-corei7-32-nxosmllib32-linux
==============================
```

source environment-setup-x86-wrsmllib32-linux および **source environment-setup-x86_64-wrs-linux** コマンドを使用して、シェル環境に SDK 固有のパスを追加します。SDK で使用する予定のシェルごとに、SDK 固有のパスを追加します。SDK 固有のパスを追加することは、ビルド ツールとライブラリの正しいバージョンを使用するように SDK をセットアップする点で重要です。

手順

ステップ 1 インストール先ディレクトリを参照します。

ステップ 2 Bash プロンプトで次のコマンドを入力します。

```
bash$ source environment-setup-x86-wrsmllib32-linux
bash$ source environment-setup-x86_64-wrs-linux
```

SDK を使用したアプリケーションの構築

一般的な Linux ビルドプロセスの多くは、このシナリオで機能します。状況に最適な方法を使用してください。

アプリケーションパッケージのソースコードは、さまざまな方法で取得できます。たとえば、tar ファイル形式で、またはパッケージが存在する git リポジトリからダウンロードして、ソースコードを取得できます。

次は最も一般的なケースの一例です。

（オプション）アプリケーション パッケージが標準の configure/make/make install を使用してビルドされることを確認します。

```
bash$ tar --xvzf example-app.tgz
bash$ mkdir example-lib-install
bash$ cd example-app/
bash$ ./configure --prefix=/path/to/example-app-install
bash$ make
bash$ make install
```

場合によっては、./configure スクリプトに追加のオプションを渡す必要があります。たとえば、必要なオプションのコンポーネントと依存関係を指定する場合などです。追加オプションを渡すかどうかは、構築するアプリケーションに完全に依存します。

例：Ganglia とその依存関係の構築

この例では、ganglia と、必要なサードパーティライブラリ（libexpat、libapr、および libconfuse）を作成します。

libexpat

```
bash$ wget 'http://downloads.sourceforge.net/project/expat/expat/2.1.0/expat-2.1.0.tar.gz'
bash$ mkdir expat-install
bash$ tar xvzf expat-2.1.0.tar.gz
bash$ cd expat-2.1.0
bash$ ./configure --prefix=/home/sdk-user/expat-install
bash$ make
bash$ make install
bash$ cd ..
```

libapr

```
bash$ wget 'http://www.eu.apache.org/dist/apr/apr-1.5.2.tar.gz'
bash$ mkdir apr-install
bash$ tar xvzf apr-1.5.2.tar.gz
bash$ cd apr-1.5.2
bash$ ./configure --prefix=/home/sdk-user/apr-install
bash$ make
bash$ make install
bash$ cd ..
```

libconfuse

（注） confuse には、./configure に追加の --enable-shared オプションが必要です。そうでない場合、必要な共有ライブラリの代わりに静的にリンクされたライブラリが構築されます。

```
bash$ wget 'http://savannah.nongnu.org/download/confuse/confuse-2.7.tar.gz'
bash$ mkdir confuse-install
bash$ tar xvzf confuse-2.7.tar.gz
bash$ cd confuse-2.7
bash$ ./configure --prefix=/home/sdk-user/confuse-install --enable-shared
bash$ make
bash$ make install
bash$ cd ..
```

ganglia

（注） 必要なすべてのライブラリの場所が ./configure に渡されます。

```
bash$ wget
'http://downloads.sourceforge.net/project/ganglia/ganglia%20monitoring%20core/3.7.2/ganglia-3.7.2.tar.gz'
bash$ mkdir ganglia-install
bash$ tar xvzf ganglia-3.7.2.tar.gz
bash$ cd ganglia-3.7.2
bash$ ./configure --with-libexpat=/home/sdk-user/expat-install
--with-libapr=/home/sdk-user/apr-install/bin/apr-1-config
--with-libconfuse=/home/sdk-user/confuse-install --prefix=/home/sdk-user/ganglia-install
bash$ make
bash$ make install
bash$ cd ..
```

RPM を使用したアプリケーションのパッケージ化

「make」を使用してアプリケーションが正常にビルドされた場合は、RPM にパッケージ化できます。

（注） **RPM および仕様ファイル**

RPMパッケージ形式は、特定のアプリケーションの完全なインストールに必要なすべてのファイル（バイナリ、ライブラリ、設定、ドキュメントなど）をパッケージ化するように設計されています。したがって、RPM ファイルを作成するプロセスは簡単ではありません。RPM ビルドプロセスを支援するために、ビルドプロセスに関するすべてを制御する .spec ファイルが使用されます。

（注） 多くのサードパーティ製アプリケーションは、tarball にパッケージ化されたソースコードの形式でインターネット上で入手できます。多くの場合、これらの tarball には RPM ビルドプロセスに役立つ .spec ファイルが含まれています。残念ながら、これらの .spec ファイルの多くは、ソースコード自体ほど頻繁には更新されません。さらに悪いことに、spec ファイルがまったくない場合もあります。このような場合、RPM を構築できるように、仕様ファイルを編集または最初から作成する必要があります。

RPM ビルド環境の作成

SDK を使用して RPM をビルドする前に、RPM ビルド ディレクトリ構造を作成し、いくつかの RPM マクロを設定する必要があります。

手順

ステップ 1 ディレクトリ構造を作成します：

```
bash$ mkdir rpmbuild
bash$ cd rpmbuild
bash$ mkdir BUILD RPMS SOURCES SPECS SRPMS
```

ステップ 2 上で作成したディレクトリ構造を指すように topdir マクロを設定します：

```
bash$ echo "_topdir ${PWD}" > ~/.rpmmacros
```

（注）
この手順は、現在のユーザーがすでに設定されている .rpmmacros ファイルを有していないことを前提としています。既存の .rpmmacros ファイルを変更するのが不便な場合は、すべての rpmbuild コマンドラインに次を追加できます。

```
--define "_topdir ${PWD}"
```

ステップ 3 RPM DB を更新します。

```
bash$ rm /path/to/sdk/sysroots/x86_64-wrlinuxsdk-linux/var/lib/rpm/__db.*
bash$ rpm --rebuilddb
```

（注）
SDK の rpm および rpmbuild ツールは、RPM データベースとして通常の /var/lib/rpm の代わりに /path/to/sdk/sysroots/x86_64-wrlinuxsdk-linux/var/lib/rpm を使用するように変更されました。この変更により、SDK を使用していない場合にホストの RPM データベースと競合することが回避され、ルート アクセスの必要性がなくなります。SDK のインストール後、この手順に従って SDK RPM データベースを再構築する必要があります。

一般的な RPM ビルド手順の使用

一般的な RPM ビルド手順は次のとおりです。

```
bash$ wget --no-check-certificate --directory-prefix=SOURCES http://<URL of example-app
 tarball>
bash$ # determine location of spec file in tarball:
bash$ tar tf SOURCES/example-app.tar.bz2 | grep '.spec$'
bash$ tar xkvf SOURCES/example-app.tar.bz2 example-app/example-app.spec
bash$ mv example-app/example-app.spec SPECS/
bash$ rm -rf example-app
bash$ rpmbuild -v --bb SPECS/example-app.spec
```

結果は RPMS/ に作成されるバイナリ RPM で、スイッチにコピーしてインストールできます。アプリケーションのインストール方法と構成には様々なバリエーションがあり得ます。これらの手順については、アプリケーションのドキュメントを参照してください。

この rpm ビルドとスイッチへのインストールは、アプリケーションをサポートするために必要なすべてのソフトウェアパッケージで必要です。SDK にまだ含まれていないソフトウェアの依存関係を満たすことが必要な場合は、ソースコードを取得して、依存関係のあるソフトウェアもビルドする必要があります。ビルド用のマシンでは、パッケージを手動でビルドして、依存関係を検証することができます。次に、最も一般的な手順の例を示します。

```
bash$ tar xkzf example-lib.tgz
bash$ mkdir example-lib-install
bash$ cd example-lib/
bash$ ./configure --prefix=/path/to/example-lib-install
bash$ make
bash$ make install
```

これらのコマンドは、ビルドファイル（バイナリ、ヘッダー、ライブラリなど）をインストールディレクトリに配置します。ここから、標準のコンパイラとリンカのフラグを使用して、依存関係を満たすための新しい場所を選択できます。ライブラリなどのランタイム コードがあれば、それらもスイッチにインストールする必要があるため、必要なランタイム コードを RPM にパッケージ化しなければなりません。

（注） Cisco devhub の Web サイトには、すでに RPM 形式にまとめられているサポート ライブラリが多数あります。

オプションのプラグインを使用しない collectd RPM の構築例

ソース tarball をダウンロードし、仕様ファイルを抽出します。

```
bash$ wget --no-check-certificate --directory-prefix=SOURCES
https://collectd.org/files/collectd-5.5.0.tar.bz2
bash$ tar tf SOURCES/collectd-5.5.0.tar.bz2 | grep '.spec$'
collectd-5.5.0/contrib/redhat/collectd.spec
collectd-5.5.0/contrib/aix/collectd.spec
collectd-5.5.0/contrib/sles10.1/collectd.spec
collectd-5.5.0/contrib/fedora/collectd.spec
bash$ tar xkvf SOURCES/collectd-5.5.0.tar.bz2 collectd-5.5.0/contrib/redhat/collectd.spec
bash$ mv collectd-5.5.0/contrib/redhat/collectd.spec SPECS/
bash$ rm -rf collectd-5.5.0
```

この tarball には 4 つの spec ファイルがあります。Red Hat 仕様ファイルは最も包括的であり、正しい collectd バージョンを含む唯一のファイルです。これを例として使用します。

この仕様ファイルは、/sbin/chkconfig を使用して collectd をインストールするように RPM を設定します。ただし、スイッチでは、代わりに /usr/sbin/chkconfig を使用します。仕様ファイルで編集された以下を編集します。

```
bash$ sed -r -i.bak 's%(^|\s)/sbin/chkconfig%\1/usr/sbin/chkconfig%' SPECS/collectd.spec
```

collectd には多数のオプションプラグインがあります。この仕様ファイルは、デフォルトで多くのプラグインを有効にします。多くのプラグインには外部依存関係があるため、これらのプラグインを無効にするオプションをコマンドラインに渡す必要があります。**rpmbuild** 1 つの長いコマンドラインを入力する代わりに、次のように Bash 配列でオプションを管理できます。

```
bash$ rpmbuild_opts=()
bash$ for rmdep in \
> amqp apache ascent bind curl curl_xml dbi ipmi java memcachec mysql nginx \
> notify_desktop notify_email nut openldap perl pinba ping postgresql python \
> rrdtool sensors snmp varnish virt write_http write_riemann
> do
>  rpmbuild_opts+=("--without")
>  rpmbuild_opts+=(${rmdep})
> done
bash$ rpmbuild_opts+=(--nodeps)
bash$ rpmbuild_opts+=(--define)
bash$ rpmbuild_opts+=("_unpackaged_files_terminate_build 0")
```

その後、次のように rpmbuild に渡され、ビルドおよび RPM パッケージ プロセス全体が開始されます。

```
bash$ rpmbuild "${rpmbuild_opts[@]}" -bb SPECS/collectd.spec
```

その後、RPMS ディレクトリで collectd の結果の RPM を見つけることができます。

これらの RPM ファイルをスイッチにコピーし、スイッチの Bash シェルからインストールすることができます：

```
bash$ rpm --noparentdirs -i /bootflash/collectd-5.5.0-1.ia32e.rpm
```

オプションの Curl プラグインを使用した collectd の RPM のビルド例

collectd curl プラグインには、依存関係として libcurl があります。

RPM ビルドプロセス中にこのリンクの依存関係を満たすには、SDK で curl をダウンロードしてビルドする必要があります。

```
bash$ wget --no-check-certificate http://curl.haxx.se/download/curl-7.24.0.tar.gz
bash$ tar xkvf curl-7.24.0.tar.gz
bash$ cd curl-7.24.0
bash$ ./configure --without-ssl --prefix /path/to/curl-install
bash$ make
bash$ make install
bash$ cd ..
```

（注） curl バイナリとライブラリは、/path/to/curl-install にインストールされます。このディレクトリが存在しない場合は作成されるため、現在のユーザーには書き込み権限が必要です。次に、ソース tarball をダウンロードし、spec ファイルを抽出します。この手順は、プラグインがない場合の collectd の例とまったく同じです。

```
bash$ wget --no-check-certificate --directory-prefix=SOURCES
https://collectd.org/files/collectd-5.5.0.tar.bz2
bash$ tar tf SOURCES/collectd-5.5.0.tar.bz2 | grep '.spec$'
collectd-5.5.0/contrib/redhat/collectd.spec
collectd-5.5.0/contrib/aix/collectd.spec
collectd-5.5.0/contrib/sles10.1/collectd.spec
collectd-5.5.0/contrib/fedora/collectd.spec
bash$ tar xkvf SOURCES/collectd-5.5.0.tar.bz2 collectd-5.5.0/contrib/redhat/collectd.spec
bash$ mv collectd-5.5.0/contrib/redhat/collectd.spec SPECS/
bash$ rm -rf collectd-5.5.0
```

この spec ファイルは、/sbin/chkconfig を使用して collectd をインストールするように RPM をセットアップします。ただし、スイッチでは、代わりに /usr/sbin/chkconfig を使用する必要があるため、spec ファイルで次のように編集します。

（注） この tarball には 4 つの spec ファイルがあります。Red Hat の spec ファイルは最も包括的であり、正しい collectd バージョンを含む唯一のファイルです。これを例として使用します。

```
bash$ sed -r -i.bak 's%(^|\s)/sbin/chkconfig%\1/usr/sbin/chkconfig%' SPECS/collectd.spec
```

この点は、前の例と違っています。collectd rpmbuild プロセスは、libcurl の場所を認識する必要があります。collectd の spec ファイルを編集して、以下を追加します。

SPECS/collectd.spec で文字列 *%configure* を検索します。この行とそれに続く行は、rpmbuild が ./configure スクリプトに渡すオプションを定義します。

次のオプションを追加します：

```
--with-libcurl=/path/to/curl-install/bin/curl-config \
```

次に、rpmbuild コマンド オプションを含む Bash アレイが再度構築されます。次の違いに留意してください。

- *curl* をビルドされないプラグインのリストから削除
- *--with curl=force* の追加

```
bash$ rpmbuild_opts=()
bash$ for rmdep in \
> amqp apache ascent bind curl_xml dbi ipmi java memcachec mysql nginx \
> notify_desktop notify_email nut openldap perl pinba ping postgresql python \
> rrdtool sensors snmp varnish virt write_http write_riemann
> do
>  rpmbuild_opts+=("--without")
>  rpmbuild_opts+=(${rmdep})
> done
bash$ rpmbuild_opts+=("--with")
bash$ rpmbuild_opts+=("curl=force")bash$ rpmbuild_opts+=(--nodeps)
bash$ rpmbuild_opts+=(--define)
bash$ rpmbuild_opts+=("_unpackaged_files_terminate_build 0")
```

それからこれは次のように rpmbuild に渡され、ビルドおよび RPM パッケージ プロセス全体が開始されます：

```
bash$ rpmbuild "${rpmbuild_opts[@]}" -bb SPECS/collectd.spec
```

RPMs ディレクトリ内の結果の RPM には、collectd-curl も含まれるようになりました。これらの RPM ファイルをスイッチにコピーし、スイッチの Bash シェルからインストールすることができます：

```
bash$ rpm --noparentdirs -i /bootflash/collectd-5.5.0-1.ia32e.rpm
bash$ rpm --noparentdirs -i /bootflash/collectd-curl-5.5.0-1.ia32e.rpm
```

翻訳について
このドキュメントは、米国シスコ発行ドキュメントの参考和訳です。リンク情報につきましては、日本語版掲載時点で、英語版にアップデートがあり、リンク先のページが移動/変更されている場合がありますことをご了承ください。あくまでも参考和訳となりますので、正式な内容については米国サイトのドキュメントを参照ください。