Cross-compilation in pkgsrc

Taylor 'Riastradh' Campbell
campbell@mumble.net
riastradh@NetBSD.org

AsiaBSDcon 2015
Tokyo, Japan
March 15, 2015

pkgsrc: portable package build system

- `https://www.pkgsrc.org/`
- Framework for building third-party software on Unix-like operating systems.
- $> 15,000$ packages.
- Supported platforms:
  - NetBSD (first platform, based on mid-'90s FreeBSD ports)
  - GNU/Linux, GNU/kFreeBSD
  - FreeBSD, OpenBSD, DragonflyBSD, MirBSD
  - Haiku, MINIX 3
  - Solaris / SmartOS / illumos
  - OS X
  - IRIX, AIX, OSF/1, HP-UX, QNX, Cygwin

pkgsrc: portable package build system

- `https://www.pkgsrc.org/`
- Framework for building third-party software on Unix-like operating systems.
- $> 15,000$ packages.
- Supported platforms:
  - NetBSD (first platform, based on mid-'90s FreeBSD ports)
  - GNU/Linux, GNU/kFreeBSD
  - FreeBSD, OpenBSD, DragonflyBSD, MirBSD
  - Haiku, MINIX 3
  - Solaris / SmartOS / illumos
  - OS X
  - IRIX, AIX, OSF/1, HP-UX, QNX, Cygwin
- Works unprivileged, so you can use it in your home directory on a server you don't administer.

Anatomy of a pkgsrc package

- `DESCR` – Human-readable description.
- `Makefile` – Machine-readable description.
  - Tells where to download source code.
  - Rules for how to configure, build, install.
  - Etc.
- `distinfo` – Names, sizes, and hashes of source distribution. Provides cryptographic integrity check.
- `PLIST` – Packing list: lists files installed by package.
- `/usr/pkg/etc/mk.conf` – Site configuration for package options.

pkgsrc example: security/nettle, part 1

```
# $NetBSD: Makefile,v 1.13 2013/11/26 09:22:19 martin Exp $

DISTNAME= nettle-2.7.1
PKGREVISION= 1
CATEGORIES= devel security
MASTER_SITES= http://www.lysator.liu.se/~nisse/archive/ \
ftp://ftp.lysator.liu.se/pub/security/lsh/

MAINTAINER= pkgsrc-users@NetBSD.org
HOMEPAGE= http://www.lysator.liu.se/~nisse/nettle/
COMMENT= Cryptographic library
LICENSE= gnu-lgpl-v2.1

USE_LANGUAGES= c
USE_LIBTOOL= yes
USE_TOOLS+= gm4 gmake
GNU_CONFIGURE= yes
SET_LIBDIR= yes
CONFIGURE_ARGS+= --disable-openssl --disable-shared
```

pkgsrc example: security/nettle, part 2

```
.include "../../mk/bsd.prefs.mk"

.if !empty(USE_CROSS_COMPILE:M[yY][eE][sS])
CONFIGURE_ENV+= CC_FOR_BUILD=${NATIVE_CC:Q}
.endif

INFO_FILES= yes
TEST_TARGET= check
PKGCONFIG_OVERRIDE= hogweed.pc.in
PKGCONFIG_OVERRIDE+= nettle.pc.in

.include "../../devel/gmp/buildlink3.mk"
.include "../../mk/bsd.pkg.mk"
```

Building and installing a package[1]

```
# which socat
socat not found
# cd /usr/pkgsrc/net/socat
# bmake install
=> Bootstrap dependency digest>=20010302: found digest-20121220
=> Fetching socat-1.7.2.4.tar.gz
...
=> Checksum SHA1 OK for socat-1.7.2.4.tar.gz
...
===> Installing dependencies for socat-1.7.2.4
...
=> Tool dependency checkperms>=1.1: found checkperms-1.11
=> Full dependency readline>=6.0: found readline-6.3nb3
...
=> Creating binary package /tmp/.../socat-1.7.2.4.tgz
===> Install binary package of socat-1.7.2.4
# which socat
/usr/pkg/bin/socat
```

[1] On NetBSD, can use base system's `make`, but everywhere else we bootstrap `devel/bmake` for pkgsrc.

Binary packages: build once, install many times

- Building from source is necessary: verify source, audit programs, modify, etc.
- Building from source is slow: run compiler on lots of source code.
- Do it once, save the result, install binary packages after.

```
builder# cd /usr/pkgsrc/net/socat
builder# bmake package

client# PKG_PATH=/nfs/builder/usr/pkgsrc/packages
client# export PKG_PATH
client# pkg_add socat
```

Binary package bulk builds

- NetBSD provides binary packages for NetBSD on many architectures[2].
- Joyent provides binary packages for OS X[3] and illumos[4].
- I build binary packages for my own machines.
- You can too!

[2] `ftp://ftp.NetBSD.org/pub/pkgsrc/packages/NetBSD/`
[3] `http://www.perkin.org.uk/pages/pkgsrc-binary-packages-for-osx.html`
[4] `http://www.perkin.org.uk/pages/pkgsrc-binary-packages-for-illumos.html`

Cross-compiling NetBSD

- Every NetBSD build is a cross-build.
- `build.sh tools` builds cross-toolchain.
- `build.sh kernel=GENERIC distribution` builds NetBSD with the cross-toolchain.

Cross-compiling pkgsrc

- Use NetBSD `build.sh tools distribution` to get started.[5]
- `USE_CROSS_COMPILE=yes`
- `MACHINE_ARCH=powerpc`
- `TOOLDIR=/usr/obj.evbppc/tooldir.NetBSD-6.1.amd64`
- `CROSS_DESTDIR=/usr/obj.evbppc/destdir.evbppc`

```
# uname -m
amd64
# cd /usr/pkgsrc/net/socat
# bmake package
...
# cd /usr/pkgsrc/packages.powerpc/All
# pkg_info -Q MACHINE_ARCH socat-1.7.2.4.tgz
powerpc
```

[5] See `doc/HOWTO-use-crosscompile` for details.

Dependencies

- Some packages **depend** on other packages.
  - `tor` program uses `libevent` library at run-time.
    - `net/tor` **depends** on `devel/libevent`.
  - Compiling `tor` program requires `event.h` at build-time
    - `net/tor` also **build-depends** on `devel/libevent`.
  - Compiling `libxcb` requires turning XML into C header files with `xsltproc`.
    - `x11/libxcb` **tool-depends** on `textproc/xsltproc`.
  - Also **bootstrap-depends**, like tool-depends but for parts of the pkgsrc infrastructure.

Cross-compiling dependencies

- Use Intel Xeon to build `x11/xterm`, run on your powerpc-based thin client.
- `x11/xterm` must be built for `MACHINE_ARCH=powerpc`.
- `x11/xterm` depends on `x11/libxcb`[6].
  - `x11/libxcb` must be built for `MACHINE_ARCH=powerpc`.
- `x11/libxcb` *tool-depends* on `textproc/xsltproc`.
  - `textproc/libxsltproc` must be built for `MACHINE_ARCH=x86_64`.

[6] Via `x11/libX11`.

Build-depends vs tool-depends

- Both build-depends and tool-depends need to exist at build-time.
- *Build-depends* are cross-built and installed into `/usr/obj.evbppc/destdir.evbppc/usr/pkg/...`
  - Example: C libraries, needed for linker.
- *Tool-depends* are natively built and installed into `/usr/pkg/...`
  - Example: `xsltproc`, cross-compiler.
  - When built, `TARGET_ARCH` set to cross-compilation target.

Complications part 1: mixing up build-depends and tool-depends

- Originally, pkgsrc had only build-depends.
- `x11/libxcb` build-depended on `textproc/xsltproc`.
- Solution: change build-depends to tool-depends where appropriate.

Complications part 2: package builds tools internally

- Some packages depend on external tools like `x11/libxcb` depends on `textproc/xsltproc`.
- Others use internal tools, like `security/nettle` above.
- These try to use `CC`, which may be `powerpc--netbsd-gcc` for cross-compilation.
- Can't run the result on x86!
- Solution: set `CC_FOR_BUILD`, maybe patch package to use it instead.

```
.if !empty(USE_CROSS_COMPILE:M[yY][eE][sS])
CONFIGURE_ENV+= CC_FOR_BUILD=${NATIVE_CC:Q}
.endif
```

Complications part 3: file existence tests

- Package wants to know whether `/dev/urandom` will exist when run.
- Uses GNU `autoconf` to ask whether `/dev/urandom` exists *now*, when built.
- Build machine and target system may be different!
- But we know `/dev/urandom` will exist.
- Solution: tell `autoconf` up front.

```
.if !empty(USE_CROSS_COMPILE:M[yY][eE][sS])
.if ${OPSYS} == "NetBSD" || ${OPSYS} == "OpenBSD" || ..
CONFIGURE_ENV+= ac_cv_file__dev_urandom=yes
.endif
.endif
```

Complications part 3': file existence tests in pkgsrc

- From `x11/libdrm` in the past:

```
.if !exists(/usr/include/sys/atomic.h)
# libdrm won't find system atomic ops, use a package.
.  include "../../devel/libatomic_ops/buildlink3.mk"
.endif
```

- Solution: don't look in `/usr/include` — look in `/usr/obj.evbppc/destdir.evbppc`:

```
.if !exists(${CROSS_DESTDIR}/usr/include/sys/atomic.h)
# libdrm won't find system atomic ops, use a package.
.  include "../../devel/libatomic_ops/buildlink3.mk"
.endif
```

Complications part 4a: configure run-tests

- Similar to file existence tests.
- Program wants to know `sizeof(long)` at compile-time.
- Compiles a test program to print it, runs test program.
- Can't do that if building on 64-bit amd64 for 32-bit powerpc!
- Solution: binary search with compile-time assertions using cross-compiler.
- (Yes, seriously! GNU `autoconf` supports this with `AC_CHECK_SIZEOF`.)

Complications part 4b: configure run-tests

- Some are harder to replace.
- Tell the answers up front, maybe with patches.
- From `shells/zsh`:

```
.if !empty(USE_CROSS_COMPILE:M[yY][eE][sS])
.if ${OPSYS} == "NetBSD"
CONFIGURE_ENV+= zsh_cv_shared_environ=yes
CONFIGURE_ENV+= zsh_cv_shared_tgetent=yes
CONFIGURE_ENV+= zsh_cv_shared_tigetstr=yes
CONFIGURE_ENV+= zsh_cv_sys_dynamic_execsyms=yes
.endif
.endif
```

Complications part 5: problem children

- Some packages go to great effort to resist cross-compilation.
  - Perl
  - Python
- Workaround: just build on your powerpc thin client and ship binary packages back to x86 build machine to continue.
- (Solution: chainsaws and rototillers. Fix the build systems![7])

[7] It can be done: OpenWrt does it, with a *lot* of work. If you would like to help adapt their approach to pkgsrc, talk to me!

Related work

- OpenWrt: cross-compiled packages for Linux-based network appliances.
  - Linux-only.
  - Not general-purpose package system.
  - Much smaller than pkgsrc.
- `distcc`: run pkgsrc on thin client, run compiler remotely on x86 build machine.
  - Complex to set up: many moving parts (literally).
  - Hard to parallelize.
  - Compiler is a big part but not all of run-time.
- FreeBSD ports: run native compiler in user-mode emulator.
  - Many moving parts (figuratively).
  - Emulators are slow.
  - Less clean separation between host and target.

Future work

- Cross-OS compilation. Use SmartOS x86 cloud cluster to build for `MACHINE_PLATFORM=NetBSD-7.0-powerpc`.
- User interface improvements.
  - Can't do `bmake package MACHINE_ARCH=powerpc` for stupid reasons.
  - (When we switch to `MACHINE_PLATFORM` reasons will go away.)
  - Setting up cross-compiling requires a manual step to work around broken GNU `libtool`.
- Bulk builds.
  - `pbulk` doesn't understand build-depends vs tool-depends.
- Unprivileged builds for privileged installs.
  - Native and cross packages must both point at `/usr/pkg`.
  - (Unprivileged builds for unprivileged installs work fine — not a problem with privileges, just with different paths.)

Thank you!

Questions?