BuildStream
A distribution agnostic integration tool

10/13/2022

A BuildStream talk at
ApacheCon 2022

Historical overview

- Origins in the Baserock project
- Founded by Codethink in 2016 under the GNOME umbrella
- Used to build GNOME and Freedesktop-SDK releases
- BuildStream 1.0 released early 2018
- Bloomberg contributes to the project, accelerating development
- Build Meetup: Collaboration with projects in the build space
- BuildStream moves to Apache umbrella in 2020
- Ready for BuildStream 2.0

What is BuildStream ?

- A payload agnostic build orchestration tool
- Use BuildStream to build anything
  - Bootable system images
  - Bootstrapped compilers and runtimes
  - Static binaries, binary packages and bundles, container images, …
- Easily repeat and reproduce the same build on any build host
- Cache and share built artifacts with peers

Why BuildStream ?

- Integration engineering is not fun
  - Maintaining downstream patches
  - Fixing broken builds
  - Build automation / CI load balancing act
- There is a lot of integration engineering work to do
  - Better tooling makes integration engineering less painful
- Codethink does a lot of integration engineering
  - We’re always trying to push the needle in build & integration

Mission

- Payload agnostic
- Deterministic build sandbox
- Long term build repeatability
- Long term backwards compatibility
- Extensible / ability to build anything
- Developer facing convenience
- Project modularity and encapsulation

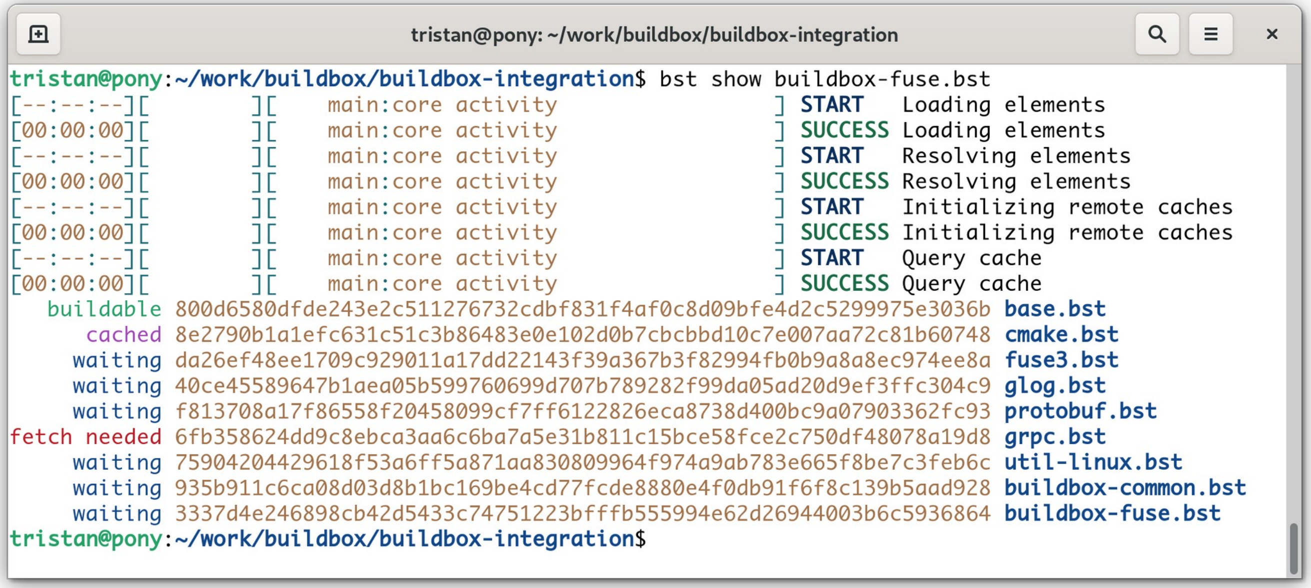
Details

Lets get familiar with the tool a bit

Elements

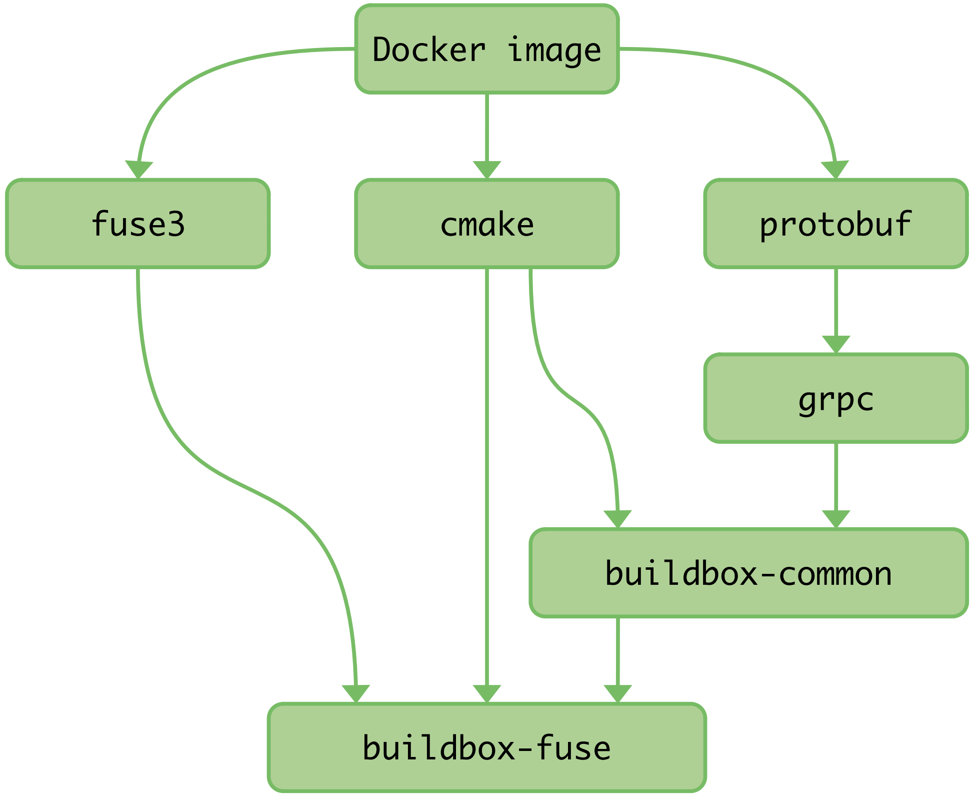
```
kind: cmake
build-depends:
- base.bst
- cmake.bst
- fuse3.bst
- ...
sources:
- kind: git_tag
  url: buildbox:buildbox-fuse.git
  track: master
  track-tags: True
  match:
  - "[0-9]*.[0-9]*.[0-9]*"
  ref: 0.0.61-0-ge363fdc88adef5db9ee40be8e89e68d4fd2c14a5
variables:
  cmake-local: |
    -DCMAKE_EXE_LINKER_FLAGS="-static-libgcc -static-libstdc++"
```

Cache keys


tristan@pony: ~/work/buildbox/buildbox-integration

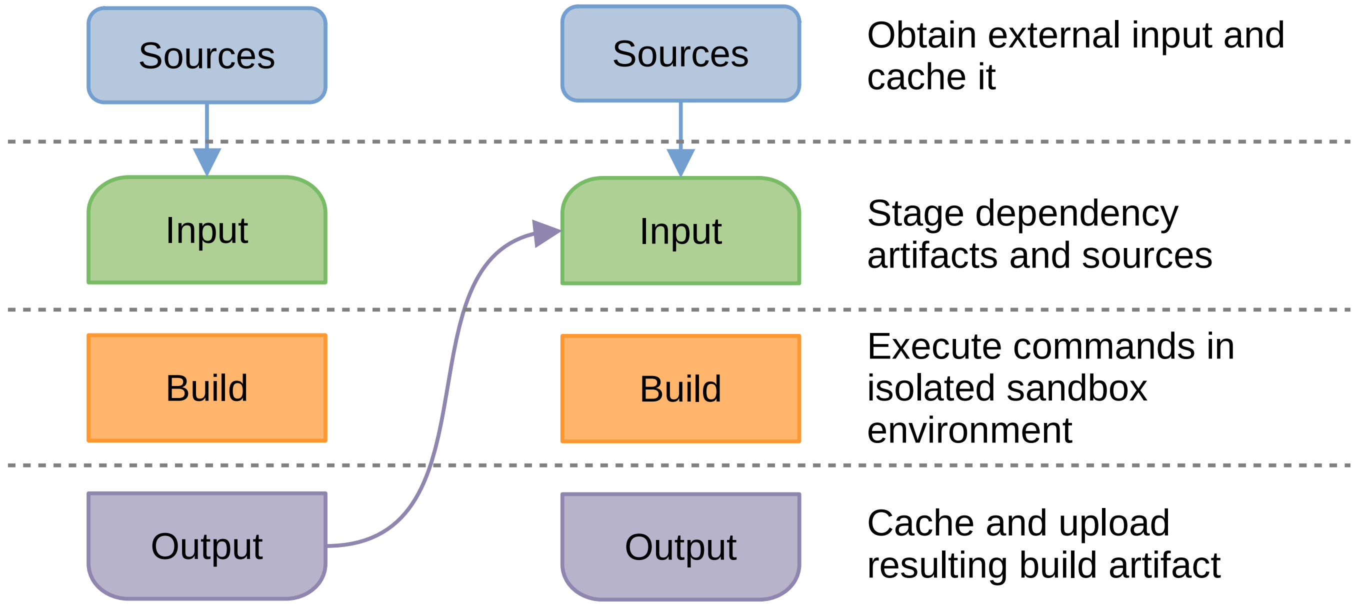
```
tristan@pony:~/work/buildbox/buildbox-integration$ bst show buildbox-fuse.bst
[--:--:--][        ][    main:core activity                 ] START   Loading elements
[00:00:00][        ][    main:core activity                 ] SUCCESS Loading elements
[--:--:--][        ][    main:core activity                 ] START   Resolving elements
[00:00:00][        ][    main:core activity                 ] SUCCESS Resolving elements
[--:--:--][        ][    main:core activity                 ] START   Initializing remote caches
[00:00:00][        ][    main:core activity                 ] SUCCESS Initializing remote caches
[--:--:--][        ][    main:core activity                 ] START   Query cache
[00:00:00][        ][    main:core activity                 ] SUCCESS Query cache
   buildable 800d6580dfde243e2c511276732cdbf831f4af0c8d09bfe4d2c5299975e3036b base.bst
      cached 8e2790b1a1efc631c51c3b86483e0e102d0b7cbcbbd10c7e007aa72c81b60748 cmake.bst
     waiting da26ef48ee1709c929011a17dd22143f39a367b3f82994fb0b9a8a8ec974ee8a fuse3.bst
     waiting 40ce45589647b1aea05b599760699d707b789282f99da05ad20d9ef3ffc304c9 glog.bst
     waiting f813708a17f86558f20458099cf7ff6122826eca8738d400bc9a07903362fc93 protobuf.bst
fetch needed 6fb358624dd9c8ebca3aa6c6ba7a5e31b811c15bce58fce2c750df48078a19d8 grpc.bst
     waiting 75904204429618f53a6ff5a871aa830809964f974a9ab783e665f8be7c3feb6c util-linux.bst
     waiting 935b911c6ca08d03d8b1bc169be4cd77fcde8880e4f0db91f6f8c139b5aad928 buildbox-common.bst
     waiting 3337d4e246898cb42d5433c74751223bfffb555994e62d26944003b6c5936864 buildbox-fuse.bst
tristan@pony:~/work/buildbox/buildbox-integration$
```

The pipeline



$$\begin{tikzcd}
 & \text{Docker image} \arrow[dl] \arrow[d] \arrow[dr] & \\
\text{fuse3} \arrow[ddd, bend right] & \text{cmake} \arrow[ddd] \arrow[dd] & \text{protobuf} \arrow[d] \\
 & & \text{grpc} \arrow[dl] \\
 & \text{buildbox-common} \arrow[d] & \\
 & \text{buildbox-fuse} &
\end{tikzcd}$$

Data flow



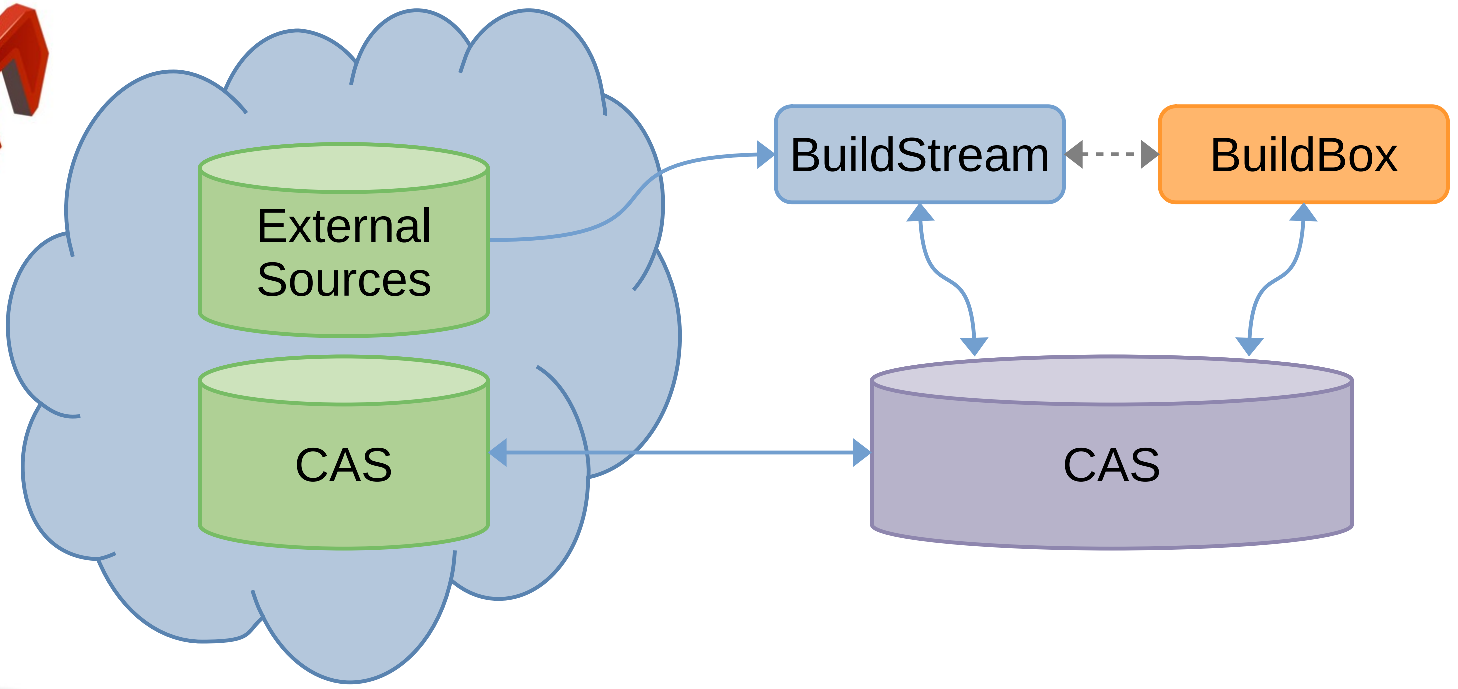
Obtain external input and cache it

Stage dependency artifacts and sources

Execute commands in isolated sandbox environment

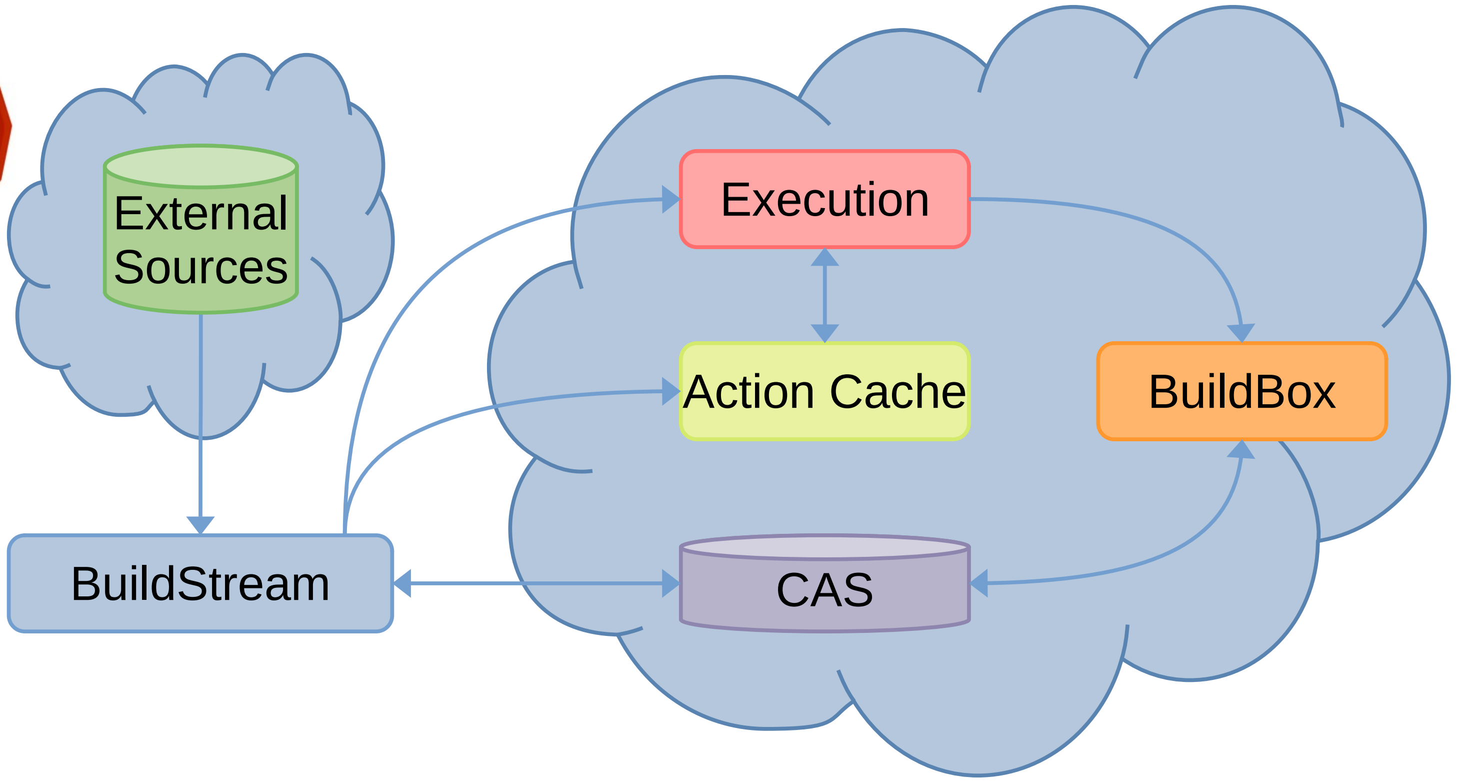
Cache and upload resulting build artifact

Data flow



CAS = Content Addressable Storage

Remote Execution



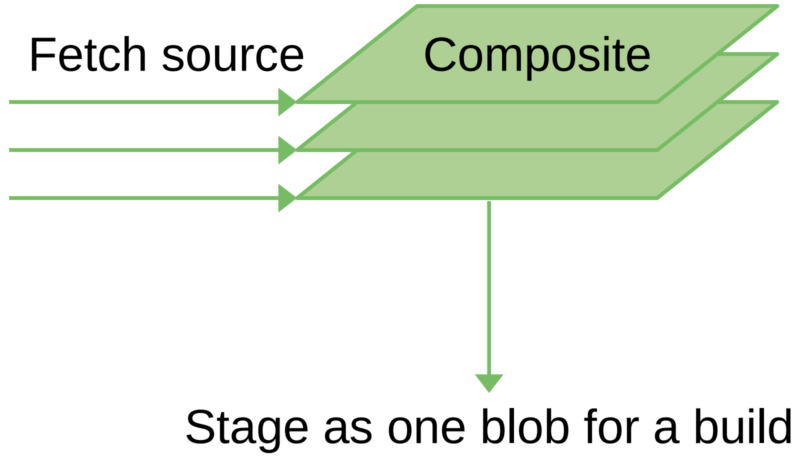
CAS = Content Addressable Storage

Building without internet

- The vast majority of FLOSS projects behave well
- Some projects attempt to download at build time
  - E.g: cracklib’s Makefile will try to download it’s words database
  - In this case we just place the words tarball in place before building
- Language oriented build systems introduce package management
  - pip (python)
  - cargo (rust)
  - npm (node.js)

Enter source composition

- Elements can already have multiple sources
- What if each source can have access to what was previously staged ?
- A more convenient experience for users of automated dependency consumption
- Caveat: Host tool trust



Source composition: pip

```
kind: pip
depends:
- base.bst
- python.bst
- ...
sources:
- kind: git
  url: github:mypythonproject.git
  track: master
- kind: pip
  requirements-files: requirements.txt
```

- Download a python project from a git repository
- Pip source can now read the requirements.txt file
- Use host pip to determine exact dependencies (pip freeze)
- Use host pip to download dependencies (pip download)

Source composition: cargo

```
kind: autotools
depends:
- base.bst
- rust.bst
- ...
sources:
- kind: git
  url: gnome:librsvg.git
  track: librsvg-2.48
- kind: cargo
  cargo-lock: Cargo.lock
```

- Download a rust project from a git repository
- Cargo source can now read the Cargo.lock file
- In this case, the Cargo.lock file provides enough information to download the deps as tarballs
- No need for host cargo

The Maven problem

- Looking grim when building java with mvn
- So far unable to automate the process of obtaining dependencies
- The `mvn dependencies:go-offline` command seems intended to address this, but fails
- Since the `pom.xml` supports minimal bound dependencies, much like pip does, we need to determine the latest version of any loosely defined dependencies (e.g. `pip freeze`)

Wanna help ?

- Any canonical method of *vendoring* dependencies with maven ?
- How can I determine all of the exact dependency versions which maven would download if the build were run *today* ?
- Our experiments attempted to recreate the .m2/ repository when staging, but this contains metadata and files which appear internal to maven

Questions ?

Project Website	https://buildstream.build/
Documentation	https://docs.buildstream.build/
Git	https://github.com/apache/buildstream/