Native Toolset Bootstrapping Design

This document is intended to provide insight into the design of native toolset bootstrapping / installation mechanisms. This document will guide the implementation of native toolset bootstrapping.

Overview

Arcade will provide a set of common installation libraries which will be provided to participating repos via Maestro. The common libraries will be used to acquire "installers" for native components.

Repos will be provided a local bootstrapping file (both a ps1 and an sh file). The bootstrapper uses the common libraries to install native toolset dependencies.

Definitions

Tool – a native toolset dependency (cmake, python, etc…)

Native Asset – Packaged native artifact, also known as a tool but specifically related to the asset as provided by the publisher of the tool. ie, in the case of cmake, one of the zip or tar files from https://cmake.org/download/

Installer – a script(s) used to deploy a native asset

Shim – wrapper script which is deployed to a platform that is referenced to execute the provided tool

Common Library – set of libraries available for native asset deployment to a platform

Arcade toolset libraries

Arcade will provide a set of libraries which will be used to install native tools. These libraries will include entry point scripts, the common libraries, and tool-specific install scripts. **The Arcade toolset libraries will be distributed via Maestro updates.**

Entry-point scripts

The entry-point scripts are the scripts which repos will use to bootstrap their defined native toolset dependencies. The entry-point scripts will read the repo's `global.json` file to determine which tool(s) and version to install. Only one version of each tool should be defined, though there is not (yet) logic to detect multiple tool versions being installed (currently if this occurs, last one installed will win).

Entry-point scripts are:

- init-tools-native.cmd

- init-tools-native.sh

Add a call to these scripts in your build script to bootstrap all tools specified in your global.json.

Common libraries

The common libraries will be used to determine which individual native tool installers are required by the repo and will execute the installers. They will be written in commonly supported formats (ps1 or bash).

Common library scripts

Development will show which common libraries will actually be required, but some examples of common library tasks include the following:

- Parse dependency requirements
- Install Xcopy Toolset
- Generate shims
- Download a file
- Extract an archive
- Determine current architecture
- Determine current OS

Native tool installers

The Arcade repo will define the installers for each supported native tool. The native tool installer will define how to install a tool locally (from blob storage). Certain, common install scenarios (xcopy deployable) may use a common library to perform the install. The install scripts will (initially) be generic install scripts (per tool) used to install any version of the native asset which has been published to Azure storage. If tool install formats noticeably change from version to version, we may need to adjust the install scripts accordingly (while maintaining backward compatibility).

Native tool installers will be responsible for supporting an "install" operation and a "clean" (uninstall) operation. See existing installers for examples of "clean".

Shims Most native tools will need to provide a "shim" via the common library scripts. "Shims" are used to execute the native toolset. Shims bridge the gap so that we have a single well-known entry point that can be used for our native toolset. Having shims allows us to put all of them in a single folder (in a given repo) so that we can use them to access the tools rather than managing path access to every known toolset.

It is possible that a native toolset will require more than one shim.

Azure Blob Storage Format

Native toolset assets will be placed in an Azure blob storage container. The default location is https://netcorenativeassets.blob.core.windows.net/resource-packages

You can browse the installers available using this url - https://netcorenativeassets.blob.core.windows.net/resourc
packages/?restype=container&comp=list

Blob storage layout

```
\external
  \linux
    \tool1
      -installer
      -additional resources needed for install
    \tool2
  \windows
    \tool1
    \tool2
```

external resources folder structure

The `external` folder is a folder structure that contains all installers and resources for external dependencies. These are zips / tarballs /etc… provided by a tool publisher which we have republished into Azure blob storage, organized in folders by the operating system and tool to be installed.

Example - resource-packages container

```
\external
  \linux
    \cmake
      -cmake-3.11.1-Linux-x86_64.tar.gz
      -cmake-3.11.0-Linux-x86_64.tar.gz
  \windows
    \cmake
      -cmake-3.11.1-win32-x86.zip
      -cmake-3.11.1-win64-x64.zip
    \python
      -python-3.6.5-win64-x64.zip
      -python-3.7.0b3-win32-x86.zip
```

Questions

How do I know what installers are already available in the container?

https://netcorenativeassets.blob.core.windows.net/resource-packages/?restype=container&comp=list

How will we handle installers if there are distro specific requirements?

This will likely come up very quickly and deserves consideration. The current plan is to allow each installer to handle this as needed.

I need to onboard a new native tool. What do I do?

- Upload the required files for tool installation to the https://netcorenativeassets.blob.core.windows.net/reso packages azure storage container (contact @dnceng if you need assistance) using the specified structure and naming convention:
- **Windows**
  - Packages should be uploaded to the `windows/<tool-name>` folder structure in the container.
  - Naming convention for packages is as follows `<tool-name>-<version>-<win32 | win64>-<x86 | x64>.zip`
  - If a zip file will be unpacked on Windows, create it only using a tool that follows the zip format's specifications and allows for reliable extraction using .NET's `System.IO.Compression` APIs or `powershell`
    - Creating the zip file in `powershell` using `Compress-Archive` would be one recommended approach.
  - Once the package has been uploaded, you should be able to add a reference to it in your global.json file.
- **Linux**
  - There is no requirement for a strict naming convention for Linux tools, taking into account that there may be different packages available for different distros and configurations. Upload the tool packages to the `linux/<tool-name>` folder structure in the container.
  - Write an installer for the tool. This should be a script called 'install-tool.sh' and create a Pull Request to the dotnet/arcade repo to introduce the installer in the eng/common/native folder.
    - Example: install-cmake.sh

How do you determine which version of the common libraries / installers to use?

Maestro will provide updates to the scripts.

How do you determine which native tools to install?

The native tools will be defined in a `native-tools` section of the repo's `global.json` file.

Example:

```
{
    "sdk": {
        "version": "2.1.100-preview-007366"
    },
```

```
    "msbuild-sdks": {
        "RoslynTools.RepoToolset": "1.0.0-beta2-62719-04"
    },
    "native-tools": {
        "cmake": "3.11.1"
    }
}
```

Why is each native dependency required to have an "installer", why isn't the local repo handling unzipping and laying out the assets?

I think that this model will allow us to be a bit more flexible in the types of dependencies that we install and provide a method for non-xcopy deployable dependencies to be installed in the future. The tool installers may make use of common libraries for installs though.

We are looking into improving this experience to genericize the installers and reduce boilerplate.

Was this helpful?