Toolset Principles and Guidance

For purposes of this document, 'Toolset' refers to tools which create and/or modify the shipping product.

Principles

- **Consistent and serviceable across the stack:** The minimum number of tool versions should be used with the ideal being one (serviceable) version per tool for the entire .NET Core product.
- **Visible:** It must be obvious which tool is being used in which situation.
- **For tools MSFT owns, the latest shipping version is used to build .NET Core:** Our product/s should be built using the last shipping version of our toolsets.

Policy / Guidance

General

- Which tool version used across the stack should be determined in one place. For more comments on this, see farther below.
- The latest shipped version of a tool should be used. This implies ongoing diligence of the product teams to update toolset dependencies.
- As needed, dependencies can be taken on brand new, non-shipping versions of a toolset, and conversely, older versions too - so long as tactics approves.
- Tools should be bootstrapped into the build where ever possible. It is recognized that this is not always reasonable (or even the best approach), but is still desireable as we try and get as close as possible to 'clone and build'. NOTE: There will still need to be provision for source-build to build "offline".

Servicing (LTS)

- All toolsets should be the latest RTM (serviceable) version for LTS.
- Toolsets should only change as needed, and with explicit consent of tactics.

VC toolset

- VC tools are brought in via Visual Studio, preferably via the build sku. Given the install limitations of VS and the Windows SDK, VC tools are provided via VM images which make up our build/test pools.

- Tools from the latest public preview of VS are available via a machine pool. Private previews of VS are not widely available and are used for targeting testing only.
- The guidance is to build against the latest RTM'd version of VC, and do testing on the latest public preview versions of VS, such that upgrades

can be done with confidence.

.NET SDK / Managed toolset

- Arcade SDK (shared infra for .NET Core) must always reference the latest preview version of the .NET Core SDK at a minimum. Immediately after shipping a preview, Arcade will update its .NET Core SDK reference.
- In cases where a newer version of the .NET Core SDK is needed, a non-shipping (newer) version can be referenced by the Arcade SDK with approval from tactics.
- In cases where an older version of .NET (and by implication Roslyn) is needed, exceptions are supported via Arcade, but should be approved by tactics and considered highly temporary.
- Roslyn version is brought in as a dependency via the Arcade SDK. It is not recommended for any build to take a direct dependency on Roslyn

Linux native toolsets

- The native *nix build tools are mostly managed via Docker containers.
- Where possible, the Docker containers should be shared across the teams.

Mac native toolsets

- Mac tools are managed via O/S itself. Here too we have both hosted and private machine pools.

Other

- Any other tools not directly called out should be managed via Arcade, and preferably bootstrapped in as part of the build.
- All toolset updates will be communicated in advance by the engineering services team.
- The implication here is that product teams should expect to deal with the toolsets updating on a relatively high cadence. However, this in turn means that the delta will be much smaller for each update.
- The latest version principle does not apply to .net runtimes and refpacks when used for compat testing

Was this helpful?