SBOM Generation Guidance

Background

An SBOM (Software Bill of Materials) is a formal record containing the details and supply chain relationships of various components used in building software. Physically, an SBOM is usually a single file (such as .json) that captures this information about the software from the build. As part of the requiements to be compliant with the Executive Order on Cyber Security, each repository involved in the composition of .NET needs to produce and upload an SBOM for each job of their official build that produces or modifies any assets. In this initial phase, we are focusing our efforts to have 6.0 and 7.0 builds generate the SBOMs.

Use with Arcade

As an initial phase for SBOM generation, Arcade provides a new YAML template to encapsulate the creation and upload of the SBOM to the build's artifacts, leveraging the ADO SBOM generator task. This section describes how to integrate this template into your repo's official build depending on your current usage of the YAML templates that Arcade provides.

Repositories using Arcade's job(s).yml templates

If you are using Arcade from the ".NET Eng - latest" or ".NET 6 Eng" channels, and using the provided job templates (job.yml, jobs.yml), Your build legs should start attempting to generate and upload SBOMs automatically.

For most cases, this is all that a repository using the arcade templates will need to do to generate and upload their SBOMs. If the generation doesn't work as expected, follow the troubleshooting instructions. Once you have verified the SBOMs are being generated for all of your jobs, you can review your SBOMs for correctness, and set up retention rules for your release builds

Repositories not using Arcade's job(s).yml templates

We encourage the usage of the job templates, as it's the best way for newer changes to the infrastructure to be added to your repository via the Arcade dependency flow. In cases where it's not possible for your repository to use the job templates, you will have to insert the generate-sbom.yml template directly into each of your jobs that produce or modify assets.

A minimal example follows:

```yaml
# One stage, one job: just build and generate an sbom
stages:
- stage: build
  displayName: Build
  jobs:
```

```yaml
- job: build (Windows)
    pool:
      name: NetCore1ESPool-Internal
      demands: ImageOverride -equals windows.vs2019.amd64

    steps:
    - checkout: self
      clean: true

    - script: eng\common\cibuild.cmd
        -configuration release
        -prepareMachine
      displayName: Windows Build / Publish

    - template: eng\common\templates\steps\generate-sbom.yml
```

Much of this template can be used as-is for most repositories, with defaults based on the common Arcade configurations. The template allows customization of behavior via the following parameters:

- `PackageVersion`: Version that will be reported by the SBOM. For repositories based on Arcade's main branch, this should be "7.0.0", and "6.0.0" for .NET 6 release branches.
- `PackageName`: default is '.NET'. Contact @dotnet/dnceng if you think your repo should use a different name.
- `ManifestDirPath`: Determines where in the build agent the SBOM will be generated to, defaults to `$(Build.ArtifactStagingDirectory)/sbom`
- `BuildDropPath` : Determines the directory that the SBOM tooling will use to find build outputs. Defaults to `$(Build.SourcesDirectory)/artifacts` to match Arcade's convention.
- `sbomContinueOnError`: By default the tasks are set up to not break the build and instead continue on error if anything goes wrong in the generation process.

Reviewing generated SBOMs for correctness

To verify that the functionality worked as expected in your repository:

1. The following steps should have been added to every job that is using the templates:
    - **Prep for SBOM generation**: Prepares the output directory for the sbom manifest and sets up the name of the artifact that will be uploaded.
    - **Create SBOM output folder**: Creates the directory where the sbom will be stored in the build agent.
    - **Generate SBOM manifest**: calls the SBOM generator task to produce the SBOM.

- **Publish generated SBOM**: Uploads the SBOM to the build's artifacts

2. In the build's artifacts, you should find a new artifact for every job that added these new steps. The artifacts will be named with a pattern of `<stage>_<job>_<SBOM>`

For each SBOM produced, you should download the `spdx2.2/manifest.spdx.json` file to make sure all the build outputs, OSS (Open Source Software) dependencies are captured in the manifest and that the product name and version for the SBOM match expectations.

It's recommended to open the manifest files with VSCode and format them so they are slightly more readable.

- `Build outputs`: These should be present in the `files` collection of the manifest. At this time, we are not concerned with the tooling overreporting the build outputs, but the SBOMs should be checked for missing output packages.
- `Dependencies`: These should be present in the `packages` collection of the manifest, and use the same detection mechanism as the component governance tasks.
- `Package name and version`: After the packages section, the last entry should mention the correct name and version for the software that the SBOM is about. The `name` property should read as ".NET 7.0.0" for main branches, and ".NET 6.0.0" for .NET 6 release branches.


```
"spdxVersion": "SPDX-2.2",
    "dataLicense": "CC0-1.0",
    "SPDXID": "SPDXRef-DOCUMENT",
    "name": ".NET 7.0.0",
    "documentNamespace": "https://sbom.microsoft/1:7eRdtVpLFUKo5AoKX3Hn2A:bhGpfqyfPUCyW
    "creationInfo": {
        "created": "2022-02-04T21:15:00Z",
        "creators": [
            "Organization: Microsoft",
            "Tool: Microsoft.SBOMTool-2.1.4"
        ]
    },
    "documentDescribes": [
        "SPDXRef-RootPackage"
    ]
```

Retention rules for release build SBOMs

We are required to store the SBOMs for builds that are released to the public in that pipeline's artifacts so that they are available if they are requested by customers. In order to achieve this for your release builds:

- **For builds of repositories that produce assets referenced by the .NET release pipeline**: Retention rules will be applied automatically via the .NET release pipeline for the following repositories (which may appear in .NET's build drops):
  - https://dev.azure.com/devdiv/DevDiv/_git/vs-code-coverage
  - https://dev.azure.com/dnceng/internal/_git/dotnet-wpf-int
  - https://github.com/dotnet/aspnetcore
  - https://github.com/dotnet/command-line-api
  - https://github.com/dotnet/diagnostics
  - https://github.com/dotnet/efcore
  - https://github.com/dotnet/emsdk
  - https://github.com/dotnet/format
  - https://github.com/dotnet/fsharp
  - https://github.com/dotnet/icu
  - https://github.com/dotnet/installer
  - https://github.com/dotnet/linker
  - https://github.com/dotnet/llvm-project
  - https://github.com/dotnet/msbuild
  - https://github.com/dotnet/msquic
  - https://github.com/dotnet/razor-compiler
  - https://github.com/dotnet/roslyn
  - https://github.com/dotnet/roslyn-analyzers
  - https://github.com/dotnet/runtime
  - https://github.com/dotnet/sdk
  - https://github.com/dotnet/source-build
  - https://github.com/dotnet/source-build-reference-packages
  - https://github.com/dotnet/templating
  - https://github.com/dotnet/test-templates
  - https://github.com/dotnet/windowsdesktop
  - https://github.com/dotnet/winforms
  - https://github.com/dotnet/wpf
  - https://github.com/microsoft/vstest
  - https://github.com/nuget/nuget.client
- **For repositories that have their own release process and cadence**: Retention rules will need to be applied manually for any repository that isn't present in the above list. Not every build should be retained indefinitely, but rather whichever builds will be used to release assets to customers. To help with this, Arcade provides a PowerShell script and a helper YAML template that can be added together or individually to

build and release pipelines. The YAML template by default will attempt to retain the same build where the pipeline is running, and the script can also be ran by itself by providing the required parameters.

Following up from the minimal example above:

```yaml
# One stage, one job: Generate an SBOM, and retain the build forever
# if a variable to do so is passed to the build

parameters:
- name: retainBuild
  type: boolean
  default: false

stages:

- stage: build
  displayName: Build
  jobs:
    - job: build (Windows)
        pool:
          name: NetCore1ESPool-Internal
          demands: ImageOverride -equals windows.vs2019.amd64

        steps:
        - checkout: self
          clean: true

        - script: eng\common\cibuild.cmd
            -configuration release
            -prepareMachine
          displayName: Windows Build / Publish

        - template: eng\common\templates\steps\generate-sbom.yml

        - ${{if eq(parameters.retainBuild, true}}
          - template: eng\common\templates\steps\retain-build.yml
```

The template takes the following parameters if more configuration is required:

- `Token`: Default is the build's `(System.AccessToken)`. In cases where the pipeline that retains the build is in a different AzDO organization than the build to be retained, this should be configured to an Azure DevOps PAT with the build read + execute scopes.
- `AzdoOrgUri`: Azure DevOps organization URI for the build to be retained, in the 'https://dev.azure.com/ format. Default is the organization where the build is running.

- `AzdoProject`: Azure DevOps project for the build to be retained. Defaults to the project where the current pipeline is hosted.
- `BuildId` : Azure DevOps Build ID for the build to be retained. Default is the build where the template is running.

If the retention was successful, you should see the following in the Azure DevOps build view:

Troubleshooting

- If the SBOM generation task fails with the message:

  ```
  Encountered error while running ManifestTool generation workflow. Error: BuildDropPath
  ```

  It means that your build outputs might not match with the expected location for Arcade: `$(Build.SourcesDirectory)/Artifacts`. In this case you should modify the `BuildDropPath` parameter of the template to point to your build's output directory.

- For any other problems with the tasks or templates, you can reach out to the .NET Engineering Services team.

Further Reading

- More information on SBOMs
- Manifest Generation task documentation

Was this helpful?