Arcade SDK Publishing Infrastructure

This document describes the infrastructure provided by the Arcade SDK for publishing build assets.

What is V1 publishing?

The publishing infrastructure has multiple stage(s), these stages represent available channels. Only the stages corresponding to the default channel will execute. This is for arcade3.x only.

V1 came into existence when we branched for release/3.x in arcade. Main and arcade/3.x initially had the same publishing logic. Over time the publishing stage in arcade main evolved so that became V2 publishing.

Asset manifest Example :

`publishingVersion` is not present in V1.

```
<Build Name="https://dnceng@dev.azure.com/dnceng/internal/_git/dotnet-arcade-validation"
BuildId="20200915.7"
Branch="refs/heads/release/3.x"
Commit="0f733414ac0a5e5d4b7233d47851a400204a7cac"
AzureDevOpsAccount="dnceng"
AzureDevOpsBranch="refs/heads/release/3.x"
AzureDevOpsBuildDefinitionId="282"
AzureDevOpsBuildId="816405"
AzureDevOpsBuildNumber="20200915.7"
AzureDevOpsProject="internal"
AzureDevOpsRepository="https://dnceng@dev.azure.com/dnceng/internal/_git/dotnet-arcade-valic
InitialAssetsLocation="https://dev.azure.com/dnceng/internal/_apis/build/builds/816405/artif
IsStable="False"
Location="https://dotnetfeed.blob.core.windows.net/arcade-validation/index.json">
```

All the 3.1 servicing branches of repos use this version of the infrastructure.

What is V2 publishing?

V2 is a legacy publishing infrastructure that is no longer utilized. It's essentially V1 publishing with explicit publishing version info. It uses a stage per channel and repositories must take Arcade updates to get publishing updates (e.g. new channels or fixes).

What is V3 publishing?

In V3, a single job or stage 'Publish Using Darc' handles all publishing for all available channels. Even if the repo branch is associated to more than one default channel(s) there will be only one stage. V3 uses `darc`

`add-build-to-channel` to promote builds based on the current configured default channels for the branch just built. The maestro promotion pipeline is a pipeline used to publish the packages to the target channel(s). `Add-build-to-channel` queues a new build of this pipeline and waits for it to publish assets to the appropriate locations. The publishing job is run against Arcade's main branch by default, meaning that repositories do not need to take an Arcade update to be able to publish to newly created channels or get most publishing fixes.

Example from arcade-validation:

V3-publishing

Figure 1: V3-publishing

Basic onboarding scenario for new repositories to the current publishing version (V3)

In order to use the new publishing mechanism, the easiest way to start is by turning your existing build pipeline into an AzDO YAML stage, and then making use of a YAML template (eng/common/templates/post-build/post-build.yml) provided by Arcade to use the default publishing stages. The process is explained below step by step.

1. Update the Arcade SDK version used by the repository to `5.0.0-beta.20461.7` or newer.
2. Disable asset publishing during the build. There are two common situations here. Some build definitions make use of the `jobs.yml` template and others make use of the `job.yml` (singular). The former is a wrapper around a few things, among them the `job.yml` and `publish-build-assets.yml` templates. If your build definition doesn't use `jobs.yml` you'll need to directly pass the `PublishUsingPipelines` parameter to the included templates. See examples below.
    1. If the build job uses the `eng\common\templates\jobs\jobs.yml` template, set the parameter `enablePublishUsingPipelines` to `true`. See example below:

        ```yaml
        jobs:
        - template: /eng/common/templates/jobs/jobs.yml
          parameters:
            enablePublishUsingPipelines: true
        ```

    2. If the build job makes direct use of `eng\common\templates\job\job.yml` you will have to do the following changes.
        1. Set the `enablePublishUsingPipelines` parameter to `true` when instantiating `job.yml`:

```
jobs:
...
- template: /eng/common/templates/job/job.yml
  parameters:
    ...
    enablePublishUsingPipelines: true
    ...
```

2. Make sure that you use the template `eng\common\templates\job\publish-build-assets.yml` to inform Maestro++ that *all* build jobs have finished executing. Also, make sure that you are setting the template parameter `enablePublishUsingPipelines` to `true`:

```
jobs:
...
- ${{ if and(ne(variables['System.TeamProject'], 'public'), notin(variables['Bu
  - template: /eng/common/templates/job/publish-build-assets.yml
    parameters:
      ...
      publishUsingPipelines: true
      ...
```

3. You'll also need to pass the below MSBuild property to the Arcade build scripts.

Name	Value
/p:DotNetPublishUsingPipelines	true

For example, if the repo has the following configuration for invoking `cibuild.cmd`:

```
- _InternalBuildArgs: /p:DotNetSignType=$(_SignType)
    /p:TeamName=$(_TeamName)
    /p:DotNetSymbolServerTokenMsdl=$(microsoft-symbol-server-pat)
    /p:DotNetSymbolServerTokenSymWeb=$(symweb-symbol-server-pat)
    /p:OfficialBuildId=$(BUILD.BUILDNUMBER)

- script: eng\common\cibuild.cmd
    -configuration $(_BuildConfig)
    -prepareMachine
     $(_InternalBuildArgs)
```

after setting the needed MSBuild properties it should looks like this: "'YAML - _InternalBuildArgs: /p:DotNetSignType=($_SignType)/p : $TeamName$ =(_TeamName) /p:DotNetSymbolServerTokenMsdl=($microsoft-symbol-server-pat)/p : $DotNetSymbolServerTokenSymWeb$ =(symweb-

symbol-server-pat) /p:OfficialBuildId=($BUILD.BUILDNUMBER)/p :
$DotNetPublishUsingPipelines =(_PublishUsingPipelines)

```
- script: eng\common\cibuild.cmd
    -configuration $(_BuildConfig)
    -prepareMachine
     $(_InternalBuildArgs)
```

1. Transform your existing build-definition to a single stage. Do that by nesting the curren

   ```YAML
   jobs:
   - template: /eng/common/templates/jobs/jobs.yml
     parameters:
       enablePublishUsingPipelines: true
   ...
   ```

   should be changed to:

   ```YAML
   stages:
   - stage: build
     displayName: Build
     jobs:
     - template: /eng/common/templates/jobs/jobs.yml
       parameters:
         enablePublishUsingPipelines: true
   ...
   ```

   We suggest you to use the stage name *build* and have only one build stage. However, that'

1. Import the`eng\common\templates\post-build\post-build.yml` Arcade template at the end of

   ```YAML
   - ${{ if and(ne(variables['System.TeamProject'], 'public'), notin(variables['Build.Reason'
     - template: eng\common\templates\post-build\post-build.yml
       parameters:
         publishingInfraVersion: 3
         enableSourceLinkValidation: false
         ...
   ```

   The `post-build.yml` template accepts the following parameters:

```
  | Name                                    | Type     | Description
  | ----------------------------------------- | -------- | -----------------------------------
  | publishingInfraVersion                  | int      | Publishing infrastructure version -
  | enableSourceLinkValidation              | bool     | Run SourceLink validation during th
  | enableSigningValidation                 | bool     | Run signing validation during the p
  | enableNugetValidation                   | bool     | Run NuGet package validation tool c
  | symbolPublishingAdditionalParameters    | string   | Additional arguments for the Publis
  | artifactsPublishingAdditionalParameters | string   | Additional arguments for the Publis
  | signingValidationAdditionalParameters   | string  | Additional arguments for the Signing
  | publishInstallersAndChecksums           | bool     | Publish installers packages and che
  | SDLValidationParameters                 | object   | Parameters for the SDL job template
  | validateDependsOn | [array] | Which stage(s) should the validation stage depend on. | bu
  | publishDependsOn | [array] | Which stage(s) should the publishing stage(s) depend on. |

  After these changes the build job(s) will publish the build assets to Azure DevOps build a

  Examples of the use of the basic onboarding scenario can be found in  the following repos:

  * [Arcade](https://github.com/dotnet/arcade/blob/main/azure-pipelines.yml)
  * [Arcade-Validation](https://github.com/dotnet/arcade-validation/blob/main/azure-pipeline
  * [Arcade-Services](https://github.com/dotnet/arcade-services/blob/main/azure-pipelines.ym

2. Create or update eng/Publishing.props, adding the following MSBuild property:
  ```XML
      <PublishingVersion>3</PublishingVersion>
  ```

 Sample:
   ```XML
      <Project>
         <PropertyGroup>
            <PublishingVersion>3</PublishingVersion>
         </PropertyGroup>
      </Project>
   ```

 Example of the use of Publishing.props can be found in the following repos :

 * [Arcade-Validation](https://github.com/dotnet/arcade-validation/blob/6009d37b7ecacbb0bc1e

The pipeline for a build with stages enabled will look like the one shown below.

![V3-publishing](./images/V3-publishing.PNG)

Validating the changes
```

Since the post-build stages will only trigger during builds that run in the internal project

1. Create a branch on the Azure DevOps internal mirror of the repo that includes the pipelin
1. Set up the "General Testing Channel" as a default channel for the internal repo + branch

  ``` Powershell
  darc add-default-channel --channel "General Testing" --branch "<my_new_branch>" --repo "ht
  ```

1. Queue a build for your test branch
1. Once the Build and Validate Build Assets stages complete, the *Publish Using Darc* stage

Checksum generation

Arcade also includes support for automatically generating checksum files. To opt in to this

Example:

  ```XML
  <ItemGroup>
    <GenerateChecksumItems Include="@(OutputFile)">
      <DestinationPath>%(FullPath).Sha512</OutputPath>
    </GenerateChecksumItems>
  </ItemGroup>
  ```

Ensure that you do not set `publishInstallersAndChecksums=false` in your call to the `post-b

Enabling 'faster' publishing

There are generally two shapes for official builds:
- Build -> Publish to Build Asset Registry -> Validate Assets (SDL, NuGet, etc.) -> Publish
- Build -> Publish to Build Asset Registry -> Publish

In the second case, the use of an additional stage and job for publishing is superfluous. In

Eligibility

A build is eligible for faster publishing if:
- It does not wish to gate publishing on any logic after 'Publish to Build Asset Registry' -
- It is on V3 publishing - This is the case for all repos beyond .NET Core 3.1.

Enabling fast publishing

1. **Set the parameter on post-build.yml** - In your call to the post-build.yml template, pa
2. **Enable publishing during the Publish to Bar job** -

```
 - **If you are using the jobs.yml template** - Pass parameter `publishAssetsImmediately: tr
 - **If you explicitly call publish-build.assets.yml** - Pass parameter `publishAssetsImmedi

## More complex onboarding scenarios

### Integrating custom publishing logic

Repositories that make direct use of tasks in Tasks.Feed to publish assets during their *bui

However, if for some reason the infra in the default publishing stages don't meet you requir

**Note:** We strongly suggest that you discuss with the *.Net Engineering* team the intended

## PublishingUsingPipelines & Deprecated Properties

Starting with Arcade SDK version **5.0.0-beta.20120.2** there is not support anymore for the

- **The build definition sets `/p:DotNetPublishusingPipelines=true`:** Arcade will handle th

| Property      |
| ------------------------------ |
| DotNetPublishBlobFeedKey      |
| DotNetPublishBlobFeedUrl      |
| DotNetPublishToBlobFeed       |
| DotNetSymbolServerTokenMsdl   |
| DotNetSymbolServerTokenSymWeb |

- **The build definition doesn't set `/p:DotNetPublishingUsingPipelines` or set it to false:

| Property      |
| ------------------------------ |
| DotNetPublishBlobFeedKey      |
| DotNetPublishBlobFeedUrl      |
| DotNetPublishToBlobFeed       |

Furthermore, starting with Arcade SDK version **5.0.0-beta.20120.2** the default value for t

## Frequently Asked Questions

### Guiding principles of the new infra?

- **Controlled by Maestro++ Channels:** The locations where packages are published to are de

- **Publishing is decoupled from the build job:** Publishing is managed by the Arcade SDK en

- **Single view for building and publishing:** The new infrastructure doesn't use Release Pi
```

What are YAML stages?

Stages are a concept introduced by Azure DevOps to organize the jobs in a pipeline. Just as

Stages are the way that Azure DevOps is bringing build and release pipelines together, and a

Why use YAML stages for publishing?

Using stages for publishing seeks to unify the Arcade SDK build artifact publishing mechanis

* Clearly separate the concepts of build, test, publish and validate.
* Support publishing and validation errors to be reported in the build page UI.
* Stages can depend on each other, which provides a natural way to extend default Arcade pub

Are there new package feeds? Which feed will be used?

Each Maestro++ channel is configured in [source](https://github.com/dotnet/arcade/blob/main/

- **A transport feed:** used for publishing packages intended for use internally in the .Net
- **A shipping feed:** used for publishing packages that will be directly used by end users.
- **A symbols feed:** symbol packages (`.symbols.nupkg`) are published to this feed as well

The target feed will be public/private depending on whether the Maestro++ channel is public/

Each stable build (i.e., [Release Official Builds](https://github.com/dotnet/arcade/blob/84f

What benefits do I get from the new infrastructure?

There are a few benefits, but the bottom line is: you can rely on Arcade SDK and Maestro++ t

What's this "Setup Maestro Vars" job?

Currently Azure DevOps does not support communicating "YAML variables" across stages. The re

How will this change affect symbol publishing?

Symbol publishing to MSDL and SymWeb will be done as a regular part of publishing the build

Can we manually assign a build to a channel?

Yes, that's possible. You need to [use Darc to do that](https://github.com/dotnet/arcade/blo

Why the build assets aren't getting published anywhere?

Most frequent cause of this is that there is no Default Channel configured for the build. [T

Why do you need the DotNetPublishUsingPipelines parameter?

The `DotNetPublishUsingPipelines` is a flag that Arcade SDK uses to determine if the repo wa

What's PackageArtifacts, BlobArtifacts, PdbArtifacts and ReleaseConfigs for?

- **PackageArtifacts**: contains all NuGet (.nupkg) packages to be published.
- **BlobArtifacts**: contains all blob artifacts (usually .symbols.nupkg) to be published.
- **PdbArtifacts**: contains all PDB artifacts to be published to symbol servers - SymWeb &
- **ReleaseConfigs**: contains configuration files used by the post-build stages. In particu

Note: only packages and blobs described in at least one build manifest will be publishe

Where can I see publishing logs in V1?

The publishing logs are stored inside an Azure DevOps artifacts container named `PostBuildLo

Where can I see publishing logs in V3?

Under the `Publish Using Darc` job get the link to the newly queued build in the [Maestro pr

How to add a new channel to use V3 publishing?

Create the channel using [darc add-channel](https://github.com/dotnet/arcade/blob/main/Docum

In the Microsoft.DotNet.Build.Task.Feed/src/Model/PublishingConstants.cs file, create a new

TargetChannelConfig takes the following attributes

Params	Description	Value
ChannelId	Id for channel to publish	
isInternal	Publishing to an internal Channel or public channel	true or false
PublishingInfraVersion	Which version of the publishing infra can use this configuration.
AkaMSChannelName	The name that should be used for creating Aka.ms links for this channel
TargetFeedSpecification	List of feeds to publish (type of asset -> feed mapping)	
SymbolTargetType	Publish to MSDL or SymWeb symbol server	PublicAndInternalSymbolTarget
FilenamesToExclude	List of files to exclude from creating aka.ms links. Should be exact
Flatten	Whether or not to flatten the path when creating the aka.ms links	Defaults to

```C#
Eg:
Publishing to General Testing channel : General Testing

          // "General Testing",
```

```
new TargetChannelConfig(
    529,
    false,
    PublishingInfraVersion.Latest,
    "generaltesting",
    GeneralTestingFeeds,
    PublicAndInternalSymbolTargets),
```

Which feeds does Arcade infra publish to?

Feed Name	Intended Usage
dotnet-eng	Packages required for engineering infra https://pkgs.dev.azure.com/dnceng/public/_packaging/dotnet-eng/nuget/v3/index.json
dotnet-tools	Tooling packages, such as Symreader, Sourcelink, etc… https://pkgs.dev.azure.com/dnceng/public/_packaging/dotnet-tools/nuget/v3/index.json
dotnet5	.NET 5 shipping packages https://pkgs.dev.azure.com/dnceng/public/_packaging/dotnet5/nuget/v3/index.json
dotnet5-transport	.NET 5 non-shipping packages https://pkgs.dev.azure.com/dnceng/public/_packaging/dotnet5-transport/nuget/v3/index.json
dotnet3.1	.NET Core 3.1 shipping packages https://pkgs.dev.azure.com/dnceng/public/_packaging/dotnet3.1/nuget/v3/index.json
dotnet3.1-transport	.NET Core 3.1 non-shipping packages https://pkgs.dev.azure.com/dnceng/public/_packaging/dotnet3.1-transport/nuget/v3/index.json
dotnet3.1-blazor	Packages specific to Blazor 3.1 This is an example of a repo-specific feed/channel https://pkgs.dev.azure.com/dnceng/public/_packaging/dotnet3.1-blazor/nuget/v3/index.json
dotnet3	.NET Core 3 shipping packages https://pkgs.dev.azure.com/dnceng/public/_packaging/dotnet3/nuget/v3/index.json
dotnet3-transport	.NET Core 3 non-shipping packages https://pkgs.dev.azure.com/dnceng/public/_packaging/dotnet3-transport/nuget/v3/index.json

Can the feeds be overriden?

Yes. The feeds can be overriden by adding the following options when calling PublishArtifactsInManifest.proj:

```
/p:AllowFeedOverrides=True
/p:InstallersFeedOverride=$(InstallersFeedOverride)
```

```
/p:ChecksumsFeedOverride=$(ChecksumsFeedOverride)
/p:ShippingFeedOverride=$(ShippingFeedOverride)
/p:TransportFeedOverride=$(TransportFeedOverride)
/p:SymbolsFeedOverride=$(SymbolsFeedOverride)
```

How are the aka.ms links formatted?

The aka.ms links are generated using the `BuildQuality` parameter that is passed to PublishArtifactsInManifest.proj, and the `akaMsChannelName` parameter passed to the `TargetChannelConfig` constructor. When akaMsChannelName is specified, we will create aka.ms links for the assets being published to that channel. Additionally, these links are "flatten," meaning that only the filename is used in addition to the build quality and the channel name when constructing the links. Finally, all version information is stripped from the filename. For example, if the `buildQuality` is `daily`, `akaMsChannelName` is `6.0`, `flatten` is `true`, and the filename is `dotnet-sdk-6.0.100-12345.12-win-x64.zip`, the aka.ms link generated will be `aka.ms/dotnet/6.0/daily/dotnet-sdk-win-x64.zip`.

What build qualities are supported?

The build qualities that are supported are daily, signed, validated, preview, and ga. All official daily builds that publish using V3 should use the `daily` build quality. Signed and validated builds are generated by the staging process of the release process. Preview and GA links are generated at release time, on release day. All builds that have preview in the release version will be of the `preview` quality. All other builds will be marked as `GA`. GA builds do not append a build quality to the links.

Can we exclude symbols from publishing to symbols server?

Yes.

Create a file eng/SymbolPublishingExclusionsFile.txt in your repo, add the file name that has to be excluded from symbol publishing in SymbolPublishingExclusionsFile.txt.

Eg: tools/x86_arm/mscordaccore.dll tools/x86_arm/mscordbi.dll tools/x64_arm64/mscordaccore.dll tools/x64_arm64/mscordbi.dll

During publishing, arcade will pick up SymbolPublishingExclusionsFile.txt and exclude the symbols mentioned in it.

Was this helpful? 

How do I publish stable packages?

Stable packages are not published by Arcade except for dependency flow and testing purposes. Stable packages go to isolated feeds (to enable rebuilds), then repo owners push these packages to Nuget.org manually. Then these packages flow to dotnet-public feed via the mirroring process

“‘mermaid flowchart LR Packages[Built Packages]–>ArcadePublishing{Arcade Publishing} ArcadePublishing–>|Non-Stable|PermanantFeeds[Permanant Feeds] ArcadePublishing–>|Stable|IsolatedFeeds[Isolated Feeds] IsolatedFeeds–>DogFoodingAndDepFlow{Dog-Fooding and dependency flow} PermanantFeeds–>DogFoodingAndDepFlow Packages–>|Release final stable packages on NuGet.org|NuGet[NuGet.Org] NuGet–>|Mirroring|DotnetPublic[dotnet-public feed] DotnetPublic–>RepoUse[Repository use] DogFoodingAndDepFlow–>RepoUse