Arcade Servicing

This document is intended to describe Arcade servicing workflow and policies. Our goal is to ensure that supported versions of the .NET Core product remain serviceable over time. As the Engineering Services team, we will need to make major changes to Arcade and our services over time to support new versions of .NET Core, while still supporting older versions.

Mechanically, this is done in two ways: - Branch our tools (Arcade) with the product. - Rev the APIs of our services as needed.

For those inserting into Visual Studio, careful consideration should be given to which Arcade to take a dependency on. If latest features are needed, then 'n' (or main) might make sense, otherwise 'n-1' (servicing) will be more stable and have much less churn.

It's important to note that we *must* continue to validate any supported versions of Arcade or services.

Details

Where do we do work on Arcade SDK?

- `main` for future releases
- `release/<NET Core Major Version Number>` for servicing

How do I get my servicing fix into main?

- Fixes that apply to both main as well as servicing releases must be first checked into main and then cherry picked into the appropriate servicing branches.
- Code flow happens from servicing branches back into main for the purposes of completeness, but developers must **not** rely on this to get fixes into main.

When do we branch?

- Major releases of .NET Core, not minor

What do we 'branch'?

- **Yes**: Arcade repo (templates, SDK, etc.)
- **As Needed**: Services should 'branch' as needed via API versioning to maintain compatibility for versions used in various servicing releases.
- **No**: OSOB images do not branch, but old images must be able to be resurrected.

What is the bar for changes in servicing branches of Arcade?

- Uses same bar as product servicing, where our customers are the repositories.
  - Security
  - Exceptionally high impact issues reported by customers.
  - External dependencies change.
- See Change Bar and Changes Policy for additional details on change bars and communication policies.

When can Arcade servicing changes be merged?

Arcade servicing changes may be merged when product branches are open. When product branches are not open, they may be merged in very special cases (e.g. targeted fixes for specific repositories, non-aligned servicing schedules, etc.). Otherwise, they should wait in PR until the product branches open for the next round of serivcing fixes.

How do Arcade servicing changes flow to repositories

Arcade servicing changes flow like any other product change, through dependency flow. These subscriptions shall have two states: - Only triggered on demand - Triggered on every build of arcade. After branches open for a servicing release, servicing subscriptions shall be set to flow every build. This ensures that any set of changes checked in that timeframe will flow as quickly to repositories as possible. When branches close for the stabilization and coherency process, those subscriptions shall be set to flow only on demand to reduce risk that an accidental merge to the servicing branch can reset the coherency process. The coherency QB may choose to flow changes selectively (e.g. ones approved in tactics to get build ready) during this timeframe.

It is the repsonsibility of the coherency QB to ensure that the correct changes are merged, the correct updates flow, and the subscriptions are in the correct state.

What is the bar for changes affecting non-current (not latest) APIs in engineering services?

- Uses same bar as product servicing, where our customers are the repositories.
  - Security
  - Exceptionally high impact issues reported by customers.
  - External dependencies change.
- See Change Bar and Changes Policy for additional details on change bars and communication policies.

Mechanics of branching Arcade and services?

The mechanics of ‘branching’ our services tends to be service specific, but generally involves generating new API versions for breaking changes. For Arcade, the mechanics are a little more complex. The following is the process by which Arcade can be branched for major release ‘N’ of .NET Core. Examples are given from the branching of the .NET Core 7.0 Arcade version.

1. Create a new branch named `release/<N>` off of the `main` branch of `dotnet/arcade`, and push it to the public repository. Ensure the pool provider names in all yaml of this branch are updated to their “-Svc” versions.
2. Create a new branch named `release/<N>` off of the `main` branch of `dotnet/arcade-validation`, and push it to the public repository. Ensure the pool provider names in all yaml of this branch are updated to their “-Svc” versions.
3. Create and merge a pull request to update the package version numbers in the main branch to match the next major version of .NET (N+1). (example)
4. Add new channels for the new branches, classified (`-c`) as `tools`. Make a note of these channel ids, as they will be used later in updating the publishing constants used by Arcade for these channels.
   - `.NET <N or next version> Eng`
   - `.NET <N or next version> Eng - Validation` Example: “‘ darc add-channel –name “.NET 7 Eng - Validation” -c “Tools” Successfully created new channel with name ‘.NET 7 Eng - Validation’ and id 3115

darc add-channel –name “.NET 7 Eng” -c “Tools” Successfully created new channel with name ‘.NET 7 Eng’ and id 3114.

```
5. [Add default channel associations](https://github.com/dotnet/arcade/blob/main/Documentati
   for Arcade `release/<N>` to point to `.NET <N> Eng - Validation`
Example:
```

darc add-default-channel –channel “.NET 7 Eng - Validation” –branch release/7.0 –repo https://github.com/dotnet/arcade

```
6. [Create a subscription](https://github.com/dotnet/arcade/blob/main/Documentation/Darc.md#

Example from .NET 7 Subscription: (this in the text editor popped by `darc add-subscription`
```

Channel: “.NET 7 Eng - Validation” Source Repository URL: https://github.com/dotnet/arcade Target Repository URL: https://github.com/dotnet/arcade-validation Target Branch: release/7.0 Update Frequency: everyBuild Batchable: False Merge Policies: - Name: Standard Pull Request Failure Notification Tags: ”

```
7. Modify the new release branch  `release/<N>` of arcade-validation to promote builds
   to `.NET <N> Eng`. ([example of below changes](https://github.com/dotnet/arcade-validatio
   In the update-channel.ps1 remove calls to:
   - Get-AzDO-Build
   - Get-AzDOHeaders
   - Get-LatestBuildResult
   - Remove the reference to bellweather repos runtime, aspnetcore, installer, and the for I

   In azure-pipelines.yml
   - Include the new release branch under trigger `release/<N.x>`.
   - Update the branch name to `refs/heads/release/<N>` for the Validate_Publishing conditio
   - Update the display name from `Promote Arcade to '.NET Eng - Latest' channel` to `Promot
   - Update the display name from  `Promote Arcade to 'Latest' channel` to `Promote Arcade t

9. Create and merge a pull request to update [PublishingConstants.cs](https://github.com/dot
   in Arcade's `main` branch for new channels ([example](https://github.com/dotnet/arcade/pu
10. Reset arcade Maestro++ subscriptions targeting .NET release branches to source from `.NE

Example from .NET 7:
```

darc get-subscriptions –source-repo https://github.com/dotnet/arcade –channel ".NET Eng - Latest" –target-branch "release/7.0" –exact darc get-subscriptions –source-repo https://github.com/dotnet/arcade –channel ".NET Eng - Latest" –target-branch "release/7.0.1xx" –exact "`For each of these subscriptions, call`darc update-subscription –id –no-trigger`, updating the channel to`.NET Eng`. Once completed, the above two commands should show no subscriptions, while adjusting the commands for`.NET Eng' should show all the ones from the initial versions of the command.

Was this helpful?