Dependency flow onboarding

There's a set of steps that need to be completed so the versions of assets your repository depends on are updated and also the assets your repository produces are updated in upstream repositories.

- Dependency flow prerequisites
- Publishing dependencies
- Consuming dependencies

Prerequisites

- If your repository is not part of the Dotnet GitHub organization, contact an administrator for the organization to install the dotnet-maestro app in your repository: https://github.com/apps/dotnet-maestro

The following pre-requisites are not hard requirements, but enabling dependency flow will be much simpler if you are using these processes:

- Internal builds produced out of https://dev.azure.com/dnceng/internal
- Publishing using the Arcade SDK

Publishing

Steps for publishing

1. Copy files from eng/ folder Copy `Versions.props`, `Versions.Details.xml` and the `common/` folder from the `eng/` folder of the arcade-validation repo.

This folder contains required version files as well as Pipeline templates used for publishing assets.

For more information about version files go to: https://github.com/dotnet/arcade/blob/master/Documentation

2. Enable Arcade publishing and publishing to the Build Asset Registry To enable publishing of your assets, follow the instructions outlined in https://github.com/dotnet/arcade/blob/master/Documentation/CorePackages/Publishing.md#basic-onboarding-scenario

Consuming

Steps for consuming

1. Add or copy Version.Details.xml Dependency consumption depends on the details file defining what dependencies your repo is going to consume.

You can create this file or copy the file from arcade-minimalci-sample's Version.Details.xml file as a starting point.

2. Add or copy your expression files Dependency flow will update versions in your expression files.

You can create these files or copy them from the arcade-minimalci-sample repo:

- Version.props
- global.json

3. Add subscriptions and channels More about Channels, Branches, and Subscriptions

Scenarios

3.1. Join the `arcade-contrib` team

1. Go to https://github.com/orgs/dotnet/teams/arcade-contrib/members
2. Click on "Request to join"

3.2. Set up your darc client Once you are part of the `arcade-contrib` team

1. Go to https://maestro-prod.westus2.cloudapp.azure.com/
2. Click "Sign in" in the upper right of the page
3. Give consent to "DotNet Maestro"
4. Click on your name and then on "Tokens"
5. Choose a name for your token and then "Create"
6. Copy the created token
7. Open a powershell or bash prompt and navigate to a repository that has the arcade toolset.
8. Run `.\eng\common\darc-init.ps1` or `./eng/common/darc-init.sh`. This will install darc as a global tool.
9. Run `darc authenticate`
10. Place the token into the `bar_password` field. You may leave the rest of the fields as-is.
11. Save and close.

3.3. Get all existing channels

1. Run `darc get-channels` to display available channels. Arcade builds are published to the '.NET Eng - Latest' channel.

3.4. Create a subscription to get Arcade updates Darc can be used to create new subscriptions, either in interactive mode or non-interactive mode. Interactive will open an editor to modify the fields, while non-interactive expects all fields on the command line.

Interactive mode

1. Run `darc add-subscription`
2. Fill out the fields. For Arcade, this typically looks like:

```
Channel: .NET Eng - Latest
Source Repository URL: https://github.com/dotnet/arcade
Target Repository URL: <your repository URL>
Target Branch: <target branch for arcade updates, e.g. master>
Update Frequency: everyDay
Batchable: False
Merge Policies:
- Name: Standard
  Properties: {}
Pull Request Failure Notification Tags: ''
```

3. Save and close

Non-interactive mode

1. Run `darc add-subscription --channel ".NET Eng - Latest" --source-repo https://github.com/dotnet/arcade --target-repo <your repo> --target-branch master --update-frequency everyDay --ignore-checks WIP,license/cla --all-checks-passed -q`

These steps can be altered for additional subscriptions to other repositories.

Supported values

Merge Policies

Merge policies are a set of rules that, if satisfied, mean that an auto-update PR will be automatically merged. A PR is only merged automatically if policies exist and all are satisfied.

- AllChecksSuccessful. All PR checks must be successful, potentially ignoring a set of checks specified in `ignoreChecks`. Checks might be ignored if they are unrelated to PR validation. The check name corresponds to the string that shows up in GitHub/Azure DevOps.
- RequireChecks. Require that a specific set of checks pass. Check names need to be defined in the `checks` property. The check name corresponds to the string that shows up in GitHub/Azure DevOps.
- NoExtraCommits. If additional non-bot commits appear in the PR, the PR should not be merged.
- NoRequestedChanges - If changes are requested on the PR (or the PR is rejected), it will not be merged.
- Standard. Combines the AllChecksSuccessful and NoRequestedChanges policies. This is the recommended merge policy to use unless there's a reason to use the others.

Update frequencies

How often does a source repo flows dependencies into a target repo.

- everyDay. The target repo only gets the dependencies updated once a day.
- twiceDaily. The target repo gets the dependencies updated twice a day.
- everyBuild. The target repo gets the dependencies updated after every source repo build.
- everyWeek. The target repo gets the dependencies updated once a week (On Mondays).
- none. No updates will happen in the target repo.

3.5. Create a channel (optional, typically not needed) You only need to create a channel in rare cases. Most .NET 5 builds should be assigned to the ".NET 5 Dev" channel. However, if you do need to create a new channel:

1. Run the following darc command:

   ```
   darc add-channel --name "<channel name>" --classification "<classification>"
   ```

   The classification is typically 'product'. Later on this will be used to differentiate dev or non-product builds, etc.

3.6. Associate a branch with a channel This will associate each new build of a specific branch in a repository with a channel. This is not technically required. Outputs from any build of any branch could be associated with a channel after the upload to the build asset registry. For example, I might decide that 'dev/foobar', containing a one off fix, is what I want to flow into downstream repositories.

In practice, it's tedious to manually associate each new build with a channel. Most of the time, each build 'master' is intended for '.NET 5 Dev', each build of release/2.1 is intended for '.NET Core Release 2.1', etc. Associating a branch with a channel causes every new build of that branch to be automatically applied to the channel.

For most .NET Core repositories, 'master' should be associated with '.NET 5 Dev'.

1. Run the following darc command:

   ```
   darc add-default-channel --branch refs/heads/<branch> --repo <repo URI> --channel <targ
   ```

   Example: For corefx master (the .NET 5 development branch), this would be:

   ```
   darc add-default-channel --branch refs/heads/master --repo https://github.com/dotnet/co
   ```

2. Verify with `darc get-default-channels`

5. Validate At this time we don't have a way to notify users if something went wrong while updating dependencies but this work is tracked by https://github.com/dotnet/arcade/issues/821.

To validate that created subscriptions and channels work as expected you'd need to verify that a PR has been created on your subscription's `targetRepository` once a build from `sourceRepository` has successfully completed. If a PR was not created something went wrong and we can use darc to find out what happened

1. Obtain the id of the non-functioning subscription:

   ```
   darc get-subscriptions
   ```

   You can use various parameters to filter the list and find the subscription you're interested in. For instance:

   ```
   darc get-subscriptions --target-repo corefx
   https://github.com/dotnet/arcade (.NET Eng - Latest) ==> 'https://github.com/dotnet/cor
   - Id: c297d885-0692-40f8-6b97-08d61f281b4c
   - Update Frequency: everyDay
   - Merge Policies:
     AllChecksSuccessful
       ignoreChecks =
                      [
                        "WIP",
                        "license/cla"
                      ]
   - Last Build: N/A
   ```

2. Incase of any errors contact @dnceng.

Was this helpful? 👍 👎