Proposal:
Angular Development Phases

Merging, Branching & Deployment

Author: Pete Bacon Darwin

Creation Date: 9 January 2019

Last update: 09 June 2020

State: draft / pre-review → leads review → team review → **in effect**

Implementation Target: v10 timeframe (in effect by August 2020)

Approvals

Framework	Pawel, Alex R	
Tooling	Keen	LGTM
DevInfra	Joey	
DevRel	Stephen	
Components	Jeremy	LGTM
Cross leads	Igor, Jen	LGTM

Outstanding Questions/Issues

1. When to create a new version-branch
    a. feature-freeze
    b. release-candidate

    Jeremy: Branch on feature-freeze, but don't change the version to "rc" for ~two weeks

    Pete: See Feature-freeze branching alternatives section below to expand on the repercussions

2. When to start releasing minor version "N.1.x-next.y" versions
    a. If during the release-candidate of N.0.0 phase, then how to tag these releases

Jeremy: Don't publish any N.1.x versions until N.0.x is released to `latest`

3. How to target PRs to the release-train that is in the feature-freeze phase

   Jeremy: use the `target: rc` label (even though it's not RC yet, the change is still *targeting* the forthcoming RC) for changes that *must* target RC but *not* patch; normal `patch` changes will continue to land there

   Pete: I would continue to use the normal labels but just disallow the caretaker from merging `major` and `minor` labelled PRs during this period. The author should not need to care whether the phase is feature-freeze or not. The `target: rc` could be used to special case a PR that would otherwise have been `target: major/minor` but needs to be forced into the feature-frozen branch.

4. How to handle cherry-picking PRs that fail
   a. How to label manually backported PRs

   Jeremy: change author manually creates PR(s) against branch(es) that would receive cherry-pick, continues to label based on the type of change (should almost always be either patch or lts)

   Pete: I was wondering about a tag called something like: `target: specific` or `target: base` or `target: cherry-pick`, which would be used in this case and also remove the need for the LTS target. The semantic of this target label would be "merge to the branch that this PR is based upon only".

Feature-freeze branching alternatives

Create new **version-branch** at **feature-freeze [CHOSEN]**

*The mental model is that **feature-freeze** is the **release-candidate** phase but without a release-candidate version (i.e. ...-rc.x) being published to npm etc.*

During **feature-freeze** the following would happen:

- A new **"next" release-train** is created
- PRs marked **target:major** or **target:minor** will be merged to the **next release-train** (i.e. `master`) as per normal constraints
- PRs targeting the **feature-freeze release-train** would need to be marked **target:patch** or **target:rc**. (**target:patch** PRs would also be merged to **next** and **latest release-trains**)
  - There could be some complexity computing whether we are in feature-freeze and so which git branch is the feature-freeze branch.
- Deployments to angular.io
  - The **next release-train** continues to be deployed to **next.angular.io**
  - The **feature-freeze** branch is deployed to **rc.angular.io** (??)

- Releases to npm
    - From **feature-freeze** are marked `...-next.x`.
    - No releases from **next**

~~Create new **version-branch** at **release-candidate** **[DISCARDED]**~~

~~*The mental model is that **feature-freeze** is a continuation of the **next** phase (with continued `...-next.x` releases) but with constraints on what can be merged to master.*~~

~~During **feature-freeze** the following would happen:~~

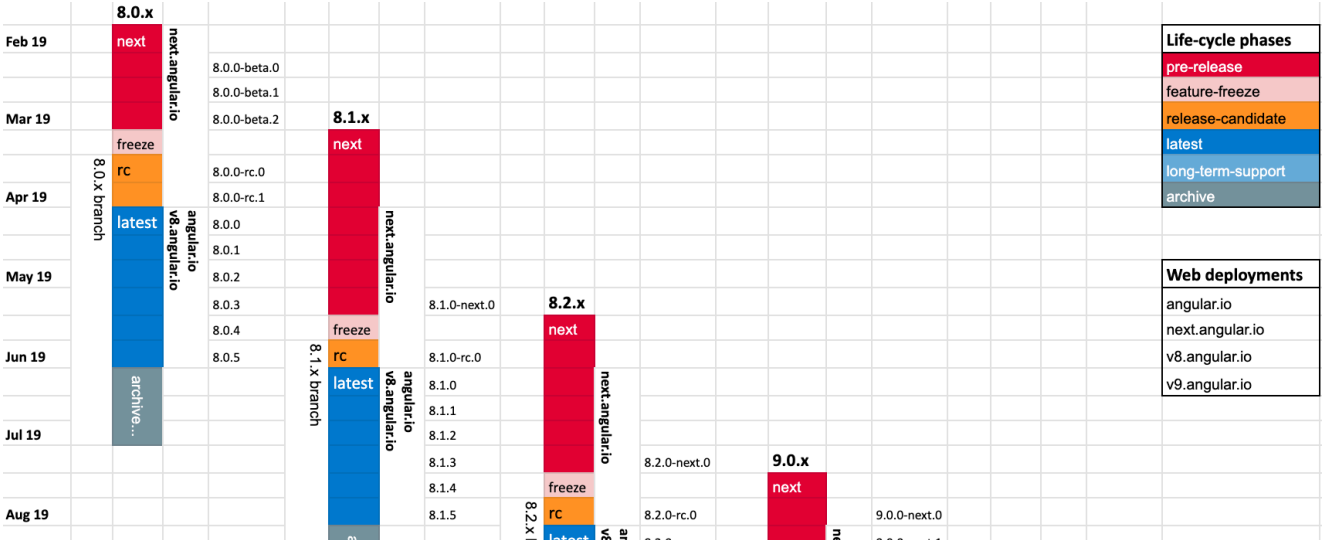
- ~~No new **"next" release-train** is created~~
- ~~PRs marked **target:major** or **target:minor** will be blocked~~
    - ~~This would need to be manually controlled by the caretaker~~
- ~~PRs marked **target:patch** will be merged to **feature-freeze** and **latest** **release-trains**~~
- ~~PRs marked **target:rc** will be merged only to **feature-freeze**??~~
- ~~Deployments to angular.io~~
    - ~~The **feature-freeze release-train** is deployed to **next.angular.io**~~
    - ~~No deployments to **rc.angular.io**~~
- ~~Releases to npm~~
    - ~~From **feature-freeze** are marked `...-next.x`.~~
    - ~~No releases from **next** (because there is no **next release-train** at this point)~~

Audience

- Developers who need to label PRs with targets.
- Caretakers who are merging PRs and cutting releases.
- Developers who are responsible for doc-gen tooling.
- General community who want to understand our release and deployment strategies.

TL;DR

Check out the following diagram that gives an illustration of the versioning, branching and deployment strategies defined here.



Scope

This document describes a proposed strategy for managing the merging, branching and deployment of the angular/angular repository, in particular during the release-candidate phase of development.

- What git branches are created and when.
- How to target different git branches when authoring PRs.
- Where to cherry-pick PRs (if any) when they are being merged.
- What sub-domains (if any) of angular.io to deploy to when commits land on branches.
- What and when to deploy to npm and what dist-tags to use.
- What and when to deploy to the builds repositories.
- From which branch to pull CLI documentation source files for each branch.
- Syncing into Google3.

This document is written from the point of view of how it will work. There is some work needed to align what is currently happening to this document (see Implementation).

There are various key terms used throughout the document. These are often highlighted in **bold**. They have specific meaning in the context of this document. You can find a quick list of these in the Glossary at the end of this document.

Overview

In the angular/angular project there is a **master** branch and **version-branches** (previously referred to as "patch branches") for all versions of Angular that have the same minor version. When a PR is accepted it is normally merged into the **master** branch and cherry-picked into zero or more **version-branches**.

When the current **master** branch is due to be released there may be a series of **release-candidate** releases. Previously there was ambiguity on when new **version-branches** should be created. This ambiguity caused confusion among team members, blocked **master** from receiving changes during this period, and created a risk of a bad change making it into **master** between the last release-candidate and the **production** release of this version.

In this document the branching strategy is explicitly defined: A new **version-branch** is created when the current **release-train** leaves its **next** phase and enters the **feature-freeze** phase.

This policy in turn exposes further ambiguity in other aspects of our process during the release-candidate mode. In particular, it is not clear how to target PRs to branches and where to deploy versions of angular.io.

*This document suggests a strategy based around **release-train** life-cycles that should simplify the development process for both developers and caretakers.*

See the Implementation section for a list of the changes that need to be implemented.

Release-trains

In this document a **release-train** refers to the life-cycle of a major/minor version of Angular (e.g. 7.2.0-rc.0, 7.2.0, 7.2.4 are all part of the 7.2.x **release-train**), from its initial development during its **next** phase, through a series of releases (e.g. release-candidate and patch) and deployments to angular.io.

Additionally the **last-minor**, is the **release-train** whose minor version number is the highest among all **release-train**s with the same major version number (e.g. once 8.0.0 is released then, out of 7.2.x, 7.3.x and 7.4.x, 7.4.x is the **last-minor**).

It is helpful to think of each **release-train** having a life-cycle of development **phases**:

- **next** - during this phase new features can be developed and, if the major version is being incremented, breaking-changes too. Throughout this **phase**, there are a series of releases (e.g. 8.1.0-next.0, 8.1.0-next.1, etc). This **phase** ends when there is a freeze on new features being accepted.

- **feature-freeze** - this phase lasts for up to two weeks between the **next** and **release-candidate** phases. Any "major features" or "features that require integration" (e.g. a framework feature requiring work in the CLI or Components projects) cannot land during this phase. They will need to be moved to the **next** phase of the following **release-train**. If such a feature is partially landed already, it must be scoped down, put behind a flag, or entirely removed.

- **release-candidate** - this phase begins when the first **release-candidate** for a **release-train** is published. During this **phase**, **release-candidates** will be published (e.g. 7.2.0-rc.0, 7.2.0-rc.1, etc). In these releases, there are no features, and only bug fixes for regressions and low-risk existing issues. This phase ends when the first patch version is released (e.g. 7.2.0).

- **latest -** this **phase** begins when the first production version for a **release-train** is published. During this **phase**, only patch increments to the version will be published (e.g. 7.2.0, 7.2.1, etc.). Only bug fixes, performance improvements and docs updates are included in these releases. This phase ends when the next **release-train** enters the **latest** phase, at which point this **release-train** will enter the **archive** or **long-term-support** phase.

- **archive** - this **phase** begins when the a **release-train** is no longer in the **latest** or **long-term-support** phase (e.g. the 7.2.x **release-train** moves to **archive** when the 7.3.x **release-train** has a **production** release and becomes **latest**). During this **phase**, no more development is planned for the given **release-train** and there should be no more versions published.

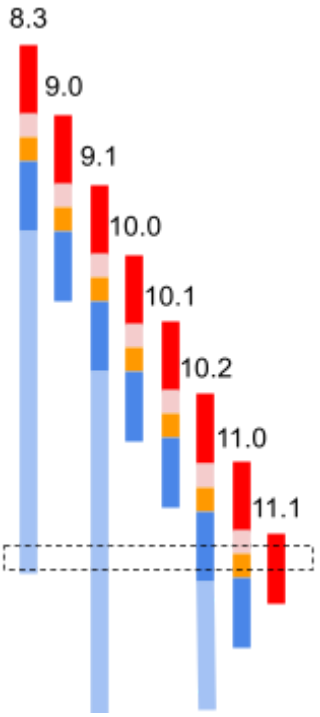
- **long-term-support** - This **phase** is only relevant for **release-trains** that are **last-minors**. When a new **release-train** that increments the major version number moves into the **latest** phase, the **last-minor** of the previous major version number enters the **long-term-support** phase instead of immediately entering the **archive** phase. During this **phase**, the **release-train** will continue to receive critical fixes and security patches only. Patch releases of such **release-trains** will be published ad-hoc when it is deemed necessary. This **phase** lasts for 12 months, then the **release-train** moves to the **archive** phase.

In normal operation once work begins on the next major **release-train** there will be no more minor releases and the current release train becomes the **last-minor** for its major version. See the **Exceptional minor release train** section below for more details about when this can change.

Active release-trains

A **release-train** is **active** if it can accept new code from PRs. At any one time there are multiple active release-trains based on their phase:

- **next** - there is always exactly one **release-train** in this phase at any time.
- **latest** - there is always exactly one **release-train** in this phase at any time.
- **feature-freeze/release-candidate** - a **release-train** in its **feature-freeze** or **release-candidate** phases is also active. There can be at most one release-train in this pair of phases at any time
- **long-term-support** - there are 2 or 3 **release-trains** in this phase at any time. This is because this phase is a timeboxed period of 12 months after each **last-minor** leaves the **latest** phase, during which time we expect to release two more major versions
- **archive** - these **release-trains** are never **active**.



The diagram to the side illustrates this. The dotted box indicates the active **release-trains** at a particular point in time. (The **archive** phases are not shown). You can see that a number of **release-trains** are active for the different phases.

- **next**: 11.1
- **release-candidate**:11.0

- **latest**: 10.2
- **long-term-support**: 8.3 and 9.1

Git Branches

The git repository contains named branches that receive commits, merged or cherry-picked from PRs that are accepted into the code base. Branches are created based on the life-cycle **phase** of the **release-train**. Here are the proposed branches:

- **master** - there is always a single **master** branch.
    - This branch receives commits and beta releases (e.g. 8.0.0-beta.0, 8.0.0-beta.1, etc) for the active **release-train** that is in its **next** phase.
- ***"version-branches"*** - once a **release-train** leaves the **next** phase a **version-branch** (e.g. 7.2.x) is branched off of master.
    - This branch will receive all commits that will go into each of the release-candidate and patch releases (e.g. 7.2.0-rc.0, 7.2.0-rc.1, etc; then 7.2.0, 7.2.1, etc) throughout the **feature-freeze**, **release-candidate**, **latest**, **long-term-support** (if any) and **archive** phases of the **release-train**.
- **g3** - this branch always points to the latest commit that has been synced into the internal Google3 repository. *The purpose of this branch is just informational, and primarily exists as a tool for caretakers to quickly compute the unsynced diff.*

The **release-train** of any particular git branch can be computed by reading the major and minor version numbers from the `version` property of the `package.json` file on that branch.

The minor version number can also be used to identify if this is a major or minor **release-train**: if it is 0 then this is a **major release-train** otherwise is it a minor **release-train**.

Targeting PRs

Each PR needs to be merged into one branch and cherry-picked into zero or more branches in the github repository. The decision of which branches should receive the commits from a PR is specified by labelling the PR with exactly one **target** label. Each **target** label maps to one or more **release-trains** (and so to their current **branch**) depending upon the phase of the **release-train** and whether it is a new major version.

Labelling

The author (or reviewer) should label the PR with *exactly one* appropriate **target: xxx** label to indicate which **branch(es)** are to receive the commits from the PR. The target labels available during normal development are:

Normal work labels - the developer will mostly use these, and only applies one to their PR.

target: major	a breaking change that *cannot* be part of a **minor** or **patch** release.
target: minor	a new feature that *cannot* be part of a **patch** release.
target: patch	a bug fix, refactoring, documentation change, etc that *can* be part of a **patch** release.

Special case labels - the developer applies one of these labels in special cases

target: rc	a critical fix for the **release-train** that is in the **feature-freeze** or **release-candidate** phase.
target: lts	a critical fix for a specific **release-train** that is in the **long-term-support** phase.

Merging/cherry-picking

The caretaker's merge script should compute to which branches to merge/cherry-pick the PR.

The git **branch** where a PR should be merged can be computed from the **release-train**:

- **next** is always pushed to **master** branch
- **rc** is pushed to the **version-branch** that has the highest semantic version, if it is also an "rc" *pre-release* version. (e.g. 7.2.0-rc-1 *but not 8.0.0-next.3 nor 7.1.4*)
- **latest** is pushed to the **version-branch** that has the highest semantic, *non-prerelease*, version.

The computation depends on whether the **release-train** in the **next** phase is a **major** or **minor release-train**; and whether there is currently a **release-train** in the **feature-freeze** or **release-candidate** phase.

The following tables identify the **release-train(s)** that should receive the "*merge*" of a PR, and which should receive the "*cherry-picks*", for a given **target label**. The "active pre-release" columns indicate whether there is a **release-train** in **feature-freeze**/**release-candidate** phase or not.

The **release-train** in the **next** phase is a **major**

	No active pre-release	Active pre-release
target: major	merge: **next** cherry-pick: none	
target: minor	merge: **next** cherry-pick: none	
target: patch	merge: **next** cherry-pick: **latest**	merge: **next** cherry-pick: **latest** + **active pre-release**
target: rc	*invalid (CI should error)*	merge: **next** cherry-pick: **active pre-release**
target: lts	merge: use the PR base-branch to identify the **release-train** cherry-pick: none	

The **release-train** in the **next** phase is a **minor**

	No active pre-release	Active pre-release
target: major	*blocked (cannot be merged)*	*blocked (cannot be merged)*
target: minor	merge: **next** cherry-pick: none	merge: **next** cherry-pick: none
target: patch	merge: **next** cherry-pick: **latest**	merge: **next** cherry-pick: **latest** + **active pre-release**
target: rc	*Invalid (CI should error)*	merge: **next** cherry-pick: **active pre-release**
target: lts	merge: use the PR base-branch to identify the **release-train** cherry-pick: none	

Failed Merges

PRs sometimes will not merge correctly or cannot be cherry-picked cleanly. In such cases the caretaker needs to know what branches map to the **release-train** phases.

- This should be provided by scripts - e.g. `currentNextBranch`, `currentLatestBranch`, etc

If the PR commits do not cherry-pick cleanly to one of the target branches then we need to have a process for how the PR gets processed.

- We could have a CI check that attempts to rebase the PR onto the branches defined by the target label?
- The caretaker should merge those branches that do merge cleanly in any case and then contact the PR author to ask them to create a new PR that has the conflicts resolved. But how should this new PR be labelled?

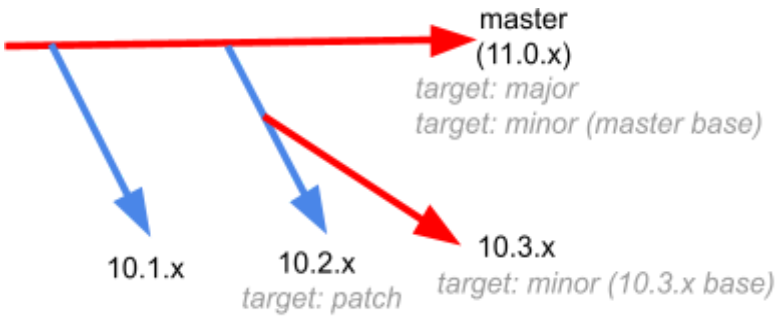
Notes

- The aim is to keep the development process simple for the PR author and for the caretaker. Therefore in normal processing there are no targets for exceptional minor release-trains nor any development-branches
    - Automated merging or cherry-picking into these branches is not supported.
    - It would be the responsibility of the caretaker to manually handle the creation of the **exceptional-minor** branch.
    - It would be the responsibility of the designated development-branch owner to handle creation and merging into the **development-branch**.
- PRs that target **long-term-support release-trains** should be rare and it is not possible to distinguish which of the many active release-trains to target based on the label. So these are a special case, where the author must create a specific PR for the **release-train** being targeted.

Other scenarios

Exceptional minor release-trains

In rare cases it might be necessary to backtrack and release an extra **minor** version once we have already started work on the next **major release-train**. In this case a new branch will be split off from the **last-minor** branch, rather than the **master** branch.



In this exceptional case, a special **exceptional minor branch** is created by the caretaker.

Note that any PRs with **target:minor** that have been merged into **master** since the **latest release-train** was branched from **next release-train** may need to be backported to the new **minor release-train**.

When such a branch has been created, the merge and release scripts will need to include this branch as appropriate. *This will be defined in more detail in a future design document.*

Development-branches

A development-branch is a special case where a collaborative team wants to merge PRs to a shared branch that is not part of a release-train. In general the development-branches should be avoided because the dev-infra team does not support them & there is a high risk of introducing regressions to the development-branches during a potentially very messy rebase process.

The best practice should be to break the work into smaller pieces that can be merged to the **next release-train**, hidden behind a flag if necessary.

Deployment

Google3

The **master** branch (and only the **master** branch) is continuously synced into google3 by the caretaker.

npm

The Angular packages (e.g. @angular/core, @angular/common, etc) are published to npm for releases of **versions** that are in the **next**, **feature-freeze**, **release-candidate**, **latest** and **long-term-support** phases. When we deploy to npm we tag the release with a **dist-tag** which is computed from the life-cycle phase of the **version** being released.

- **next**, **feature-freeze** and **release-candidate** phases => **next** tag
- **latest** phase => **latest** tag
- **long-term-support** phase => **vx-lts** tags (where x is major version number)

Notes

- Since **next**, **feature-freeze** and **release-candidates** will all use the same **next dist-tag**, there can only be one of these released to npm at any one time.
- It should be an error to release the first production release with open PRs labeled with "**target: rc**" - the release script / release instructions should check for that condition.

github "builds" repositories

Each time commits are merged into a branch on github the Angular packages are built and tested by CI. If the results are green then the packages are pushed to its associated **build repository** (e.g. https://github.com/angular/core-builds**),** by a CI script.

There is one **build-repository** for each Angular package. In these repositories, the branches mirror the branches in the main **source-repository** (https://github.com/angular/angular). The built packages are pushed to the same branch name in the build repositories as the source

repository (e.g. 7.4.x => 7.4.x and master => master, release-candidate => release-candidate). Notably, this setup supports publishing build-artifacts for **development-branches** as well.

angular.io websites

The angular.io domain and subdomains host the documentation for the Angular project. There is a separate subdomain for the **next/feature-freeze release-train**, the **release-candidate release-train** and for the **last-minor** of each major version.

Subdomains

- **next.angular.io** - shows the docs for the **release-train** that is in its **next** phase. It represents the bleeding edge of what is being developed in Angular.
- **angular.io** - shows the docs for the **release-train** that is in its **latest** phase. No beta or release-candidate releases are deployed here. This represents the latest stable release of Angular.
- **rc.angular.io** - shows the docs for the **release-train** that is in its **feature-freeze** or **release-candidate** phases. When there is no **release-train** in a **feature-freeze** or **release-candidate** phase, then this subdomain redirects to angular.io.
- **vN.angular.io** (where N is a major version number) - shows the docs for the **last-minor release-train** that matches the major version number.
    - When the **release-train** is in its **next** phase, then this domain redirects to next.angular.io.
    - When the **release-train** is in its **feature-freeze** or **release-candidate** phases, then this domain redirects to rc.angular.io.
    - When the **release-train** is in its **latest** phase, then this domain redirects to angular.io.
    - When the **release-train** is in **long-term-support** or **archive** phases, then this domain points to its own **hosting-site** for that version.

Firebase hosting

Each subdomain is either a redirect to another subdomain (as described in the previous section) or is hosted by a Firebase **hosting-site**.

The following Firebase **hosting-sites** are created:

- **next-angular-io** - this project always hosts the **next.angular.io** subdomain. Each PR that is merged to the **master** branch triggers a new deployment to this project.
- **rc-angular-io** - this project always hosts any **release-train** that is currently in its **feature-freeze** or **release-candidate** phase. If there is no **release-train** in this **phase** then this site lies dormant. Each PR that is merged to a **version-branch** of a **release-train** that is in this phase, triggers a new deployment to this project.
- **vN-angular-io** (where N is a major version number) - this project always hosts the **last-minor release-train** that matches the major version. Such **hosting-sites** are created when a **release-train** starts with a new major version.

Deployment example

This example walks through the changes to subdomains and hosting as we step through the phases of **release-trains** for v10.0.x and v10.1.x. Things that change at each stage are highlighted in yellow.

10.0.x release train is in active development

Release-trains:

- 10.0.x is in **next** phase
- 9.1.x is (**last-minor** for v9) in **latest** phase
- 8.4.x is (**last-minor** for v8) in **long-term-support** phase
- 7.2.x is (**last-minor** for v7) in **archive** phase

Branch	Firebase site	Subdomain	DNS redirects from
master	next-angular-io	next.angular.io	v10.angular.io
9.1.x	v9-angular-io	angular.io	v9.angular.io rc.angular.io
8.4.x	v8-angular-io	v8.angular.io	-
7.2.x	v7-angular-io	v7.angular.io	-

10.0.x enters feature-freeze

Release-trains:

- 10.1.x is in **next** phase
- 10.0.x is in **feature-freeze** phase
- 9.1.x is (**last-minor** for v9) in **latest** phase
- 8.4.x is (**last-minor** for v8) in **long-term-support** phase
- 7.2.x is (**last-minor** for v7) in **archive** phase

Branch	Firebase site	Subdomain	DNS redirects from
master	next-angular-io	next.angular.io	
10.0.x	rc-angular-io	rc.angular.io	v10.angular.io
9.1.x	v9-angular-io	angular.io	v9.angular.io
8.4.x	v8-angular-io	v8.angular.io	-
7.2.x	v7-angular-io	v7.angular.io	-

10.0.0-rc.0 is published

Release-trains:

- 10.1.x is in **next** phase
- 10.0.x is in **release-candidate** phase
- 9.1.x is (**last-minor** for v9) in **latest** phase
- 8.4.x is (**last-minor** for v8) in **long-term-support** phase
- 7.2.x is (**last-minor** for v7) in **archive** phase

Branch	Firebase site	Subdomain	DNS redirects from
master	next-angular-io	next.angular.io	
10.0.x	rc-angular-io	rc.angular.io	v10.angular.io
9.1.x	v9-angular-io	angular.io	v9.angular.io
8.4.x	v8-angular-io	v8.angular.io	-
7.2.x	v7-angular-io	v7.angular.io	-

10.0.0 is published

Release-trains:

- 10.1.x is in **next** phase
- 10.0.x is in **latest** phase
- 9.1.x is (**last-minor** for v9) in **long-term-support** phase
- 8.4.x is (**last-minor** for v8) in **archive** phase
- 7.2.x is (**last-minor** for v7) in **archive** phase

Branch	Firebase site	Subdomain	DNS redirects from
master	next-angular-io	next.angular.io	
10.0.x	v10-angular-io	angular.io	v10.angular.io rc.angular.io
9.1.x	v9-angular-io	v9.angular.io	-
8.4.x	v8-angular-io	v8.angular.io	-
7.2.x	v7-angular-io	v7.angular.io	-

10.1.x enters feature-freeze

Release-trains:

- 10.1.x is in **feature-freeze** phase
- 10.0.x is in **latest** phase
- 9.1.x is (**last-minor** for v9) in **long-term-support** phase
- 8.4.x is (**last-minor** for v8) in **archive** phase
- 7.2.x is (**last-minor** for v7) in **archive** phase

Branch	Firebase site	Subdomain	DNS redirects from
master	next-angular-io	next.angular.io	-
10.1.x	rc-angular-io	rc.angular.io	-
10.0.x	v10-angular-io	angular.io	v10.angular.io
9.1.x	v9-angular-io	v9.angular.io	-
8.4.x	v8-angular-io	v8.angular.io	-
7.2.x	v7-angular-io	v7.angular.io	-

10.1.0-rc.0 is published

Release-trains:

- 10.1.x is in **release-candidate** phase
- 10.0.x is in **latest** phase
- 9.1.x is (**last-minor** for v9) in **long-term-support** phase
- 8.4.x is (**last-minor** for v8) in **archive** phase
- 7.2.x is (**last-minor** for v7) in **archive** phase

Branch	Firebase site	Subdomain	DNS redirects from
master	next-angular-io	next.angular.io	
10.1.x	rc-angular-io	rc.angular.io	
10.0.x	v10-angular-io	angular.io	v10.angular.io
9.1.x	v9-angular-io	v9.angular.io	
8.4.x	v8-angular-io	v8.angular.io	
7.2.x	v7-angular-io	v7.angular.io	

10.1.0 is published

Release-trains:

- 10.1.x is in **latest** phase

- 10.0.x is in **archive** phase
- 9.1.x is (**last-minor** for v9) in **long-term-support** phase
- 8.4.x is (**last-minor** for v8) in **archive** phase
- 7.2.x is (**last-minor** for v7) in **archive** phase

Branch	Firebase site	Subdomain	DNS redirects from
master	next-angular-io	next.angular.io	
10.1.x	v10-angular-io	angular.io	v10.angular.io rc.angular.io
9.1.x	v9-angular-io	v9.angular.io	
8.4.x	v8-angular-io	v8.angular.io	
7.2.x	v7-angular-io	v7.angular.io	

Deployment Theming

In addition to selecting the sub-domain to deploy to, the life-cycle **phase** indicates what theme to use for the rendered site:

- **next** (next.angular.io) - red + a "next" banner
- **feature-freeze/release-candidate** (rc.angular.io) - orange + a release-candidate banner
- **latest** (angular.io) - blue
- **archive + long-term-support** (vN.angular.io) - grey + an archive banner

The banner is a message at the top of the docs pages that tells the reader that this is not the latest version:

> This is the **archived documentation for Angular v6.** Please visit angular.io to see documentation for the current version of Angular.

The deployment theme to use is computed by the tooling (docs-infrastructure) at deployment time.

Extraction of CLI documentation

The Angular CLI docs, built as part of angular.io, are extracted from the angular/cli-builds repository, based on a hard-coded SHA in a file in the angular/angular repository (/aio/package.json).

By storing different SHAs in each of the angular/angular branches it is possible to map the released version of CLI to the version of the docs. For example

- CLI **master** => angular **master**
- CLI **latest** => angular **latest**

- CLI **7.2.x** => angular **7.1.x**

Note that, since Angular CLI may release minor versions independently of the rest of Angular, there is no guarantee that the archive branches will have the same names.

There is a custom bot that can track changes to the CLI docs on specified branches, then create a suitable PR in the angular/angular repository that updates the SHA in an appropriate branch. Currently this bot maps branches as follows:

- Angular **master** => CLI **master**
- Angular **last-minor** with highest major => CLI **last-minor** with same major
- Angular **latest** => CLI **last-minor** with same major

Notes

Alan Agius is investigating whether we can use RenovateJS rather than the homegrown bot..

Implementation

This section describes all the tasks that need to be completed to implement these new ideas.

- Update AIO project
    - Create a new angular.io deployment theme for rc.angular.io.
    - Update aio deployment scripts to use correct branch names and push to correct sites
    - Implement vx.angular.io redirect strategy
    - Create necessary sites.
    - Setup rc.angular.io DNS
- Update CLI docs integration
    - Update CLI project to use new branches
    - Update CLI docs integration scripts
- Update github
    - Update the "target" labels
- Update caretaker merge script
    - Use new branch names when merging
- Update "builds" deployment scripts that run in CI (Angular and CLI)
    - Ensure branches are added to the builds repositories when there are new versions
- Update caretaker release scripts
    - Use the correct branch names
    - Apply the correct npm dist-tags
    - Increment the version in package.json on master branch when a new version-branch is created.
- Update public documentation (write a blog post?) about the changes.
- Update the release steps doc Update labels doc

- Rollout these changes to angular/angular-cli and angular/components repos

Next steps

- Discuss renaming **master** git branch to **next**

Related Documents

- Caretaker Instructions
- Release Instructions

Glossary

Here are brief definitions for easy reference of key terms used in this document. More detailed explanations of these terms appear in the document.

- **active:** A **release-train** is **active** if it can accept new code from PRs.
- **archive**: the phase of a **release-train** after it has stopped being **latest** or **long-term-support**.
- **development-branch**: a branch used by a group of developers to collaborate which can be the target of PRs but is not a **release-train**.
- **exceptional-minor:** a **release-train** for the unusual case where a minor version needs to be released after work has already started on the next major version.
- **feature-freeze**: the phase of a **release-train** between the **next** phase, where features are actively developed, and the **release-candidate** phase, where versions are published for external assessment.
- **hosting-site:** a site in the Firebase hosting project that will hold the distribution for a particular instance of an angular.io application. These will usually be accessible via a subdomain: e.g. next.angular.io, angular.io, v9.angular.io, etc.
- **last-minor**: the **release-train** that has the highest minor release number for a given major release.
- **latest**: the phase of a **release-train** after its first production version is published (when it leaves **release-candidate** phase) until the next **release-train** becomes **latest**. *Only one **release-train** can be **latest** at any given time.*
- **major release-train:** a **release-train** that will cause a major version bump (i.e. X.0.0).
- **minor release-train:** a **release-train** that will not modify major version (i.e. X.Y.0 - where Y is not 0).
- **master**: the git branch that receives commits for the current **release-train** while it is in its **next** phase.
- **next**: this has two different usages:
    - the phase of a **release-train** where new features (including breaking changes) are developed. *Only one **release-train** can be **next** at any given time.*

    - the dist-tag on npm that developers can use to subscribe to new releases of **release-trains** during **next**, **feature-freeze** or **release-candidate** phases.
- **phase**: a period of the life of a **release-train** where specific development and release activities are taking place.
- **pre-release:** A **release-train** that is either in **feature-freeze** or **release-candidate** phase. We have an **active pre-release** if there is a release train in **pre-release**.
- **production:** when the **release-train** is in the **latest** or **long-term-support** phases. This begins when the X.Y.0 version is released.
- **release-candidate**: the **phase** where a **release-train** is believed to be ready for production and is being published to get wider audience feedback before its first **production** version is released, moving it to the **latest** phase.
- **target**: a label on a PR that indicates a mapping between the PR and one or more branches where the PR should be merged.
- **release-train**: the life-cycle of a major/minor version of Angular (e.g. v8.0.x or v7.1.x), which encompasses the development and releases for that version as it passes through its life-cycle **phases** (e.g. **next**, **feature-freeze**, **release-candidate**, **latest**, etc).
- **version-branch**: a git branch that is used to track commits for a **release-train** once it leaves the **next** phase of its life-cycle, e.g. (8.2.x).