# High-level Process Lifecycle and Role of Pull Requests

This lesson gets us started with the high-level process lifecycle and the role of a pull request in Jenkins X.

**We'll cover the following**

- High-level process of a lifecycle of an application
- What not to do?

**Pull Requests** (or whatever their equivalents are called in your favorite Git distribution) are a norm. Most of us have adopted them as the primary way of reviewing and accepting changes that will ultimately be deployed to production. They work hand-in-hand with feature branches.

> Some of you might cringe thinking about pull requests. You might prefer trunk-based development, and work directly on master. Even if that is your preference, I still recommend you go through this chapter and do all the exercises. If processes were manual, getting pull requests merged would be a slow, hands-on, and time-consuming effort. If you use pull requests, you should use automation to speed up the process, and reduce the human effort required. That's precisely what Jenkins X is doing for us. So, put your reservations about pull requests aside, and follow along.

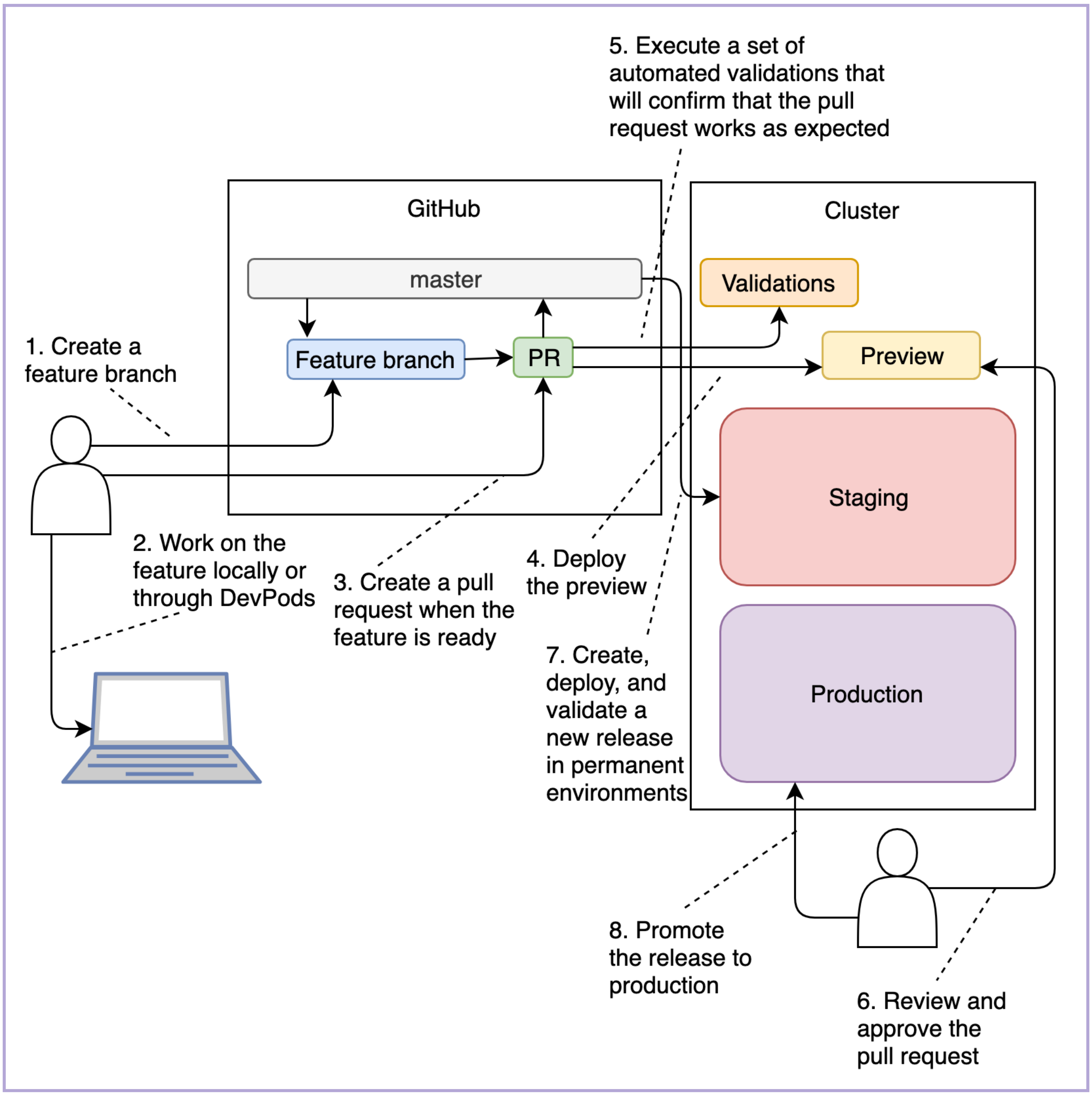
## High-level process of a lifecycle of an application

A common (and recommended) workflow is to create a new branch for each feature or change we want to release. Once we create a branch dedicated to a feature or change, we either work locally or in DevPods until we are satisfied with the outcome. From there on, we make a pull request, which should execute a set of automated steps that will deploy the preview and validate it. If all the steps are successful, we might have some manual actions like code review. Once finished, we merge the pull request and that, as you already saw, results in yet another round of automated tests that end with the deployment to one or more environments (those set to receive new releases automatically). Finally, the last

step is to promote a release to production whenever we feel we’re ready (unless the promotion to production is set to be automatic). Hopefully, the whole process from creating a branch all the way until it is deployed to permanent environments (e.g., staging, production) is measured in days or even hours.

The high-level process of a lifecycle of an application usually contains the following steps:

1. Create a feature branch
2. Work on the feature locally or through DevPods
3. Create a pull request when the feature is ready
4. Deploy the preview
5. Execute a set of automated validations that will confirm that the pull request works as expected
6. Review and approve the pull request
7. Create, deploy, and validate a new release in permanent environments (e.g., staging)
8. Promote the release to production (unless that part is automated as well)


5. Execute a set of automated validations that will confirm that the pull request works as expected
GitHub
Cluster
master
Validations
1. Create a feature branch
Feature branch
PR
Preview
Staging
Production
2. Work on the feature locally or through DevPods
3. Create a pull request when the feature is ready
4. Deploy the preview
7. Create, deploy, and validate a new release in permanent environments
8. Promote the release to production
6. Review and approve the pull request

The high-level process of a lifecycle of an application

There could be variations in the process, but at a very high level, the process works reasonably well and is widely adopted. The problem is that the process based on feature branches is in stark contrast to how we were developing applications in the past.

## What not to do? #

*A long time ago in a galaxy far, far away* we used to have long application lifecycles, cumbersome and mostly manual processes, and an infinite number of gates with silly approval mechanisms. That reflected in our branching strategies.

gates with silly approval mechanisms. That reflected in our branching strategies. We'd have project or development branches that lived for months and moving

from one environment to another usually meant merging from one branch to another. We do not live in 1999 anymore, and those practices are obsolete today.

We split projects into features. We reduced lifecycles from months to weeks to days to hours. And, more importantly, we learned that it is pointless to test one set of binaries and deploy another to production. All that resulted in the feature branches model. Each feature gets a branch, and each branch is merged back to the master. There's nothing in between. There are no staging, integration, pre-production, or other branches. The reason for this lies in the process that tells us that we should build something only once and move the same artifact through environments. With such an approach, there is no reason for the existence of all those branches. You develop in a feature branch and you merge it to the master, you build the artifacts as part of the merge process and you move them through environments until they reach production. Even that can be questioned, and many are now pushing directly to the master branch without feature or any other branches and without pull requests. We won't go that far, and I'll assume that, if you do want to push directly to master, you should be able to adapt the process we'll use. What I do NOT expect you to do is create a complicated branching schema only because you're used to it. Move forward from whatever year you live in, come to the present.

We already explored how to *work on a feature locally or through DevPods*. In this chapter, we'll cover all the other steps except for the promotion to production. We'll go through most of the lifecycle of an application and see how we can add pull requests into the GitOps and Jenkins X processes we explored so far.

---

Let's get started by creating a Kubernetes cluster (if you deleted the previous one).