CloudBees®

eBook

Let's Get Back to Coding! 5 Approaches to Solving Pipeline Bottlenecks

On an average day, the typical developer writes no more than about 100 lines of code. And that's what the more productive developers achieve: Some developers may produce as few as 5 or 10 lines of code each day.

That's not an especially high number, given that the typical phone app contains something like 50,000 lines of code. It takes many, many developer-days to write an application—let alone maintain it on an ongoing basis—if programmers can churn out only a few hundred lines of code per week.

That's why development teams must be constantly on the lookout for ways to accelerate code production. First, they need to identify bottlenecks that slow the development of software. Then they should find tools and processes that can mitigate the challenges of continuously writing and deploying applications.

To Improve Software Delivery Velocity, Improve Your CI Pipelines

To achieve that goal, continuous integration (CI) pipelines are a great place to start. Although there are many reasons code production is slowed down, some of the most common are initiating, configuring, and executing CI pipelines. The more time developers need to spend on tasks like setting up and managing CI pipelines or troubleshooting failed builds, the less time they have to do their main job: writing innovative code.

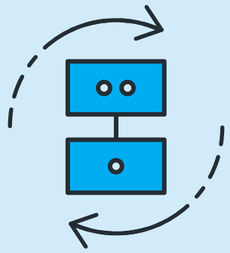
Plus, developers who are able to spend more time coding and less time fighting with CI tools tend to be happier. Most programmers chose their careers because they like coding, not because they like manually setting up software or sorting through CI logs to figure out why a build is taking longer than it should.

This eBook will help developers improve CI pipelines, starting with the most common pain points. In the following pages, you'll learn about the most common pain points that arise within CI pipelines. You'll also learn which practices and tools — including features unique to CloudBees—can mitigate CI pipeline challenges so developers can spend more time doing what they love and less time on CI administration, context-switching, and other tasks that disrupt efficient software delivery. Some of the practices you'll also learn how to optimize resource availability for builds to improve the speed of both writing and delivery.

Top Developer Pain Points Around CI Pipelines

To unlock the full potential of your CI tools and processes, developers must be empowered to focus on writing code, not bogged down with pipeline management. Here's a walkthrough of five common challenges related to this effort–and how CloudBees can help.

Although there are many reasons code production is slowed down, some of the most common are initiating, configuring, and executing CI pipelines.



Challenge 1: Slow Pipeline Initialization

CI pipelines are the foundation for modern software delivery. A well-structured CI pipeline ensures that developers can write, integrate, and build new application code quickly and efficiently.

However, setting up a streamlined CI pipeline can be a lot of work. That's especially true in cases where development teams need to initiate multiple pipelines, each of which requires a somewhat different configuration. If developers have to set up each CI pipeline from scratch and tweak it manually, they may end up spending weeks initiating pipelines before they write their first line of code.

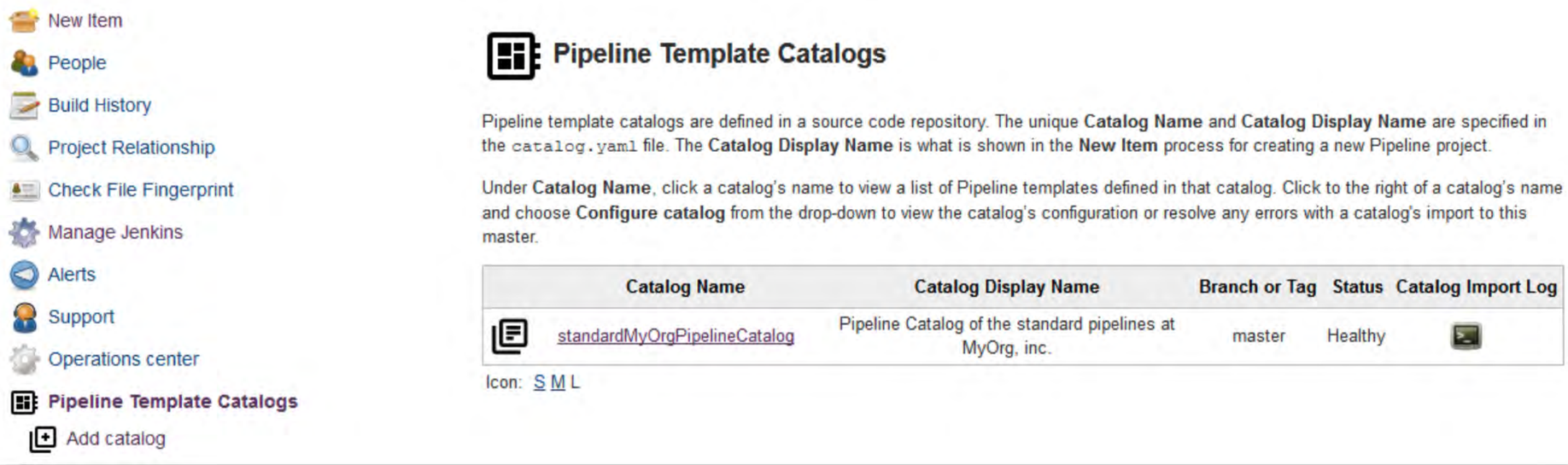
The need to ensure that pipeline configurations conform with organizational standards or regulatory requirements makes the issue only more complicated. It can be challenging for developers to know which standards each pipelines need to meet, let alone implement those standards in an efficient way.

CloudBees offers a solution to these challenges in the form of Pipeline Templates and Pipeline Template Catalogs. This feature lets CI admins and development teams define standard pipeline configurations in the form of templates. For example, here's what a Pipeline Template might look like for a Maven pipeline:

```
version: 1
type: pipeline-template
name: Maven Pipeline Template
templateType: MULTIBRANCH
description: Provides a pipeline template for maven builds.
parameters:
...
multibranch:
markerFile: cloudbees-ci.yml
branchSource:
github:
...
```

After creating a template like this, admins or developers can share it across the organization by configuring a Pipeline Template Catalog. The catalog is also a simple YAML file, such as the following:

```
version: 1
type: pipeline-template-catalog
name: customerPortalFrontendTeamCatalog
displayName: Shared Template Catalog for
Teams Working on the Customer Portal
```



By creating Pipeline Templates and sharing them across the organization using a Pipeline Template Catalog, CI admin and development teams can easily configure pipelines that conform to organizational standards, then deploy them automatically. The result is not only much less time and effort on the part of developers to initiate pipelines, but also a lot less guesswork in determining how to configure pipelines to meet domain-specific requirements.

Challenge 2: Customizing CI Configurations

While templates are a great way to streamline the initiation of pipelines across multiple projects and development teams, the pipeline configuration that works for one team may not suffice for another. For that reason, developers need a way to customize their CI configurations. They may want to add plugins, for example, or tweak pipeline configurations.

One way to do this is to give every developer full admin control over the CI pipeline. That solves the challenge of allowing developers to customize their configurations. The problem with this approach, however, is that it's a recipe for chaos and nonconformance: If individual developers can deploy any plugin or configuration change they want without oversight or controls in place, teams are likely to end up with pipelines that violate organizational or regulatory standards, and/or that are unstable.

A better solution is to leverage CloudBees CI Configuration as Code (CasC), which provides the foundation for a manageable, Git-based workflow that lets developers request CI configuration changes, then validates them using automatic tests before the changes are applied. The process works like this:

- When a developer wants to change the configuration of a managed controller (in other words, a Jenkins® instance) in CloudBees, the developer issues a pull request on Git.
- The pull request triggers a new managed controller instance, where the configuration changes will be tested automatically.
- If the tests pass, the pull request will be merged into the main branch of the targeted managed controller, so that they take effect.

This approach solves two key challenges. First, it provides a self-service, automated process through which developers can request CI configuration changes. There is no need to navigate institutional bureaucracy or track down admins to request a change.

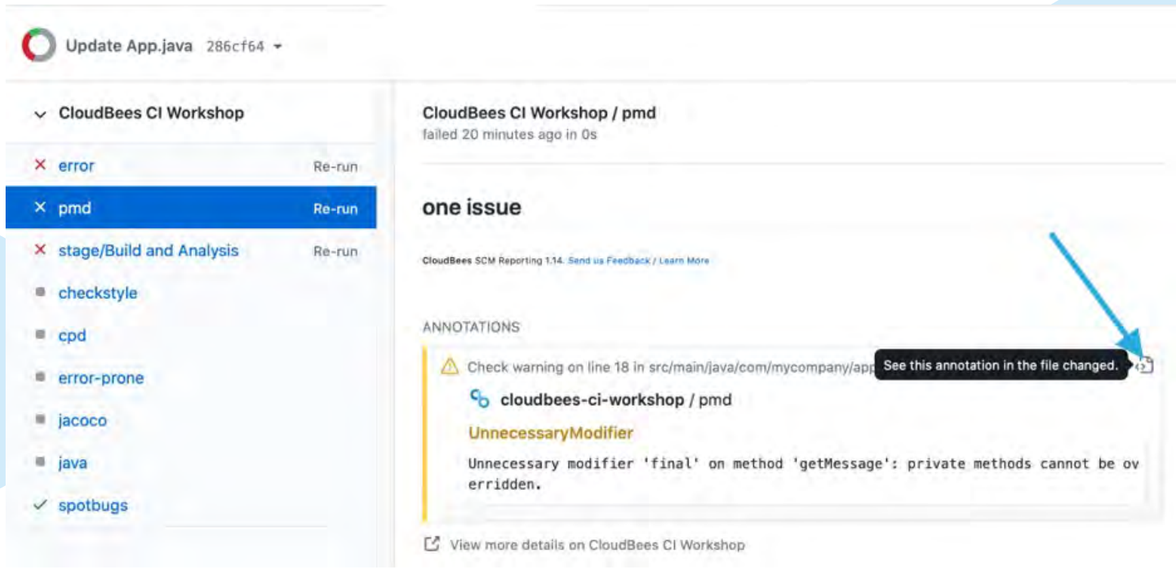
Second, the Git-based approach ensures that configuration changes are properly tested and validated before they are applied. In this way, development teams avoid the risk of introducing changes that cause pipeline instability or nonconformance.

Challenge 3: Troubleshooting Pipeline Execution Issues

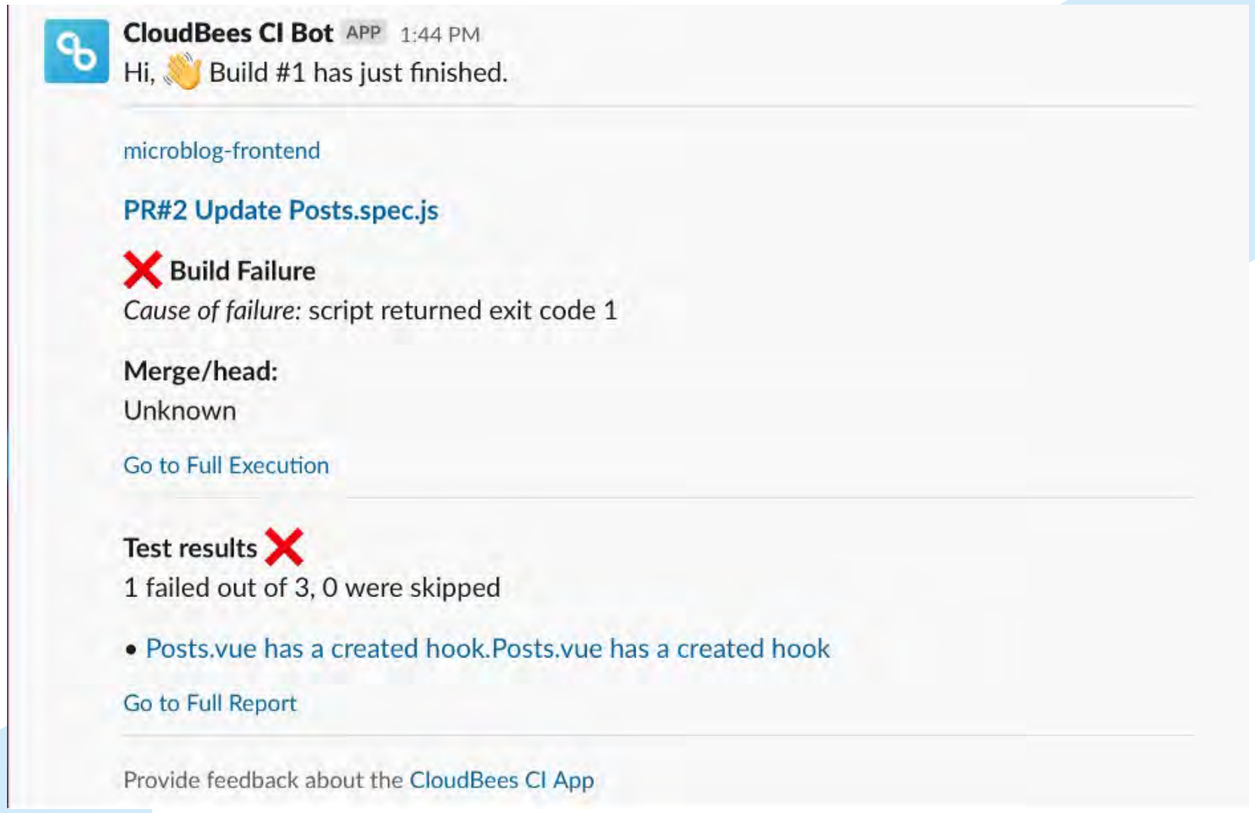
Even when pipeline configurations are thoroughly tested and validated, there's always a chance that they won't run as expected. And when something goes wrong—like when a build fails to complete or takes much longer than expected—the responsibility for figuring out why falls to developers.

In traditional CI environments, troubleshooting pipeline execution issues is not very efficient. It typically requires developers to check build logs or CI console interfaces in order to track down the source of a build problem. That translates to context-switching, which is a big distraction from coding.

In CloudBees, however, there's no need to context-switch when pipeline execution issues arise. Through the SCM Reporting feature, CloudBees automatically pushes information about build issues into GitHub or BitBucket.

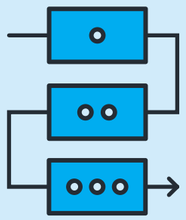


This means developers can track build statuses and identify issues without having to switch out of the coding environment that they rely on for doing their main jobs.



In addition, the CloudBees Slack Plugin makes it possible to push build information to Slack, eliminating the need for devlopers to switch contexts.

Plus, because CloudBees sends Slack messages directly to the individual developers who committed the code associated with a CI job, only those developers who need to know about build issues are notified, without distracting other developers from their coding work.



Challenge 4: Managing Pipeline Dependencies

CI pipelines don't exist in a vacuum. In many cases, one pipeline can't run until another pipeline completes.

The challenge that pipeline dependencies present in a traditional CI environment is that developers must manually keep track of the status of multiple pipelines and trigger new jobs by hand based on that status. Monitoring pipelines manually is not only tedious, but it also creates a huge distraction for developers whose time is better spent coding.

That's why CloudBees provides the Cross Team Collaboration feature, which makes it possible to coordinate pipelines by generating notification events in one pipeline, then sharing the notifications with other pipelines.

For example, imagine a situation where a pipeline generates a JAR file. The JAR file is then used by a different pipeline to complete a separate task. With Cross Team Collaboration, you could configure a notification in the first pipeline using code such as the following:

```
// Declarative //
pipeline {
agent any

stages {
stage('Example') {
steps {
echo 'new maven artifact got published'
publishEvent simpleEvent('com.example:my-jar:0.5-SNAPSHOT:jar')
}
}
}

// Script //
node {
stage("Example") {
echo 'new maven artifact got published'
publishEvent simpleEvent('com.example:my-jar:0.5-SNAPSHOT:jar')
}
}
```

Then, you can configure a trigger for the second pipeline based on the event notification that you configured above:

```
pipeline {
agent any

triggers {
eventTrigger simpleMatch('com.example:my-jar:0.5-SNAPSHOT:jar')
}

stages {
stage('Maven Example') {
steps {
echo 'a new maven artifact triggered this Pipeline'
}
}
}

// Script //
properties([pipelineTriggers([eventTrigger simpleMatch('com.example:my-jar:0.5-SNAPSHOT:jar')])])

node {
stage('Example') {
echo 'a new maven artifact triggered this Pipeline'
}
}
```

The result is a fully automated solution for connecting the two pipelines. Instead of relying on the team managing the first pipeline to notify the second pipeline's team when the JAR file is ready, each pipeline can be kept in sync and executed automatically.

Challenge 5: Inefficient Use of Build Infrastructure

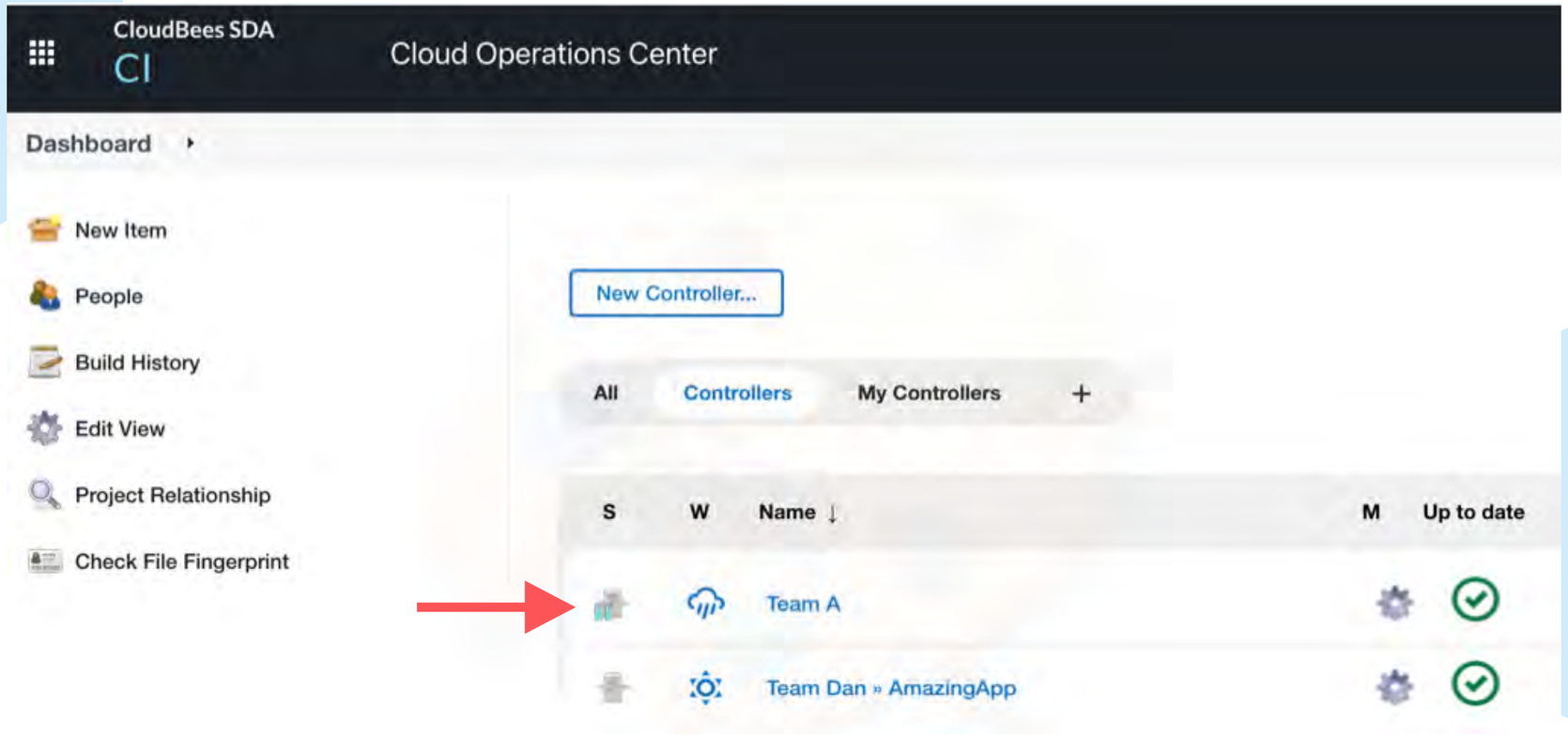
In a perfect world, each CI instance in your organization would be active only when necessary. It would shut down or hibernate when there are no builds to run, in order to make its infrastructure resources available to other pipelines.

In the real world, however, achieving this optimization of resource consumption is very difficult. Developers routinely leave CI instances active even when they shouldn't be, and underlying infrastructure platforms (like Kubernetes) offer no automated way of knowing when an instance should be shut down. As a result, fewer resources are available to jobs that actually need them.

Plus, the business may be left paying for infrastructure it doesn't actually need.

Development teams can solve this challenge with help from CloudBees CI Hibernating Managed Controllers. This feature automatically shuts down or hibernates CI instances after a predefined period of inactivity.

To set up the feature, you simply enable it in the Helm chart that you use to configure the Kubernetes cluster that hosts your managed controllers. You can then configure automatic hibernation, and track the status of hibernated managed controllers through the CloudBees interface.



By shutting down inactive managed controllers, developers make the resources that were tied up by those controllers available for other controllers that are actually active. The result is faster builds, thanks to more memory and CPU for each build.

In addition, when inactive managed controllers shut down or hibernate, Kubernetes can automatically scale down its total node count. That reduces infrastructure costs for the business.

Improve Developer Efficiency to Enable Innovation and Value Creation

CI software should help developers work faster and more efficiently. But when developers have to spend lots of time manually setting up, modifying, and troubleshooting CI pipelines, these tools can become more trouble than they're worth. They become a source of distraction and frustration for developers who would rather be writing code.

By leveraging CloudBees CI features like Pipeline Templates, Configuration as Code, Contextual Feedback, Cross Team Collaboration, and Hibernating Managed Controllers, development teams can avoid these pitfalls. They'll benefit from an experience that turns CI software from a liability into an engine of developer productivity. The ultimate result is more value delivered to customers in less time, not to mention greater developer satisfaction, since efficient CI means that developers can focus on what they love and are good at—writing code—instead of the toil of managing inefficient CI environments.

Contact a CloudBees CI expert today to learn how to bring efficiency to your development teams.

CloudBees, Inc., 4 North 2nd Street, Suite 1270 San Jose, CA 95113 United States
www.cloudbees.com • info@cloudbees.com