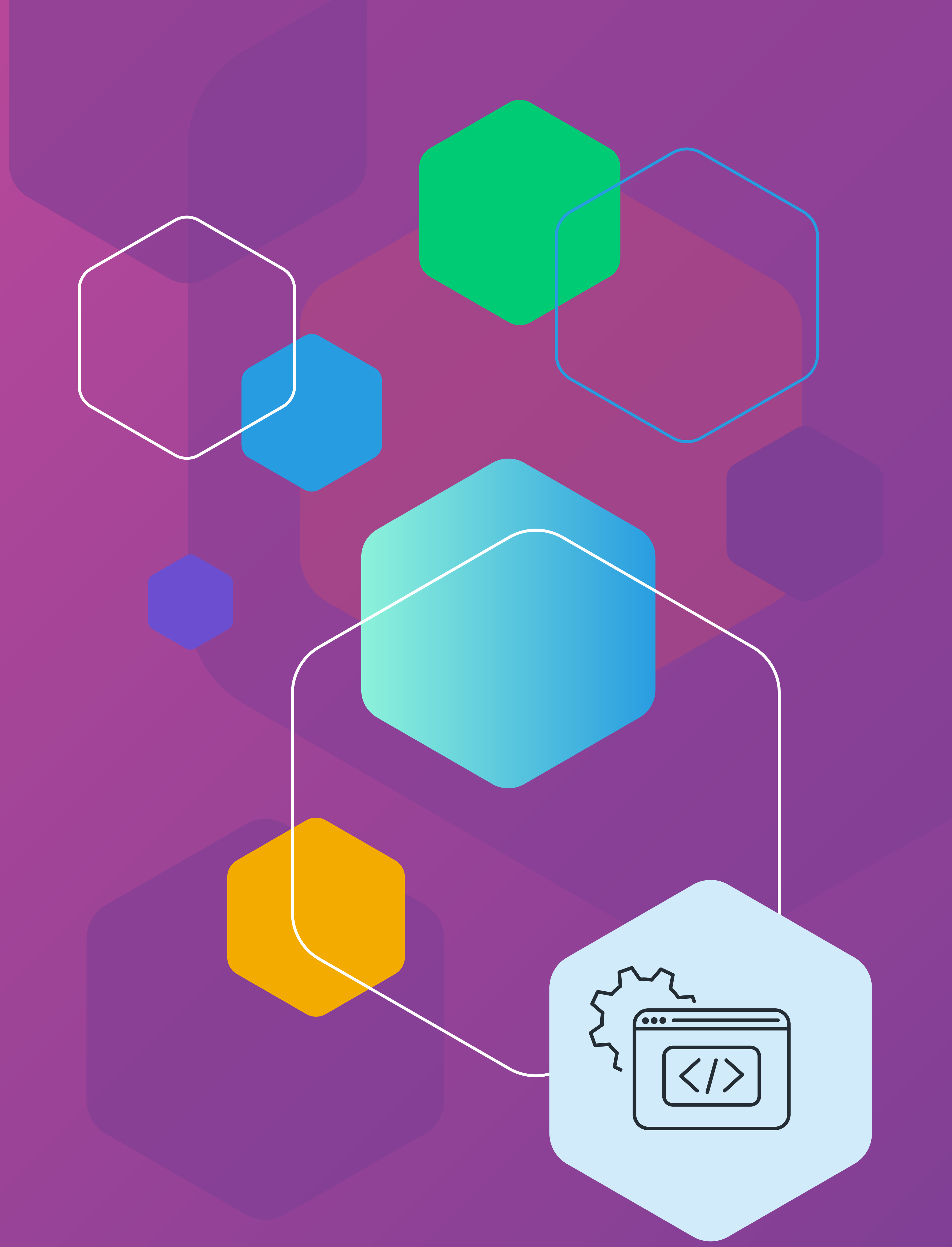
CloudBees®

eBook

Configuration as Code: A guide to the approach and benefits



The push for "X as code" has moved many aspects of the software lifecycle to standardized text files.

This trend has so far included defining infrastructure, environments, tests, documentation, and much more as code. Continuous integration and continuous delivery (CI/CD) tools are crucial to running automated processes; they take many of these other "X as code" files and both trigger and configure tasks based on their contents.

However, CI tools are complex in themselves. If a team is responsible for configuring them manually or maintaining long-running CI servers, they are likely to encounter challenges as demands on their usage scale. Switching the configuration of CI/CD to code alongside other "X as code" efforts helps create consistent, reproducible, and reliable processes and components across an entire application stack, giving teams the ability to scale when and as much as needed.

This practice, which is aptly named Configuration as Code, or CasC, is tool-agnostic and clarifies the management of Jenkins® instances by capturing their configurations in human-readable, declarative configuration files. Managers can then maintain these files as first-class revision-controlled artifacts that they can easily apply back to controllers if a configuration fails, saving everyone time.

Other key benefits of CasC for Jenkins users include:

- Prevention of configuration drift
- Easily auditable changes
- Automatically reproducible Jenkins instances

All of this means your teams spend less time creating, maintaining, and debugging integration infrastructure and more time on application code that adds value to your business.

Read on to learn more about the CasC approach and the benefits of scale, governance, and autonomy it can bring to development teams.

Considerations for starting with CasC

Switching to CasC involves rethinking your current CI workflow —from pipelines to plugins to security and everything in between—as a series of requirements. Although the implementation details of using CasC vary depending on the CI tools, infrastructure, and processes you use, there are some common questions to consider.

What authentication and security information does the configuration need? These might be details to ensure the requester is permitted to run CI processes, details for the provider that runs the CI, as well as details from other third-party services needed for access—for example, version control.

What about the system resources a CI process needs to run tasks, such as the amount of memory, CPU, and disk space? Do you want to run tasks on bare metal servers, virtual machines, or Kubernetes clusters?

Using CasC means you can break out of using one centralized CI server and, instead, use short-running processes. This is possibly the biggest impact of switching to CasC, as you can stop thinking of one multipurpose service and create discrete, task-oriented processes. As you think through how to represent your current setups and workflows as CasC, consider what you're missing and have needed in the past. With CasC, you can represent all the potential CI setups you need to reach business goals.

For a full walkthrough of getting started with CasC, see our documentation.

5 key software delivery benefits of CasC

1. Standardization

Configuration via UI enforces a form of standardization. Yet with dozens of features, plugins, and screens contributing to hundreds of options, it's all too easy for a user to become overwhelmed and select an incorrect or different option accidentally. An incorrect configuration option can range from an inconvenience to a security or large-scale customer-facing problem.

Using a UI to configure a CI can also result in configuration drift. Even if you switch to CasC, if others continue to use the UI, then the configuration differs from the desired state, and it becomes difficult to know which value is correct.

Defining that configuration as code instead results in something you can rely on, as its values will always create a Jenkins instance configured the way you expect. Although understanding the depths and details of configuration files has a learning curve, there are standard formats and approaches in the industry that are commonly used.

This standardization means that a developer who has previously followed CasC practices (or something similar) can understand most of the principles of a configuration file with a new project or company. Even if a new project or company uses different CI tools or providers, it's likely the developer can understand what tasks the configuration is supposed to run and how they can implement it.

"Using a UI to configure a CI can also result in configuration drift.

2. Reliability

A standardized and consistent format for configuration files helps reduce the risk of configuration errors to a large degree. For further reassurance, another advantage of using text formats is the ability to check and test their contents. Certain tools, such as linters, check code and configuration files and make recommendations on errors, warnings, and best practices. You can create test CI processes to ensure your configuration changes are correct and that everything will run as expected before potentially wasting team members' time on deploying production code.

Using a standardized format means that developers can write configuration files with the tools they are happiest using, and take advantage of the variety of customization and functionality they offer. These may include some of the aforementioned linting and testing tools, meaning that errors and best practices are easier to find and fix as early as possible.

3. Reusability and flexibility

Reusability means making configuration as applicable to as many situations and uses as possible so that it isn't useful to only one specific function or use case.

Using environment variables and templating is the first step to making CasC reusable. These are variables set when triggering the CI process, and they are passed by commands themselves or detected from the environment the command runs in. An environment variable can determine the code to clone into a CI instance, the branch to checkout, the directory to clone into, and many other factors. Variables and templates are one way to allow different developers with different access rights to use the same CasC and trigger similar but subtly different CI processes.

A CI process exists to run tasks, so any CasC definition needs to tell it where to find those tasks. Again, through the use of variables, those tasks can change, allowing for the triggering of different tasks based on the environment.

Beyond internal developer flexibility, you can leverage some of the same principles to configure CI processes for different releases—for example, releasing different software versions to different customers or regions. This is useful in larger-scale testing instances, but also if you need to release different versions for different cultural or regulatory purposes.

You can typically extend CasC with plugins that add functionality and configuration to CI capabilities. These could be additional version control providers, notification services, or authentication providers.

4. Versioning for collaboration and traceability

Versioning CasC makes it easier for team members to collaborate and configure services for the many automated tasks of software development. Teams and individual members can contribute to the parts of the configuration they're responsible for with minimal impact on others. As members propose changes, they can see them in the context of existing configuration and discuss suggestions and ideas around those changes. This discussion and visibility are not possible when configuring with a UI or with a long-running CI server that few people have access to. After discussing the changes, you can test the proposed changes' functionality and impact by using branches in the repository.

You can keep configuration in its own repository, or you can keep configuration changes located in the same code repositories as application code and in line with product and code changes. Keeping it in the same location means you can tie everything to a specific version number or release of application code.

Versioning offers an audit trail to see who contributed what and when to the configuration. This audit trail is useful for identifying issues or changes to performance, but also for regulatory reasons. This audit trail helps recognize configuration drift and identify the discussion and decision behind a particular change.

If a team member or customer notices a problem, versioning means you can roll back changes to previous states and reconfigure systems to a previously working state in minutes.

Most version control providers integrate with hundreds of other tools. This means you can connect configuration to other services and processes, triggering other manual or automated steps during collaboration on the CasC, or when creating an infrastructure based on it—for example, checking for certain standards in the configuration files or triggering notification services.

5. Team and infrastructure scalability

Reproducible CI configuration managed from a distributed but centrally managed source (version control) makes it easier to onboard new employees and reassign them to different projects. Many hours are lost with new hires asking questions about where to find the resources they need in order to work. If instead, you can direct them to one source of knowledge with a handful of commands, they can start work much faster.

Using CasC removes the need for centrally managed and running CI services that only a few team members have access to. If that fails or has problems, team members can no longer run CI jobs and are blocked from working. A similar situation can occur if a particular CI task consumes compute resources. If you only have a handful of dedicated instances, they block other team members' progress. But, if you spin them up dynamically as requested, you reduce this risk. The reproducible nature of CasC means that you can work around other unforeseen hits to productivity. If you run CI processes on one provider and they experience downtime, you can switch to another. If you need to run CI processes in another region for data privacy or regulatory reasons, you can. If one provider changes its methods for charging and another is more cost-effective, switching is a matter of changing the configuration code. If those requirements change, you can switch again, each time with minimal impact on team members.

CasC gives development teams the power to spin up CI processes when they need them and in the configurations required for specific projects or tasks. CasC also reduces the need for specialized DevOps functions to keep long-running CI servers functioning efficiently, and "looking after them" CI becomes a series of ephemeral processes.

As you add new teams and products, it's much easier to create the new configuration you need with CasC. Copy an existing and similar file, make changes, and you'll have the required configuration relatively quickly. In some cases, if you make smart use of the environment and other variables when creating the configuration, you may not even need to duplicate the existing configuration.

With CasC, not only can you help mitigate configuration drift, but differences in configuration values become a feature you can explore and experiment with.

Improving developer productivity and delivery velocity

In the current market, it's hard to recruit and keep quality developers with production-level experience. One area where you can improve the experience for developers you already employ is with internal productivity. Using practices such as CasC keeps developers engaged as they can focus on the work they want to do—and the work that will bring the most value and innovation to your customers—instead of maintaining integration tools or negotiating with DevOps teams to resolve CI issues.

Configuration as code helps reduce the amount of time spent on configuring, maintaining, and debugging CI infrastructure, and it enables DevOps teams to scale alongside the business without worrying about the manual management of hundreds or even thousands of Jenkins controllers. At the same time, it helps developers test new features, experiment with code changes freely, and leverage flexible plugins without burdening managers with complicated maintenance.

Simply put, CasC puts the power to run CI processes into the hands of the developers who rely on it the most—in turn, giving businesses the automation capacity and flexibility they need to deliver software when they and their customers need it.

Learn how CloudBees CI can supercharge your growing software delivery engine.

CloudBees®

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