



EBOOK

How Jenkins® Admins Can Provide Scalable CI for Software Delivery Teams

It's One Thing to Set up a Continuous Integration (CI) Pipeline for Your Developers.

It's quite another to create CI pipelines that can scale seamlessly and are easy to manage.

That's why simply deploying CI software isn't enough to maximize the value that this approach brings to the business. Admins who manage CI pipelines must also take steps to ensure that the processes for deploying, scaling and managing those pipelines are as efficient as possible.

This eBook provides guidance on achieving that goal. By walking through the most common pain points that Jenkins admins face when setting up CI pipelines, then highlighting CloudBees features that mitigate those challenges, we'll explain how businesses can make the most of CI by ensuring pipeline scalability and manageability.

As you'll learn, by investing in pipeline management tools and practices that make pipeline administration as efficient as possible, everyone benefits—the admins who manage pipelines, the developers who depend on them, and the business as a whole. This way, CI becomes a vehicle of value creation rather than a cause or delays of inconsistency in software delivery operations.



Admins who manage CI pipelines must also take steps to ensure that the processes for deploying, scaling, and managing those pipelines are as efficient as possible.



The Benefits of Scalable and Manageable CI

Before diving into the top CI administration pain points and solutions, let's discuss why having efficient CI is so critical for the success of all stakeholders.

When your pipelines can scale and are easy to manage, your developers benefit in several ways:



Avoiding needless bureaucracy

The less time it takes for shared services admins to set up a new pipeline or modify an existing one, the easier it is for developers to request changes. In turn, developers get to spend more time coding, and less time navigating enterprise bureaucracy in order to get the pipelines they need.



Ensuring scalable teams

When pipelines are easy to deploy, manage and scale, development teams can add or remove users quickly. That's important because individual developers often join or leave projects, and the team doesn't want to delay development operations while it waits for admins to update CI software to reflect team changes.

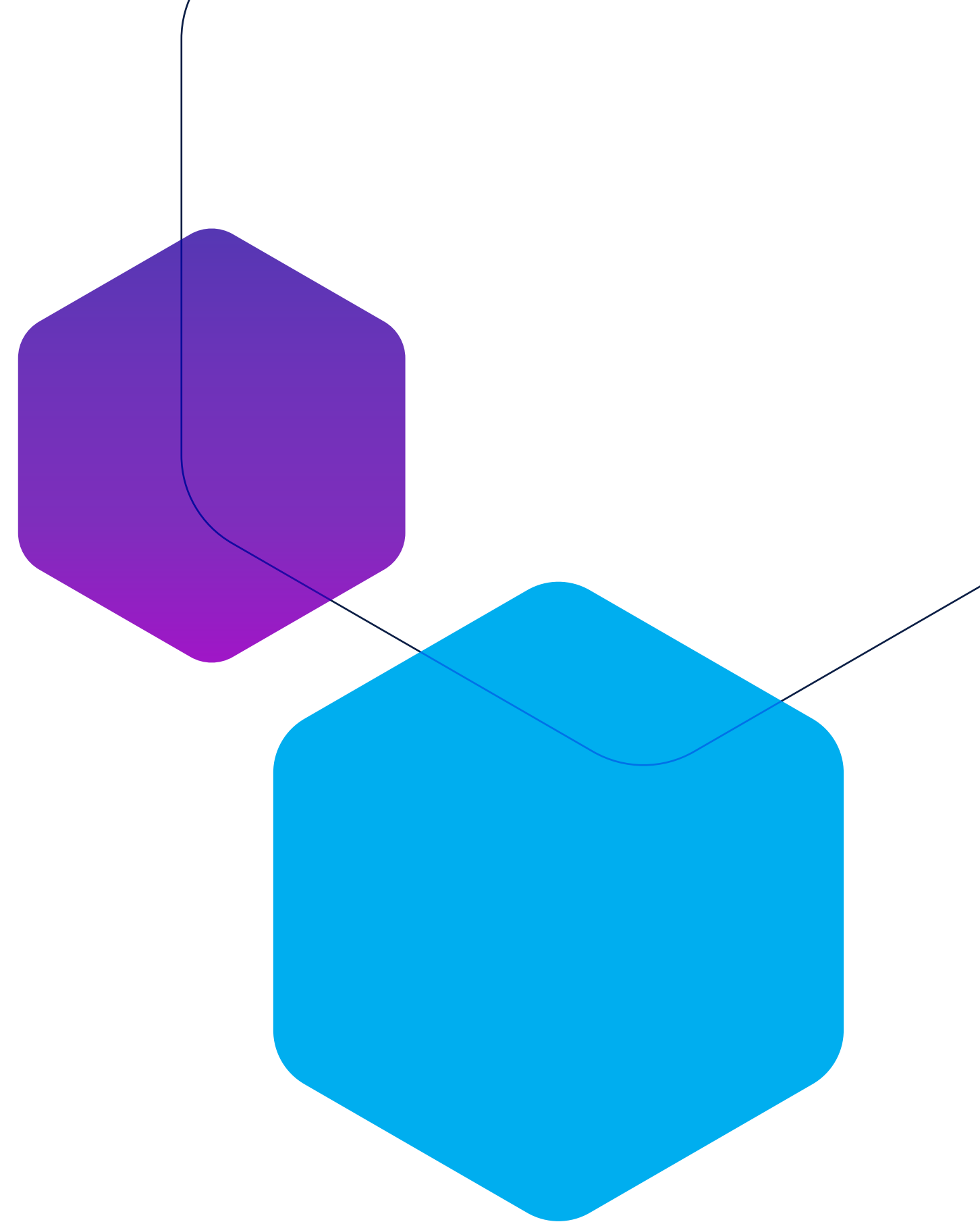


Tightening CI security

Pipelines that can be scaled and managed efficiently are consistent and predictable pipelines. By extension, they are secure. The less manual work that is required to set up or update a pipeline, the lower the risk that secrets will be exposed to the wrong stakeholders, or that admins will accidentally configure the wrong access controls.

In short, efficient pipeline management and scalability translates to a smoother, more flexible, and more secure software delivery experience for developers. In a similar fashion, pipeline admins waste less time applying manual changes or performing redundant tasks. They also become more productive, and their jobs are more enjoyable because they can automate tedious pipeline management processes.

Finally, the business as a whole benefits when both developers and shared services admins are as productive and satisfied as they can be. With well-managed pipelines, developers and admins can focus on creating value and innovation for the business, rather than troubleshooting pipeline issues or navigating inefficient pipeline admin processes.

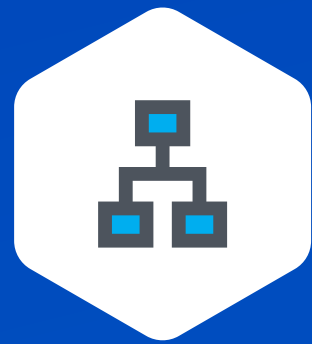


The Top Pain Points in Jenkins CI Administration

To unlock the full potential of CI, admins responsible for pipeline deployment and management must avoid the most common challenges that get in the way of efficient CI. Here's a walkthrough of those challenges, and how CloudBees can help.



Jenkins



Challenge 1: Slow Pipeline Setup



Setting up pipelines can be one of the biggest drains on efficiency. If admins have to create each pipeline that developers need from scratch, they'll spend enormous time configuring pipelines. They're also likely to end up with pipelines that are inconsistent and unpredictable, because the pipelines won't be configured in a uniform way based on centralized standards.

CloudBees addresses this challenge with Pipeline Templates and Pipeline Template Catalogs. Using these features, admins can define pipelines using code, then make those pipelines available to whichever developers or development teams need to use them.

The screenshot shows the Jenkins Pipeline Template Catalogs page. On the left is a navigation sidebar with items like 'New Item', 'People', 'Build History', 'Project Relationship', 'Check File Fingerprint', 'Manage Jenkins', 'Alerts', 'Support', 'Operations center', 'Pipeline Template Catalogs', and 'Add catalog'. The main content area is titled 'Pipeline Template Catalogs' and contains explanatory text and a table.

Pipeline template catalogs are defined in a source code repository. The unique **Catalog Name** and **Catalog Display Name** are specified in the `catalog.yaml` file. The **Catalog Display Name** is what is shown in the **New Item** process for creating a new Pipeline project.

Under **Catalog Name**, click a catalog's name to view a list of Pipeline templates defined in that catalog. Click to the right of a catalog's name and choose **Configure catalog** from the drop-down to view the catalog's configuration or resolve any errors with a catalog's import to this master.

Catalog Name	Catalog Display Name	Branch or Tag	Status	Catalog Import Log
 standardMyOrgPipelineCatalog	Pipeline Catalog of the standard pipelines at MyOrg, inc.	master	Healthy	

Icon: [S](#) [M](#) [L](#)

The templates are defined using simple YAML files. For instance, here's a Pipeline Template for a basic Java app:

```
version: 1
type: pipeline-template
name: Java Maven App Pipeline Job
description: Simple Java App With Maven
parameters:
- name: emailRecipient
  type: string
  displayName: Email address to be used for sending build notifications
- name: github_repo
  type: string
  displayName: GitHub HTTPS repo endpoint
- name: github_creds
  type: credentials
  displayName: GitHub Credentials ID
  defaultValue: github-creds-example
```

After creating a Pipeline Template, admins can create a separate catalog.yaml to share the template. For example, here's what the catalog file might look like for a template shared by front end development teams working on a customer portal:

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

Pipeline Templates and Pipeline Template Catalogs mean that admins don't have to reinvent the wheel each time a development team needs a new pipeline. They can instead reuse pipelines they've already created by making the templates easily available through the catalog. The result is much less time spent by admins in creating pipelines, and much less effort on the part of developers to find and deploy the pipelines they need.



Challenge 2: Maintaining Instance Consistency

Typically, admins want to give each development team or project its own CI instance. A common challenge that admins face, however, is the risk that each instance is configured differently. If the instances are set up separately and manually, there's a good chance that their configurations won't be consistent. Even if you do manage to set up all of your instances in a uniform way initially, changes that your developers request over time may lead to configuration drift, resulting in inconsistent instances.

With CloudBees, you can mitigate this challenge by taking advantage of Configuration as Code (CasC). CasC captures the configuration of each of your Jenkins instances (known in CloudBees as managed controllers) and defines it as human-readable configuration files.

In this way, CasC solves two key challenges related to achieving instance consistency:

1. Apply new instances using code. This makes it fast and easy to reproduce identical instances across your enterprise environment.
2. Instance configurations can be managed as code, which means you can version it, test it, and validate it. Code-based configuration makes it much easier to identify configuration inconsistencies, as well as track how your instance configurations change over time.

The core of CasC configuration data is the jenkins.yaml file, which defines the Jenkins configuration of each instance. For example, the file might include a credentials configuration like the following:

```
credentials:
  system:
  - credentials:
  - string:
  description: "GitHub PAT from JCasC - secret text"
  id: "cbdays-github-token-secret"
  scope: GLOBAL
  secret: "{AQAAABAAAAAwhY0iqxnrlWCCLvk+2TLChLxlT}"
```

In addition, CasC includes several other YAML files. Beyond bundle.yaml, which defines the CasC configuration files themselves, you can optionally create files that define other instance configuration data, such as rbac.yaml (which defines root-level RBAC configuration for individual managed controllers) and plugins.yaml (which defines which plugins to install or update on managed controllers).

Example CasC configuration bundle with items
Example bundle.yaml file

```
id: "bundle-1"
version: "1"
apiVersion: "1"
description: "My CloudBees Configuration as Code (CasC) bundle"
allowCapExceptions: true
availabilityPattern: "folder1/*.*"
parent: "bundle-global"
jcas:
  - "jenkins.yaml"
jcasMergeStrategy: "errorOnConflict"
plugins:
  - "plugins.yaml"
catalog:
  - "plugins-catalog.yaml"
items:
  - "items.yaml"
rbac:
  - "rbac.yaml"
variables:
  - "variables.yaml"
```



Challenge 3: Maintaining Pipeline Governance Standards

When a CI workflow fails to run properly, it threatens to disrupt not only the operations of whichever team depended on it, but also other teams, whose pipelines may be negatively impacted by the buggy pipeline. For this reason, pipeline admins must take steps to ensure that they can identify and address situations where pipeline execution does not conform to organizational standards. They must also detect pipelines that violate regulatory requirements.

One way to do this is to monitor each pipeline as it runs. But that, of course, would require tremendous effort. It's not practical if you have dozens or more pipelines across your organization.

That's why CloudBees provides Pipeline Policies. Pipeline Policies let admins define runtime validations that are performed automatically on both scripted and declarative pipelines. Using YAML, admins can impose conditions that govern pipelines at runtime, and automatically take action when pipelines violate the rules.



For example, consider the following policy:

```
cloudbees-pipeline-policies:
config:
policies:
- action: "warning"
customMessage: "Please check your pipeline
as it should use less than 20m in our infra"
description: "This policy helps to reduce
the use of agent for a long time"
filter: "acme/dev-team/*"
name: "Avoid build running for more than 20m"
rules:
- "pausedActionInAgentRule"
- entirePipelineTimeoutRule:
maxTime: 20
- agentTimeoutRule:
maxTime: 20
- pausedActionTimeoutRule:
maxTime: 20
```

This policy will trigger a warning message in the event that the entire pipeline or a pipeline agent times out for more than 20 minutes, or if there is a paused action for more than 20 minutes. The policy could alternatively be configured with action: "fail", which would stop the build completely.

Using a policy like this, admins can prevent execution delays from one pipeline's instability affecting other pipelines. They can also automatically warn developers about pipelines that are not executing as required. And they can do all of this without having to monitor each pipeline manually, or generate feedback by hand when something goes wrong during pipeline execution.



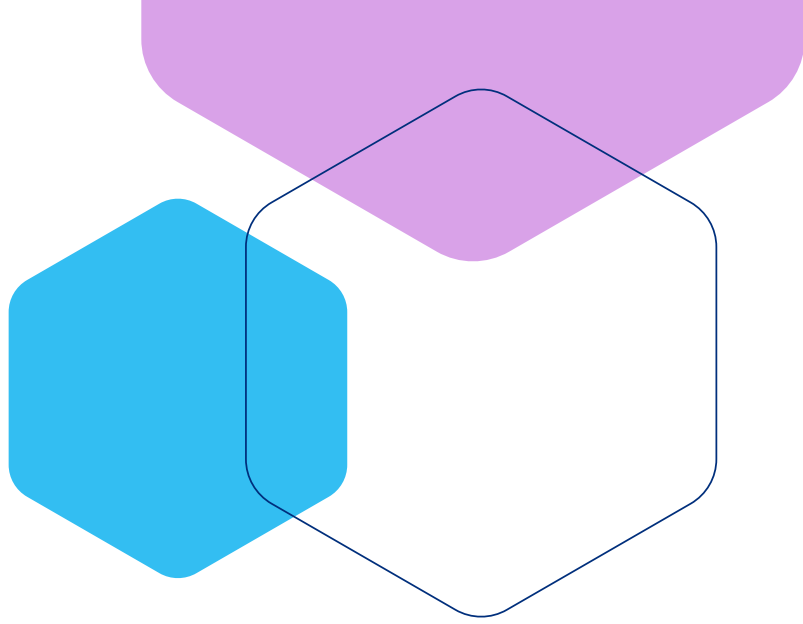
Challenge 4: Enforcing Access Controls

Admins typically don't want every developer in their organization to have complete access to and control over every resource in a CI pipeline. Instead, they want to configure access rights granularly, such that a developer can view build jobs but not trigger them, for example. They may also want an individual developer to have one set of permissions for one CI instance, but a different set for a different instance.

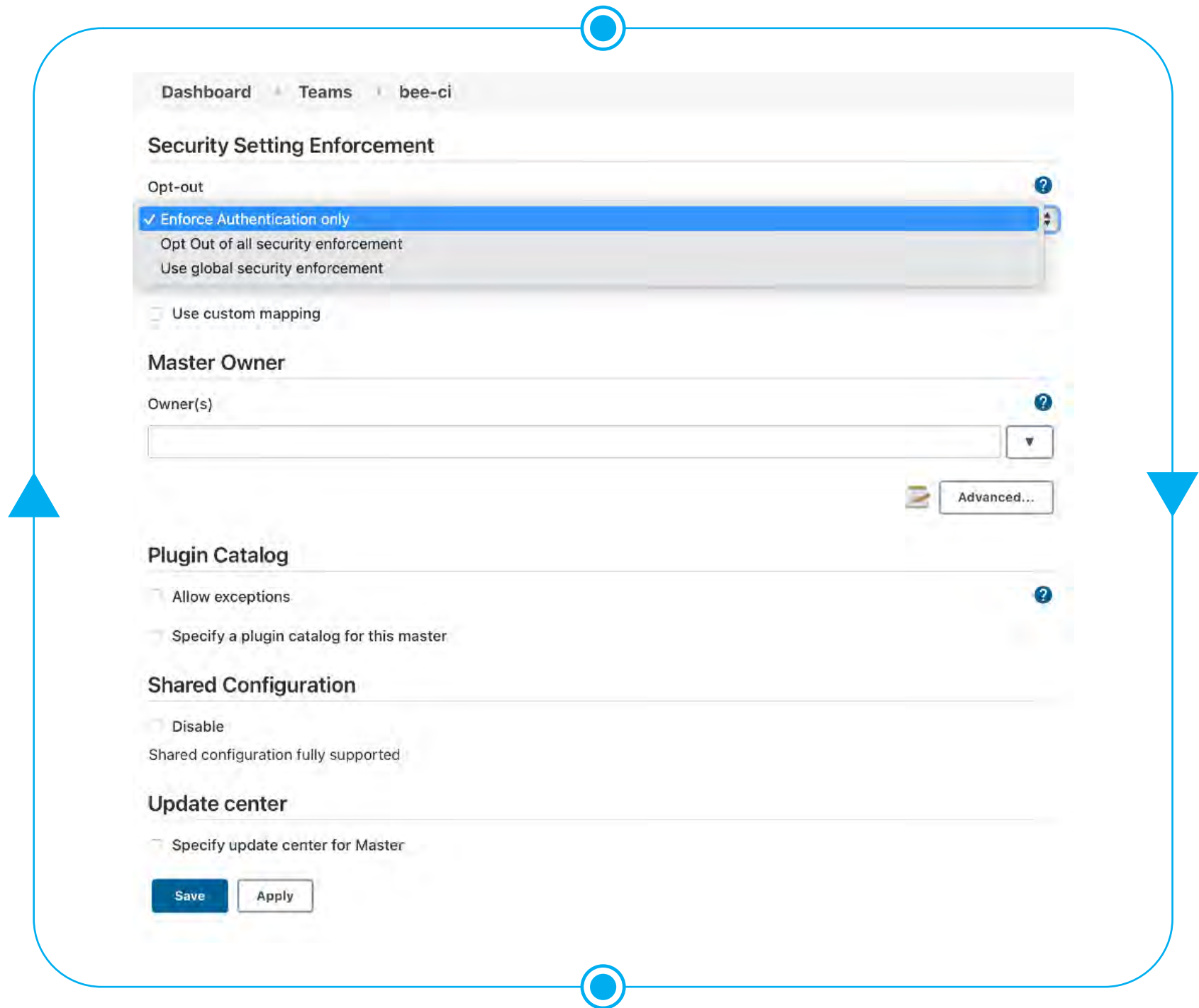
CloudBees makes granular access control rights easy to enforce using a role-based access control (RBAC) model. As noted above, the CasC bundles that CloudBees can generate for CI instances can optionally include a `rbac.yaml` file, which defines access rights in a granular fashion. For example - the RBAC yaml file could look like this:

```
removeStrategy:  
rbac: "SYNC"  
roles:  
- name: administer  
permissions:  
- hudson.model.Hudson.Administer  
- name: developer  
permissions:  
- hudson.model.Hudson.Read  
- hudson.model.Item.Read  
- hudson.model.Item.Create  
- hudson.model.Item.Configure  
filterable: "true"  
- name: browser  
permissions:  
- hudson.model.Hudson.Read  
- hudson.model.Item.Read  
filterable: "true"  
- name: authenticated  
filterable: "true"  
permissions:  
- hudson.model.Hudson.Read  
groups:  
- name: Administrators
```

```
roles:  
- name: administer  
grantedAt: current  
members:  
users:  
- admin  
external_groups:  
- ${external_admin_group}  
- name: Developers  
roles:  
- name: developer  
members:  
users:  
- developer  
internal_groups:  
- "some-other-group"  
external_groups:  
- "ldap-cb-developers"  
- name: Browsers  
roles:  
- name: browser  
members:  
users:  
- read
```



When applied as part of a CasC bundle, a rbac.yaml file like this allows admins to enforce a consistent set of granular access control permissions across all of the CI instances they manage. Note, too, that permissions can also be managed through the CloudBees Operations Center, for admins who prefer a graphical user interface (GUI):



Enhance Software Delivery Operations and Execute on Scalable CI

Creating pipelines that are easy to manage and scale is hard—if you rely on manual configuration processes and workflows, that is.

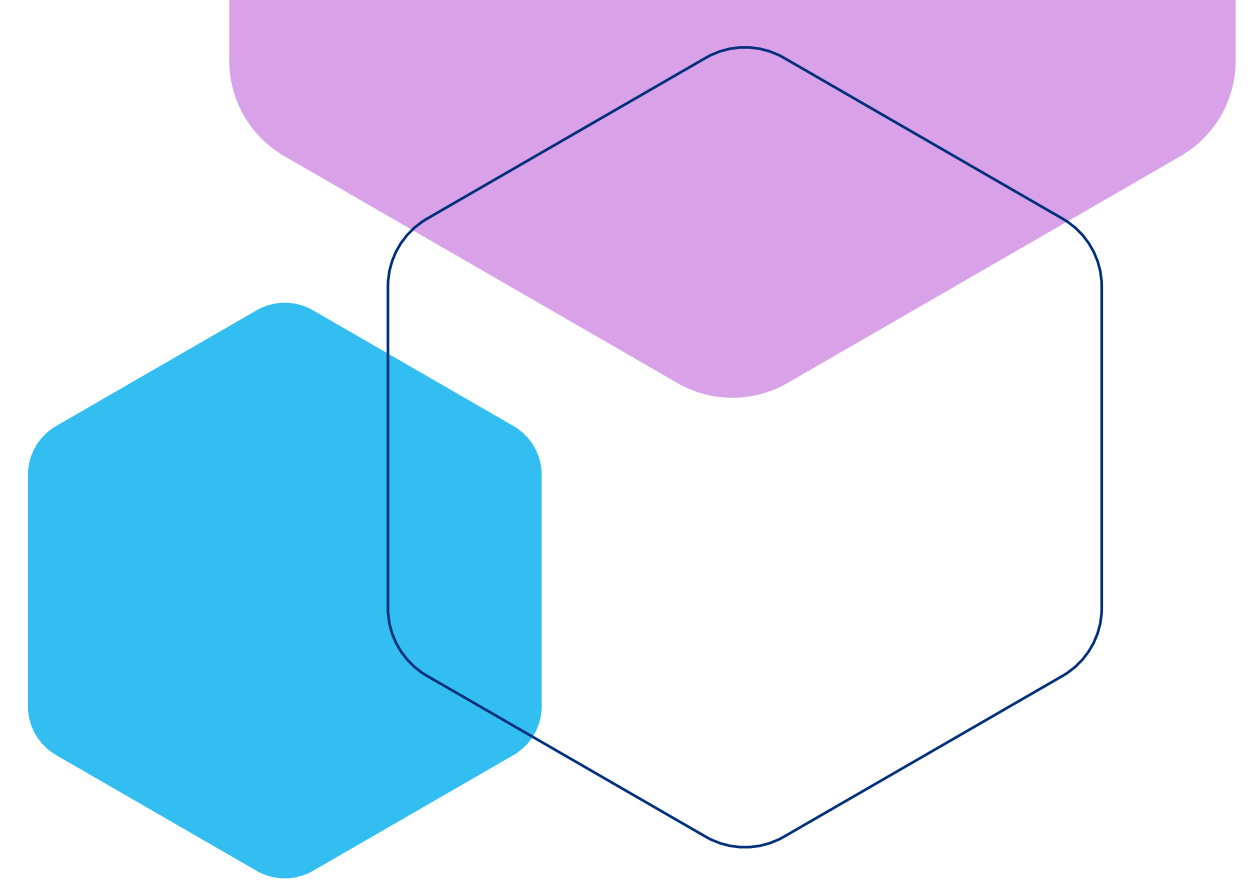
But with CloudBees, you can automate and streamline pipeline management tasks that would otherwise require tremendous time and effort on the part of admins. In particular, you can:

- Use **Pipeline Templates and Pipeline Template Catalogs** to reuse and share template configurations across your organization.
- Maintain CI instance consistency and reproducibility using **Configuration as Code**.
- Automatically enforce organizational and regulatory standards during pipeline execution through **Pipeline Policies**.
- **Configure granular access controls** using the `rbac.yaml` file in your CasC bundle or through the CloudBees Operations Center.



When you take advantage of tools like these, you gain the ability to create, modify and share pipelines across your organization quickly and easily. At the same time, your developers get fast and easy access to the pipeline configurations they need, while also benefitting from more consistent pipeline operations and a lower risk of disruptions to pipeline execution. And the business as a whole gets shared services admins and developers who are happier and more productive—a key to delivering innovation and value to customers.

Contact a CloudBees CI expert today to learn how to bring scale and manageability to your software delivery teams.





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

2023 CloudBees, Inc., CloudBees and the Infinity logo are registered trademarks of CloudBees, Inc. in the United States and may be registered in other countries. Other products or brand names may be trademarks or registered trademarks of CloudBees, Inc. or their respective holders.

Jenkins® is a registered trademark of LF Charities Inc.