

Continuous Governance: The Guardrails for Continuous Everything



Introduction

The continuous economy represents one of the most inspiring changes for how companies approach software development and delivery. The rousing shift towards continuous integration (CI) and continuous delivery (CD) essentially spawned the continuous everything movement. Companies can now create, test, deliver and deploy applications frequently and predictably by automating processes using open source tools like [Jenkins](#) and [Jenkins X](#).

But the shift towards the continuous everything (CE) model inherently creates a perception of risk and question about who governs the process. Understanding how governance plays into the continuous economy is one of the most important pieces that is not yet well understood.

This eBook briefly describes some of the foundational pillars on which continuous everything stands. Continuous governance (CG) is one of the numerous pillars of continuous everything, and one that encounters an unfair share of confusion. This eBook explains how governance applies to the continuous everything paradigm to form the basis of continuous governance.

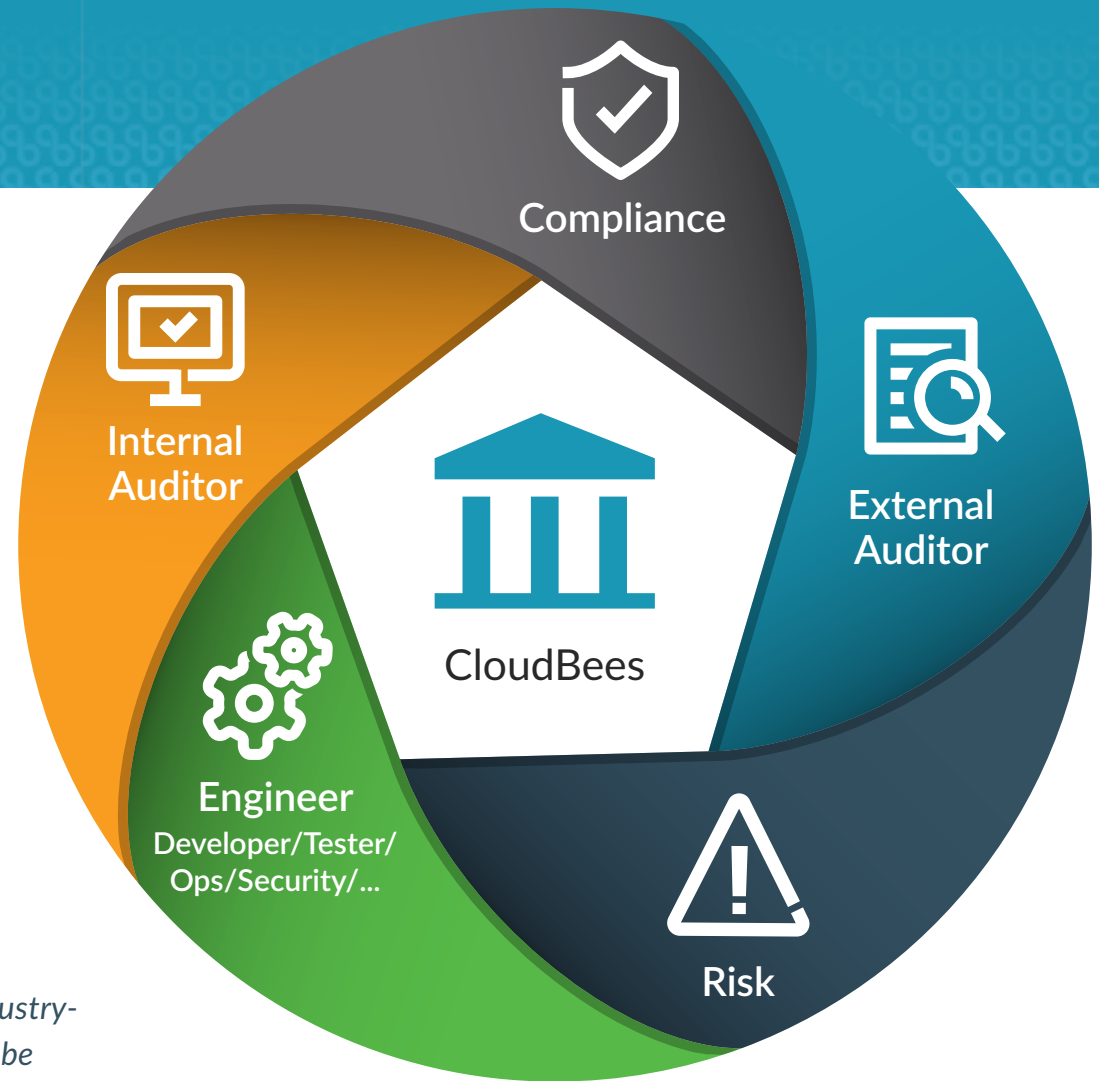
What is governance?

To internalize continuous governance, it is imperative to first understand what governance is. According to Wikipedia, governance is all of the processes of governing undertaken by a government, a market or a network over a social system through norms of an organized society.

But who governs governance? The governing body has leverage over others and hence ownership could get political. If every organization comes up with their own governance standards, the industry will continue to remain deeply fragmented on this controversial topic. CloudBees has identified the major players as risk, compliance, engineering organizations and auditors (both internal to organizations and external), and is leading the charge to bring the industry under a unified governance umbrella.



This eBook illustrates continuous governance protocols that can be productized into an industry-standard governance engine, which can then be integrated with continuous everything pipelines to ensure continuity of governed processes. However, first, let us dive into the continuous paradigm and then hone in on continuous governance to understand how it fits.

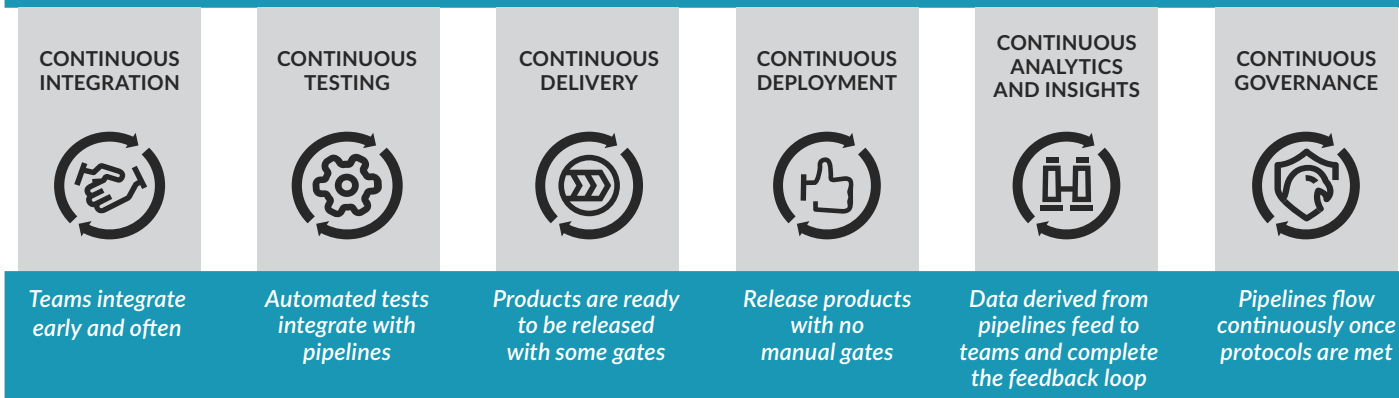


Continuous Everything

Continuous everything or the continuous paradigm encompasses many pillars as seen in the illustration below.



Continuous Everything



Source: CloudBees, Inc.

We will define each pillar, provide a concrete example for further clarification and identify governance use cases. We will then dive into the the tenets of continuous governance.



Continuous integration is a process where teams integrate early and often to detect issues earlier in the cycle.



Continuous integration

Continuous integration is a process where teams integrate early and often to detect issues earlier in the cycle.

As the code for features, bug fixes, tests, configuration, data, and infrastructure are committed, pipelines can run static analysis security testing (SAST) with the help of integrated scanners. These can flag security vulnerabilities in minutes that could potentially weaken (and sometimes cripple) the organization's security posture. In this case, continuous integration reduces the organization's susceptibility to cybersecurity threats.

The cost of a defect is:

- » Small in development/test environment
- » Medium in staging
- » Large in production

Hence, fixing defects earlier in the pipeline lowers the total project cost.

Governance use cases

These are governance use cases that need to be included in the continuous integration framework for teams to integrate seamlessly.

The following two are particularly important with the culture of distributed workforces gaining momentum.

- » Everyone should not be able to read all parts of the source code repository.
- » Everyone should not be able to commit code to all parts of the source code repository.

Also, the following use cases should be addressed.

- » Secrets should not be committed to version control in clear text.
- » Versioned artifacts and/or images should be stored in version control and/or an artifact repository.
- » For source code and artifacts to be safe in the cloud, cloud providers should provide evidence of periodic audits.



Continuous testing is a process where automated tests integrate with pipelines and form gates to determine whether code should be promoted from one pipeline stage to the next.

Continuous testing

Continuous testing is a process where automated tests integrate with pipelines and form gates to determine whether code should be promoted from one pipeline stage to the next.

Last-minute feature additions and bug fixes are normal and when under pressure from tight deadlines, teams may add them without updating the test suite. These can inadvertently introduce fatal problems at the last minute, like a performance issue. Pipelines, with integrated performance tests, will abort and send notifications to potential culprits in the team. However, without continuous testing, this can cause customer dissatisfaction and downtime.

Governance use cases

These are governance use cases that need to be included in the continuous testing framework for teams to test seamlessly.

- » Teams should not be allowed to bypass test execution in pipelines. Zero test failures is not always a good thing since zero test failures can result from zero tests executed.

- » Test code, scripts, configuration and data should be in version control and colocated with the system under test.
- » Test execution results should not be editable.
- » Production data sets should be masked/scrubbed before use in development/test and staging environments to protect consumer privacy.
- » A common concern is: How do we know that tests don't have bugs? Tests should not be written to validate tests; otherwise, we need to write tests for those tests too. This recursive pattern of testing the tests does not scale or even make sense. Govern, but don't be paranoid.
- » Auditors require "test evidence" or "audit evidence." The data model should be clearly defined, as in:
 - Test id
 - Test description
 - Test duration
 - Test status (pass | fail)



This "test evidence" data model should be standardized across internal and external auditors.

- » Pipelines should abort even if one test fails. An incomplete test amounts to a failed test.
- » Tests, and test suites, should be idempotent and independent, so that they can execute in parallel. This significantly reduces test cycle time, feature lead time and, eventually, time to market.



Continuous delivery is an automated approach where products are ready to be released from a source control repository to production with one or more manual gates.



Continuous delivery

Continuous delivery is an automated approach where products are ready to be released from a source control repository to production with one or more manual gates.

Once an application is validated by automated tests, it is promoted from the development/test environment to staging. Before deployment to production, pipelines can file change requests automatically and poll for a human to verify and approve. Once the human approves, pipelines continue from where they left off. This is an example of a manual gate. Pipelines generate an audit trail by recording the approval details (time of approval, for instance), along with the approver's details (identification, for instance).

Note that the manual gate can be in other parts of the pipeline too. This is just one example prevalent in the industry.

Governance use cases

These are governance use cases that need to be included in the continuous delivery framework for teams to deliver seamlessly.

- » How many environments are too many? CloudBees recommends three – one development/test environment per pull request, one staging environment and one production environment. You can extend the guiding principles mentioned in this eBook to more, although be warned that more environments could lead to more maintenance overhead. “Death by a thousand environments” isn’t worth the trouble.
- » Secrets should be managed securely by the pipeline. The definition of secrets should not be constrained to just passwords but should extend to keys, certificates and all sensitive information.
- » Key rotation policies should be designed and implemented to reduce the possibility of breaches.
- » Pipelines should be configured to touch only those production assets that they need to.
- » If a pipeline is configured to seek approval, only a selective set of people should be enabled to provide approval, thus honoring segregation of duties.
- » Pipelines are designed to integrate with SaaS/ IaaS/PaaS providers, and if those vendors get breached, pipelines could be affected. Audit/ compliance protocols should be standardized for SaaS vendors.
- » Artifacts that have been deployed to development/test environments should be retained for X days, staging for Y days and production for Z days. Typically, $Z > Y > X$.
- » CAB (Change Approval Board) or similar councils that perform manual change approvals should be discouraged, since it nullifies the continuous bandwagon.



Continuous deployment is an automated approach to release products from version control to production with no human intervention, which means there are no manual gates.



Continuous deployment

Continuous deployment is an automated approach to release products from version control to production with no human intervention, which means there are no manual gates.

Once an application is validated by automated tests, it is automatically promoted from the development/test environment to staging to production with no human intervention. Change requests are filed and approved by the pipeline automatically. You may question the legitimacy of pipelines providing approvals. However, this automated audit trail radically enhances traceability.

Some organizations (and not just large enterprises) do not allow continuous deployment based on their internal risk, governance and compliance constraints, and instead practice continuous delivery with manual gates. However, be aware that some auditors have not invested in continuous everything deeply enough and this causes them to discourage this practice, mostly due to lack of education and awareness.

Governance use cases

These are governance use cases that need to be included in the continuous deployment framework for teams to deploy seamlessly.

- » Removing the manual gate does not violate segregation of duties (SoD). Techniques like GitOps, where environments are Git repos and automated promotion of code and artifacts happens via pull requests and pipelines, help automate SoD.
- » Your customers may not want to see code go live every time there is a commit and the pipeline runs. In this case, we need to find a middle road to enable upgrades when it's safe and at the same time not dwell too long on older versions of the product.

- » We need to appreciate the nuances in industries involving firmware, embedded systems, hardware and IoT (internet of things). Here are a few cases to consider:
 - Medical devices may not be able to withstand new software upgrades in the middle of a procedure, especially ones that pose a risk to the patient's life.
 - Electric cars may not be able to accept every kind of over the air (OTA) upgrade, especially while driving.
 - Mobile users upgrade apps only by choice and so teams may only be able to continuously deploy to the app stores.



Continuous analytics and insights means that nuggets of information are derived from pipelines and fed back to the teams in order to complete a feedback loop.



Continuous analytics and insights

Continuous analytics and insights means that nuggets of information are derived from pipelines and fed back to the teams in order to complete a feedback loop.

In this case, facts rule. Opinions, however interesting, can be irrelevant.

Pipelines generate hordes of transactional data that can be mined into nuggets of information as part of pipeline analytics. These nuggets feed into teams in a continuous feedback loop, such that they constantly unlearn and learn from fresh data.

Wherever we are in our continuous everything journey, continuous improvement is at the heart of it. Pipeline analytics and insights fuel continuous improvement and the organic growth of teams. The trends observed in the data position organizations for long-term success in a scientific and data-driven manner.

We can analyze pipeline failures and observe that 50% of the failures are due to violation of performance benchmarks. This could mean that performance tests and data are buggy, the product is indeed sluggish, or even that the benchmarks themselves are inaccurate.

Either way, we know what work to prioritize for the following sprint.

Or, we can look at repeat offenders, who are instrumental in over 20% of pipeline failures, and help them develop new skills and learn new tools. We can also partner with teams who haven't deployed to production in the last seven days to understand what's causing their throughput to drop.

Governance use cases

These are governance use cases that need to be included in the continuous analytics and insights framework for teams to analyze seamlessly.

- » Pipeline logs should not display consumer confidential data, either personally identifiable information (PII), financial information or health-related data.
- » Certain team members should have access to selective portions of pipeline logs.
- » Some people should have access to pipeline data, in case it is streamed to a persistent store for eventual mining.

- » Key performance indicators (KPIs) should be defined for the organization's success, and should not skew towards any one department's vested interests.

For example, the metric "number of tests executed in a sprint" does not reflect the effectiveness of tests. In fact, unless tests are executed in parallel, they inflate the total pipeline execution time. Some organizations have used this as a success metric in the past and need to be educated.

Similarly, the metric "number of releases per sprint" reflects how fast we can move bits from point A to point B, but does not reflect the value delivered to the customer.

- » KPIs should support both business and engineering goals. Technical craftsmanship is great, but we need to solve business problems at the same time.



Continuous governance is a special flavor of governance that is pertinent to the continuous paradigm, or in other words, continuous everything.



Continuous governance

Continuous governance is a special flavor of governance that is pertinent to the continuous paradigm, or in other words, continuous everything. It is the sixth pillar in the continuous everything model.

On the other hand, the general term of governance applies as much to organizations as it does to countries and governments. Governance is the process that ensures practices which conform to an organization's policies, whether driven by compliance, operational efficiency or other business objectives. It determines:

- » Interactions between individuals, teams and organizations.
- » Decisions and the decision-making process itself.
- » Roles and responsibilities.

Governance is a wide umbrella and encompasses lots of domains.

In the next section, we will expand on the main tenets of continuous governance to understand how it fits into the continuous everything world.

The Main Tenets of Continuous Governance

Pipelines are incarnations of the continuous everything paradigm and are products in their own right. They release high-quality and secure products frequently and predictably to customers. Continuous governance defines principles that can be designed into automated controls or software gates that promote code from one stage of the pipeline to the next.

Let's study each tenet in more detail.

Register your pipelines

Pipelines can sprout from anywhere in the organization. However, to attain legitimacy, all pipelines should be registered with the following metadata:

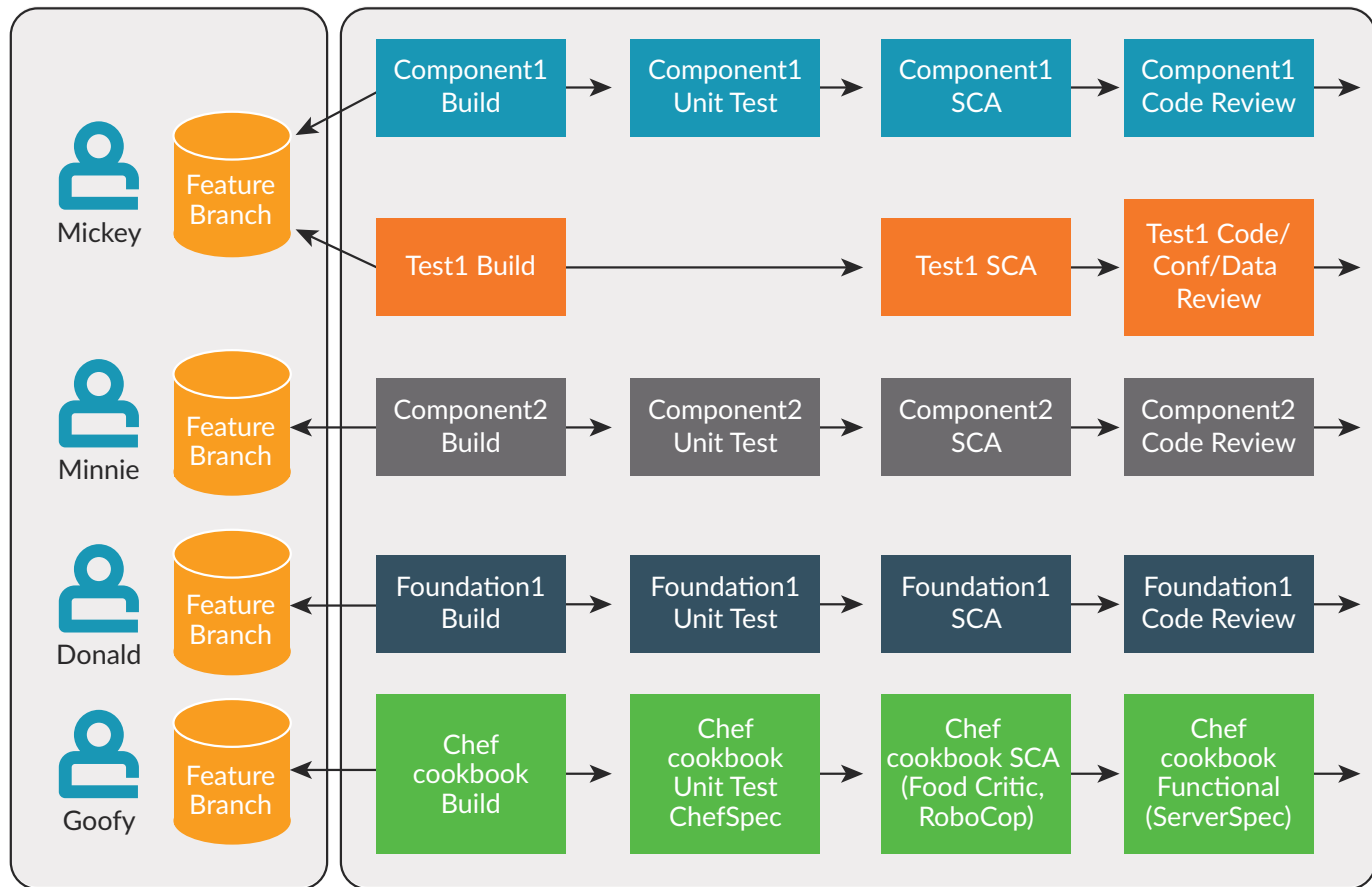
- » **Product that flows through the pipeline.** Pipelines are associated with the product that flows through the pipeline. Artifact(s) that constitute the product(s) being built, tested and deployed should be registered alongside the pipeline.
- » **Team(s) that use the pipeline.** Pipelines are associated with teams of people who build and maintain the product that flows through the pipeline.
- » **Owner who is responsible for the pipeline.** Pipelines have shared ownership since different people work on different pieces that constitute the pipeline. For example, some engineers could have built the pipeline, others could have integrated tests with the pipeline and *aaS third-party vendors could have provided the infrastructure for the pipeline. Even though there appears to be multiple owners, it pays to have a single point of contact for accountability. Ownership could be rotated, as long as the person has the necessary expertise to understand the nuances.

Version control

For the continuous paradigm to succeed, teams should integrate code, scripts, data and configuration in a central place to ensure repeatable behavior. This embodies principles such as configuration as code, infrastructure as code and in this particular context, pipeline as code.

Version control is the only source of truth for:

- » Source code
- » Tests
- » Configuration
- » Data
- » Infrastructure
 - Pipelines
 - Tools
 - Network
 - Cloud or data center resources



Source: CloudBees, Inc.

In the figure above, teams write code for components, tests, infrastructure and configuration and commit to version control. Pipelines then integrate with version control and run different validations before promoting the code to the next stage.

For example, Mickey writes features and tests for the first component, Minnie does the same for the second component, Donald writes foundational pieces like pipelines and tools, whereas Goofy builds a Chef pipeline for Chef cookbooks and recipes. Everyone commits every day to version control, taking special care not to break the pipeline(s) before they leave for Disney World for the weekend.

Some teams successfully use version control for documentation as well. Additionally, GitOps is the latest trend to manage environments, applications and application versions, where environments like staging and production are Git repos and code gets promoted through automated pipelines. Long story short, version control improves traceability and auditability required in order for the continuous everything paradigm to succeed.



Integrate all tests with pipelines

Tests are co-located with the system under test in version control. Tests are versioned artifacts, just like the system under test. Test execution results are automatically generated and should not be editable. They are archived in a format that can be submitted to auditors as test evidence or audit evidence.

Also, teams are not allowed to bypass tests during pipeline execution, since that enables pipelines to shoot defective artifacts into production, only faster. Pipelines are meant to bless us with responsible speed and not suicidal speed.

“Death by a thousand test types” is a syndrome of teams who over-think the problem. The following types of validations ensure quality and security of not just pipelines and associated infrastructure but also products flowing through the pipeline.

- » **Unit test.** These tests do not interact with the database or the network and help during code refactoring. They are closest to the source code and execute fast. Code coverage is a way to measure how much code is covered with these tests. For example, we can measure how many classes, methods and functions these tests cover. A common malpractice is to mandate a certain percentage of code coverage without understanding the implications. Also, a common oversight is to forget unit tests for security.

- » **Static code analysis.** Static code analysis checks for defects without executing code, and is inexpensive and fast. There are two main purposes of static code analysis.

a) Static analysis security testing (SAST).

DevSecOps advocates that a product be designed with security in mind, rather than sending a finished product for evaluation. Static analyzers can detect security vulnerabilities and these are a few areas where SAST should be applied.

AREA	DESCRIPTION
Open source software (OSS) libraries, plugins and dependencies	These can contain vulnerabilities that become part of your artifact. Although you are not the author, the onus lies on you to scan and fix problems that could otherwise weaken your organization's security posture.
Owned and operated software	This is code that you write and maintain.
Containers	Containers are the new normal, and securing containers is key to successfully running your applications securely inside containers.

- b) **Coding violations.** Static code analyzers ensure coding best practices are honored by team members in an automated fashion rather than only through manual code reviews.



» **Functional.** This is the largest bucket of tests, depending on the complexity of your product. Functional tests validate customer use cases involving but not limited to the following:

TYPE	DESCRIPTION
Positive scenarios	These are "happy path" use cases that are regularly experienced by customers.
Negative scenarios	These are corner cases that do not occur every day but cause havoc when they do.
Accessibility	This enables employees with additional accessibility needs to use the product.
I18N (internationalization)	Based on business requirements, the product should be ready for both domestic and international markets.
L10N (localization)	The product should account for regional constraints.
Data quality	The integrity of the data generated by the product should be unquestionable.

» **Integration.** During integration testing, various parts of the system communicate with each other over the wire. Integration tests primarily validate the network along with product interfaces, and do not hone in on functionality.

» **Performance.** Performance benchmarks are established with product owners so that customer expectations are met. These benchmarks are used to pass or fail performance tests which in turn helps the pipeline to proceed or abort based on whether the benchmark was attained or not.

Under normal conditions, traffic is expected to hit the established benchmarks with a variance of 5-10%. Under extreme conditions, traffic may swing wildly due to unusual circumstances. These validations are often termed load or stress tests, and can be simulated programmatically.

» **Security.** Unlike SAST, which happens pre-deployment, dynamic analysis security testing (DAST) is executed after deploying the product. The product is expected to be functional and executing in containers or on servers at this time. DAST explores for vulnerabilities the same way an attacker would in real life. DAST is an incarnation of DevSecOps - just like SAST - and ensures we address security vulnerabilities while building the product instead of sending a finished product for evaluation.

Version and retain artifacts

Artifacts are generated only through the pipeline. Manually generated artifacts can have unpredictable consequences and should not be enabled for downstream consumption. An artifact retention policy details retention periods for artifacts deployed in development/test, in staging and in production environments.

- » **Versioning.** Each time the pipeline runs, the version of the artifact is updated. The data model for artifact versions could be {pipeline stage}.{major version}.{minor version}. The pipeline, by default, increments the {minor version} every time an artifact is built or rebuilt. You could use timestamp as the minor version because in the continuous world, any versioned artifact is eligible to be a release candidate. Versions should not include static strings like “snapshot” and “release_candidate.”

Updating the {major version} happens only when major functionality or a breaking change is introduced in the product flowing through the pipeline. This requires human intervention in the artifact metadata in source control. {Pipeline stage} reflects how far the artifact has reached in its lifecycle. For example, if you have three milestones – development/test, staging, and production, you could have a simple enumeration like [1, 2, 3]. Every time an artifact gets promoted from a lower environment to a higher environment, {pipeline stage} is updated from 1 (development/test) to 2 (staging) to 3 (production).

- » **Retention.** Sometimes, production issues can happen long after deployments. Versioned artifacts in development/test are retained for X months and versioned artifacts in staging are retained for Y months. When a versioned artifact reaches production, and the customer sees it, the organization retains that version for Z months. Typically, $Z > Y > X$. A retention policy is necessary for audits as well.

Do not expose consumer data

Under no circumstances should personally identifiable information (PII), financial and/or health-related data appear unmasked in pipeline logs.

- » **PII.** Production data (or even snippets of it) containing PII cannot be used in lower environments like development/test or staging for testing unless the data is masked or scrubbed. The masking process is also known as scrubbing. Examples of PII are government identifications like social security numbers and birth date.
- » **Financial.** Similar constraints apply to financial data. Employees are at risk of becoming insiders if they come into contact with financial data. Examples of financial data are credit card and bank account information.
- » **Health.** Rigid constraints apply to health-related information as per Health Insurance Portability and Accountability Act (HIPAA) guidelines.



Manage secrets

With hackers continuously attacking systems, keeping sensitive data hidden is paramount to avoid cybersecurity breaches. Passwords, certificates, API keys and other sensitive data should not appear in clear text in pipeline logs or in source control. This data should only be accessible by privileged roles set by the administrator.

Keys should be rotated (meaning, periodically changed) to better defend against potential breaches and compromises.

Automate segregation of duties

Segregation of duties (SoD) implies that no one person or group has control to release software from version control to production. Essentially, SoD means we have checks and balances that prevent any one person or group from becoming too powerful.

The guiding principles of segregation of duties date back to the U.S. Constitution where the total power of the U.S. government was distributed into three branches – judicial, executive and legislative. This division enables checks and balances and prevents any one branch from becoming too powerful.

The three branches of the U.S. Government



Source: CloudBees, Inc.



Source: CloudBees, Inc.

However, blind applications of these SoD principles to continuous everything can be detrimental. In the continuous world, we want to steer clear from hand offs, where person/group 1 finishes some work and hands off to person/group 2 for sign off. For example, we should not encourage developers to write the feature, and then hand off to a tester. Similarly, we should not encourage testers to approve and throw it over the wall to operations, or reject and pass it back to developers.

In the figure, notice the long chain of hand offs happening between various silos of an engineering organization before the product goes live and the organization stands to make money. In this example, check in to go live is the time it takes for a commit to reach the customer. It is a subset of feature lead time, which in turn, is a subset of time to market.

Having said that, let's explore methods by which we can make modernized applications of SoD to the continuous world.

- » **GitOps.** Pull requests let you tell others about changes you have pushed to a source control repository. Once a pull request is sent, interested parties can review the set of changes, discuss potential modifications and even push follow-up commits if necessary.





Code review is a systematic examination of source code for features, tests, data, configuration, infrastructure and related functions. It is intended to find logic mistakes and violations of standards. Code reviews can happen alongside pull requests or even otherwise to prevent one author from having total control.

GitOps is the latest technology where environments like staging and production are Git repos. Promotion of artifacts happens via pull requests and through automated pipelines. GitOps is a modernized application of SoD without manual hand off or sign off. **Kube CD** is an offering from CloudBees that wraps around Jenkins X and implements GitOps. Kube CD enables teams to develop and deliver cloud native apps on Kubernetes in a streamlined fashion.

- » **RBAC.** Role-Based Access Control (RBAC) enables certain actions to be allowed for certain roles. Roles should not be muddled with organizational hierarchy or titles. Roles are defined based on expertise and experience. Well-defined roles prevent errors in pipelines and protect teams from overstepping lines. **CloudBees Core** offers flexible and governed continuous delivery that has a slick implementation of RBAC.

For example, role A might be able to update secrets, certificates and keys and role B might be able to update expected test outcomes. Also, in the spirit of RBAC, headless users should be single-purpose. Headless users are nothing but automated agents created by admins and authorized to perform work on behalf of named accounts. Also known as service accounts, headless users form the backbone of the continuous paradigm to enable automation and reduce manual intervention.

However, there are some negative side effects of headless users. As we know, with great power comes great responsibility. Since headless users can perform delicate actions the same way as humans can, we need to establish clear ground rules of what they can and can not do. To ease troubleshooting in case of unforeseen accidents, headless users need to be segregated by teams and also by functions. Let's consider a couple of scenarios.

Team A:	Team B:
 A headless user that deploys to production has access to this team's production assets.	 A third headless user that deploys to production has access to a different set of production assets.
 Another headless user who has write access to the artifact repository.	 A fourth headless user who has only read access to the artifact repository.

Essentially, at no time, can Team A accidentally step over Team B's assets, and vice versa.

Source: CloudBees, Inc.



- » **Manual gates.** While we know that manual gates are instituted to allow human intervention, it is important to understand there are two kinds of manual gates.

GATE TYPE	DESCRIPTION
Classic manual	<p>This type of gate creates disjointed pipelines. Let's simplify and use the example where one pipeline got split into two due to the presence of a manual gate.</p> <ul style="list-style-type: none"> » Upstream pipeline #1 finishes. » A human intervenes and performs some manual work. » After manual inspection of the work results, downstream pipeline #2 is either: <ul style="list-style-type: none"> - Not kicked off, or - Manually kicked off.
Automated manual	<p>Automated manual gates can be programmatically introduced in pipelines to pause and ask for human input. This prevents the pipeline from making one headless end-to-end run that now has a coded gate.</p> <ul style="list-style-type: none"> » Pipeline pauses and programmatically asks for input. » A human provides the input. » Depending on the human input, the same pipeline either: <ul style="list-style-type: none"> - Restarts from where it was paused, or - Aborts with appropriate notifications sent off.

Selective people are empowered to provide human input to pipelines to perform sensitive actions, especially in production where customers may become affected. The identities of these people are logged by pipelines for traceability and auditability. Moreover, automated manual gates should not be abused to poll for long periods of time, since this inflates test cycle time, feature lead time and, eventually, time to market.

Irrespective of what kind of gates we use, or which method (or combination) of methods we choose, the central idea is to gain responsible speed through our pipelines, and not suicidal speed.



Pipelines should be auditable production assets

Pipelines are classic examples of process as code, and processes should be auditable. Hence, pipelines should be auditable. Let's explore a few aspects of pipeline audits.

- » **Pipeline infrastructure.** To be auditable, pipelines should be instituted as production assets with production-grade hardware and network. Auditors, both internal and external, should look at pipelines as auditable assets, and should break out of old habits of looking only into “blessed change management servers” that are previously known to them.
- » **Pipeline logs.** Pipelines, like any other application, generate a wealth of information and data. This data is transactional in nature and reflect a chain of events that can be eventually traced back to commits in version control. These pipeline transactions can also be archived and aggregated to perform analytics to help organizations make informed decisions. To enable auditability and segregation of duties, pipeline logs are:
 - » **Configurable.** An enumeration of verbosity levels exist, like [Info, Warning, Error] and we can switch between different levels, depending on business requirements.
 - » **Access-controlled.** Pre-defined sets of people have access specific sets of logs and specific sections inside those sets.
- » **Timestamped.** Each row in the log records not just the activity but also the time it happened.
- » **Retained.** Logs should be retained for X months, per a log retention policy on different pieces that constitute the pipeline. For example, some engineers could have built the pipeline, others could have integrated tests with the pipeline and third-party *aaS vendors could have provided the infrastructure for the pipeline. Even though there appears to be multiple owners, it pays to have a single point of contact for accountability. Ownership could be rotated, as long as the person has the necessary expertise to understand the nuances.
- » **Pipeline as an app.** Pipelines are mission-critical applications for organizations which release quality and secure products to their customers frequently and predictably. Just like any other application, pipelines should follow the gold standards of software development and delivery and should be built as a [twelve-factor application](#). Hence, pipelines, like any other production asset, should be released via continuous delivery/ deployment pipelines.

Visualize software gates and pipelines

Pipeline visualization tools depict the state of products flowing through registered pipelines. Visualization helps teams understand not just their throughput but bottlenecks as well.

Additionally, different kinds of gates (for example, simple, composite and weighted composite) when visualized add color to understanding code promotion criteria. Visualizers reduce/eliminate the cognitive and collaboration overhead.

There are three types of gates:

- » Simple
- » Composite
- » Weighted composite

» **Simple gates.** Simple gates are linked to one KPI, for example:

- » Number of test failures = 0
- » Percentage of code coverage \geq X%



Business Value per Sprint

Source: CloudBees, Inc.

One important flaw is that in case the lone KPI is biased, the entire gate gets biased. For example, the number of releases per sprint measures how fast we move bits from point A to point B, but does not reflect the business value achieved in that sprint. Similarly, the number of tests executed may not be a true reflection of results as much as it is of an effort.

» **Composite gates.** Composite gates remove the one critical deficiency of simple gates and this helps automate the segregation of duties. To avoid skewing on a single metric (that could be championed by one influential person or group), composite gates rely on a diverse portfolio of metrics instead.

For example, let's consider a couple of indices that comprise a portfolio of metrics:

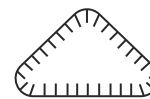
- » Code quality index = function of (cyclomatic complexity, code duplication, unit test coverage)
- » Stability index = function of (check in to go live, number of escaped defects, customer delight)

These indices will now be balanced uniformly between the chosen set of metrics.



Stability Index = function of (check in to go live, # of escaped defects, customer delight)

Source: CloudBees, Inc.



Code Quality Index = function of (cyclomatic complexity, code duplication, unit test coverage)

» **Weighted composite gates.** What if you don't want the balance between the participating metrics to be uniform? A weighted index helps you put more emphasis on certain metrics over others. For example, if you are a great believer of cyclomatic complexity, you could deliberately let it influence half of the overall index.

- » Code quality index = function of (50% cyclomatic complexity, 25% code duplication, 25% unit test coverage)
- » Stability index = function of (20% check in to go live, 40% number of escaped defects, 40% customer delight)

Honor compliance

Governance is often open to interpretation and is mistakenly used interchangeably with compliance. In this section, we will hone in specifically on compliance so that the differentiation is clear. Compliance means an organization's adherence to regulations, norms, standards and other protocols related to its business. Violations often result in legal punishment and/or fines, along with PR damage and loss of brand equity.

» **FISMA and NIST.** Federal Information Systems Act (**FISMA**) requires government agencies to implement an information security program that effectively manages risk. The National Institute of Standards and Technology (**NIST**) is a non-regulatory agency that has authored protocols on how to comply with FISMA. Let's study how FISMA requirements are addressed by the continuous governance tenets we discuss in this eBook.

- » **Maintain an inventory of information systems.** This aligns well with our continuous governance tenet to register pipeline metadata before accepting any pipeline as legitimate.
- » **Conduct continuous monitoring.** This aligns well with our requirement to perform pipeline analytics at all times. The nuggets of insights are then fed back into the system to complete the feedback loop. [CloudBees DevOptics](#) offers visibility and insights to measure, manage and optimize your software delivery and is well-positioned to be a continuous monitor of this kind.

» **Ensure the integrity, confidentiality and availability of sensitive information.** This aligns well with our discussion on the secure management of secrets, that is, passwords, keys, certificates and similar sensitive material.

- » **HIPAA.** The Standards for Privacy of Individually Identifiable Health Information ("Privacy Rule") establishes a set of national standards for the protection of certain health information. The U.S. Department of Health and Human Services (**HHS**) issued the Privacy Rule to implement the requirement of **HIPAA**. HIPAA standards address the use and disclosure of individuals' health information, called "protected health information" by organizations subject to the Privacy Rule. HIPAA also standardizes the individuals' privacy rights to understand and control how their health information is used.

This aligns well with our discussion on protecting consumer PII data from inadvertently appearing in pipeline logs.



- » **PCI.** The Payment Card Industry Data Security Standard (**PCI DSS**) is a set of security standards designed to ensure that companies who accept, process, store or transmit credit card information maintain a secure environment.

We are in the era of e-commerce and this aligns well with our data governance principle of protecting consumer financial data. Financial data, such as payment-related information or account-related information, cannot be exposed in pipeline logs or elsewhere. Note that the same constraint applies just as much to the product flowing through the pipeline as the pipeline itself.

- » **EU General Data Protection Regulation.** The EU General Data Protection Regulation (**GDPR**) is designed to harmonize data privacy laws across Europe, protect and empower all EU citizens and reshape the way organizations across the region approach data privacy.

GDPR redefined not just the data management process, but also the roles and responsibilities of the C-Suite. The latter now must ensure that they have watertight consent management processes in place, while requiring effective data rights management systems to ensure they don't lose their most valuable asset – data.

From the perspective of continuous governance, this aligns with protecting consumer data from being exposed in pipeline logs or elsewhere.

- » **Sarbanes-Oxley Act.** In 2002, the United States Congress passed the Sarbanes-Oxley Act (**SOX**) to protect shareholders and the general public from accounting errors and fraudulent practices in organizations, and to improve the accuracy of corporate disclosures. The act sets deadlines for compliance and publishes rules on requirements. All public companies now must comply with SOX, both on the financial side and on the IT side. The way in which IT departments store corporate electronic records changed as a result of SOX. While the act does not specify how a business should store records or establish a set of business practices, it does define which records should be stored and the length of time for the storage.

To comply with SOX, corporations must save all business records, including electronic records and electronic messages for a certain period of time. This aligns well with our continuous governance protocol of establishing:

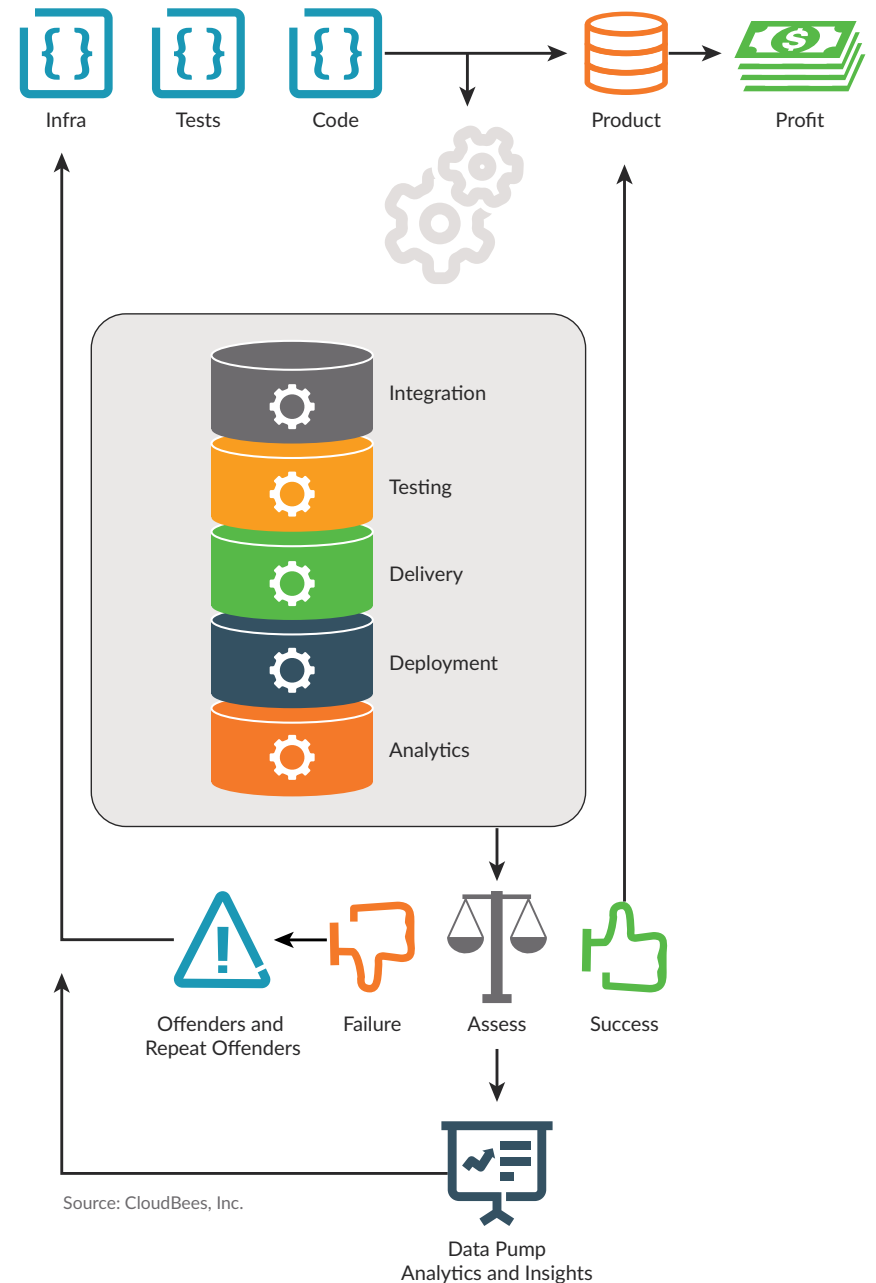
- » Artifact retention policies for development/test and staging environments and particularly for production.
- » Filing change requests automatically through pipelines before deploying in production (and sometimes even staging) and thereby create an audit trail.

Governance Engine

Now that there is an understanding of the continuous governance tenets, the next step is to deliver a model the industry can follow. In an ideal world, a standardized governance engine integrates with all pipelines to ensure they are operating in a governed manner. Which brings us back to the original question: Who governs governance? [CloudBees](#) is an industry leader in DevOps and continuous everything and is in a unique position to author governance protocols that encompass auditors, risk, compliance and engineering.

In the following figure, we illustrate the architecture of a governance engine. The engine feeds off the data emitted by pipelines, assesses the results and then feeds back into the pipelines to close the feedback cycle. If the assessment is positive, pre-configured software gates allow pipelines to promote code from one stage to the next. If the assessment is negative, alerts are generated and notifications sent to potential culprits. Repeat offenders are required to undergo rigorous training to help eliminate the root causes of failures.

Assessment data emitted by the governance engine is pumped into a reservoir for analytical processing. The nuggets of insights mined thereafter can then enable the organization to make informed data-driven decisions.





Common Misconceptions

There are some misconceptions associated with continuous governance. This section aims to bust as many myths as possible to reduce friction during execution.

Agile rests on collaboration principles, and hence we should govern in a collaborative manner. Apart from software, agile has eaten the world. While it is important to consider everyone's inputs, it is essential to keep the brightest and non-political minds as the governing body. Also, this body should govern without being paranoid. This enhances accountability and positions the organization for success.

Governance is superfluous, since we want our teams to be self-organizing. Another agile misconception. Teams can self-organize while being governed. In fact, with proper guardrails, teams can avoid shooting themselves in the foot, while releasing software frequently.

Governance is the same as compliance. No, it is different. In this eBook, we discussed compliance as one of the many tenets of governance.

Governance applies to large organizations only. Governance applies to organizations of all sizes. Large organizations might be under stricter regulations but that should not discourage small and mid-sized players from fastening their seat belts.

Continuous governance applies to software only. It applies to all product types, like software, firmware, embedded systems and IoT. Software is further ahead than the others, but the gap is narrowing as we write this eBook.

*We've won some governance battles in the past.
It's now time to win the war. Let's be a practitioner!*

Conclusion

To power the continuous economy is no mean feat! CloudBees is strategically positioned to drive continuous everything and has taken a definitive stand on continuous governance. While continuous integration, continuous testing, continuous delivery, continuous deployment and continuous analytics are seen as the formative pillars of continuous everything, continuous governance gets sidelined more often than not. Our lives and not just our work are tied to the success of the continuous economy and we don't want to cut corners on what's the right thing to do and how to do it.

Continuous governance has predictably emerged as the bone of contention since the undertaker of governance gains superiority over the governed. Organizations have been torn by powerful factions aspiring to dominate others. However, the solutions they provide are often political and not technical. They also mandate large departments of personnel to do manual verifications. Not only is this manual solution error-prone, it is also costly since in general, people are more expensive than tools.

The bottom line is that continuous governance is overdue. The industry is paying hefty penalties in terms of political divide, cybersecurity threats, privacy concerns and misconstrued separation of duties that lead to disrupted lives and loss of jobs. At CloudBees, we think we should remove the duct tape and introduce ironclad scientific solutions. As the market leader, we should not just create pockets of excellent, but scale those automated continuous governance recipes across the globe.

We've won some governance battles in the past. It's now time to win the war.
Let's be a practitioner!

About

CloudBees is powering the continuous economy by building the world's first end-to-end system for automating software delivery, the CloudBees Suite. The CloudBees Suite builds on emerging DevOps practices and continuous integration (CI) and continuous delivery (CD) automation adding a layer of governance, visibility and insights necessary to achieve optimum efficiency and control new risks. As every company in the world is now a software company, this new automated software delivery system will become the most mission-critical business system in the modern enterprise. As today's clear leader in continuous CI/CD, CloudBees is uniquely positioned to define and lead the automated software delivery category. CloudBees puts companies on the fastest path to transforming great ideas into great software and returning value to the business more quickly.

Backed by Matrix Partners, Lightspeed Venture Partners, Verizon Ventures, Golub Capital and Delta-v Capital, CloudBees was founded in 2010 by former JBoss CTO Sacha Labourey and an elite team of continuous integration, continuous delivery and DevOps professionals. Follow CloudBees on [Twitter](#), [Facebook](#), [LinkedIn](#) and [Google+](#).



The registered trademark Jenkins® is used pursuant to a sublicense from the Jenkins project and Software in the Public Interest, Inc. Read more at: www.cloudbees.com/jenkins/about

© 2018 CloudBees, Inc. CloudBees and CloudBees DevOptics are registered trademarks and CloudBees Core, CloudBees CodeShip, CloudBees Jenkins Enterprise, CloudBees Jenkins Platform and DEV@cloud are trademarks of CloudBees. Other product or brand names may be trademarks or registered trademarks of their respective holders.

1018v00