Configuration Promotion Automation

June 11, 2025

Copyright

All product technical documentation is
Ping Identity Corporation
1001 17th Street, Suite 100
Denver, CO 80202
U.S.A.

Refer to https://docs.pingidentity.com for the most current product documentation.

Trademark

Ping Identity, the Ping Identity logo, PingAccess, PingFederate, PingID, PingDirectory, PingDataGovernance, PingIntelligence, and PingOne are registered trademarks of Ping Identity Corporation ("Ping Identity"). All other trademarks or registered trademarks are the property of their respective owners.

Disclaimer

The information provided in Ping Identity product documentation is provided "as is" without warranty of any kind. Ping Identity disclaims all warranties, either express or implied, including the warranties of merchantability and fitness for a particular purpose. In no event shall Ping Identity or its suppliers be liable for any damages whatsoever including direct, indirect, incidental, consequential, loss of business profits or special damages, even if Ping Identity or its suppliers have been advised of the possibility of such damages. Some states do not allow the exclusion or limitation of liability for consequential or incidental damages so the foregoing limitation may not apply.

Table of Contents

Configuration Promotion at Ping Identity

Effective configuration promotion is paramount to the success and stability of modern Ping Identity solutions within an overall multi-vendor solution. By adopting a systematic and well-planned approach to configuration management and configuration promotion automation, organizations can mitigate deployment risks, ensure consistency, and streamline the configuration promotion process.

Configuration promotion is not just a technical process. It is a strategic endeavor that demands collaboration, communication, and adherence to best practices. Through version control, automated testing, thorough code reviews, and staged environment deployments, teams can confidently and reliably push configurations from development to test, then test to production.

Learn about configuration promotion automation

Learn about configuration promotion automation concepts and how Ping Identity deployments align with industry-standard GitOps methodology.

- Overview
- Configuration as Code (CaC)
- Testing Automation
- Auditing and Reviewing Changes
- Promoting and Deploying Configuration
- Verifying Deployed Configuration

GitOps CI/CD Pipeline Examples

Use the provided examples and templates to start your own Ping Identity Configuration Promotion Automation project.

- GitOps Pipeline Examples

Copyright © 2025 Ping Identity Corporation

Ping Identity developer tools: Terraform

Use Terraform to create configuration as code packages for Ping Identity product configuration. Declare end-state configuration and use Terraform to manage deployment of configuration to development, testing, staging and production environments. Use Terraform's drift detection and auto-healing capabilities to ensure configuration remains accurate and consistent.

- Get Started using Ping Identity's Terraform providers

Ping Identity developer tools: Ping CLI

Use Ping CLI to export configuration from fully configured Ping Identity deployments, generate Terraform import logic to import fully configured environments to Terraform state and script configuration management API calls.

- Get Started using Ping CLI

Copyright © 2025 Ping Identity Corporation

GitOps Pipeline Examples

Use the following template and example resources to get started with GitOps configuration promotion pipelines.

Ping Identity Platform Example CI/CD Pipeline

Learn how to use Terraform to manage and deploy Ping Identity platform configuration, including solutions, use cases, and third-party integrations in a GitOps CI/CD pipeline.

Example Pipeline on GitHub

Ping Identity Application Example CI/CD Pipeline

Learn how to use Terraform to manage and deploy applications (as part of application onboarding) that depend on services managed by a central IAM platform team in a GitOps CI/CD pipeline.

Example Pipeline on GitHub

Ping Identity Infrastructure Example CI/CD Pipeline

Learn how to use DevOps to deploy Ping Identity's Advanced Identity Software to self-managed infrastructure in a GitOps CI/CD pipeline.

Example Pipeline on GitHub

Copyright © 2025 Ping Identity Corporation

Configuration Automation Concepts

Automating configuration promotion brings efficiency, reliability, and risk mitigation into the deployment process. By automating the promotion of Ping Identity solution configuration through various environments, organizations ensure consistency and repeatability in their deployment practices. When delivered through a Continuous Integration and Continuous Deployment (CI/CD) pipeline, this process not only accelerates the delivery of new features but also minimizes the potential for human error, which is a common source of configuration issues.

These documented sections intend to raise considerations and provide suggestions to help build a strong foundation for automating configuration promotion as it relates to Ping Identity solutions. As such, guidance is provided generically, and details of pipeline implementation are left to the implementor. Examples provided in this guide are intended only to serve as a common foundation when incorporating Ping Identity solutions into pipeline implementations.

The reader is assumed to be past the discovery and exploration phase of Ping Identity products and is interested in adopting the DevOps and CI/CD practices for deploying into production environments. The reader should be familiar with the general concepts around CI/CD and SDLC as well as terms like:

- CI/CD, including pipelines.
- Infrastructure as Code (IaC).
- GitOps and similar methodologies.
- Tools such as Terraform, Ansible, Kubernetes, Helm.
- Foundational understanding of container technologies.
- Basic DevOps understanding, including automation and testing.

With these concepts in mind, these documented sections will explore the following areas:

- Export Configuration as code
- Validate Configuration
- Testing Automation
- Auditing and Reviewing Changes
- Promoting and Deploying Configuration
- Verifying Deployed Configuration

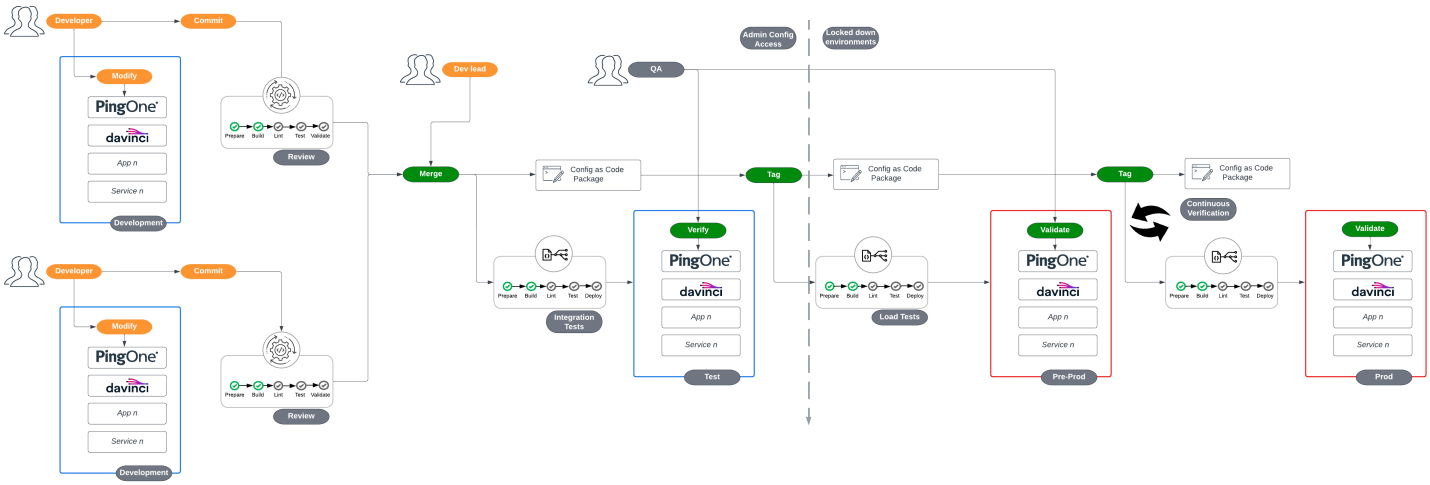
Definitions

- "Ping Identity solutions" as a term represents all forms of products and services that Ping Identity offers.
- "Environments" as a term represents deployment environments, such as development, test, pre-production, and production.
- Mentions of Terraform are relevant to Ping Identity offerings that have an available terraform provider.

Example pipeline diagram

For a common foundation on where concepts can be implemented into a pipeline, a generic pipeline diagram is shown below.

Copyright © 2025 Ping Identity Corporation



Copyright © 2025 Ping Identity Corporation

Configuration as Code (CaC)

Configuration as Code (CaC) is the practice of managing and automating the configuration of software and systems using machine-readable definition files. Instead of manually configuring settings through graphical interfaces or command-line tools, configurations are defined in code, stored in text files, and treated like any other piece of software. This allows for version control, collaboration, and automated deployments.

The key benefits of CaC include increased consistency, reduced errors, and improved auditability. By defining configurations in code, developers can eliminate the risk of human error associated with manual configuration. CaC also enables version control, allowing developers to track changes, revert to previous configurations, and maintain a history of modifications. This fosters collaboration, as teams can easily review and contribute to configuration changes. Furthermore, automated deployments ensure that configurations are applied consistently across all environments.

Export CaC

Engineers responsible for contributing features within Ping Identity solutions typically treat the GUI of the products as a development environment. Thus, they will need a process to extract the resulting configuration into code.

> **Note**
>
> Ping Identity recommends that complex use-case development (such as authentication and authorization policy definition or DaVinci flow design) occur in the admin consoles rather than manipulating exports directly.

Extracted CaC can take various forms, but it should fit within the following guardrails:

- **Readability:** Code should be human-readable and well-documented. Clear and concise code facilitates understanding, collaboration, and maintenance.
- **Modularity:** Configurations should be organized into modular components. This makes it easier to manage, update, and reuse configuration settings across different environments or components.
- **Versioning:** Like application code, configuration code should be versioned using a version control system. This enables tracking changes, rolling back to previous configurations, and collaborating effectively.
- **Idempotence:** The configuration code should be idempotent, meaning that applying the same configuration multiple times has the same result as applying it once. This ensures that repeated executions don't lead to unintended side effects or inconsistencies.
- **Reproducibility:** CaC should support the reproducibility of environments. Given a specific version of the configuration, it should be possible to recreate the same environment consistently.
- **Parameterization:** Configuration should be parameterized to allow flexibility. This enables the same codebase to be used across different environments or instances with varying requirements.
- **Security considerations:** Implement security best practices in the configuration code. Parameterize sensitive information and ensure that access controls and permissions are well-defined and enforced.
- **Error handling:** Include proper error handling mechanisms in the code. Log meaningful error messages and provide information that helps diagnose and resolve issues quickly.
- **Auditability:** Maintain an audit trail of changes to configurations. Ensure that changes are logged and the reasons for each change are documented. This aids in troubleshooting and compliance.

Copyright © 2025 Ping Identity Corporation

- **Consistency across environments:** Ensure that the same configuration codebase can be used across different environments (development, testing, production) with minimal modifications. This reduces the chances of configuration drift.
- **Scalability:** Design configurations to scale as the system grows. Consider the ability to manage configurations for a growing number of services, components, or instances.
- **Collaboration:** Facilitate collaboration among team members by following best practices for version control, code review, and documentation. Make sure that changes to the configuration are well-communicated and understood by the team.
- **Community and ecosystem support:** Choose tools and frameworks that have active community and ecosystem support. This ensures that there are resources, plugins, and extensions available to enhance and extend the functionality of the configuration code.

Forms of how this configuration can be stored include:

- Terraform HCL for solutions that have published Terraform providers (using the `terraform export` CLI feature).
- Exported configuration files containing binary or structured data.
- Postman API collection.
- Generic API collection (which can be organized in an orchestrator such as Ansible or ICEflo).
- Server Configuration Profiles (applies to Ping Identity software Docker images).

Validate configuration

Validating CaC is an important prerequisite for building confidence in the code review and promotion process. Code validation delivered through open-source tooling or developed scripts can be automatically run within the promotion pipeline.

Aspects of validation include:

- **Syntax checking:** Use syntax checkers specific to the configuration language developers are using, such as YAML, JSON, or HCL. Ensure consistent formatting and coding style across configuration files to help with reviewability.
- **Linting:** Additional linting beyond syntax checking can be used for service-specific validations. This can include dependency checking between configuration modules or logical error identification.
- **Security scanning:** Perform security scans on configuration code to identify and remediate vulnerabilities. Ensure that sensitive information, such as passwords and API keys, is properly hidden and managed.
- **Compliance Checks:** Implement checks for regulatory compliance and company policies within configuration validation processes.

Forms of configuration validation include:

- Terraform validate (`terraform validate`).
- JSON lint tools such as JSONLint.
- Postman schema validation and syntax error checks.
- Terraform sentinel policies.
- Static code analysis tools.

Copyright © 2025 Ping Identity Corporation

Copyright © 2025 Ping Identity Corporation

Testing Automation

Unit testing

Unit testing goes beyond validation by running code to ensure the configuration is functional and achieves what is intended. Unit tests aim to test individual components of configuration to isolate issues. Build tests into continuous integration (CI) pipelines to run them automatically whenever configuration changes occur. This helps catch issues early in the development process.

Aspects of unit testing include:

- **Isolation of components:** Test individual components configuration in isolation. Unit tests should focus on a specific part of the configuration, ensuring it behaves as expected independently of other components.
- **Mocking and stubbing:** Use mocking or stubbing techniques to isolate the unit under test from external dependencies. This allows developers to control the behavior of external components and focus on testing the specific unit. Mocking and stubbing Ping Identity services could mean virtualizing the backend services.

Forms of unit testing include:

- Terraform tests (`terraform test`) run against test-specific, short-lived resources, preventing any risk to existing infrastructure or state.
- Code specific frameworks, such as Jest (for Javascript) or the native `go test` (for go).
- Postman's Mock Server feature can simulate backend API responses.

Integration testing

Integration testing goes beyond unit testing by ensuring the whole configuration or modules are tested as a group. Integration tests aim to test identify regression between dependencies. Build tests into CI pipelines to run them automatically whenever configuration changes occur. This helps catch issues early in the development process.

Aspects of integration testing include:

- **Realistic environment simulation:** Create integration tests that simulate a realistic environment as closely as possible. This can involve using test environments or containers to replicate the production setup.
- **End-to-end testing:** Perform end-to-end testing to validate the entire configuration, including its interactions with non-Ping Identity system configuration that is managed simultaneously.
- **Data consistency:** Check for data consistency across different components. Ensure that data passed between various configuration elements maintains integrity and coherence.
- **Negative testing:** Conduct negative testing to evaluate how well the configuration handles unexpected or invalid inputs, errors, and adverse conditions. This type of testing aims to identify vulnerabilities, ensure graceful error handling, and enhance overall robustness and security.

Forms of integration testing include:

- Code-specific frameworks, such as Jest (for JavaScript) or the native `go test` (for go).
- Postman's Mock Server feature can simulate backend API responses.
- Open-source testing frameworks, such as Citrus and Fitnesse.

Copyright © 2025 Ping Identity Corporation

- Selenium automates browsers for testing configuration from the client perspective.

Copyright © 2025 Ping Identity Corporation

Auditing and Reviewing Changes

The audit and review process before accepting and deploying changes provide a gate for human interaction and validation of new features. This stage should provide a view of the differences between the current and desired configurations. To do this effectively, engineers should have a clear format to interpret configuration exports and determine the functional intention of changes. The audit and review process should enforce protection against users pushing through unverified changes and encourage the reviewer to check and revalidate compliance with all other areas mentioned in previous sections.

Forms of auditing and review of changes include:

- GitHub pull request (PR) with reviewers.
- QA engineer reviewing configuration within the UI of an automatically deployed Ping Identity solution.

Copyright © 2025 Ping Identity Corporation

Promoting and Deploying Configuration

After configuration changes pass review, the pipeline is ready to promote those changes into a pre-production and then production environments. Manual configuration access should not be available in pre-production or production environments, and this deployment should be automated in a similar or identical form of any other environment. Environments that are logically further in the promotion process beyond controlled testing environments (such as QA or user acceptance testing environments) should have continuous monitoring for errors as well as acceptable performance thresholds and alerting of outages. There should be an incident response plan for any anomalies that may be detected. Notification of anomalies could trigger an automated rollback if mutation outcomes can be guaranteed.

Forms of deployment of approved configuration include:

- Terraform apply (`terraform apply`).
- Postman API collection with Newman CLI.
- Collection of APIs with cURL or Ping CLI's custom API request.
- Server Configuration Profiles with `helm install`, customized Docker image builds (applies to Ping Identity software Docker images).

Copyright © 2025 Ping Identity Corporation

Verifying Deployed Configuration

Continuous verification is an automated process to detect configuration drift by calling read (GET) APIs and validating responses against the current desired configuration to identify drift and possible paths to remediation. Ping Identity Solutions could have involvement with external systems or dynamic configuration. As such, the ongoing reverification of configuration should include testing to confirm against external configuration interference.

Forms of continuous verification include cron jobs running:

- Terraform plan (`terraform plan`) command returning empty plan. This might involve HCL that includes the Terraform `check {}` block that can provide warnings if non-managed configuration is seen to drift from defined assertions.
- Postman API collection of reads that include response verification tests.
- Node.js API test package.

Copyright © 2025 Ping Identity Corporation