Liquibase

LIQUIBASE SECURE · TECHNICAL WHITE PAPER

Change Automation: Technical Overview

Architecture, Governance, and Runtime Posture of the Liquibase Secure Command Line Tool

Summary

Change Automation is the command line tool at the center of Liquibase Secure and the engine that applies database change. It installs on the developer workstations and CI/CD agents that already perform deployments, connects to more than sixty database platforms, and converts a proposed schema change into a governed operation: previewed as exact SQL before it runs, checked against policy, applied under a lock, recorded in a tamper evident ledger on the database itself, reversible one changeset at a time, and evidenced in a report a reviewer can read.

At the center of that operation is the changelog. The changelog is the standard, version controlled description of database change that every other capability consumes. It can be authored in an IDE, produced by an external tool, captured from an existing database, or generated by an AI assistant through Liquibase's own APIs. Whatever its origin, Change Automation treats it identically.

Three properties distinguish the tool in practice. Nearly every command that changes a database has a paired command that emits the exact SQL instead of running it, so approval happens on statements rather than on intent. The record of what has been applied lives on the managed database rather than in the pipeline, so it survives a CI platform migration. And policy checks evaluate a change against the same rules, held locally or governed centrally on Liquibase Secure Server, whether a person or an AI assistant wrote it.

Change Automation runs entirely inside the customer's trust boundary. It makes outbound connections only, holds no credentials beyond the moment of execution. Integration with Liquibase Secure Server adds centralized visibility (Change Intelligence) and centralized policy governance (Change Governance) to simplify governance capabilities and enable centralized visibility across database estates.

Key Outcomes for Enterprise Teams

- **Review the exact statements before they run.** Every update and rollback command has an -sql counterpart, so change boards approve SQL, not a description of intent.
- **Enforce standards in the tool, not in a person's head.** Policy checks run identically on a developer workstation and a build agent, from a local configuration or from a centrally governed policy set.
- **Keep a durable record on the asset.** Tracking tables on each managed database hold every applied changeset with checksum, author, order, and deployment identifier, independent of the CI platform.

© 2026 Liquibase Inc. All Rights Reserved

- **Reverse one mistake without unwinding a release.** Rollback operates per changeset, so a single bad change is removed while the correct work behind it stands.
- **Produce audit evidence as a by-product.** Operations reports and structured logs are written by the deployment that made the change, at the moment it ran.

The Governance Problem in Database Change

The question for a team responsible for database change is not whether they can deploy a schema change without a human touching it. Many already can. It is whether they could demonstrate, eighteen months from now and after a CI platform migration, exactly which statements ran against a given database, on what date, and who reviewed them first. When that answer lives inside a system scheduled for replacement, the record was never really a record.

Automated Is Not Governed

Most enterprises have automated their database deployments. Far fewer have governed them, and the distance between those two words is where a surprising share of production incidents still lives. The difference shows up under three questions. Who read the statements before they executed against production? For most teams the answer is a person's name rather than a process. What was applied to the staging database three releases ago? Usually a search through pipeline logs that have since rotated away. How would one bad column change be removed without unwinding the four good ones that shipped behind it? It would not be; the team would restore from backup and lose a day.

None of that reflects poorly on the people involved. The DBA who carries the replication topology in their head is not a bottleneck to be removed; they are the compensating control the organization built because nothing else was doing the job. What most organizations have is a process that is scripted rather than governed. A scripted process executes statements in order, holds no opinion about whether they are safe, keeps no durable record of which already ran where, and produces nothing a compliance reviewer would accept. It automates the execution but removes the one point where human judgment could catch a risk.

What Governed Automation Requires

For a database deployment to run unattended and still satisfy a change board, five things must be true of the tool performing it. The exact statements must be **reviewable** before they run, against that specific target, in its current state. Organizational standards must be **enforced** by the tool rather than by whoever happens to be reviewing that afternoon. What was applied where must be **recorded** somewhere durable and queryable. A single bad change must be **reversible** without sacrificing the correct work that followed it. And the **evidence** an auditor will eventually request must fall out of the deployment itself rather than being reconstructed later.

These five are not a feature list. They are the difference between a tool that types faster than a human and a tool an enterprise can hand a production database to. Every capability described in the architecture below exists to deliver one of them.

© 2026 Liquibase Inc. All Rights Reserved

Where the Record Lives

Record keeping is an architectural decision, and the one that matters most over a ten year horizon. Aviation offers the reference model: an aircraft's technical log travels with the airframe, not with the airline's maintenance system, because the airframe outlives every carrier and every software system that ever serviced it. Database change history usually lives in the deployment system instead: in GitHub Actions, in Jenkins, in whatever preceded Jenkins, or in a ticketing system. Change CI vendors and that history becomes an archive nobody can query, while the database keeps running and outlives the pipeline that changed it.

Liquibase puts the record on the database. Tracking tables on the managed database itself hold every changeset applied to it, each with a checksum, an author, an execution order, and a deployment identifier. Rewrite the pipeline, change CI platforms, or lose the workstation, and the ledger remains bolted to the asset, backed up alongside it, readable by anyone with database access. That is why the architecture places Change Tracking at the foundation rather than filing it under reporting.

What Change Automation Is, and Is Not

Change Automation is the command line execution engine of Liquibase Secure. It reads a changelog, compares it against the tracking tables on the target database to determine what has already run, and deploys only what is pending. Running it a second time therefore does nothing, which sounds unremarkable and is precisely what makes the operation safe to place in an automated pipeline. The same binary previews SQL, runs policy checks, inspects live databases for drift, orchestrates multi step flows, and writes reports and logs.

Change Automation is not a hosted service and not a control plane. It has no inbound network surface, no persistent daemon, and no Liquibase managed runtime in the deployment path. It is also not the authoring environment: changelogs are produced by developers and tools upstream of it, and the tool consumes them from version control or the filesystem. Visibility across pipelines and centralized policy governance are provided by the complementary Liquibase Secure Server, described later in this document, not by the tool itself.

Vocabulary

Changeset. A single unit of database change with an identifier and an author: add a column, create an index, deploy a stored procedure. A changeset may carry preconditions, an explicit or generated rollback, and contexts and labels that scope where it applies.

Changelog. An ordered file of changesets, written in Formatted SQL, XML, YAML, or JSON and kept in version control alongside the application code it supports. The changelog is the standard for describing database change in Liquibase Secure and the central artifact of Change Automation.

Tracking tables. The ledger resident on each managed database: DATABASECHANGELOG records applied changesets, DATABASECHANGELOGLOCK serializes concurrent runs, and DATABASECHANGELOGHISTORY (Liquibase Secure) records every operation, including those that changed nothing.

Policy checks. Deterministic rules evaluated against changelogs and live databases, each with a configured severity that determines whether a violation stops or merely annotates a run.

© 2026 Liquibase Inc. All Rights Reserved

Naming and Editions

Change Automation is the current name for the component that was called Secure Automation in the previous generation of Liquibase Secure (V5). Documentation, license keys, and support artifacts from that generation refer to Secure Automation; the two names describe the same command line tool and the same capability set, and this document uses Change Automation throughout.

Liquibase Secure is a distinct commercial product with its own distribution and license, not the free open source Liquibase Community tool with a key applied. The two are not designed to be used interchangeably on the same changesets or environments, so standardize on one per environment.

Change Automation Architecture

The diagram below shows the logical architecture. The changelog sits at the center of the tool as the artifact every capability consumes. To its left are the ways a changelog comes into being. Around it are the capability families of the command line tool. Below it are the target databases and the ledger they carry. To its right is the optional Liquibase Secure Server integration, alongside the equivalent local configuration and outputs the tool uses when no server is present.

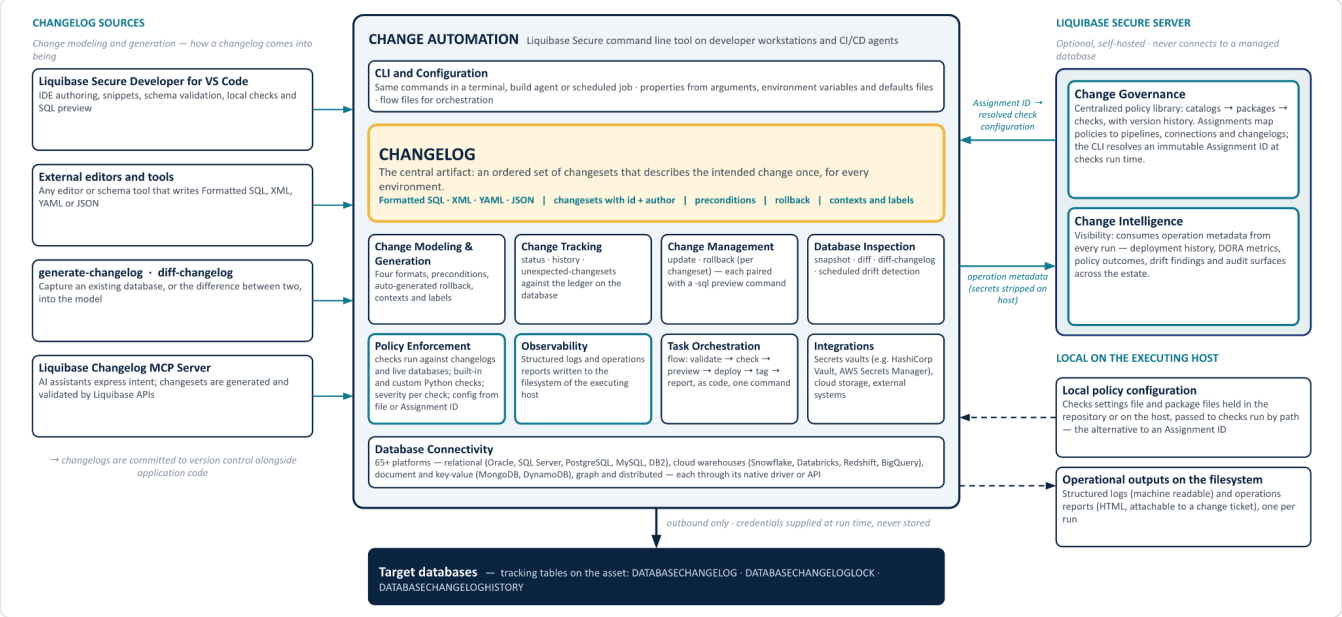


Figure: Change Automation logical architecture. Solid arrows show the Liquibase Secure Server integration; dashed arrows show local operation without the server.

The Changelog as the Central Artifact

Everything Change Automation does is an operation on, or a consequence of, a changelog. The changelog is the standard for describing database change: an ordered, version controlled set of changesets that states the intended change once and serves every environment through contexts and labels rather than through forked copies. Because it is a file in the application repository, it flows through the same pull request, review, and branch protection controls as application code, and because it carries identifiers and checksums, the tool can prove later which of its changesets were applied where.

© 2026 Liquibase Inc. All Rights Reserved

Changelogs are created as part of Change Modeling and Generation and can originate from any of four paths. Change Automation treats the resulting artifact identically regardless of source:

- **IDE authoring.** Liquibase Secure Developer for VS Code creates changelogs in all four formats with format aware snippets, bundled schema validation, and the operational commands (status, update, rollback, checks run, and their -sql previews) available from the editor. This is the primary interactive path and is described in the companion Developer Productivity Technical Overview.
- **External editors and tools.** Any editor, schema design tool, or code generator that writes Formatted SQL, XML, YAML, or JSON produces a valid changelog. Nothing in the format is proprietary to the IDE extension.
- **Capture from a live database.** The generate-changelog command captures an existing database into the model, and diff-changelog expresses the difference between two databases, or between a database and a snapshot, as a changelog. These are the entry points for bringing an existing estate under governance.
- **AI assisted generation.** The Liquibase AI Changelog Generator is an MCP server that lets AI assistants and coding agents request changesets in natural language. The model expresses intent and calls a tool with structured parameters; the changeset itself is generated by Liquibase's native Java change classes and validated against an in memory database before it is returned. The assistant never writes changelog syntax from memory.

Once a changelog exists, its lifecycle is the same on every path: committed to version control, checked against policy, previewed as SQL, deployed, tracked, and reported on.

CLI and Configuration

The command line is the primary entry point. The same commands run in a terminal, a build agent, and a scheduled job. Configuration properties can be provided as command arguments, environment variables, and a defaults file, resolved in a documented precedence, so environment differences are handled as parameters rather than as code branches. Flow files, described under Task Orchestration, allow a standard sequence of commands to be defined once and invoked as a single step.

Change Modeling and Generation

This capability family determines how change is expressed. It comprises the four changelog formats; preconditions that gate a changeset on the state it expects to find; automatically generated rollback for change types where the inverse is well defined, with explicit rollback blocks for the rest; and contexts and labels so one changelog serves development, test, and production without forking. It also includes the generation commands discussed above that produce changelogs from live databases.

Change Tracking

Change Tracking is the ledger. Commands such as status report what is pending, history reports what has run, and unexpected-changesets flags anything the database claims to have applied that is no longer in the changelog. This is the fundamental mechanism for knowing what should be present in a database, what is pending, and what is an anomaly, and because the ledger lives on the database, the answer is available to anyone with database access, without the pipeline.

© 2026 Liquibase Inc. All Rights Reserved

Change Management

Change Management applies and reverses change. update deploys pending changesets under a lock and records each in the tracking tables with its checksum. rollback reverses deployed changesets, to a tag, by count, to a date, or one changeset at a time. Granularity is the point: rollback-one-changeset reverses a single mistake without costing the release that shipped around it.

Nearly every command that changes a database has a paired command that writes the exact SQL instead of running it: update-sql for update, rollback-sql for rollback, and so on through the family. Approval therefore happens on actual statements, generated against the specific target in its current state, rather than on a description of intent. This is separation of duties implemented inside the tool rather than as a process beside it.

Database Inspection

Database Inspection reads live state. snapshot captures structure, diff compares two databases or a database against a snapshot, and diff-changelog expresses the difference as a changelog. Liquibase Secure adds scheduled drift detection with alerting. Monitoring for drift matters because any difference between actual and expected state means a change happened without going through the governed path.

Policy Enforcement

Policy Enforcement runs checks against changelogs and against live databases. Liquibase Secure ships a library of built-in checks and supports custom checks scripted in Python for organization specific rules. Each check carries a configurable severity, so a violation can fail a pipeline stage, warn, or merely annotate, and the same configuration drives checks run on a developer workstation and on a build agent. Automation is the only reliable way to apply standards consistently; at scale no DBA can be expected to remember every policy on every review.

Policies can be sourced from two locations, and the enforcement behavior is identical in both cases:

- **Local configuration.** A checks settings file (and any package files it references) held in the repository or on the executing host, passed to checks run by path. This is the self contained mode and requires no external dependency.
- **Centralized governance.** Policy assets defined once on Liquibase Secure Server under Change Governance and mapped to pipelines, connections, or changelogs through an Assignment. Change Automation is configured with the Assignment ID, the server URL, and an API token; when checks run or checks show executes, it requests the check configuration for that Assignment over HTTPS, receives the resolved checks and their settings, and runs them exactly as it would from a local file. Because the Assignment ID is immutable, policy authors can add, remove, or tune checks and retarget the Assignment without any change to pipeline configuration.

Observability

Observability produces two outputs for two audiences, both written to the filesystem of the host that executed the operation. Structured logs are machine readable records of every command, changeset, and check outcome, suitable for retention alongside the pipeline's other artifacts. Operations reports are HTML documents, rendered by the run that made the change, thorough enough to attach to a change ticket and readable by a reviewer without access to the tool. Both are produced by default and require no additional infrastructure.

© 2026 Liquibase Inc. All Rights Reserved

Task Orchestration

Task Orchestration sequences a deployment into a flow file: an as-code definition of stages such as validate, check, preview, deploy, tag, and report, with shell steps, conditionals, and error handling. A flow runs identically from a single command on any automation platform or on a workstation, so the standard steps of a database change are defined once and cannot be skipped by omission.

Database Connectivity

Change Automation covers more than sixty five database types across the relational estate (Oracle, SQL Server, PostgreSQL, MySQL, DB2, and others), cloud warehouses (Snowflake, Databricks, Redshift, BigQuery), document and key value stores (MongoDB, Amazon DynamoDB), and graph and distributed platforms. Depth matters as much as breadth: platform specific objects such as functions, triggers, and packages are modeled rather than flattened, and each platform is addressed in its own terms, relational engines through JDBC drivers or native tools, warehouses through their specific drivers, document stores through their own APIs. This matters because newer data platforms often sit outside whatever process governs the relational estate, and that gap is where audit findings originate.

Security Integrations

Integrations connect Change Automation to systems adjacent to the deployment: secrets managers such as HashiCorp Vault and AWS Secrets Manager so that a property holds a reference to a vault path rather than a credential, cloud storage for changelogs and reports, and other external systems. The common credential exposure is not an exotic attack; it is a password in a repository or a CI variable forty people can read, and the vault integrations exist to remove it.

Liquibase Secure Server Integration

Liquibase Secure Server is a complimentary, self hosted component that extends Change Automation in two directions. It never connects to a managed database and holds no credentials for one; everything it knows arrives from Change Automation instances running in the customer's own pipelines.

Change Intelligence (visibility). Change Automation reports operation metadata outbound to Liquibase Secure Server after each run, with sensitive properties (passwords, secrets, tokens, keys, credentials, and similar patterns) stripped on the originating host before any network call. Change Intelligence consumes that metadata and presents deployment history, DORA metrics, policy check outcomes, drift findings, and audit surfaces across every pipeline and environment. Change Intelligence is described in its own Technical Overview.

Change Governance (centralized policy). Change Governance is the centralized policy management layer of Liquibase Secure Server. It holds an organization's policy assets as a searchable, shareable library of catalogs, packages, and checks, with version history on every check and package and single action rollback to a prior version, and it maps those assets to the entities teams already operate (projects, pipelines, database connections, changelogs) through named Assignments. A single pipeline or connection can carry several Assignments, so a DBA group, a platform team, and a security review group can each enforce their own concerns on the same target while keeping each run narrow to what that group inspects.

© 2026 Liquibase Inc. All Rights Reserved

Change Governance is not a policy engine and does not execute checks. Change Automation remains the execution point: it requests the configuration behind an Assignment ID, authenticated by an API token scoped to the requesting group, and enforces the result on the executing host. The server stores and resolves the policy a run consumes, and, when Change Intelligence reporting is enabled, receives the outcome of that run so policy results appear alongside deployment and drift evidence. Access to catalogs and Assignments is governed by the same persona based permission templates, group overlays, entity level grants and denies, and immutable audit log that Change Intelligence uses.

Both directions of the integration are additive. A Change Automation instance that is not connected to Liquibase Secure Server operates fully from local configuration and writes its outputs locally; connecting it adds visibility and central governance without altering how a deployment executes.

Data Flow: One Change, End to End

The capabilities are easier to retain as a sequence. A single column addition, from keyboard to production, with the responsible capability in parentheses:

1. A developer adds a changeset to the changelog in the application repository, from the IDE extension, an external tool, or an AI assistant through the AI Changelog Generator. (Change Modeling and Generation)
2. Before committing, the developer runs policy checks locally and corrects what they flag. The checks come from the local configuration or from Change Governance on Liquibase Secure Server. (Policy Enforcement)
3. The change goes through pull request and review like any other code change.
4. The pipeline invokes a single flow. (Task Orchestration)
5. The flow runs validate, then policy checks as a build gate, then update-sql to produce the exact statements for the approval record. (Change Management, Policy Enforcement)
6. A DBA or change board approves those statements. Nobody approves a description.
7. The flow runs update. Changesets apply under a lock and are recorded in the tracking tables with their checksums. tag marks the release, and an operations report and structured logs are written to the host. Operation metadata, with secrets stripped, is reported to Liquibase Secure Server if configured. (Change Management, Change Tracking, Observability, Server Integration)
8. A scheduled diff runs against expected state to catch anything arriving outside this path. (Database Inspection)
9. Should the change prove wrong, rollback-one-changeset reverses it and leaves the rest of the release standing. (Change Management)

Every step is the same tool, driven by artifacts held in version control, producing a record as it proceeds.

Deployment Model and Runtime Posture

Change Automation installs in one step on the hosts that already perform deployments, with certified drivers, extensions, and integrations included. That eliminates the work of assembling and version matching drivers across a mixed estate, which otherwise consumes unbudgeted weeks and produces environment specific

© 2026 Liquibase Inc. All Rights Reserved

failures.

Aspect	Posture
Runs on	Developer workstations and CI/CD agents, as a command line process under the identity of the invoking user or service account
Network	Outbound only: to target databases, to secrets managers, and optionally to Liquibase Secure Server. No inbound path, no listening socket, no Liquibase managed runtime in the deployment path
Credentials	Supplied at execution from arguments, environment variables, defaults files, or a secrets vault reference. Never required in a changelog and never persisted by the tool
State	Held on the managed database (tracking tables). Outputs (reports, structured logs) written to the executing host's filesystem
Policy source	Local checks configuration, or Change Governance on Liquibase Secure Server
Usage and licensing	When connected to Liquibase Secure Server, operation metadata gives the organization an understanding of how license capacity and product capabilities are used across pipelines and environments; sensitive properties are stripped on the host before anything is sent

A less visible part of the runtime posture is that Change Automation is commercially maintained enterprise software. Liquibase Secure carries SLA backed support with named technical account management and available professional services, and receives frequent security patches, with critical vulnerability fixes committed inside fourteen days. For software that deploys database changes in every pipeline, the support and patch commitment is a security control in its own right.

Governing Change That an AI Wrote

Developers already ask coding assistants to write migrations, and that practice has become the default very quickly. The question for an organization is where to put the controls. An agent's judgment is not a control: an agent writing a changelog from memory is likely to produce changes that appear correct and are not, using invalid attributes, unsupported types, or silently omitting rollback logic.

Liquibase Secure places the control on the change rather than on the author. Policy checks are deterministic and do not distinguish a human authored changeset from a generated one. Upstream, the AI Changelog Generator narrows the model's role to expressing intent: the changeset is generated by Liquibase's own API and validated before it is returned, so a model that misunderstands a request produces a wrong but valid changeset the developer can see and reject, never invalid syntax that fails in a pipeline. Downstream, the SQL preview, the tracking tables, and the operations report apply unchanged. The governance path for AI written change is the same path as for any other change.

Audit Readiness as a By-Product

The expensive part of an audit is rarely the finding. The cost is the preparation: locating logs, reconstructing history, and filling gaps in the record. Liquibase Secure inverts that sequence. Tracking tables hold a tamper evident history in which every applied changeset carries a checksum and later modification is detectable. Operations reports are generated by the deployment that made the change, at the moment it ran. Structured logs

© 2026 Liquibase Inc. All Rights Reserved

are written alongside them on the executing host and can be retained under the organization's existing log retention practice. Governance regimes such as SOX, HIPAA, and PCI DSS all ask a version of the same question, who changed what, when, and under what authority, and Change Automation accumulates the answer through its standard operation.

Where It Sits in the Liquibase Secure

Change Automation is the execution engine that delivers defined changes to target databases. The other components of Liquibase Secure either help define the change or govern and report on its execution.

Component	Role	Relationship to Change Automation
Change Automation	Command line execution engine: preview, check, deploy, track, roll back, report	This document. Known as Secure Automation in Liquibase Secure V5
Change Intelligence	Deployment history, DORA metrics, drift findings, and audit surfaces across the estate	Consumes operation metadata emitted by Change Automation
Change Governance	Centralized policy governance and enforcement configuration	Supplies policies that Change Automation enforces on the host
Secure Developer for VS Code	Changelog authoring, local operations, and policy checks in the editor	Produces changelogs; invokes Change Automation locally as a child process
AI Changelog Generator	Deterministic, API generated changesets from natural language for AI assistants using MCP interface	Produces changelogs; no database connectivity

© 2026 Liquibase Inc. All Rights Reserved