TRUST CENTER

Change Management Policy

How changes to code, infrastructure, and procedures are proposed, reviewed, and deployed.

Effective: May 21, 2026 Owner: Howell & Gibbs LLC Status: v1 – initial publication

Review cadence: Semi-annual, or after any material change to CI, deploy, or hosting architecture, or any change in team size that affects the review model in §6

---

1. Purpose

This policy describes how changes — to application code, infrastructure configuration, third-party integrations, and operational procedures — are proposed, reviewed, deployed, and audited at SendTax. It supports the commitments made in the **Information Security Policy** §5.4 and aligns with SOC 2 Common Criteria CC8.1 (change management).

This policy is a companion to the **Information Security Policy**, **Access Control Policy**, **Incident Response Policy**, and **Business Continuity & Disaster Recovery Policy**.

2. Principles

- **Changes go through CI.** No change reaches production without passing automated checks. There is no "trusted bypass" — including for the co-founders.
- **Two stages, every time.** Production deploys are preceded by a staging deploy of the same artifact. Staging is not a coincidence of trunk-based development; it is a deliberate gate.
- **Reviewable history.** Every change is traceable to a pull request, a commit, an author, and a CI run. The audit trail is the same surface engineers work in — not a separate ticket system that drifts from reality.

- **Reversible by default.** Every change should have a thought-out rollback. Database migrations are the special case (§9), but even then, changes deploy with the rollback in mind.
- **Honest about scale.** SendTax operates with a two-person team. This policy reflects the controls that are achievable, audited, and enforced at that scale, not the formal Change Advisory Board process of a 200-person company.

3. Scope

This policy applies to:

- **Application code** — frontend (Next.js apps) and backend (FastAPI + Celery worker)
- **Infrastructure as code** — `fly.toml` files, GitHub Actions workflows, Dockerfiles, provisioning scripts under `st-backend/infra/`
- **Database schema** — Alembic migrations
- **Operational configuration** — Fly.io secrets, third-party service configuration accessed through the relevant providers' consoles
- **Operational procedures** — runbooks and internal documentation used in incident response and routine operations
- **Trust Center documents** — these are themselves treated as changes that flow through the same review process

It does not govern routine operator actions (a single deploy, a secret rotation) that are themselves applications of an existing documented procedure.

4. Change categories

Every change is one of four categories. Category drives the path it takes through this policy.

Category	Definition	Path
Standard	Any planned change to code, infrastructure, or schema in the normal flow of work	§6 standard flow

Pre-approved	Recurring routine changes documented as procedures (secret rotation, dependency updates, deploy to staging from an approved branch)	§6 standard flow with abbreviated review
Emergency	A change required to contain or remediate an active incident, where waiting for the full standard flow would cause material harm	§10 emergency flow
Trust Center / policy	A change to a Trust Center document or other published policy	§6 standard flow with the policy owner as required reviewer

5. Roles and responsibilities

Role	Responsibility
Author	The person making the change. Responsible for the design, the implementation, the tests, and the rollout plan.
Reviewer	The person reviewing the change. Required for non-trivial changes; see §7.
Deployer	The person triggering the production deploy. May be the author for standard changes.
Owner of the affected area	For changes touching critical paths (encryption, RLS, billing, schema), the area owner reviews even if a different reviewer was already assigned.

In a two-person team, the author and reviewer are usually the only two people involved. §7.2 describes how the single-reviewer constraint is currently mitigated.

6. The standard change flow

Every standard change moves through these stages, enforced by GitHub Actions and the deploy workflows.

6.1 Branch and commit

- Work happens on a feature branch off `main`. Direct commits to `main` are not permitted.
- Commit messages describe **what changed and why**, not just what was touched.

6.2 Open a pull request

- Pull requests target `main`.
- The PR description states the user-facing intent, the testing approach, any operational risk, and the rollback plan if non-obvious.

6.3 Local pre-commit gate

Before a PR is opened, the author's local pre-commit hooks run:

- Format and hygiene checks (trailing whitespace, large files, merge-conflict markers, debug statements left in code)
- `make check-fast` for backend changes — runs the same fast lint/type checks that CI enforces
- End-to-end gate check for backend and frontend tier-1 paths

The pre-commit hooks are defined in `.pre-commit-config.yaml` and in `st-apps/.husky/pre-commit`.

6.4 Continuous integration

Every PR (and every push to `main`) triggers CI workflows scoped to the changed area by `paths:` filters.

Backend (`.github/workflows/backend-ci.yml`):

- `lint-type` — `ruff` (linter) and `mypy` (type checker). The workflow is designed to mirror `make check-fast` so that a `git push --no-verify` cannot bypass the lint and type gate.
- `tests` — full pytest suite, running against a PostgreSQL service container

Frontend (`.github/workflows/frontend-ci.yml`):

- `pnpm -w lint`
- `pnpm -w type` (TypeScript type checking)
- `pnpm -w build` (full monorepo build)
- Lockfile discipline — CI fails if `pnpm-lock.yaml` is mutated unexpectedly during a CI run
- OpenAPI client drift check — the frontend's generated API client must match the backend's published OpenAPI schema
- Monorepo test suite
- Optional Playwright E2E suite, gated by repository secrets

A failing CI run blocks merge. There is no documented procedure to bypass CI; SendTax treats that as a policy invariant.

6.5 Review

Per §7 below.

6.6 Merge

Merges to `main` use squash-merge by default for application code. The merge commit message preserves the PR title and description.

6.7 Staging deploy

A push to `main` automatically triggers:

- `backend-deploy.yml` (staging environment) — builds and deploys to `st-api-stg` and `st-worker-stg`
- `frontend-deploy-web.yml` (staging environment) — deploys the Fly-hosted frontend apps to their staging variants

Alembic migrations run as Fly's `release_command` **before** new machines serve traffic; a migration failure aborts the deploy and keeps the old machines healthy.

6.8 Production deploy

Production deploys are **manual** and **one path only**. The `release.yml` workflow ("Release to Production") is the single sanctioned entry point. It calls `backend-deploy.yml` and `frontend-deploy-web.yml` with `environment: production`.

Inside the deploy workflow, a job gate enforces that no other trigger (no `workflow_run`, no direct `workflow_dispatch` of the deploy workflow) may set `environment=production`. This is a code-level guarantee in `.github/workflows/backend-deploy.yml`.

The deploy order is:

1. **Backend.** Migrations apply first (via `release_command`), then API and worker machines roll. Backend goes first because schema additions may be prerequisites for the new frontend.
2. **Fly-hosted frontends.** `st-filer-web`, `st-pro-web`, `st-admin-web`
3. **Vercel-hosted marketing site.** Fast-forwards the `deploy` branch to the released SHA, which the `st-www` project auto-deploys

Each stage is rolled to staging and re-validated before its production counterpart proceeds.

7. Code review

7.1 What review covers

A reviewer is expected to read the change for:

- **Correctness.** Does the code do what the PR claims?
- **Safety.** Any risk to encryption boundaries, RLS, authentication, or audit-log integrity?
- **Tests.** Are there tests for the new behavior, and do they actually exercise the path?
- **Operational fit.** Migrations safe? Secrets handled correctly? Rate limits sensible? Errors logged?

- **Trust Center alignment.** Does the change require an update to any published policy or the Sub-Processor List?

7.2 Who reviews

- **Default.** A reviewer other than the author. The current team is two people; in practice this means the other co-founder.
- **Self-merge exception.** For genuinely trivial changes (typo fixes, documentation-only edits, dependency bumps already signed off by automated scanning), the author may self-merge. Self-merges are recorded in the PR for audit. In current practice, Liam reviews Holly's changes; Liam self-merges his own changes — a documented exception that remains in place until the team grows beyond the two co-founders (§14).
- **Roadmap.** As the team grows, SendTax will move to a model where every non-emergency change requires a non-author reviewer with no self-merge exception (§14).

7.3 Sensitive-area reviews

Changes to any of the following require **explicit acknowledgment from the area's owner** in the PR before merge, in addition to any generic review:

- The encryption module (`st-backend/app/core/encryption.py`)
- The Row-Level Security policies (`st-backend/app/db/rls.sql`) and the tenancy module (`st-backend/app/core/tenancy.py`)
- Authentication flows (`st-backend/app/api/routes/auth.py`, the Clerk webhook handler)
- Audit-log writes (the `audit_service`)
- Deploy workflows and `release_command` configuration

8. Testing requirements

- **Unit tests** are expected for new application logic
- **Integration tests** are required for changes touching the encryption module, RLS policies, audit-log hardening, and cross-tenant access paths. Existing integration tests cover these areas and a change is expected to extend rather than remove them.

- **Migration tests** — schema changes that affect existing data must include a backfill plan and, where feasible, a test that exercises the backfill against a representative dataset
- **Manual testing** of UI changes happens in staging before production release. The release runbook makes staging validation an explicit prerequisite.

CI failure is sufficient to block merge; CI passage is not sufficient to authorize merge. A reviewer's judgment about test coverage is also part of the gate.

9. Infrastructure and configuration changes

Infrastructure and configuration changes follow the same flow as code changes:

- **`fly.toml` changes** are committed to source control and reviewed in PRs
- **GitHub Actions workflow changes** trigger a workflow-file validator in the frontend CI suite, in addition to standard review
- **Secrets** are set via `fly secrets set` or via the provisioning scripts under `st-backend/infra/`, with the secret values themselves sourced from environment-scoped 1Password vaults. Changes to *which* secrets exist (the schema, not the values) flow through PRs.
- **Direct console changes** to third-party providers (Cloudflare, Clerk, GCP, Stripe, Postmark, etc.) are minimized. No formal per-change log for provider-side console changes exists today; establishing a structured change log per provider-side change — so the audit trail is preserved outside source control — is on the roadmap (§15).

10. Database migration discipline

Database changes are the highest-risk category and have their own runbook at `st-backend/docs/ops/MIGRATIONS.md`. The policy-level commitments are:

- **Migrations are forward-only by default.** SendTax does not expect to run `alembic downgrade` against production. Recovery from a broken schema change uses PITR restore plus a corrective migration, not a downgrade.
- **Migrations run as Fly's `release_command`** *before* new machines serve traffic. A non-zero exit aborts the deploy and keeps the old machines healthy.

- `CREATE INDEX CONCURRENTLY` **and other non-transactional operations** are flagged in code review because they commit incrementally and partial state is possible on failure.
- **Backfills** of existing rows are written to be idempotent and run incrementally where the dataset is large.
- **Dual-deploy pattern** for breaking schema changes: deploy additive change → backfill → deploy code that uses the new shape → deploy cleanup of the old shape. The runbook describes this pattern in detail.

11. Emergency changes

An emergency change is one that must skip part of the standard flow because waiting would cause material harm — typically during an active Critical or High-severity incident (see **Incident Response Policy**).

11.1 What is permitted

During an emergency, the following may be relaxed:

- **Pre-merge review.** A change may be merged without prior reviewer approval if the Incident Commander authorizes it.
- **Test coverage.** A change may ship without tests for the new behavior if the corrective action is time-critical.

11.2 What is NOT permitted, even in an emergency

The following always apply:

- **CI must pass.** No emergency justifies merging code that fails lint, type checks, or existing tests.
- **The staging → production sequence stands.** Even an emergency production deploy runs through the staging gate; this typically takes only a few minutes and is not skipped.
- `release.yml` **remains the only path to production.** Direct production dispatch is not permitted.
- **Secrets are not committed to source control.** Always.

11.3 After the emergency

Every emergency change generates a follow-up obligation:

- A post-emergency PR adds the missing tests, refines the hotfix into a clean change, and updates documentation as needed
- The emergency itself is documented on the incident timeline per the **Incident Response Policy**
- The post-incident review evaluates whether the emergency shortcut was justified and whether the standard flow needs adjustment to avoid the same shortcut next time

12. Rollback

Every change must be rollback-able. Specifically:

- **Application code rollback** is performed by deploying the previous image tag through the same `backend-deploy.yml` or `frontend-deploy-web.yml` workflow
- **Frontend rollback (Vercel)** is performed by reverting the `deploy` branch to the prior SHA and triggering an auto-deploy
- **Schema rollback** uses PITR restore plus a corrective forward-migration. SendTax does not rely on Alembic downgrade scripts to rescue production.
- **Configuration rollback** uses `git revert` on the relevant `fly.toml` or workflow file, followed by a deploy

The deploy runbook in `st-apps/docs/RUNBOOKS.md` documents the basic deploy/rollback procedures (see also the **Business Continuity & Disaster Recovery Policy** §8.2).

13. Audit trail

The audit trail for changes is built into existing tools, not maintained separately:

- **Git history.** Every change has an author, a timestamp, a PR reference, and a commit message.
- **Pull-request history.** Reviews, approvals, and comments are preserved by GitHub.

- **CI history.** Every check, every run, every failure is preserved in GitHub Actions for the duration of GitHub's default retention.
- **Deploy history.** Fly retains deploy records (image, who triggered, outcome) and is queryable via `fly releases`.
- **Application audit log.** Sensitive actions inside the running application are recorded in the `audit_log` table per the **Access Control Policy** §8.

Together these provide a defensible record of who changed what, when, why, and with whose approval.

14. Documentation

A change is not complete until the relevant documentation is updated:

- **Operational runbooks** (`st-apps/docs/RUNBOOKS.md`, `st-backend/docs/ops/MIGRATIONS.md`) are updated when a procedure changes
- CLAUDE.md is updated when the high-level architecture changes meaningfully
- **Trust Center documents** are updated when a change affects any published commitment (sub-processors, security posture, data flows)

Documentation updates ride in the same PR as the change, not in a separate "follow-up" PR.

15. Roadmap commitments

Commitment	Current state	Gap	Target
Structured change-log entry per third-party console change (§9)	Honor-system informal log [VERIFY if any log exists at all]	Schema and discipline for capturing console-side changes systematically	Within 6 months
Removal of self-merge exception once the team grows (§7.2)	Permitted today for trivial changes [VERIFY if used in practice]	Mandatory non-author review on all changes once team size > 2	Triggered by first non-founder hire

16. Exceptions

Any deviation from this policy beyond the §11 emergency flow requires:

1. Approval from a second SendTax operator
2. A documented justification recorded with the change
3. A defined re-evaluation point

Exceptions are reviewed at each policy review and are not allowed to become permanent without explicit re-authorization.

17. Enforcement

Violations of this policy may result in revocation of access, termination of employment or contract, and, where applicable, civil or criminal referral. The §2 principle — no trusted bypass, including for co-founders — applies to enforcement as well as to the rules themselves.

18. Related policies

- **Information Security Policy** — umbrella policy; secure development commitments in §5.4
- **Access Control Policy** — audit log scope (§8); production deploy access model (§5.3)
- **Incident Response Policy** — Incident Commander authority during emergency **changes (§5)**
- **Business Continuity & Disaster Recovery Policy** — rollback procedures (§8.2, §12); data corruption recovery (§8.4)
- **Vendor Management Policy** — sub-processor change handling, which flows through Trust Center document updates (§4)

19. Document control

Version	Date	Author	Notes
1.0	May 21, 2026	Holly Gibbs	Initial publication

20. Contact

Security disclosures	security@send.tax
Privacy inquiries	privacy@send.tax
Operating entity	Howell & Gibbs LLC (operator of SendTax)