Mixpeek Change Management Policy

Version: 1.0 **Effective Date:** April 15, 2026 **Last Reviewed:** April 15, 2026
Owner: Mixpeek Engineering

1. Purpose

This policy defines the procedures for planning, testing, approving, deploying, and rolling back changes to Mixpeek production systems. It ensures that changes are made safely, with appropriate testing and verification, to maintain system reliability and security.

2. Scope

This policy applies to all changes to Mixpeek production infrastructure, including: - Application code (API, Engine, Studio, Canvas, Homepage, MCP) - Infrastructure configuration (Kubernetes manifests, Terraform, network policies) - Database schemas and migrations - Third-party integrations and dependencies - CI/CD pipeline configurations

3. Change Categories

Category	Description	Examples	Approval
Standard	Routine changes following established procedures	Feature additions, bug fixes, dependency updates	Pre-approved via automated gates
Emergency	Urgent changes to restore service or fix critical security issues	Hotfixes for outages, security patches	Post-hoc review within 24 hours
Significant		Database migrations, infrastructure changes,	

Category	Description	Examples	Approval
	Changes with broad impact or elevated risk	auth system modifications	Explicit review before deployment

4. Change Lifecycle

4.1 Development

All changes must be: 1. Developed in a local environment with the full service stack running (API, Celery, Ray engine, MongoDB, Qdrant, Redis). 2. Type-checked where applicable (`tsc --noEmit` for TypeScript, Ruff for Python). 3. Linted for code quality and security (Ruff, Bandit, ESLint). 4. Tested against the relevant test suite.

4.2 Pre-Deployment Verification

Before any change is pushed to production, the following gates must pass:

Server changes:

```
1. Local E2E test passes:
   python server/scripts/e2e/staging_e2e_test.py \
     --base-url http://localhost:8000 \
     --namespace e2e-local-verify

2. Pre-commit hooks pass (Bandit, Ruff, architecture boundaries)
```

Studio changes:

```
1. npx tsc --noEmit  (zero type errors)
2. npm run lint      (no new lint errors)
3. npm run build     (Vite frontend + proxy server)
```

Homepage changes:

```
1. Local dev server verification
2. Explicit approval before push
```

4.3 Deployment

Mixpeek uses a direct-to-main workflow with automated deployment:

Service	Deploy Trigger	Method	Rollout Time
API, Celery, MCP	Push to main	GitHub Actions → GKE rolling update	3-5 min
Engine (Ray)	Push to main (engine/ changes)	GitHub Actions or manual `deploy_service.sh`	10-15 min
Studio	Push to main	GitHub Actions → GKE	3-5 min
Homepage	Push to main (with approval)	GitHub Actions → GKE	3-5 min
Canvas	Push to main	GitHub Actions → GKE	3-5 min

Deployment safety features: - Rolling updates with `maxSurge: 1, maxUnavailable: 0` (zero-downtime) - Startup, readiness, and liveness probes on all services - Graceful shutdown with `preStop` sleep for connection draining - PodDisruptionBudgets ensure minimum availability during updates - Engine supports canary deployments (`--canary` , `--promote` , `--rollback`)

4.4 Post-Deployment Verification

After every deployment:

1. **Wait for rollout completion:** `kubectl rollout status deployment/<service> -n mixpeek-engine --timeout=300s`
2. **Run production E2E test:** `python server/scripts/e2e/staging_e2e_test.py \ --base-url https://api.mixpeek.com \ --namespace e2e-prod-verify`
3. **For homepage/studio changes:** Verify every production URL returns correct content (not a 404 page).

4.5 Rollback

If post-deployment verification fails:

Service	Rollback Method
API, Celery, MCP	`kubectl rollout undo deployment/<service> -n mixpeek-engine`
Engine	`bash infra/gke/deploy_service.sh --rollback`
All services	Git revert + push triggers new deployment

Rollback decisions should be made within 15 minutes of detecting a production issue.

5. Infrastructure Changes

Changes to Kubernetes manifests, Terraform configurations, or network policies require:

1. Review of the change diff before applying.
2. Verification that no in-progress rollouts are active.
3. Staged application (one manifest at a time where possible).
4. Immediate monitoring for 15 minutes post-apply.

6. Database Changes

- Schema migrations must be backward-compatible (no breaking changes to running code).
- Data migrations must be tested against a representative dataset.
- Direct database writes (insert, update, delete) are prohibited except through the API pipeline.
- Read-only queries for debugging are permitted.

7. Dependency Changes

- All dependencies are pinned to specific versions (Poetry lockfile for Python, package-lock.json for JavaScript).
- Dependency updates are monitored via GitHub Dependabot.
- Security-critical dependency updates are prioritized and deployed within 48 hours.
- Major version upgrades require local testing of the full E2E suite.

8. Emergency Changes

For critical outages or active security incidents:

1. Fix may be deployed without full pre-deployment verification.
2. Incident Commander authorizes the emergency change.
3. Post-deployment verification is still required.
4. A post-incident review must document: what changed, why it was urgent, and what gates were skipped.
5. Any skipped gates must be retroactively satisfied within 24 hours.

9. Change Records

All changes are recorded through: - **Git history** — commit messages with descriptive summaries on the `main` branch. - **GitHub Actions** — deployment logs with timestamps, artifact versions, and deployment status. - **ClickHouse audit logs** — API-level changes captured with actor, action, resource, and timestamp. - **Kubernetes events** — rollout history retained for 5 revisions per deployment.

10. Policy Review

This policy is reviewed annually and updated when deployment processes, infrastructure, or regulatory requirements change.

This document is maintained by Mixpeek Engineering. Questions should be directed to info@mixpeek.com.