1) Problem statement (concise)

Fintech teams deploy microservices fast. Regulations (AML, KYC, PCI, data residency, consumer-protection) require that every release complies with many written policies and controls. Today compliance checks are often manual, slow, inconsistent, and hard to enforce in CI/CD. This causes regulatory risk, slowed releases, and vulnerable services misconfigured in production.

2) Your solution (one-line + description)

Regulatory Compliance CI/CD Sandbox (FinComply) — a policy-as-code platform that automatically validates fintech microservices and their infra/configs against AML/KYC/PCI and company policy during CI/CD. It uses OPA (Rego) for deterministic policy checks, plus LLM-assisted interpretation and a RAG (retrieval) layer to map ambiguous deployment artifacts to regulation clauses. The platform produces human-readable compliance reports, remediation steps, and a sandboxed auto-remediation option for low-risk fixes.

Why this is valuable:

- Enforces rules automatically in PRs / pipelines.
- Interprets regulatory language using RAG + LLM to generate explainable mappings from regulation -> failing artifact -> remediation.
- Tracks policy drift and provides audit trails for compliance teams.

3) High-level architecture / flow

(Verbal diagram)

1. **Input sources**
    - Repo: application code, Dockerfiles, CI manifests, Kubernetes YAML, Terraform.
    - Runtime manifests: Helm charts, cloud infra, env vars, secrets configs.
    - Team docs & regulatory corpus: FATF guidance, PCI-DSS, AML policy doc, internal policies.

2. **Pre-check stage (static analysis)**
    - SAST + dependency scanning (snyk/ligo/semgrep).
    - IaC scanning (Checkov, tfsec, KICS).
    - Container image scan (Trivy).
    - Basic rule engine checks (e.g., no secret in repo, HTTPS enforced, TLS versions, CORS).
3. **OPA Policy Layer (policy-as-code)**
    - Rego policies for deterministic checks (e.g., “K8s secrets not in plain YAML”, “no wide RBAC clusterrolebinding”).
    - OPA runs in CI pipeline and returns PASS/FAIL + policy ID.
4. **ML/LLM-assisted Interpretation**
    - Retrieval (embeddings + vector DB) over regulatory corpus and prior incidents.
    - RAG + instruction-tuned LLM to:
        - Map ambiguous infra configuration to likely control(s).
        - Generate human-readable explanation: “This Dockerfile disables user namespace — maps to PCI requirement X”.
        - Suggest remediation snippets (e.g., K8s patch, sample Terraform fix).
    - A classification model identifies severity (High/Medium/Low) and which regulation sections apply.
5. **Policy decision & auto-remediation**
    - Policy engine chooses action per rules: block merge, warn, auto-create remediation PR (with patch), or escalate to security team.
6. **Dashboard & Audit**
    - UI: PR view with failing checks, graph linking artifact -> regulation text -> remediation + confidence score and provenance.
    - Audit logs, versioned policies, and retraining triggers.

7. **Feedback loop**
    - Developer feedback (mark suggested remediation accepted/rejected) feeds back to retriever and fine-tunes classifier/regression for future improvements.

4) ML components you'll need (roles & tasks)

1. **Retriever / Embeddings** — map regulatory documents, internal controls, and past incidents into a vector store for RAG.
    - Task: semantic search for regulation clauses matching a failing artifact.
2. **Instruction-tuned LLM (RAG generator)** — produce human-readable mapping and remediation suggestions.
    - Task: given artifact + retrieved clauses, generate explanation and code snippet remediation.
3. **Classifier(s)** — detect compliance violations from text/configs, and categorize severity/regulation (multi-label classification).
    - Task: label a failing file with one or more control IDs (AML/KYC/PCI/GDPR/…).
4. **NER / PII detector** — find presence of PII or sensitive fields in code/config/text.
    - Task: mark fields/lines with PII (card number, SSN, auth token).
5. **Policy mapping / rule suggestion model** — optional: rank the most likely control(s) that an artifact violates.
6. **Explainability & Confidence Estimation** — produce confidence scores and highlight provenance lines.

5) Hugging Face model recommendations (task -> model)

Below are practical, well-known HF models that work offline or can be fine-tuned locally. Use the best available model for each job; if you need a stronger LLM for generation, consider hosted LLMs (OpenAI, Anthropic) or HF inference API for large models.

Embeddings (retriever)

- `sentence-transformers/all-mpnet-base-v2` — high-quality semantic embeddings, good general-purpose (compact + accurate).
- `sentence-transformers/all-MiniLM-L6-v2` — lower-cost, fast; good for prototype.
  Use these for indexing regulatory docs into a vector DB (Milvus, Pinecone, Weaviate, or FAISS).

Instruction-tuned / RAG generation

- Lightweight/opensource options:
  - `google/flan-t5-large` — good instruction following for generation tasks (smaller cost).
  - `bigscience/bloomz-3b` / `bigscience/bloomz-1b1` — instruction tuned multilingual options (be mindful of cost).
- If you need higher-quality natural-language reasoning and production readiness, consider using **OpenAI GPT-4** or **Claude** via API for the RAG generation step (better few-shot reasoning and safety); or HF hosted models like `meta-llama/Llama-2-13b` via HF Inference if you have infra.

Classification / multi-label (control mapping)

- `nlpaueb/legal-bert-base-uncased` — legal domain BERT, good starting point if available (often fine for regulation texts).
- `roberta-base` or `deepset/roberta-base-squad2` fine-tuned for multi-label compliance classification.
- `facebook/bart-large-mnli` — zero-shot classification for mapping text -> control categories (fast prototyping).

NER / PII detection

- `dbmdz/bert-large-cased-finetuned-conll03-english` — classic NER baseline.

- `Jean-Baptiste/roberta-large-ner-english` — strong NER model.
- For PII detection specifically, consider fine-tuning a model on custom PII-labelled dataset or using regex backed by ML.

Embedding + semantic code search (for remediation snippets)

- `microsoft/codebert-base` or `microsoft/codebert-base-mlm` — embeddings for code snippets.
- `Salesforce/codet5-base` — can generate code fix snippets (fine-tune for patches).

Other utilities

- Use `sentence-transformers/msmarco-distilbert-base-v4` for high-performance retrieval if you need search tuned for QA.

Note: HF model names are evolving. Treat the above as recommended starting points; check Hugging Face to pick recent improvements (e.g., newer Flan or Llama instruction-tuned models) if you want cutting-edge generation quality. For heavy generation or sensitive contexts, using a hosted, monitored LLM (OpenAI/Anthropic/HF Inference) often gives better safety/quality.

6) Data & dataset strategy

You will need:

1. **Regulatory corpus** — official texts (PCI DSS, FATF/GAFI guidance, AML/KYC guidelines, GDPR snippets) + internal policy docs. Convert these to canonical documents: ID, section, plain text, summary.
2. **Infra & code artifacts** — collect sample code, Dockerfiles, Terraform, K8s manifests, Helm charts from open-source fintech repos (sanitize).

Labeled examples — create training examples of artifact -> violated control(s) -> remediation. Label format:

```
{
  "artifact": "<K8s YAML snippet>",
  "violations": ["PCI-3.4.1", "Internal-NoPublicIngress"],
  "severity": "HIGH",
  "remediation_patch": "<kubectl patch or terraform fix>",
  "evidence_lines": [3,5]
}
```

3.
4. **PII examples** — files containing sample PII and non-PII to train detectors (use synthetic PII like `4111 1111 1111 1111` etc).

5. **Incident logs** — if you can, anonymize historical incidents to map patterns.

If you lack labeled data:

- **Synthetic generation**: programmatically inject issues into safe configs (e.g., turn off TLS, set env var `PASSWORD=...` in YAML, add wide RBAC clusterroleBinding). Use templates to create many variants and label automatically with the rule that injected.

- **Crowdsourced labeling**: split labeling tasks inside your team; store as JSONL for fine-tuning.

7) Concrete flow / pipeline (developer experience)

1. **Developer creates PR** with code/infra change.

2. **CI job runs**:

    - Static checks (SAST / dependency scanning).

    - IaC scanners (Checkov).

    - OPA policy checks (fast deterministic).

    - Run embedding-based retriever: create a compact artifact representation (text + code) -> get top-K regulatory clauses.

    - RAG generation: send artifact + retrieved clauses to LLM to create an explanation & remediation suggestion.

    - Classifier assigns severity and control IDs.

    - Aggregate results: PASS/FAIL + human-readable report.

3. **Output**:

    - Inline comment on PR with failing checks, explanation, provenance, remediation patch (optional), and “confidence”.

    - Option for maintainer to auto-apply a suggested patch (if policy permits) or open issue in tracking system.

4. **Policy store** (gitops): Rego policies managed in a `policies/` repo, versioned and tested by a policy CI job.

5. **Auditor view**: compliance dashboard lists all failing PRs, policy coverage, policy drift, and evidence logs.

8) MVP feature list (must-have)

- Repo + CI integration (GitHub Actions / GitLab): run on PR.

- Rego policies for 10-15 common infra checks (secrets in repo, public S3, K8s RBAC wide perms, insecure TLS, disabled CSP, plain-text DB credentials).

- Vector DB with embeddings of regulatory corpus.

- RAG generator that returns: matched clauses + explanation + one remediation suggestion.

- Simple multi-label classifier that maps artifact -> policy IDs (rule + ML hybrid).

- Dashboard with audit logs and ability to accept/reject remediation suggestions.

- Auto-remediation option for 2 low-risk categories (e.g., add `.dockerignore`, set `readOnlyRootFilesystem: true`) with PR patch created.

9) Evaluation metrics (how to prove it’s better)

- **Detection metrics**: precision / recall / F1 for each control (on held-out synthetic + real labeled data).

- **False Positive Rate**: counts of false alarms per 100 PRs.

- **Time-to-feedback**: latency from PR to report.
- **Remediation acceptance rate**: % of suggested remediations accepted by devs.
- **Coverage**: % of infra files mapped to at least one compliance control.
- **Business metric (proxy)**: hypothetical reduction in compliance review time (estimate).

10) 12-week plan (high-level milestones)

Week 1–2: Specification & dataset collection

- Gather regulations, choose 10 high-value policies to automate.
- Build repo templates and synthetic artifact generator.

Week 3–4: Core infra & static scanning

- CI integration; run OPA + Checkov + Trivy.
- Create dashboard skeleton + audit logging.

Week 5–7: Retriever + RAG prototype

- Index regulatory corpus with sentence-transformer embeddings and a vector DB.
- Integrate a small instruction-tuned LLM (Flan-T5) for generation.
- Implement simple RAG pipeline.

Week 8–9: Classifier & NER

- Train multi-label classifier to map artifact -> policy IDs.
- Add PII/NER detection.

Week 10: Auto-remediation + feedback loop

- Implement patch generation for simple fixes.
- Feedback button in UI and store labels.

Week 11: Robustness, metrics, and user testing

- Evaluate, tune thresholds, reduce FP.
- Add provenance and explainability visualizations.

Week 12: Finalize, prepare demo + report + video walkthrough

- Deploy on a small k8s cluster, prepare slides + live demo scenarios.

11) CI/CD integration examples

- **GitHub Actions**: Add job `compliance-check` that runs container `fincomply/checker:latest` — outputs a JSON summary. Use GitHub Checks API to annotate files and provide comments.
- **GitLab**: Run as pipeline stage; fail pipeline for HIGH severity violations.
- **Policy as code repo**: `policies/` with Rego + unit tests using `opa test` in pipeline.

12) Example Rego checks (ideas to implement quickly)

- `no_plaintext_secrets.rego`: detect `PASSWORD=` or `AWS_SECRET_ACCESS_KEY` in repo files.
- `no_wildcard_ingress.rego`: detect `host: "*"` or `0.0.0.0`.
- `rbac_minimum_privilege.rego`: forbid clusterRoleBindings that bind to `system:unauthenticated`.
- `tls_mandatory.rego`: enforce `tls.enabled == true` in helm values.

13) Where ML helps vs pure policy

- Use **policy-as-code (Rego)** for deterministic, high-confidence checks (low-latency, must-run in CI).
- Use **ML / LLM** for:
    - Interpreting ambiguous configs and mapping to regulation text.
    - Generating remediation suggestions and human-friendly explanations.
    - Prioritizing violations by learning from historical accept/reject feedback.
    - Semantic search of regulations (RAG).

14) Example HF usage patterns (practical)

- **Indexer**: run `all-mpnet-base-v2` to embed every regulation clause and store in vector DB.
- **Retriever**: fetch top-k matches for an artifact using cosine similarity.

Generator: feed artifact + top-k clauses into `flan-t5-large` (or a hosted GPT-4 if available) with prompt template:

```
 Input: <artifact snippet>
Clauses: <clause1; clause2; ...>
Task: Explain why this artifact may violate which clauses, and
suggest a code/infra fix (max 200 tokens). Provide confidence score.
```

-
- **Classifier**: fine-tune `nlpaueb/legal-bert-base-uncased` on your JSONL labelled mapping task for multi-label classification.

15) Implementation tips & hardening

- Start with deterministic Rego rules for critical checks — they block merges reliably.
- Use ML suggestions as advisory at first (don't automatically block until confidence is high).

- Cache embeddings and responses to reduce cost and latency.
- Add “policy test suite” (unit tests for Rego) and tie it to PR merging for policy changes.

16) Security, privacy & legal considerations

- Never send real customer PII to third-party LLMs without anonymization & legal approval.
- Keep vector DB and regulatory docs access controlled; logs must be auditable.
- Document data retention & encryption (at rest/in transit).
- For production, prefer on-prem or private hosted LLM inference for sensitive generation.

17) Useful open-source tools to integrate (quick reference)

- Rego / OPA (policy engine)
- Checkov / tfsec / KICS (IaC scanning)
- Trivy / Clair (container scanning)
- Semgrep (SAST)
- FAISS / Milvus / Weaviate / Pinecone (vector DB)
- sentence-transformers (embeddings)
- Hugging Face Transformers (models)
- Kubernetes + Helm + GitHub Actions (deployment & CI)
- Sentry / Prometheus / Grafana (observability)

18) Final concrete next steps (what you should do now)

1. Pick 10 high-value policies to start (e.g., PCI: store cardholder data rules; KYC: customer data minimization; infra: no public S3; secrets: no plaintext secrets).
2. Build or synthesize a dataset of 500–2000 artifact examples labeled with the above policies (synthetic injection helps).
3. Implement a minimal CI job: OPA + Checkov + Trivy and ensure it comments on PRs.
4. Index the regulatory corpus into a vector DB using `all-mpnet-base-v2`.
5. Implement a small RAG prototype using `flan-t5-large` for explanation and `roberta-base`/`legal-bert` for classification.
6. Run pilot on a sample repo and collect developer feedback. Tune thresholds & make auto-remediation opt-in.