

# 自動化がもたらす「有用性」と「品質」の両立



## 自動化による有用性の向上

手作業を極小化し、Azure Pipelinesによりプロセスを完全自動化。開発リソースを「作業」ではなく「価値創造」に集中させます。



## 品質のゲートキーパー

SonarQubeとJacocoによる自動検査をラインに組み込み、欠陥のあるコードの流出を機械的に阻止します。



## シンプルな運用

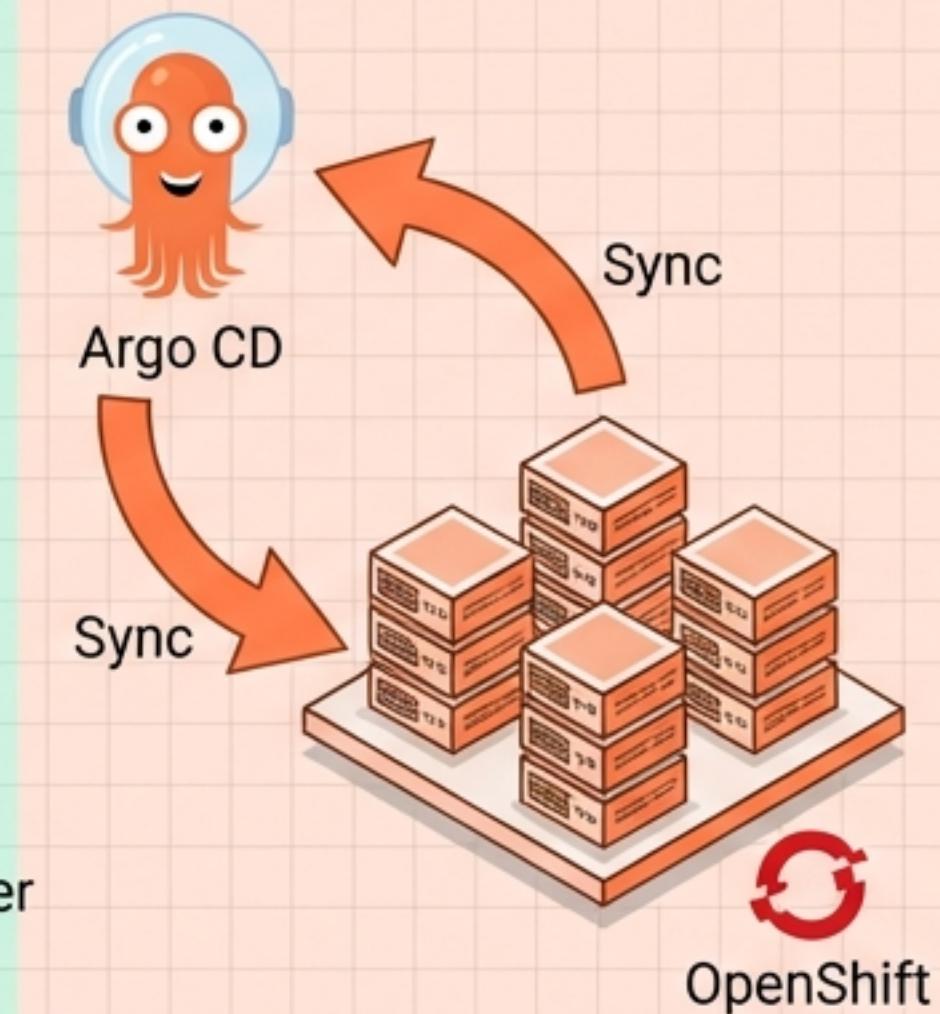
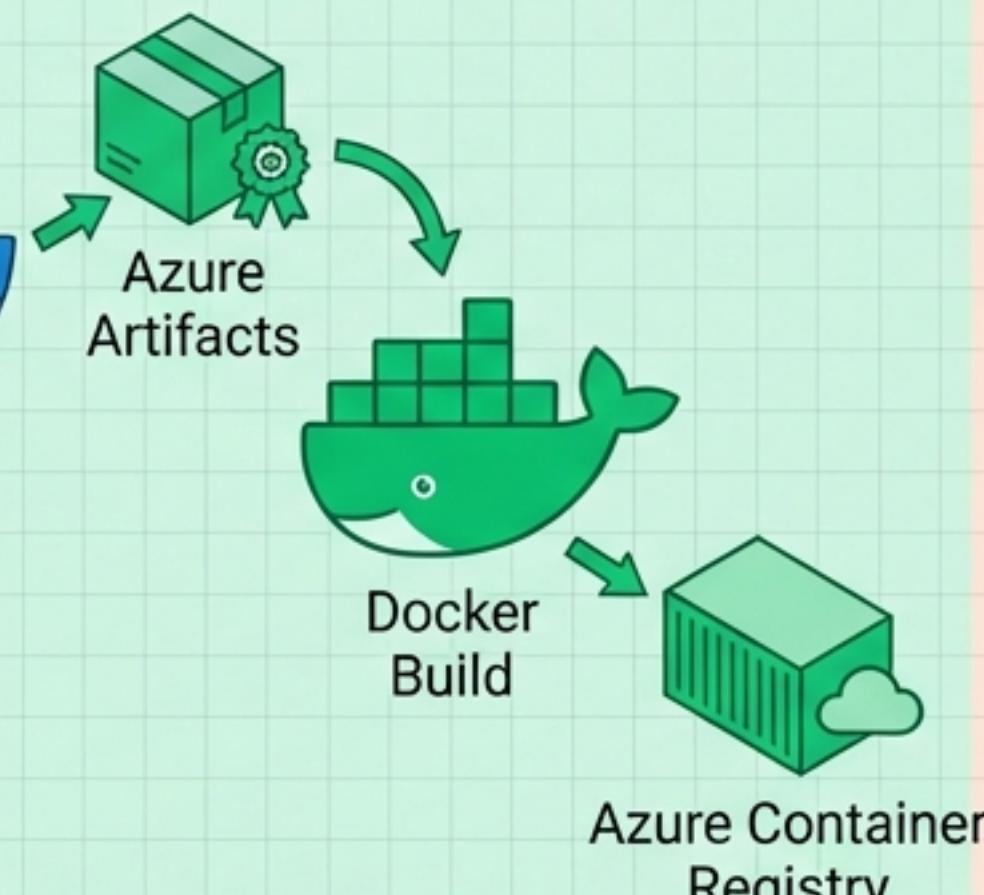
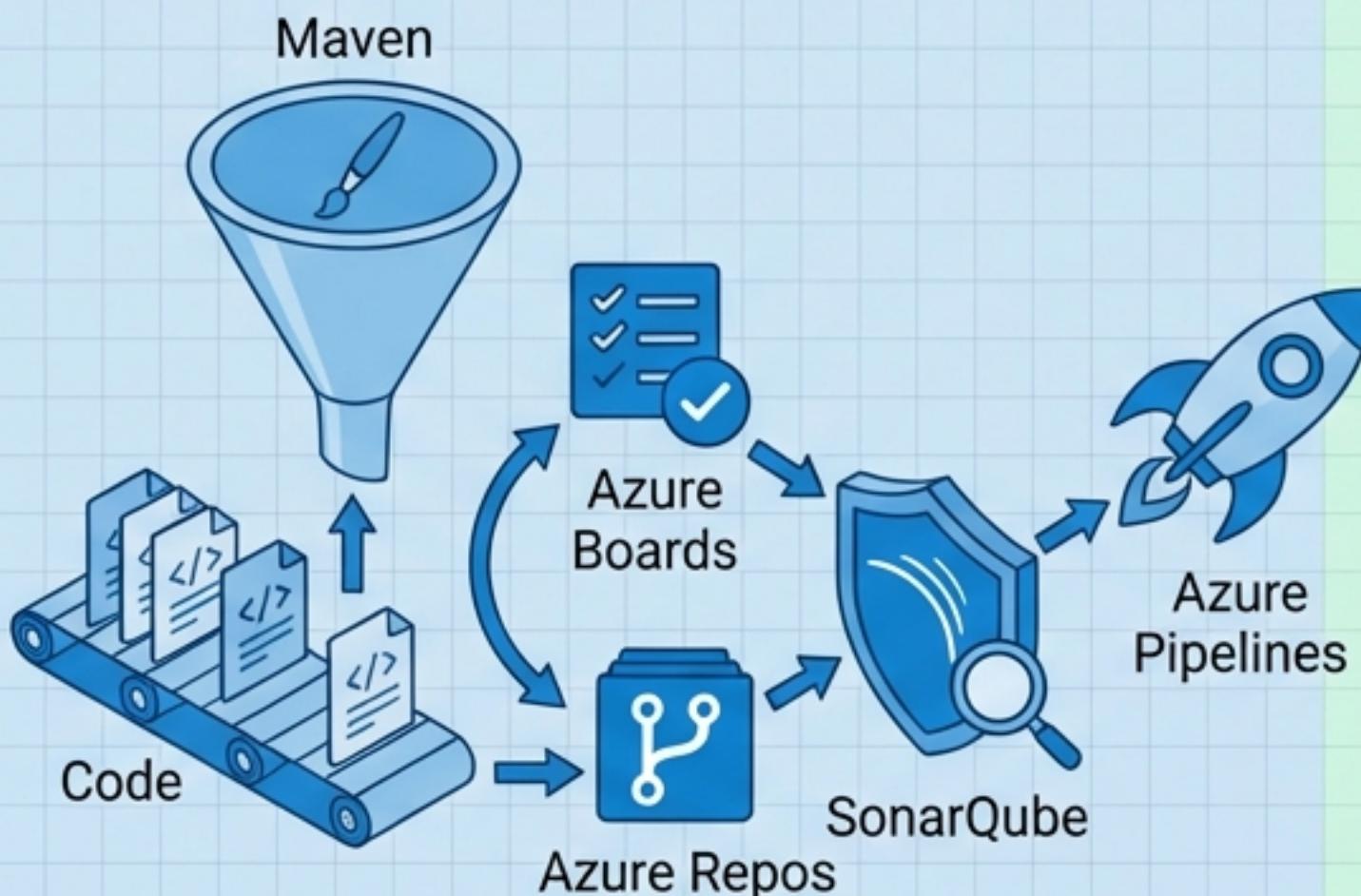
複雑なマージ作業やコンフリクト解消を回避するため、メインブランチを中心としたシンプルな開発フロー（トランクベース）を採用します。

# 全体構成：コードから本番環境への自動化された旅路

Phase 1: CI/CD (Build & Test) -  
Azure Repos & Pipelines  
(自動化された組立ライン)

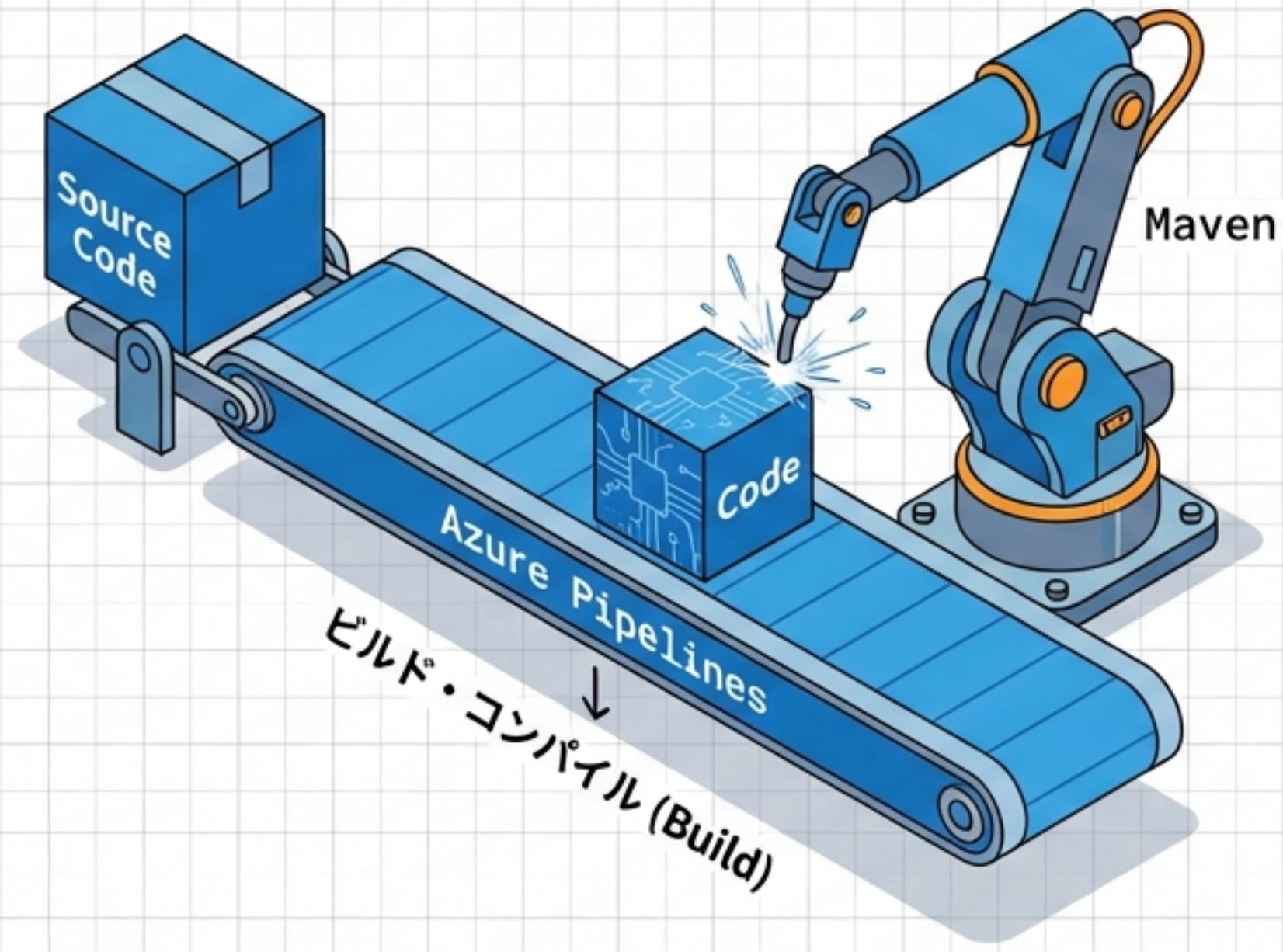
Phase 2: Container (Packaging) -  
Azure Artifacts & Docker  
(コンテナ生成と格納)

Phase 3: Release (GitOps) -  
Argo CD & OpenShift  
(継続的デリバリーと同期)



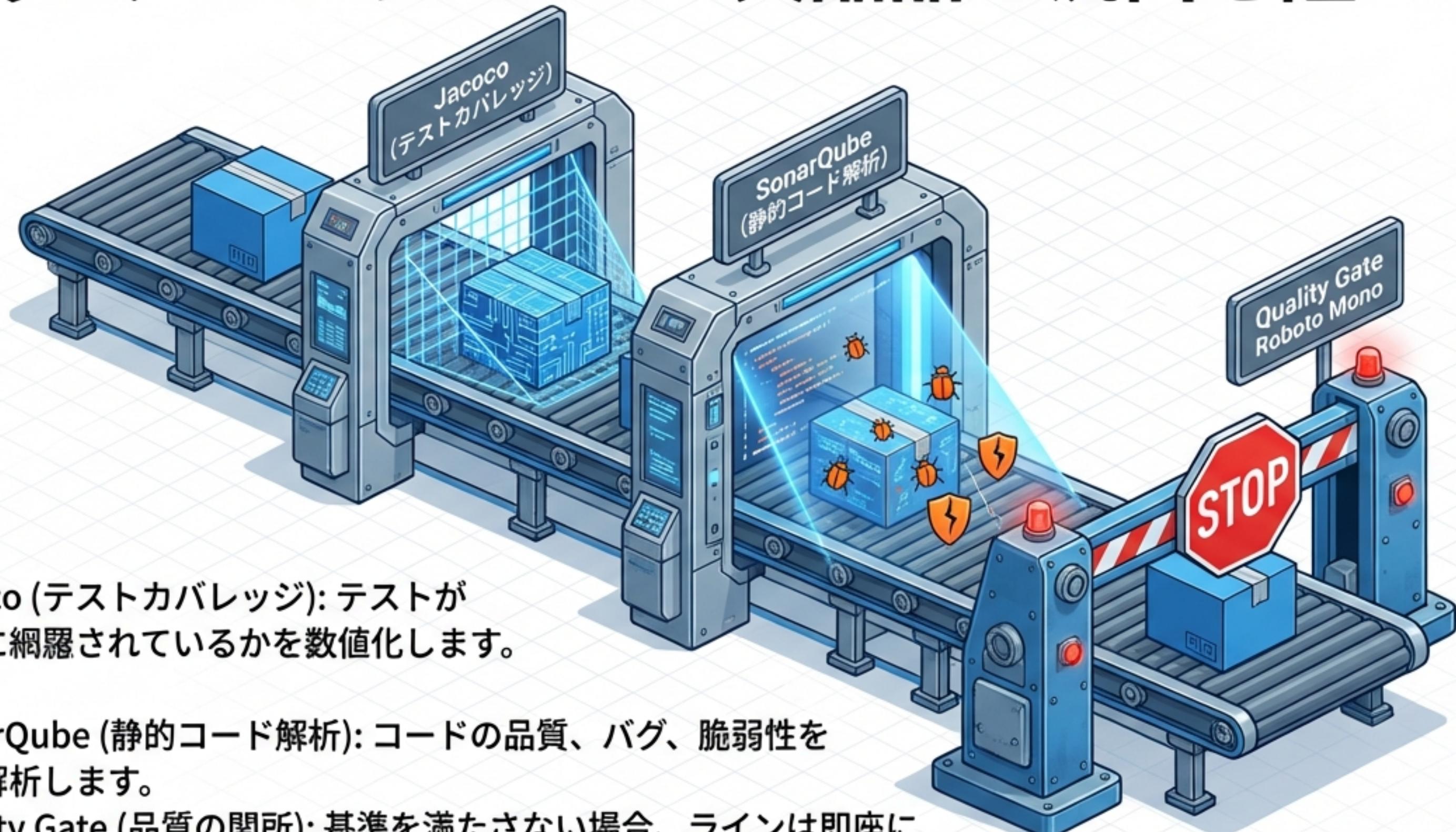
ソースコードから本番稼働まで、3つの自動化ステージを経てデリバリーされます。

# Phase 1：自動化された組立ライン (CI)



- Azure Repos: 開発者がコードをプッシュした瞬間、工場が稼働します。
- Azure Pipelines: 組立ラインを制御するエンジンです。
- Maven: ビルドロボットとして機能し、コンパイルからパッケージングまでを自動で行います。手動操作によるミスを排除し、誰が実行しても同じ結果が得られる「再現性」を担保します。

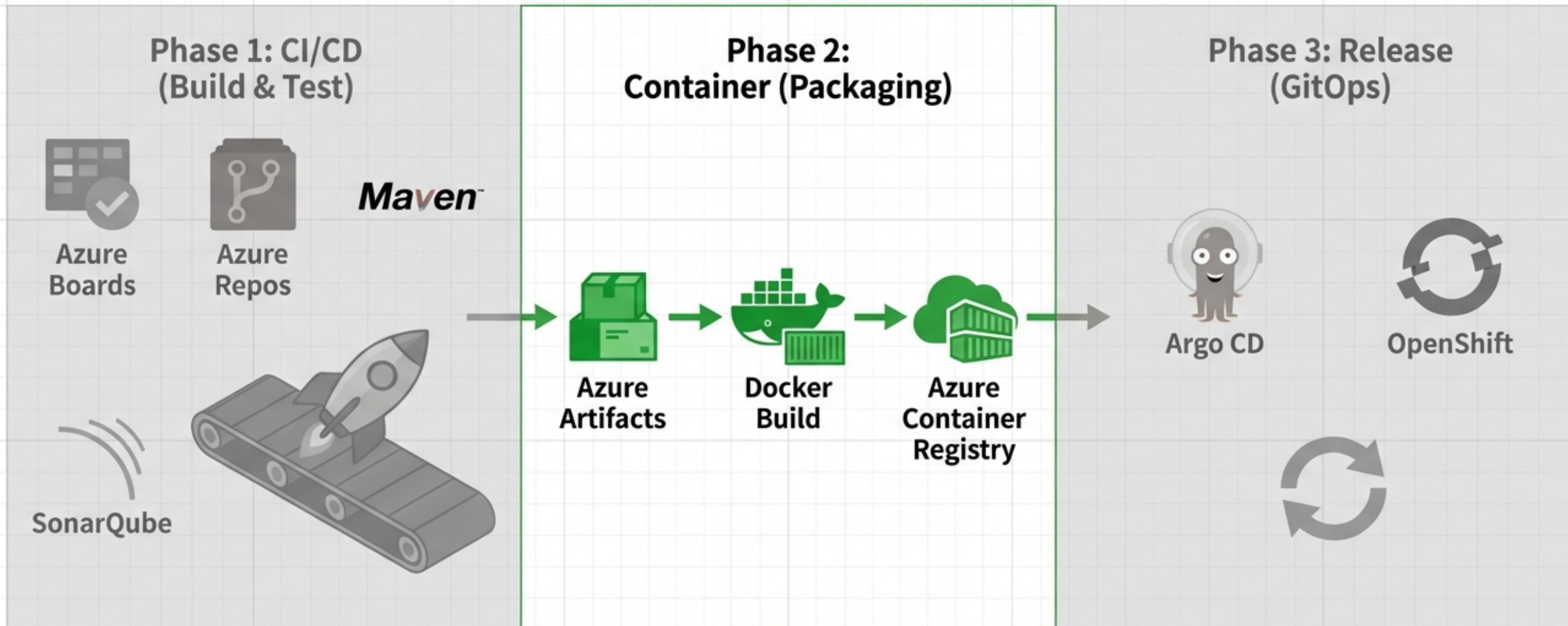
# 品質のゲートキーパー：欠陥品の流出を阻止する



- Jacoco (テストカバレッジ): テストが十分に網羅されているかを数値化します。
- SonarQube (静的コード解析): コードの品質、バグ、脆弱性を自動解析します。
- Quality Gate (品質の関所): 基準を満たさない場合、ラインは即座に停止します。不合格なアーティファクトが次工程へ進むことは物理的にありません。

# Phase 2：コンテナ生成と格納庫への登録

ビルドされた成果物を、どこでも動作する形式（コンテナ）にパッケージングする工程です。



# バージョン管理指針：「スナップショット」と「不变のリリース」

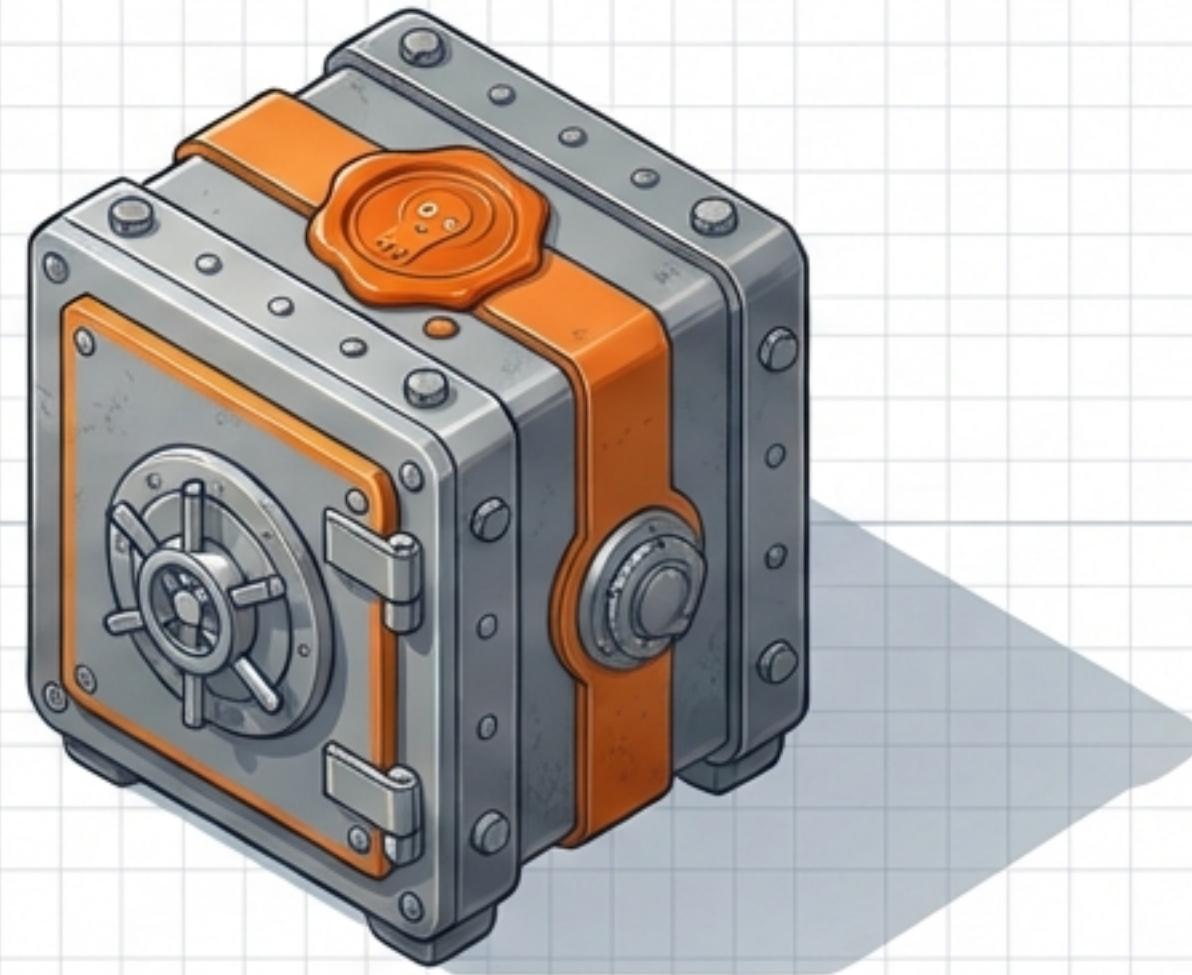
開発用：スナップショット (Snapshot)



ver 1.0.0-SNAPSHOT

日々上書きされる開発版。

本番用：イミュータブル・リリース (Release)



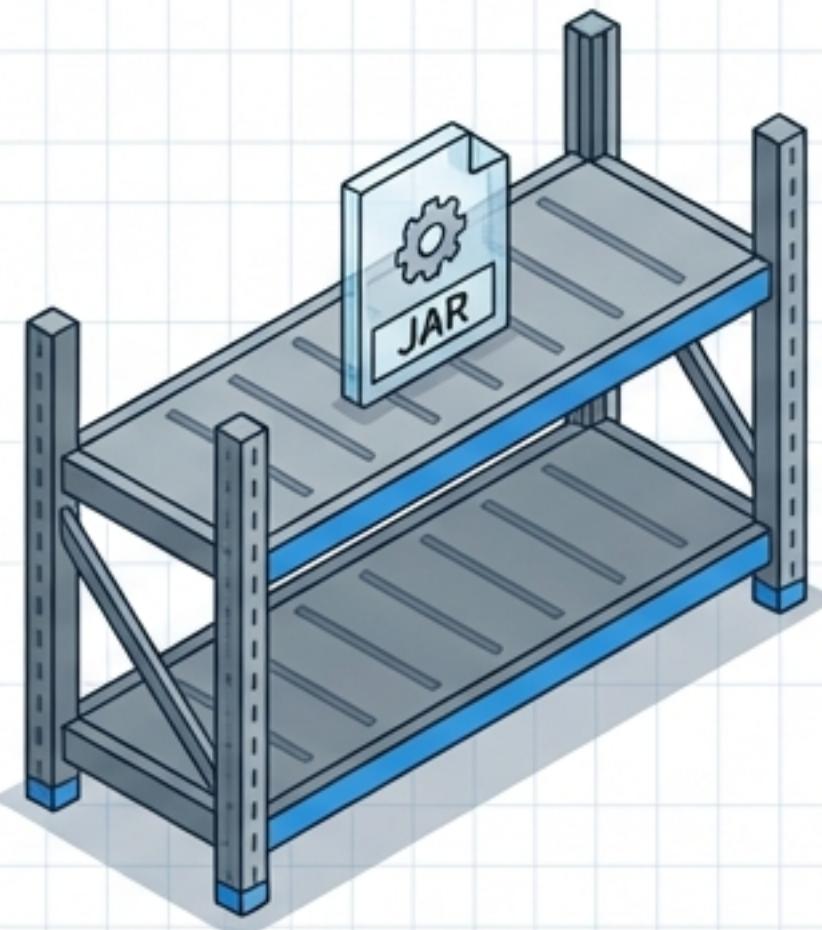
ver 1.0.0 (Immutable)

一度作成したら絶対に上書きされない不变のバージョン。

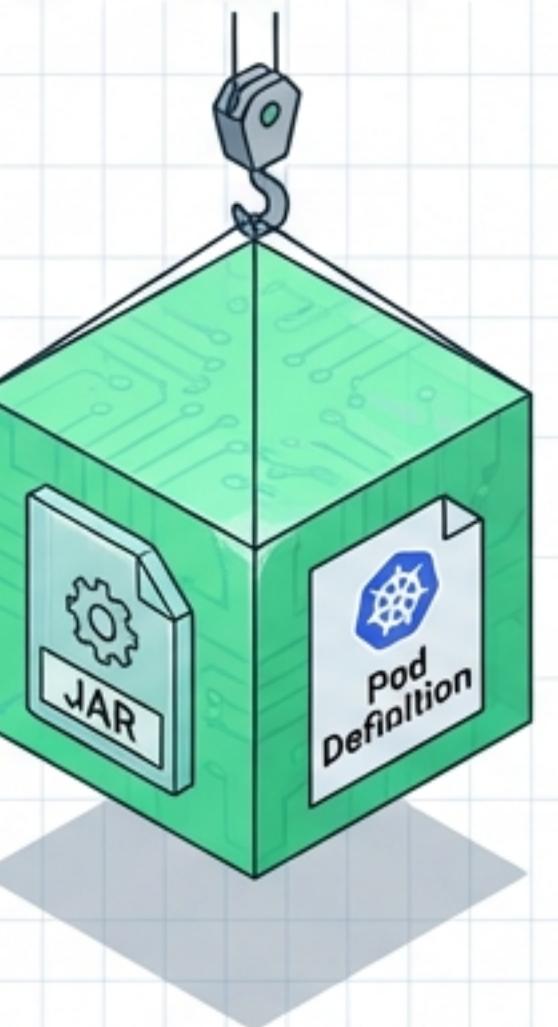
「不变のリリース」を作成することが、確実なロールバックと冪等性 (Idempotency) の条件となります。

# アプリケーションと環境の「完全なカプセル化」

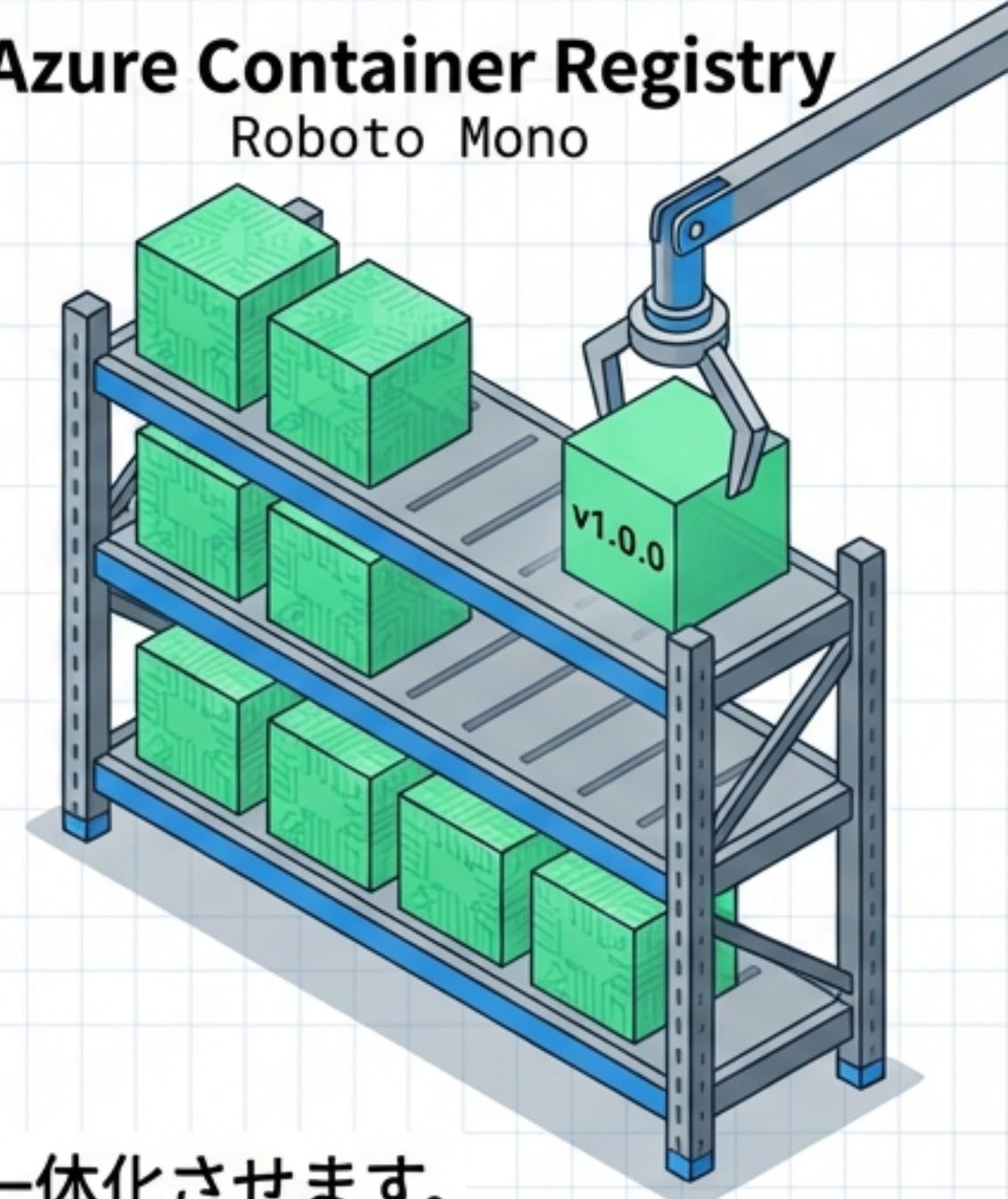
Azure Artifacts  
Roboto Mono



Docker Build



Azure Container Registry  
Roboto Mono

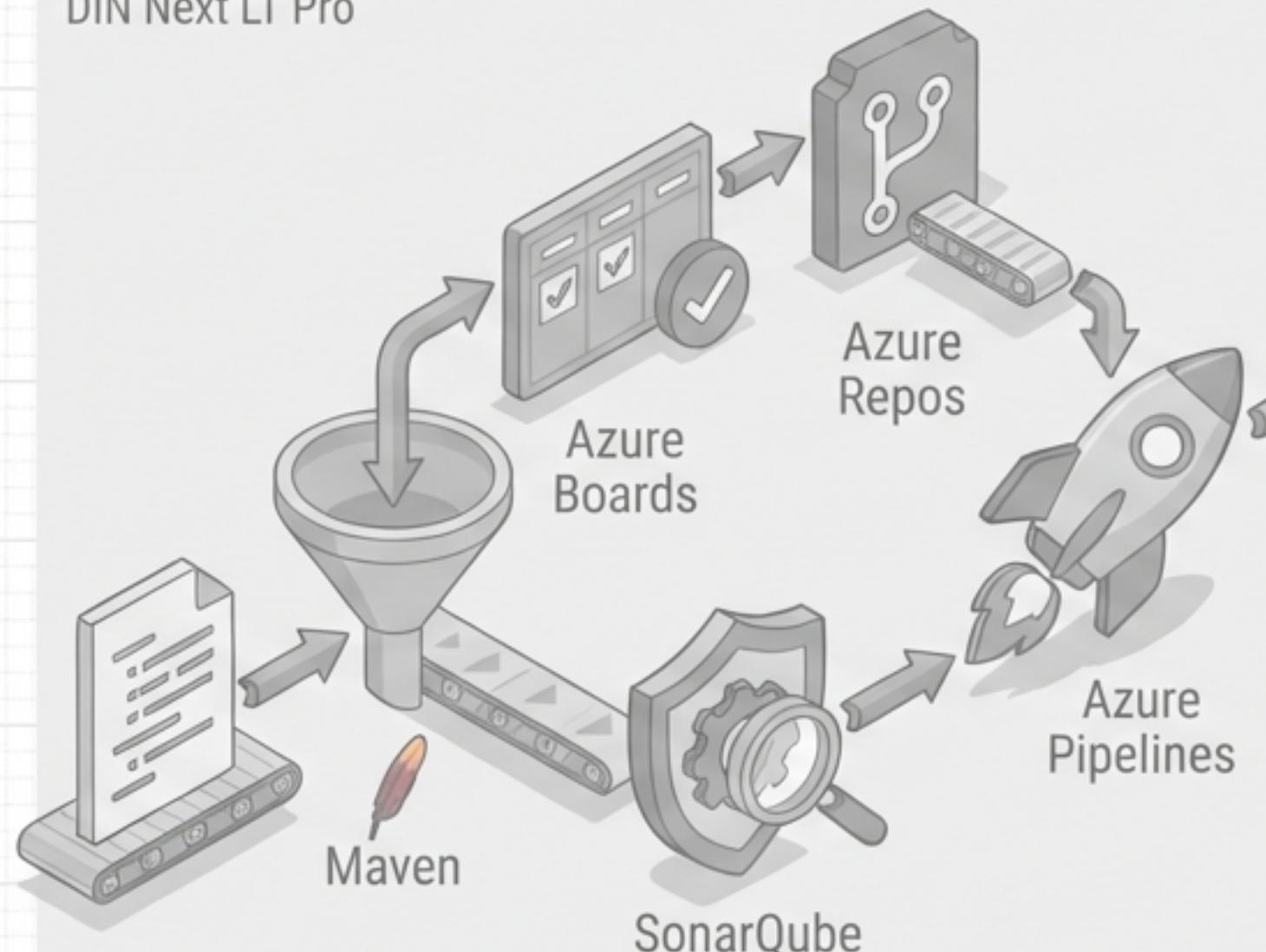


- Azure Artifacts: 生成されたJARファイルを管理。
  - Docker Build: アプリケーション(JAR)と動作環境(OS/Middleware)を一体化させます。
  - Azure Container Registry: 生成されたコンテナイメージを格納します。
- これにより、「開発環境では動いたが、本番環境では動かない」という環境起因の問題を根絶します。

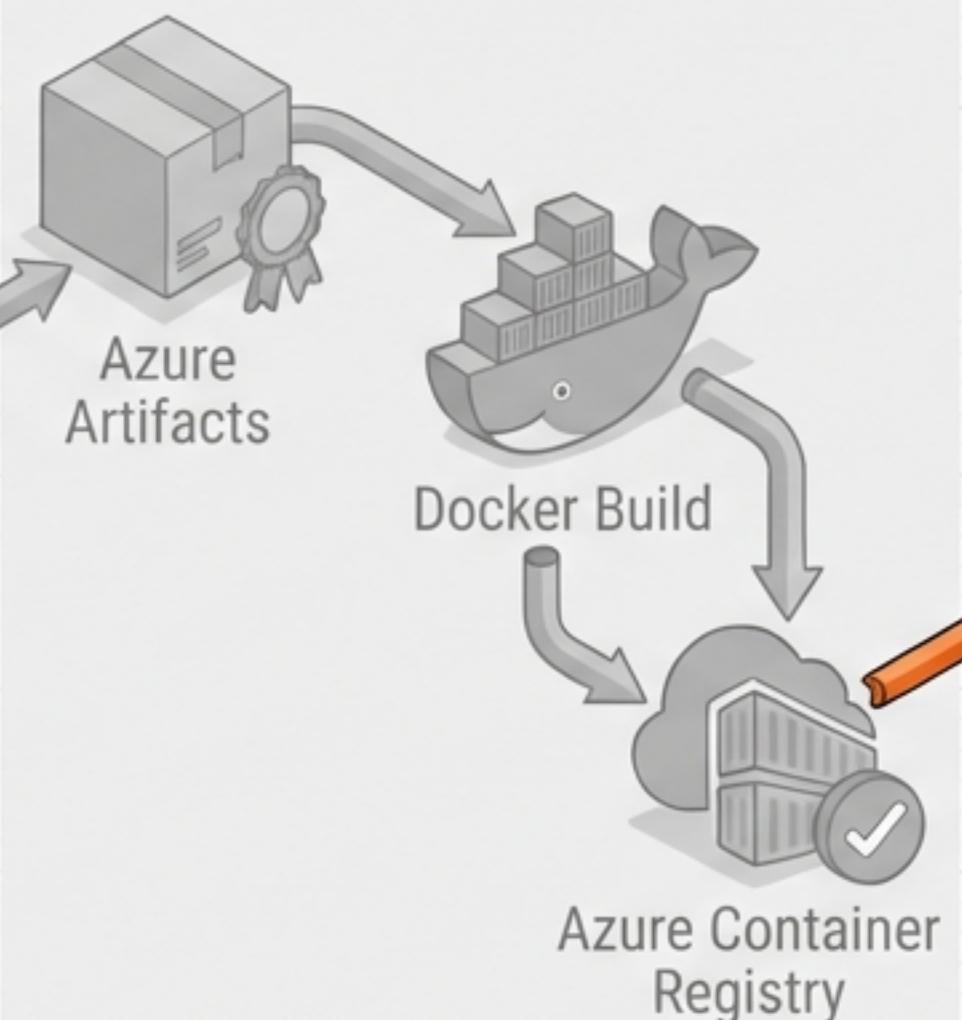
# Phase 3 : 本番と試験環境へのリリース

格納されたコンテナを、OpenShift環境へ安全かつ確実にデプロイする工程です。

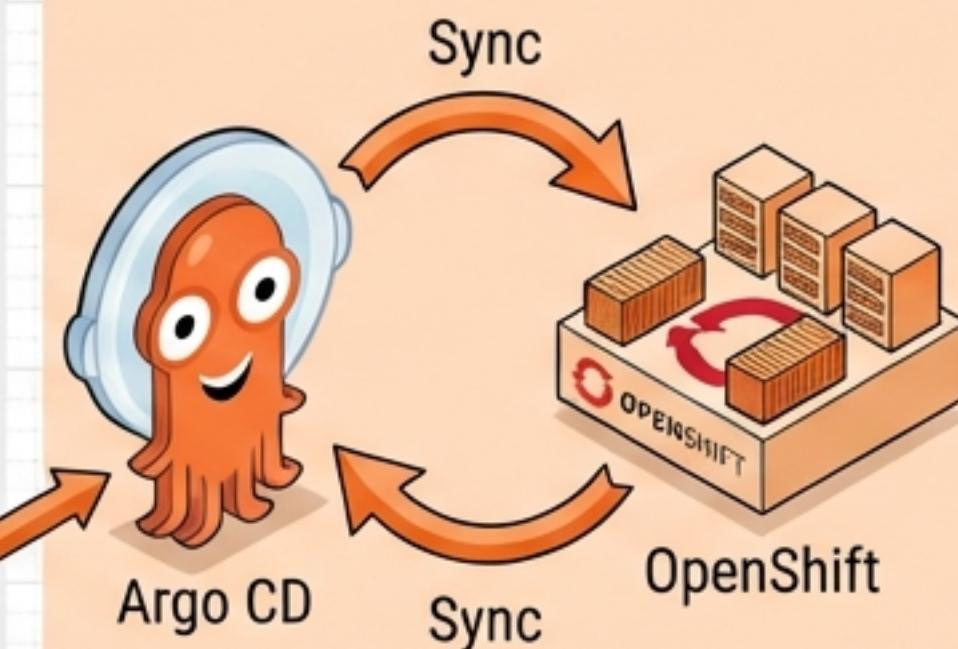
Phase 1: CI/CD (Build & Test)  
DIN Next LT Pro



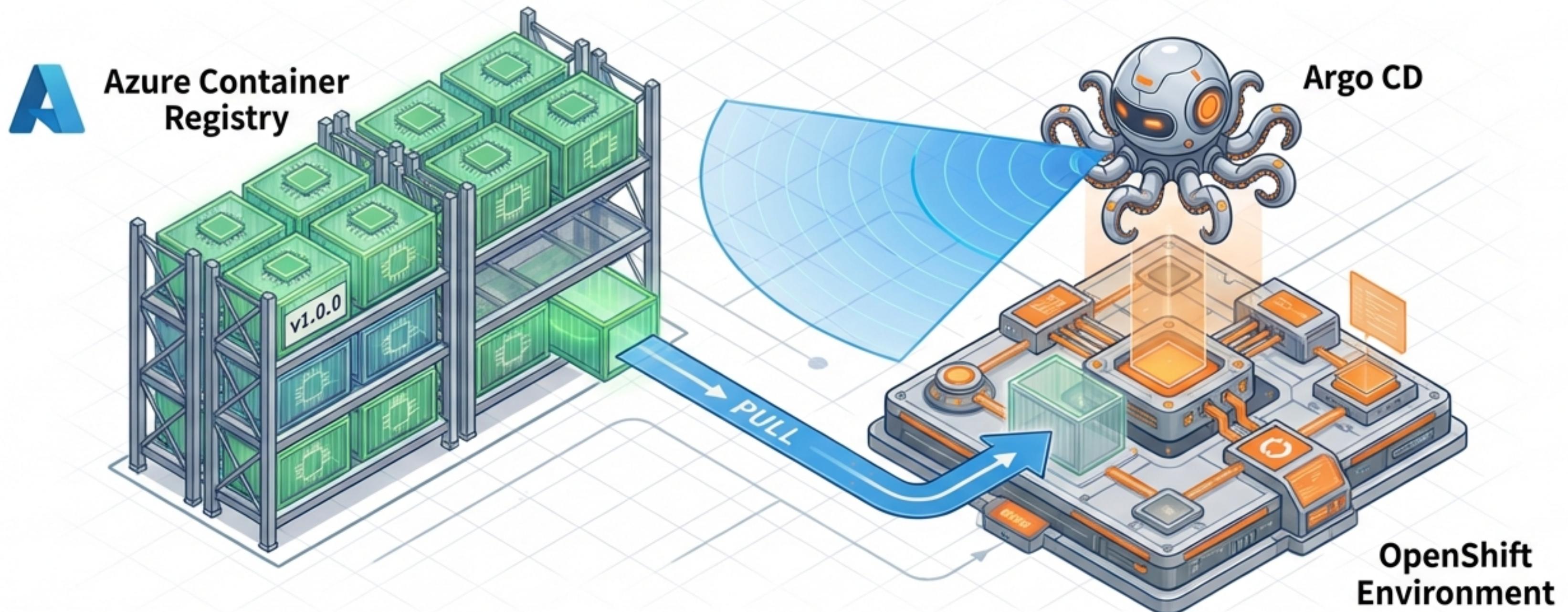
Phase 2: Container (Packaging)  
DIN Next LT Pro



Phase 3: Release (GitOps)  
DIN Next LT Pro



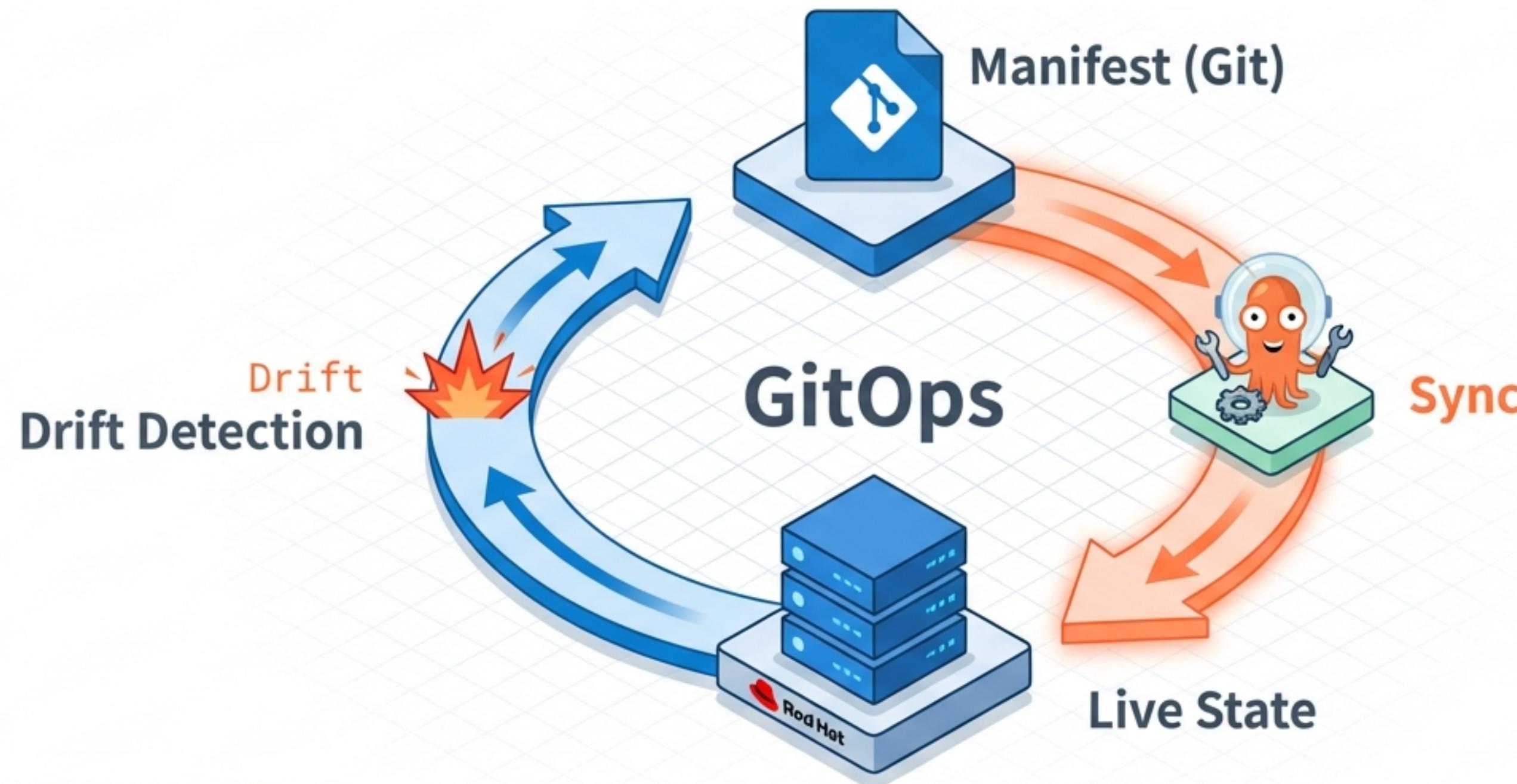
# GitOpsによる継続的デリバリー



- Traditional (Ansible等): 外部からトラックで配送する「Push型」。手順が複雑になります。
- Modern (Argo CD): OpenShift内のエージェントが、倉庫(Registry/Git)の変更を自動検知して同期する「Pull型」。

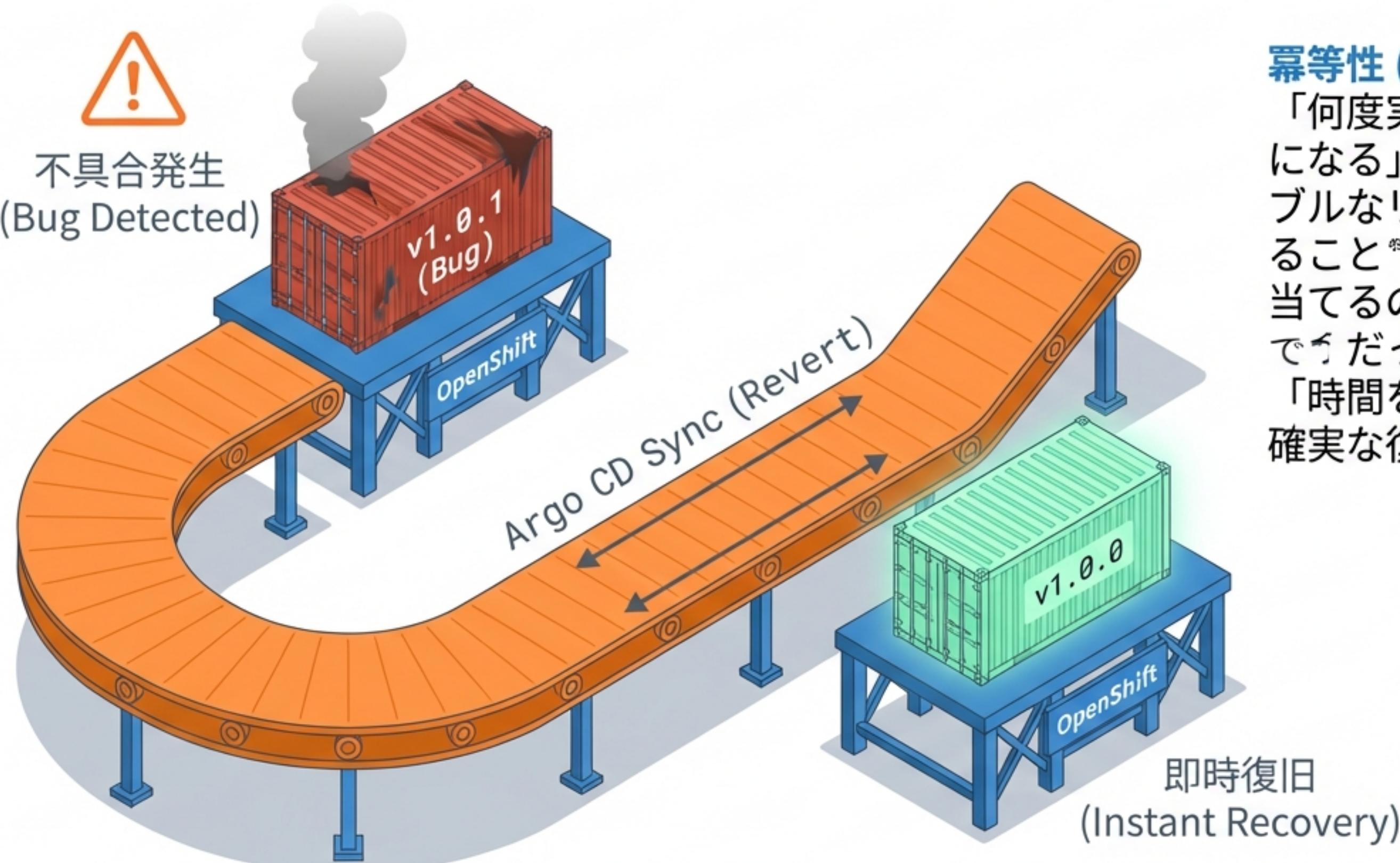
世界標準のアプローチ: 本番環境に常駐するArgo CDが、あるべき状態（マニフェスト）と現在の状態を常に監視し、差異が発生した瞬間、自動的に同期を実行します。

# 同期メカニズム：宣言的定義による自動デプロイ



- Single Source of Truth: Gitを「唯一の信頼できる情報源」として扱います。
- Drift Detection: 誰かが手動で設定を変更しても、Argo CDが即座に検知し、Gitで定義された「正解」の状態に強制的に修正します。  
これにより、環境の構成管理が自動化され、人為的な設定ミスが永続化することを防ぎます。

# 確実な復旧：GitOpsが保証する「幕等性」



**幕等性 (Idempotency):**  
「何度実行しても同じ結果になる」特性。イミュータブルなリリース版を使用すること<sup>※</sup>で、修正パッチを当てるのではなく、正常ただうだった→時間空間へ「時間を巻き戻す」ような確実な復旧を実現します。