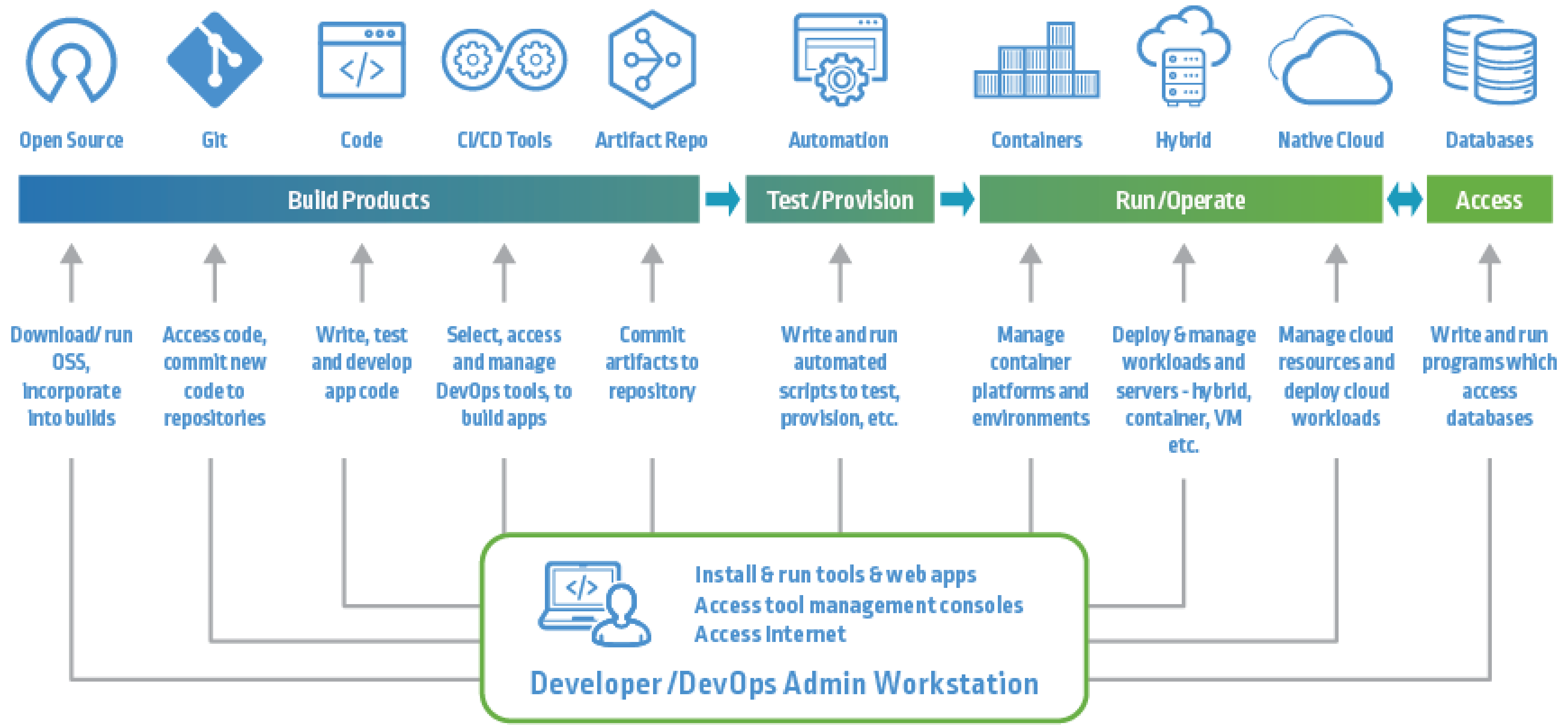
Introduction to CI/CD

Module 3: Pipelines

Pipelines

- A pipeline is a series of automated steps that take a software component from coding all the way to the operational environment



Open Source
Git
Code
CI/CD Tools
Artifact Repo
Automation
Containers
Hybrid
Native Cloud
Databases
Build Products
Test/Provision
Run/Operate
Access
Download/run OSS, incorporate into builds
Access code, commit new code to repositories
Write, test and develop app code
Select, access and manage DevOps tools, to build apps
Commit artifacts to repository
Write and run automated scripts to test, provision, etc.
Manage container platforms and environments
Deploy & manage workloads and servers - hybrid, container, VM etc.
Manage cloud resources and deploy cloud workloads
Write and run programs which access databases
Install & run tools & web apps
Access tool management consoles
Access Internet
Developer/DevOps Admin Workstation

Image Credit: https://www.cyberark.com/what-is/ci-cd-pipeline/

Pipelines

- Automation tools drive each step

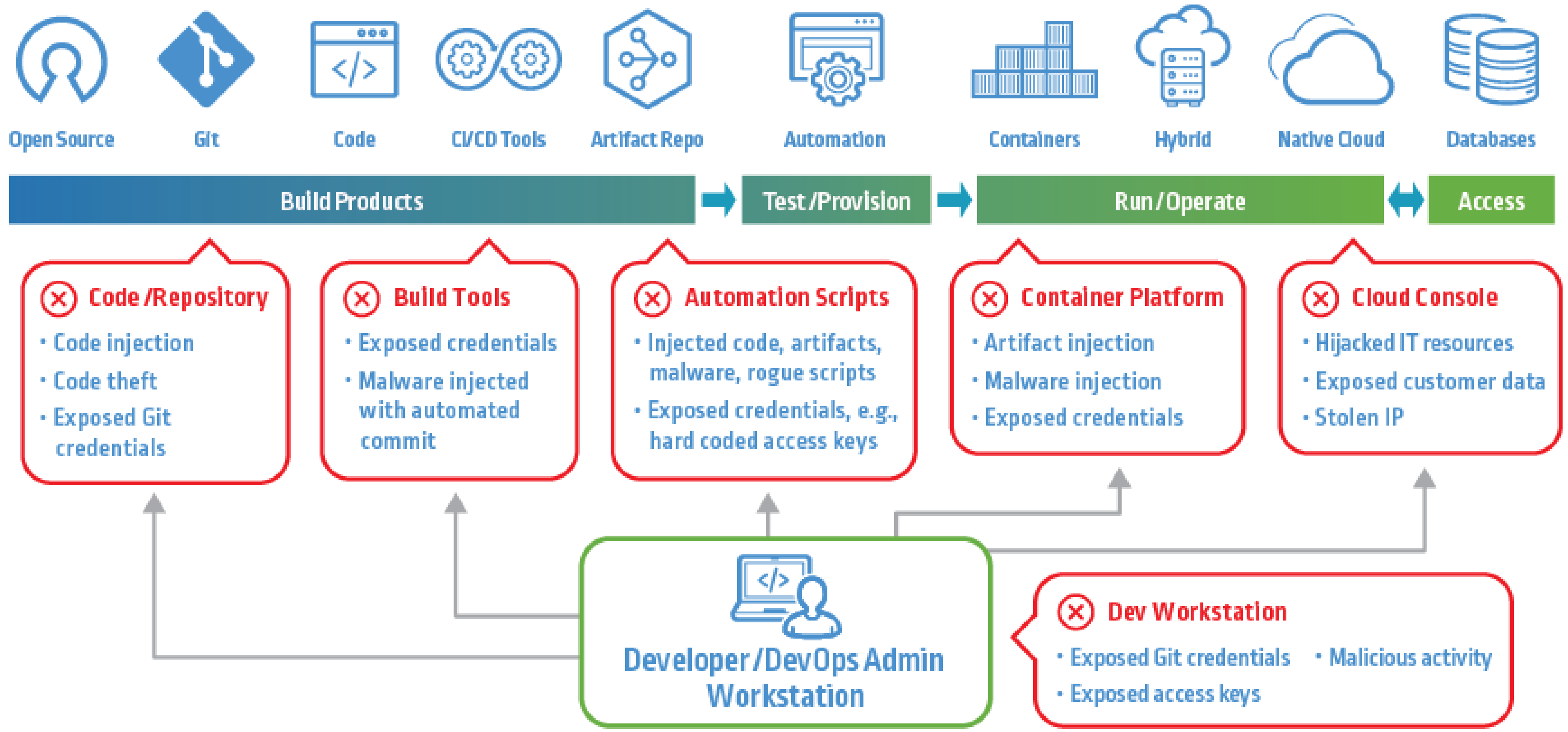
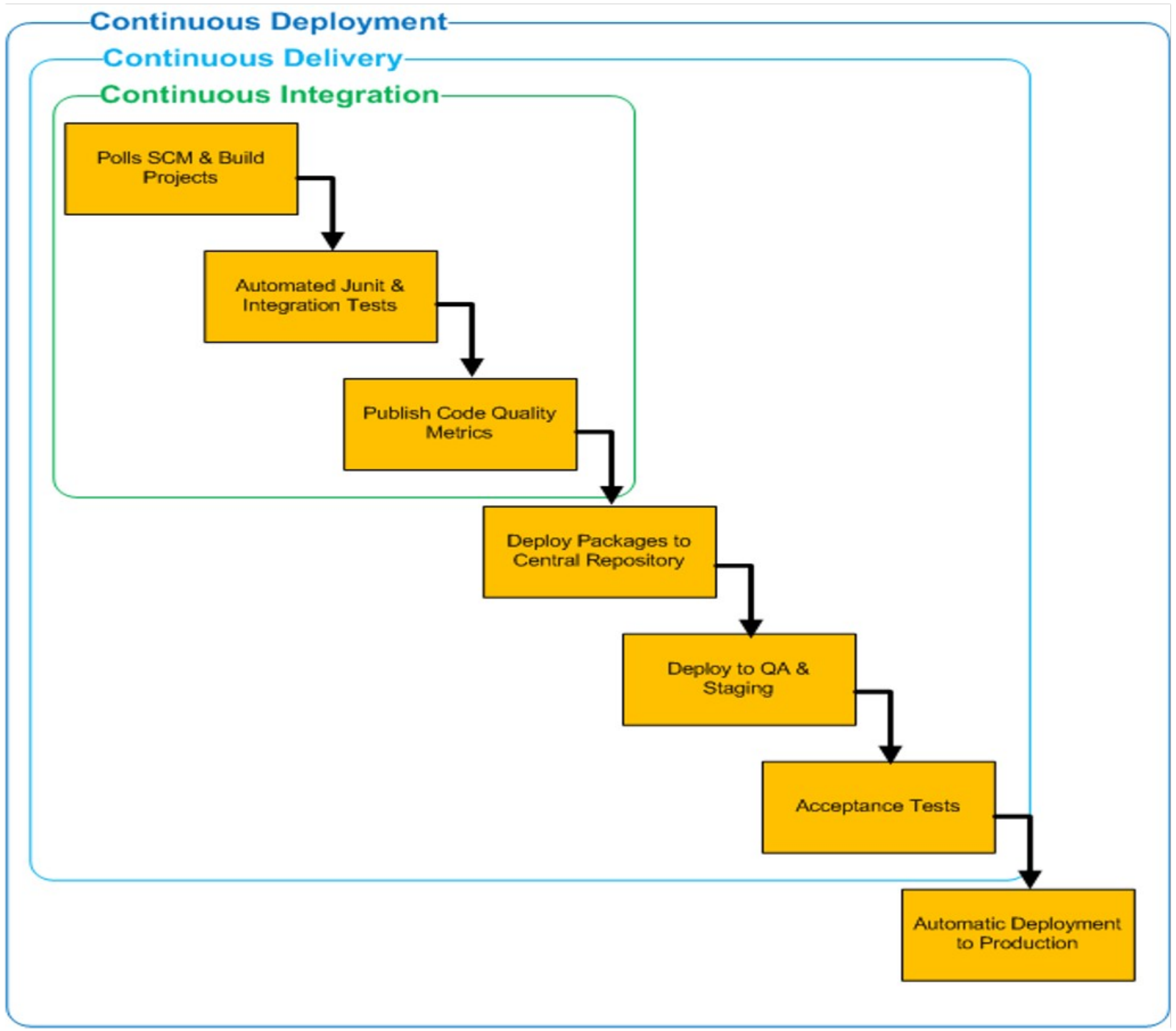


Image Credit: https://www.cyberark.com/what-is/ci-cd-pipeline/

Continuous Integration, Delivery and Deployment


Continuous Deployment
Continuous Delivery
Continuous Integration
Polls SCM & Build Projects
Automated Junit & Integration Tests
Publish Code Quality Metrics
Deploy Packages to Central Repository
Deploy to QA & Staging
Acceptance Tests
Automatic Deployment to Production

Continuous Integration (CI)

- CI is where members of a team integrate their work frequently
  - Usually each person integrates at least daily, leading to multiple integration per day
- Each integration is verified by an automated build (including test)
  - Intended to detect integration errors as quickly as possible
  - Goal is to merge and test the code continuously to catch issues early by automating integration process
- A CI project must have a reliable, repeatable, and automated build process involving no human intervention
  - CI Server (orchestration tool) is responsible for performing the integration tasks
- Automatic unit testing, static analysis and failing fast are core to CI

Continuous Integration Practices

- Single source repository for all developers
- Build automation
  - Every change to the integration branch should trigger a new build
  - Keep the builds fast and trackable
  - Make the builds self-testing
- Test the builds in production-like environment
  - Keep all verified releases in artifacts repository and available to everyone
- Publish coding metrics

Continuous Delivery (CD)

- CD is a natural extension of CI
  - Every change to the system has passed all the relevant automated tests and is ready to deploy in production
  - Team can release any version at the push of a button
  - Keep all verified releases in artifacts repository and available to everyone
- But the deployment to production is not automatic
  - The goal of CD is to put business owners in the control of scheduling of the software releases
  - The decision to release is a governance decision, not a technical one

Continuous Deployment (also CD)

- Continuous Deployment adds automatic deployment to end users in the Continuous Delivery process
  - Continuous Deployment automatically deploys every successful build directly into production
  - Deploying the build to production as soon as it passes the automated and UAT tests
- Continuous Deployment is not appropriate for many business scenarios
  - Business Owners prefer more predictable release cycles as opposed to arbitrary deployments

CI/CD as a General Pipeline Pattern

- A pipeline is an automated, ordered set of stages that transform an input artifact into a validated, deployable output.
- This idea shows up everywhere:
  - Software delivery (CI/CD)
  - Data engineering (ETL/ELT)
  - Machine learning (MLOps)
  - Infrastructure (IaC pipelines)
  - Analytics & reporting

CI/CD Pipelines (Baseline Reference)

- Source → Build → Test → Package → Deploy → Monitor
- Concrete tools
  - GitHub Actions / GitLab CI / Jenkins
  - Maven / Gradle / npm
  - Docker / Helm
  - Kubernetes / VM / PaaS

CI/CD Pipelines (Baseline Reference)

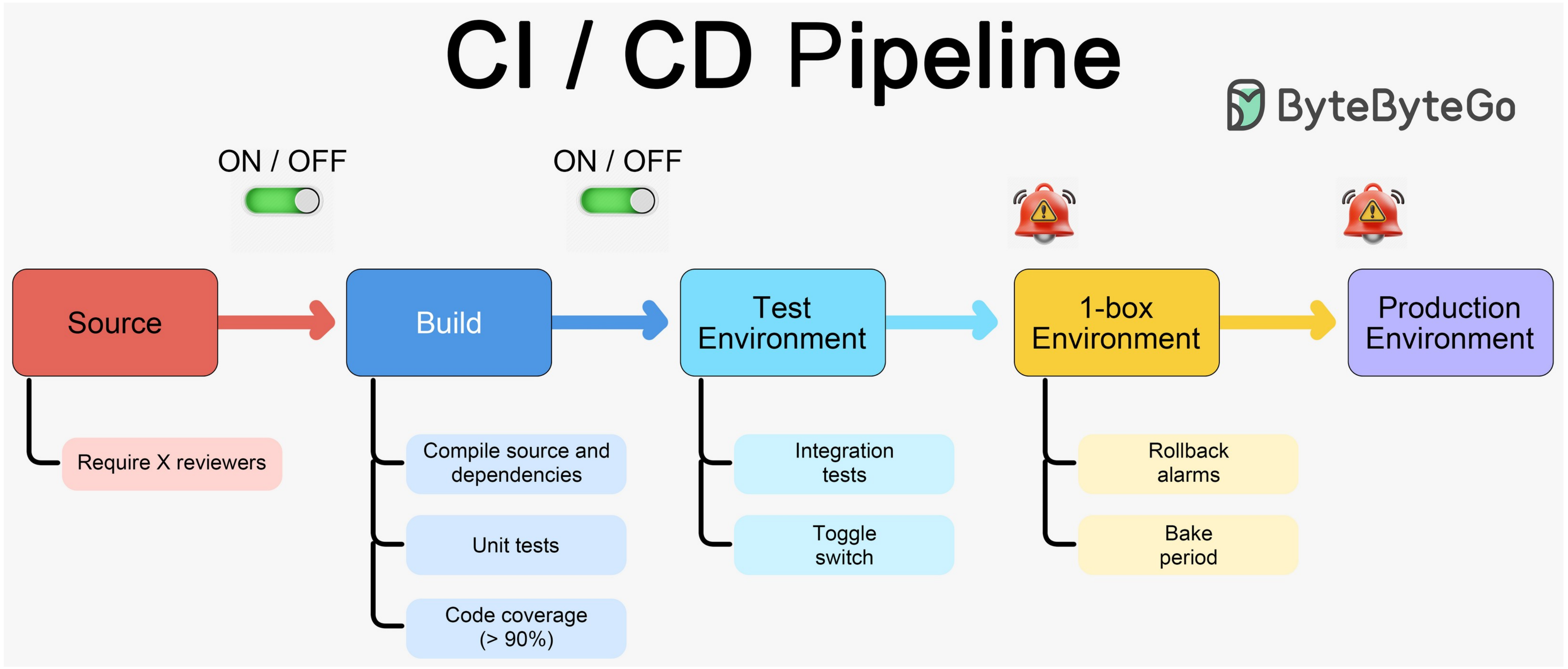


Image Credit: https://blog.bytebytego.com/p/a-crash-course-in-cicd

Data Pipelines (Data Engineering)

- Ingest → Validate → Transform → Load → Quality Check → Publish
  - Data pipelines apply the same CI/CD logic, but the “artifact” is data, not code
- Example
  - Source: Kafka topic / API / database
  - Transform: Spark / Flink / dbt
  - Load: Data warehouse (BigQuery, Snowflake)
  - Validation: Great Expectations
  - Orchestration: Airflow / Dagster
- CI/CD benefits
  - Versioned SQL and transformation logic
  - Automated schema validation
  - Test datasets before production loads
  - Promotion of pipelines from dev → prod

Data Pipelines (Data Engineering)

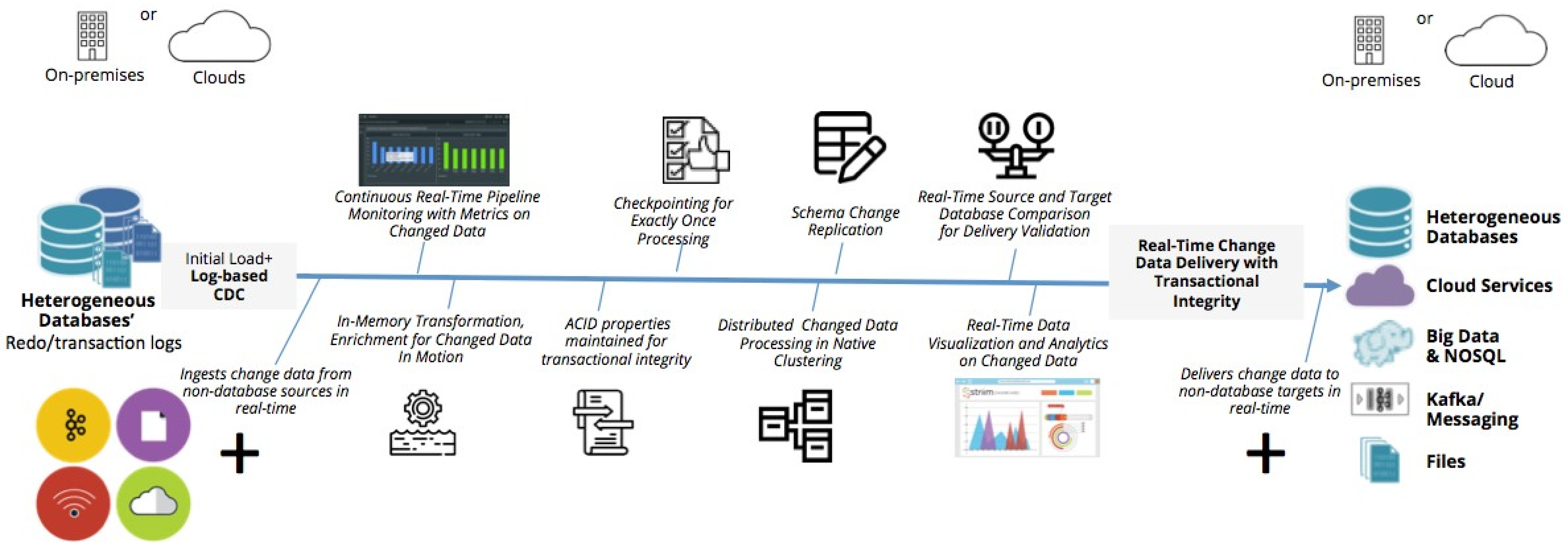

On-premises
or
Clouds
Heterogeneous Databases'
Redo/transaction logs
Initial Load+
Log-based CDC
Ingests change data from non-database sources in real-time
Continuous Real-Time Pipeline Monitoring with Metrics on Changed Data
In-Memory Transformation, Enrichment for Changed Data In Motion
Checkpointing for Exactly Once Processing
ACID properties maintained for transactional integrity
Schema Change Replication
Distributed Changed Data Processing in Native Clustering
Real-Time Source and Target Database Comparison for Delivery Validation
Real-Time Data Visualization and Analytics on Changed Data
Real-Time Change Data Delivery with Transactional Integrity
Delivers change data to non-database targets in real-time
On-premises
or
Cloud
Heterogeneous Databases
Cloud Services
Big Data & NOSQL
Kafka/ Messaging
Files

Image Credit: https://www.striim.com/blog/what-is-a-data-pipeline-and-must-have-features-of-modern-data-pipelines/

MLOps Pipelines (Machine Learning)

- Data → Feature Engineering → Train → Evaluate → Register → Deploy → Monitor
  - MLOps pipelines extend CI/CD to models, which are probabilistic artifacts
- Example
  - Training: TensorFlow / PyTorch
  - Tracking: MLflow
  - Model Registry: MLflow / SageMaker
  - Deployment: REST endpoint / batch job
  - Monitoring: Drift detection, accuracy decay

MLOps Pipelines (Machine Learning)

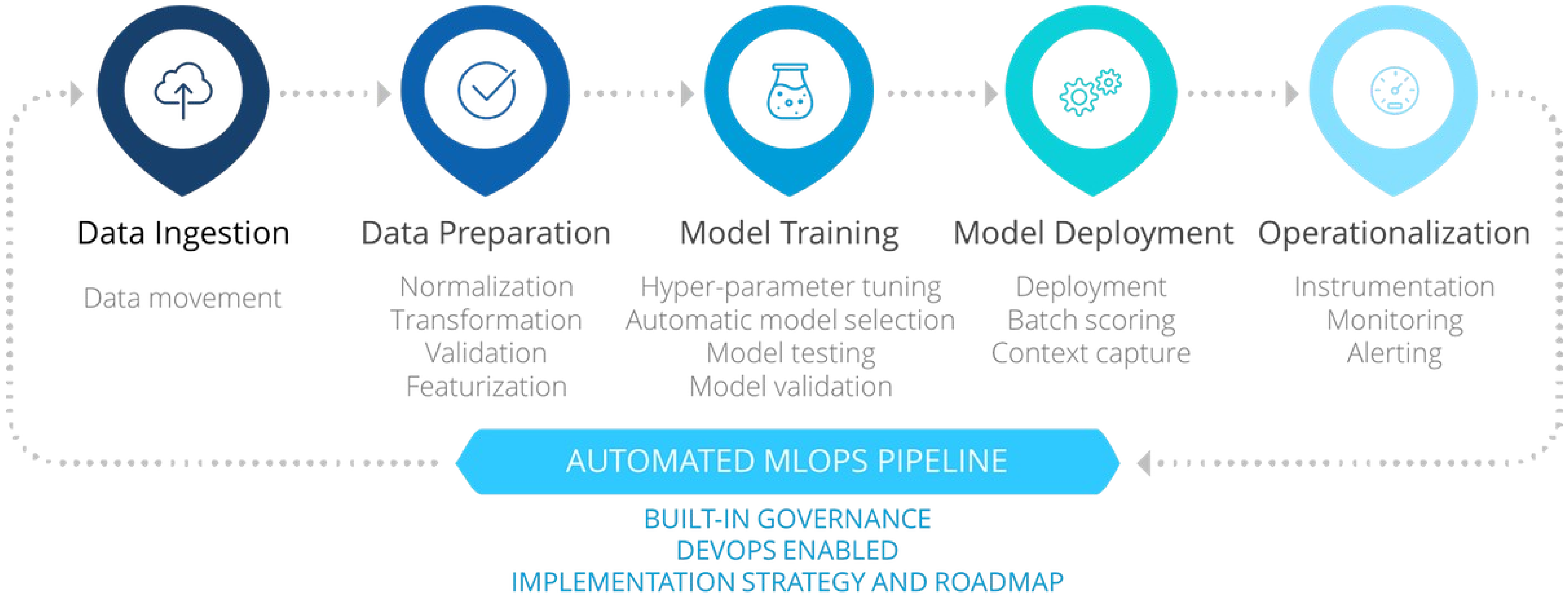


Image Credit: https://www.infracloud.io/blogs/introduction-to-mlops//

Infrastructure Pipelines (IaC)

- Define → Validate → Plan → Apply → Verify
  - Infrastructure is treated as a versioned artifact, validated and promoted like code
- Example
  - Terraform / Pulumi
  - Automated security scans
  - Environment promotion (dev → test → prod)

Infrastructure Pipelines (IaC)

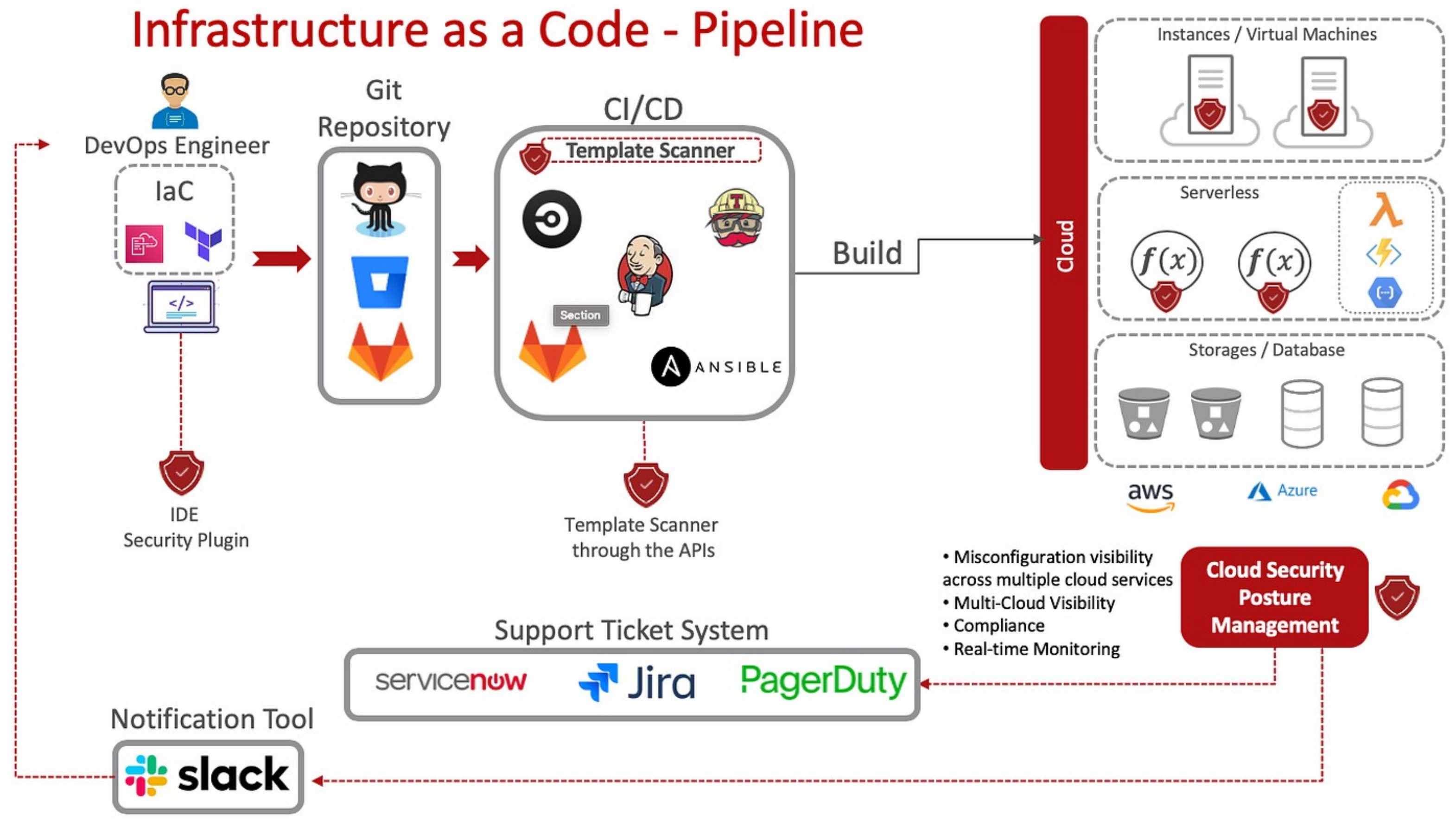


Image Credit: https://medium.com/swlh/putting-security-into-the-iac-pipeline-4de98f88ad24

Tool Chains

- Generally some form of repository tool is used for the CI environment
  - Usually git based like GitHub or GitLab
- Various packaging and build tools are used throughout the process
  - These are generally dependent on the programming tools
  - For Java, we usually see Maven and Gradle for example
- Automated testing tools are used throughout the pipeline
  - Unit testing, Cucumber/Behave integration testing
  - Code quality tools like SonarQube
- The whole process is managed by an orchestration tool
  - Commonly Jenkins is used as a standalone tool
  - GitLab and GitHub have orchestration capabilities that are often used

Questions