Platform Engineering UNIVERSITY

GITOPS FOR PLATFORM ENGINEERING

GitOps Fundamentals and Core Principles

MODULE 02

History and Origin

History and Origin

GitOps—and the ideas behind it—are older than you might think

1. Desired state: defined in cookbooks (recipes)
2. Chef client: runs at intervals on each target node (server / VM)
3. Drift detection: the client compares actual state with the cookbooks
4. Remediation: If drift is detected, the client reconciles the node to the desired state

Configuration Management (CM)

Chef, a configuration management tool, was announced in 2009 (formerly Opscode)

History and Origin – Today

In 2017, GitOps was defined based on four principles

Just four principles—no more, no less.

Declarative, Versioned and Immutable, Pull-based and Continuously reconciled!

You name it, you own it!

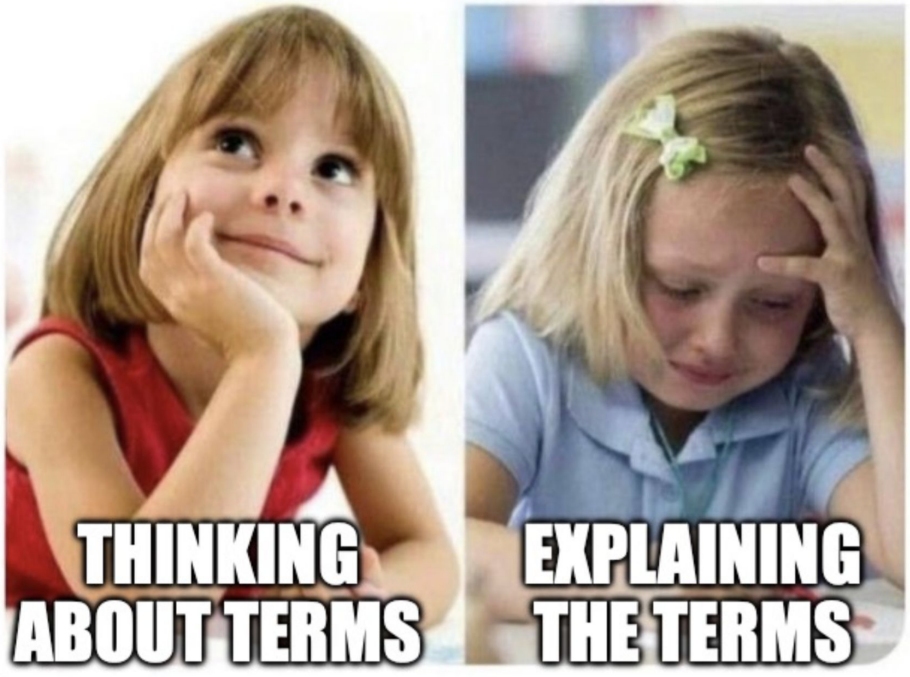
Alexis Richardson introduced the term **GitOps in 2017** while he was at Weaveworks. Today, he is the CEO of **ConfigHub**.

Terminology

Terminology

- Current State
- Desired State
- State Store
- Drift
- Reconciliation
- Declarative
- Feedback Loop
- Rollback
- Continuous Delivery
- Continuous Deployment

- Code
- Infrastructure as Code (IaC)
- Infrastructure as Data (IaD)
- Configuration as Code (CaC)
- Configuration as Data (CaD)



Terminology– Part 1/3

Current State is what is *actually running* right now.

People often treat Current State as "the truth".

Desired State is the **normative definition** of what the system *should* be.

It is also the **intent,** not a runtime artifact.

State Store is the place where the **authoritative representation** of **intent** lives.

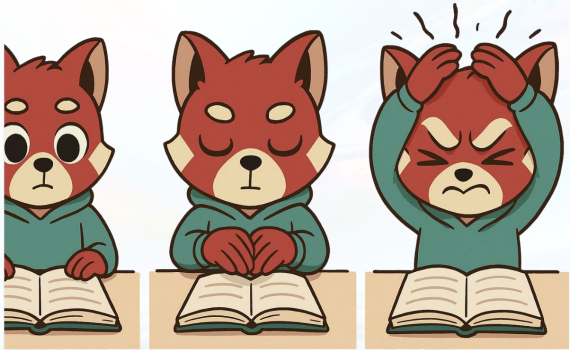
Git is typically the primary state store.

Drift is the **difference between Desired State** and **Current State.**

Drift is not "failure" by default — it's a **signal** that the system is out of alignment.

Reconciliation is the heart of GitOps: **converging Current State toward Desired State.**

Reconciliation ≠ "deployment". Deployment is just one kind of reconciliation action.



Terminology – Part 2/3

Declarative describe *what* you want, not *how* to do it.

Declarative ≠ simple.

Feedback Loop is the process where a cluster-resident **reconciler** continuously compares the **desired state** (defined in Git) with the **current state** of the cluster and automatically corrects any detected **drift** to enforce consistency.

Rollback is usually **not a special operational procedure**.

Rollback = revert a commit or move a tag. **Reconciliation** applies the previous Desired State again.

Continuous Delivery means, that **code** is always ready for production but requires a **manual** approval step before deployment.

Continuous Deployment means, that **code** is automatically deployed to production **without human intervention** once all tests pass.

Terminology – Part 3/3

Code means executable or evaluatable definitions that transform inputs into a resulting state.

Infrastructure as Code (IaC) means, that the infrastructure is created and changed by **executing code** that contains provisioning logic.

Infrastructure as Data (IaD) means, that the Infrastructure is described as plain declarative state data that controllers continuously reconcile.

Configuration as Code (CaC) is generated through templates and logic rather than expressed directly.

Configuration as Data (CaD) is stored as **plain**, declarative data without execution logic (code).

Terminology – Part 3/3

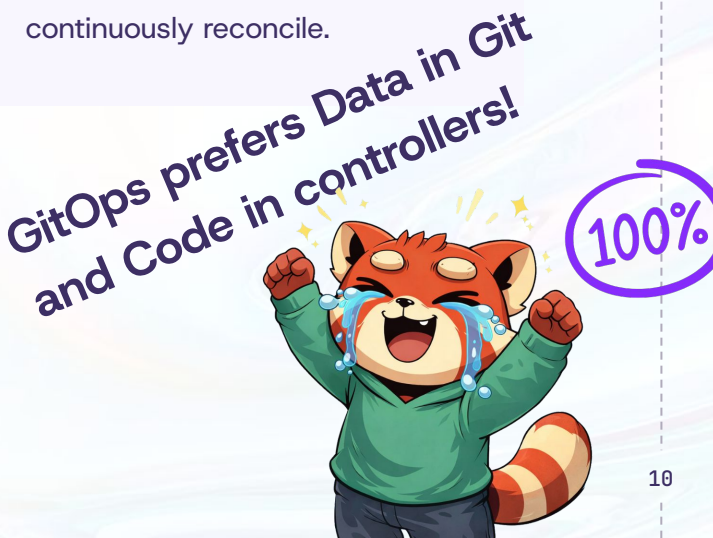
Code means executable or evaluatable definitions that transform inputs into a resulting state.

Infrastructure as Code (IaC) means, that the infrastructure is created and changed by **executing code** that contains provisioning logic.

Infrastructure as Data (IaD) means, that the Infrastructure is described as plain declarative state data that controllers continuously reconcile.

Configuration as Code (CaC) is generated through templates and logic rather than expressed directly.

Configuration as Data (CaD) is stored as **plain**, declarative data without execution logic (code).



Terminology – Example 3/3 (I)

- Infrastructure as Code (IaC)

```
variable "project_id" {

  type    = string
  default = "3432-1212..."

}

You, 2 seconds ago | 1 author (You)
module "ske_cluster" {
  source     = "../modules/ske-cluster"
  project_id = var.project_id
  name       = "ske-${substr(var.name, 0, 3)}"

}
```

```
project_id = var.project_id
name       = "ske-${substr(var.name, 0, 3)}"
```

- Infrastructure as Data (IaD)

```
resource "stackit_ske_cluster" "main" {
  project_id = "3432-1212..."
  name       = "ske-prod"
}
```

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: allow-app
spec:
  podSelector:
    matchLabels:
      app: backend
```

YAML

- Configuration as Code (CaC)

templates/ci.yaml

```
{{ if eq true .Values.letsencrypt }}
apiVersion: cert-manager.io/v1
kind: ClusterIssuer
metadata:
  name: {{ .Values.clusterIssuer.name }}
spec:
  acme:
    email: {{ .Values.clusterIssuer.email }}
    server: {{ .Values.clusterIssuer.server }}
    privateKeySecretRef:
      name: {{ .Values.clusterIssuer.name }}
    solvers:
    - http01:
        ingress:
          ingressClassName: {{ .V...
{{ end }}
```

HELM

values.yaml

```
cert-manager:
  global:
    installCRDs: true
clusterIssuer:
  email: artem@lajko.dev
  name: letsencrypt-staging
  server: https://acme-staging...
```

- Configuration as Data (CaD)

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: workload
spec:
  replicas: 3
```

YAML

Terminology – Example 3/3 (II)

- Infrastructure as Code (IaC)

```
variable "project_id" {

  type    = string
  default = "3432-1212..."

}

You, 2 seconds ago | 1 author (You)
module "ske_cluster" {
  source     = "../modules/ske-cluster"
  project_id = var.project_id
  name       = "ske-${substr(var.name, 0, 3)}"

}
```

```
project_id = var.project_id
name       = "ske-${substr(var.name, 0, 3)}"
```

- Infrastructure as Data (IaD)

```
resource "stackit_ske_cluster" "main" {
  project_id = "3432-1212..."
  name       = "ske-prod"
}
```

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: allow-app
spec:
  podSelector:
    matchLabels:
      app: backend
```

YAML

- Configuration as Code (CaC)

templates/ci.yaml

```
{{ if eq true .Values.letsencrypt }}
apiVersion: cert-manager.io/v1
kind: ClusterIssuer
metadata:
  name: {{ .Values.clusterIssuer.name }}
spec:
  acme:
    email: {{ .Values.clusterIssuer.email }}
    server: {{ .Values.clusterIssuer.server }}
    privateKeySecretRef:
      name: {{ .Values.clusterIssuer.name }}
    solvers:
    - http01:
        ingress:
          ingressClassName: {{ .V...
{{ end }}
```

HELM

values.yaml

```
cert-manager:
  global:
    installCRDs: true
clusterIssuer:
  email: artem@lajko.dev
  name: letsencrypt-staging
  server: https://acme-staging...
```

- Configuration as Data (CaD)

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: workload
spec:
  replicas: 3
```

YAML

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: allow-app
spec:
  podSelector:
    matchLabels:
      app: backend
```

YAML

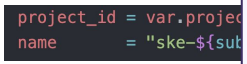
Terminology – Example 3/3 (III)

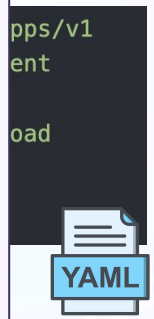
- Infrast uration as Co a (CaD)

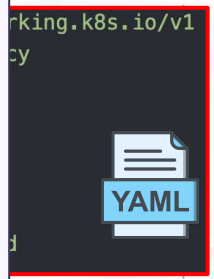
There is **no 100% fixed** definition of CaD and IaD. The meaning depends on context and can change over time.

In general, you can say that **IaD is a subset of CaD** (a semantic specialization).

- IaD = platform scope
- CaD = workload scope







The Four Principals (What is GitOps)

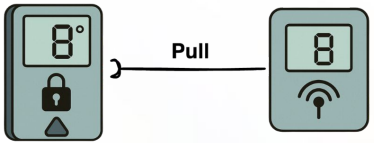
The Four Principles (What is GitOps)

Remember this.

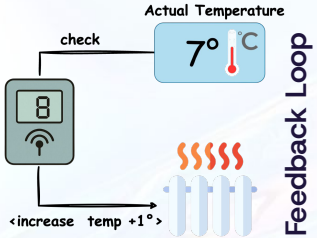
Declarative

Versioned and Immutable

Pull-Based



Continuously Reconciled



Just four simple principles—no more, no less.

What is GitOps Not

A replacement for DevOps!

A replacement for CI/CD (maybe for CD, but not CI)

A replacement for Platform Engineering!

GitOps is not

ClickOps or ad-hoc kubectl changes

A replacement for IaC tools

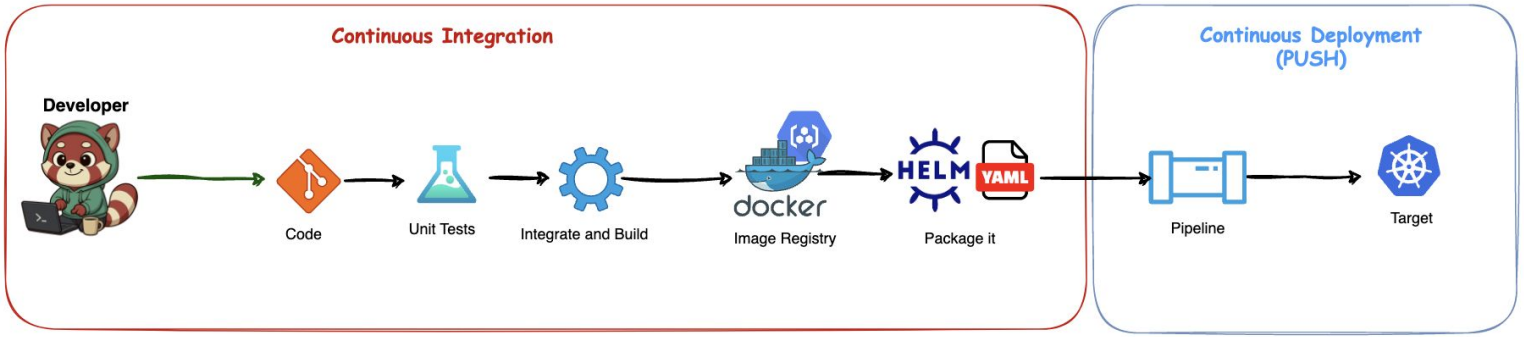
Manual, Git-based deployments

CI/CD with Git slapped on top

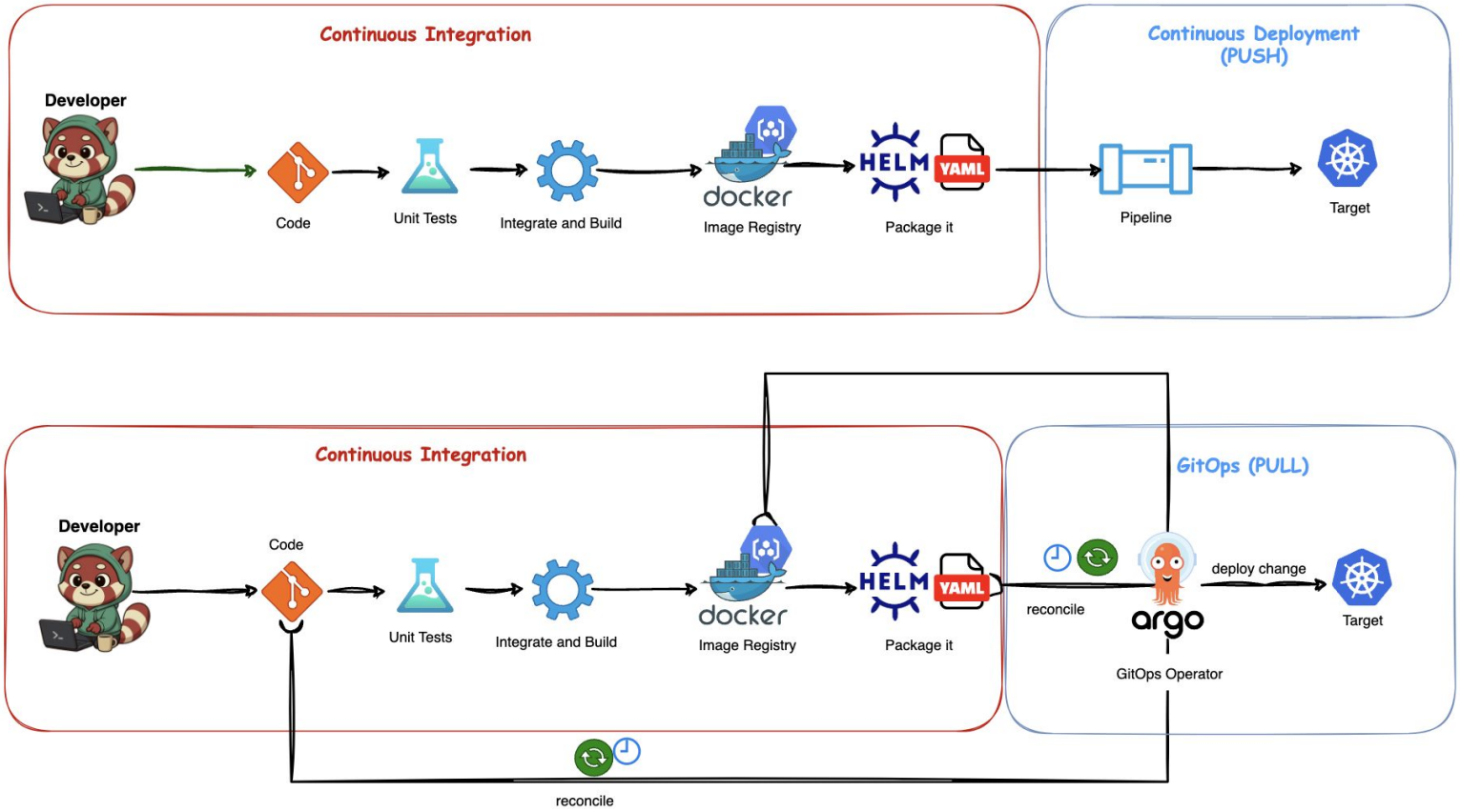
Scripting IaC from Git

Just a developer-only approach at first

CI/CD vs GitOps CD (I)



CI/CD vs GitOps CD (II)



Context: How GitOps Fits In

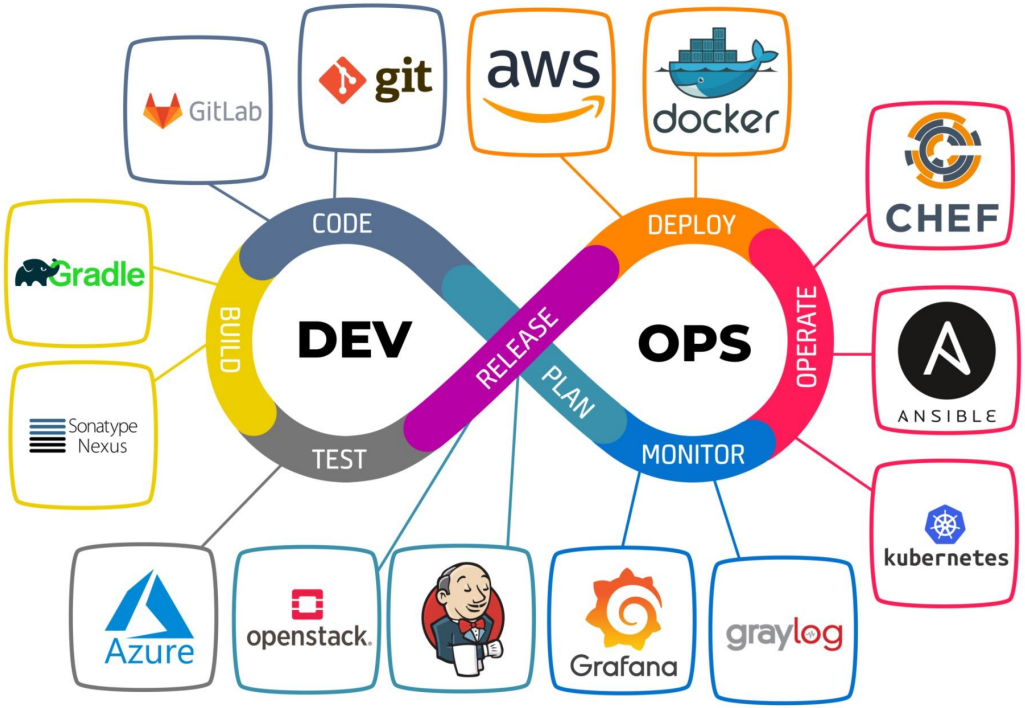
Context: How GitOps Fits In

- Devops
- DevSecOps
- Platform Engineering

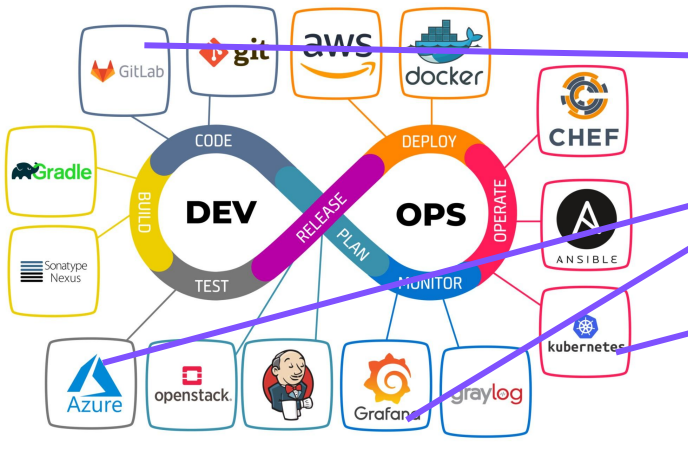
Evolution, Not Replacement: GitOps is seen as an advanced implementation of DevOps principles

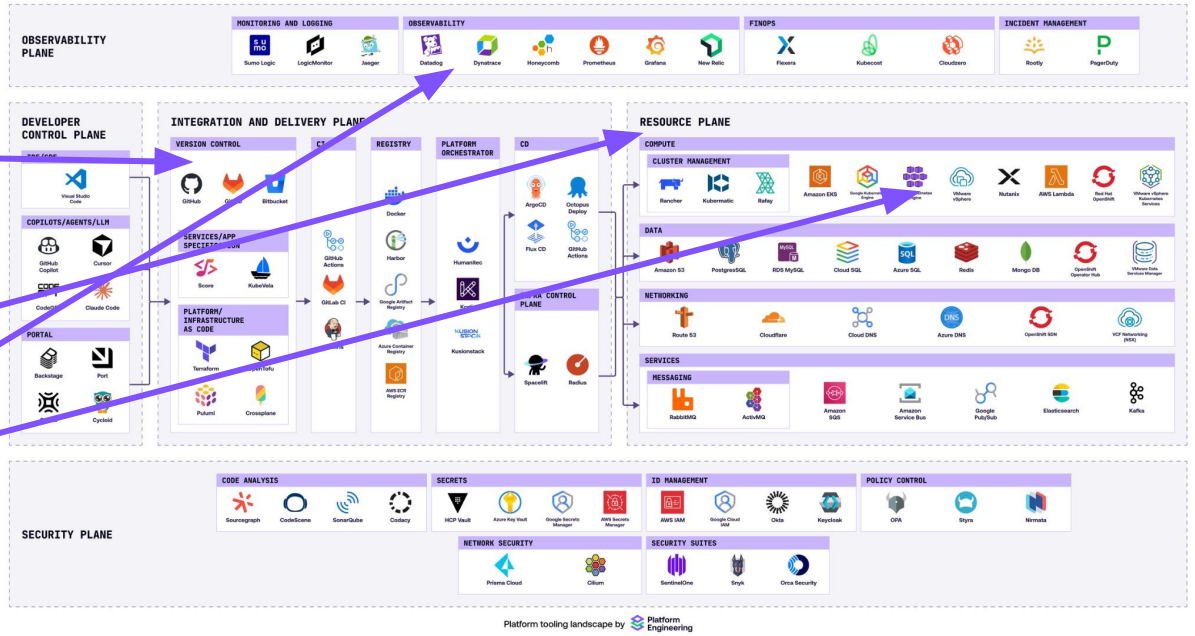
GitOps is not a tool, but an operating model that accelerates DevOps, secures DevSecOps, and scales Platform Engineering.

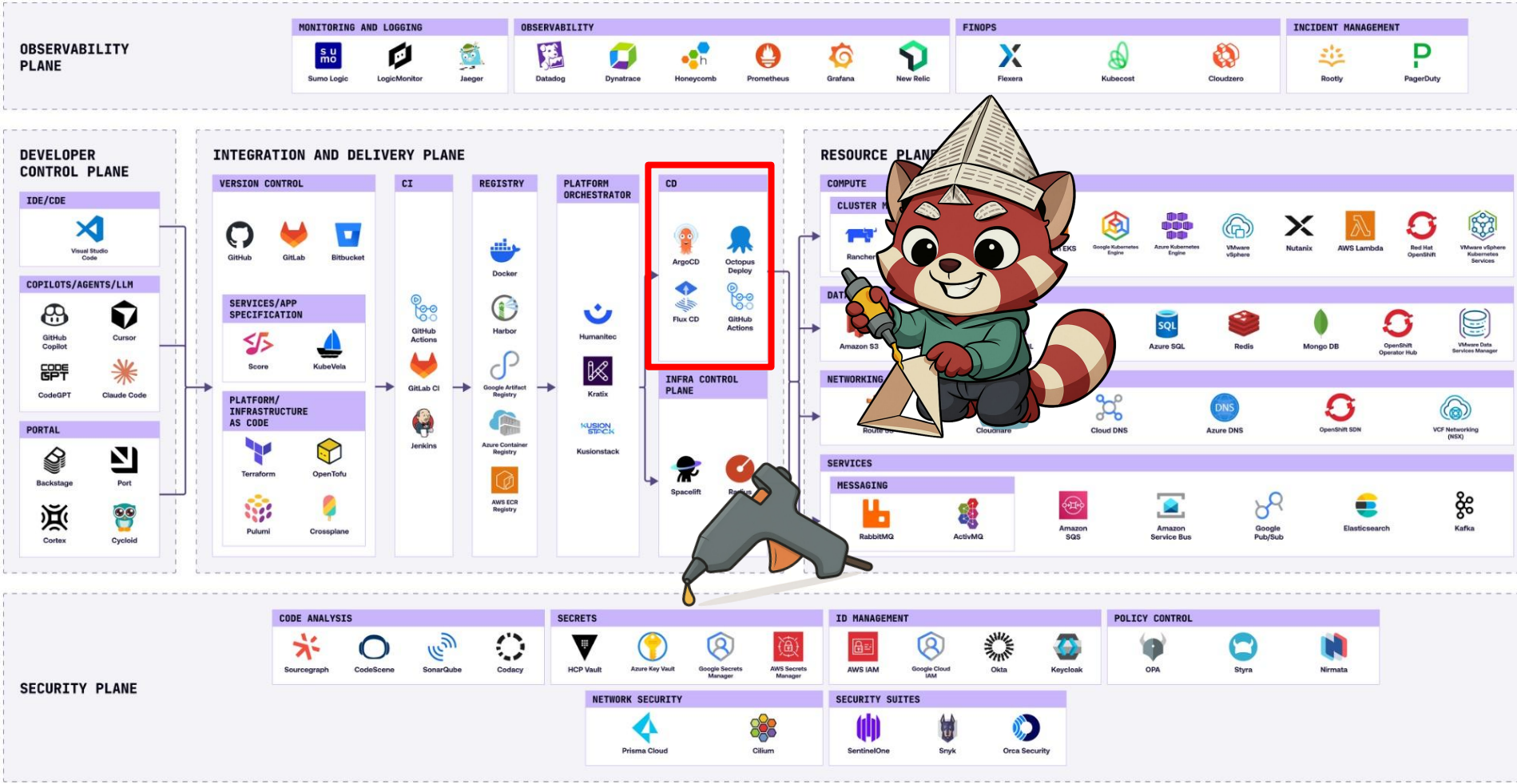
And by the way: DevOps is dead!



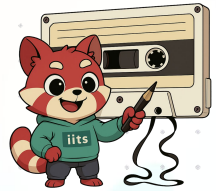
~~DevOps~~ Platform Engineering






OBSERVABILITY PLANE
MONITORING AND LOGGING
Sumo Logic
LogicMonitor
Jaeger
OBSERVABILITY
Datadog
Dynatrace
Honeycomb
Prometheus
Grafana
New Relic
FINOPS
Flexera
Kubecost
Cloudzero
INCIDENT MANAGEMENT
Rootly
PagerDuty
DEVELOPER CONTROL PLANE
IDE/CDE
Visual Studio Code
COPILOTS/AGENTS/LLM
GitHub Copilot
Cursor
CodeGPT
Claude Code
PORTAL
Backstage
Port
Cortex
Cycloid
INTEGRATION AND DELIVERY PLANE
VERSION CONTROL
GitHub
GitLab
Bitbucket
SERVICES/APP SPECIFICATION
Score
KubeVela
PLATFORM/ INFRASTRUCTURE AS CODE
Terraform
OpenTofu
Pulumi
Crossplane
CI
GitHub Actions
GitLab CI
Jenkins
REGISTRY
Docker
Harbor
Google Artifact Registry
Azure Container Registry
AWS ECR Registry
PLATFORM ORCHESTRATOR
Humanitec
Kratix
Kusionstack
CD
ArgoCD
Octopus Deploy
Flux CD
GitHub Actions
INFRA CONTROL PLANE
Spacelift
Radius
RESOURCE PLANE
COMPUTE
CLUSTER M
Rancher
EKS
Google Kubernetes Engine
Azure Kubernetes Engine
VMware vSphere
Nutanix
AWS Lambda
Red Hat OpenShift
VMware vSphere Kubernetes Services
DAT
Amazon S3
Azure SQL
Redis
Mongo DB
OpenShift Operator Hub
VMware Data Services Manager
NETWORKING
Route 53
Cloudflare
Cloud DNS
Azure DNS
OpenShift SDN
VCF Networking (NSX)
SERVICES
MESSAGING
RabbitMQ
ActivMQ
Amazon SQS
Amazon Service Bus
Google Pub/Sub
Elasticsearch
Kafka
SECURITY PLANE
CODE ANALYSIS
Sourcegraph
CodeScene
SonarQube
Codacy
SECRETS
HCP Vault
Azure Key Vault
Google Secrets Manager
AWS Secrets Manager
ID MANAGEMENT
AWS IAM
Google Cloud IAM
Okta
Keycloak
POLICY CONTROL
OPA
Styra
Nirmata
NETWORK SECURITY
Prisma Cloud
Cilium
SECURITY SUITES
SentinelOne
Snyk
Orca Security
Platform tooling landscape by
Platform Engineering

Recap: GitOps Fundamentals and Core Principles



- GitOps is not new—but in 2017, Alexis Richardson gave it a name
- Terms can be tricky. Understanding them well will help you now and in the future.
- GitOps is based on just four principles:
  - Declarative
  - Versioned and Immutable
  - Pull Based
  - Continuous Reconciliation
- GitOps is not a replacement for DevOps or Platform Engineering.
- GitOps is the glue for platform engineering. It allows you to:
  - Define declarative policies, infrastructure, budgets, dashboards, and more
  - Scale your platform by offering self-service