Kubernetes Advanced Monitoring, Logging, and GitOps

Karthikeyan Vaiyapuri

01

Monitoring with Prometheus and Grafana

What is Prometheus?

01

Open-source monitoring system with dimensional data model

02

Pull-based architecture that scrapes metrics from configured targets

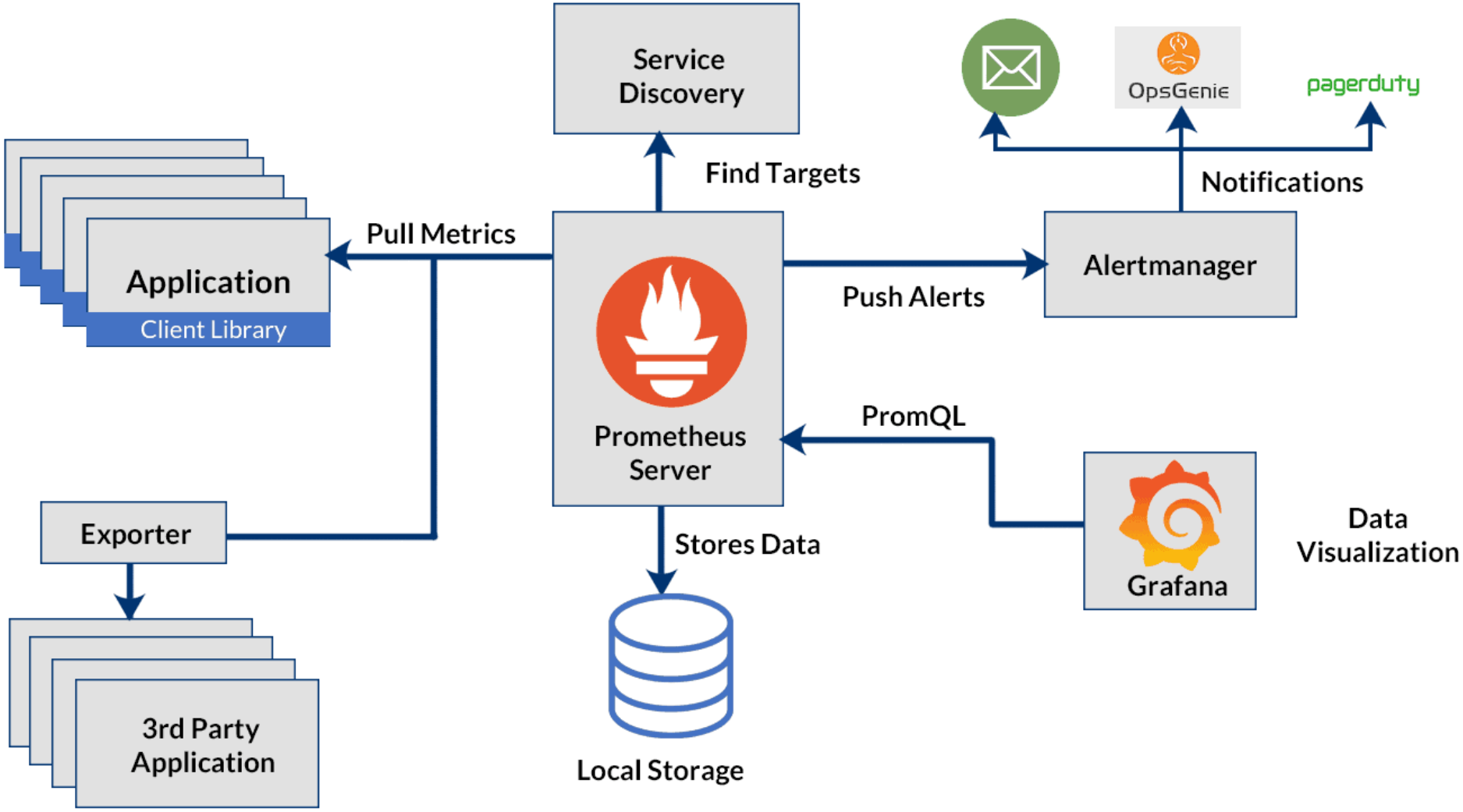
03

Built-in alerting through Alertmanager

04

Native Kubernetes integration with auto-discovery

Prometheus Architecture in Kubernetes



Prometheus Architecture in Kubernetes

Components

Prometheus Server: Core component that scrapes and stores metrics

Service Discovery: Automatically discovers Kubernetes resources

Exporters: Collect metrics from various services (node- exporter, kube-state-metrics)

Alertmanager: Handles alerts sent by Prometheus server

Key Kubernetes Metrics

01

Cluster-level metrics: CPU, memory, storage usage

02

Pod-level metrics: Container resource consumption

03

Application metrics: Custom business metrics

04

Infrastructure metrics: Node health, network performance

Prometheus Configuration in Kubernetes

```yaml
# ServiceMonitor for automatic service discovery
apiVersion: monitoring.coreos.com/v1
kind: ServiceMonitor
metadata:
  name: app-metrics
spec:
  selector:
    matchLabels:
      app: myapp
  endpoints:
  - port: metrics
```

Grafana Integration

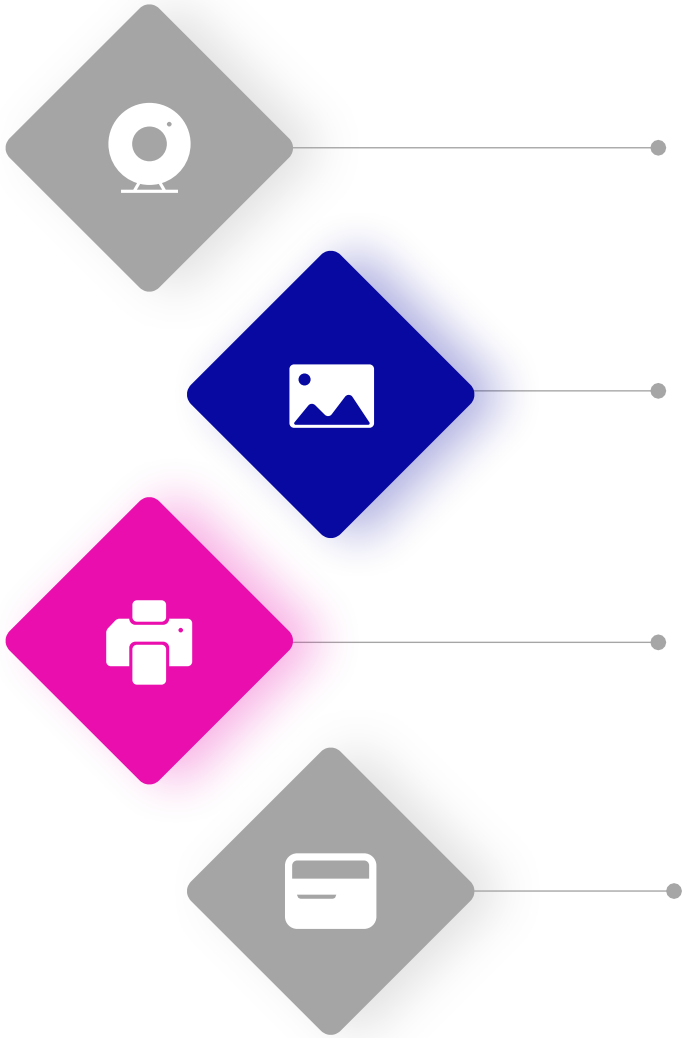
Visualization platform for Prometheus metrics

Pre-built dashboards for Kubernetes monitoring

Custom dashboards for application-specific metrics

Alerting capabilities with multiple notification channels

Deployment Strategies



Helm Charts: prometheus-community/kube-prometheus-stack

Operator- based: Prometheus Operator for CRD management

Manual deployment: Custom YAML manifests

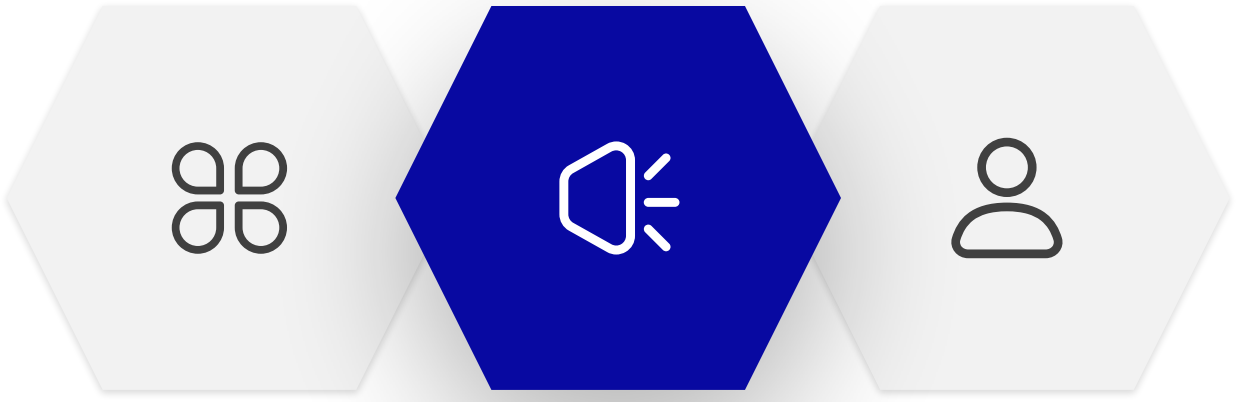
Cloud- managed: EKS/GKE/AKS integrated monitoring

02

Logging with EFK/ELK Stack

EFK Stack Components

Elasticsearch: Distributed search and analytics engine



Kibana: Data visualization and exploration platform

Fluentd/Fluent Bit: Log collection and forwarding agents

ELK Stack Alternative

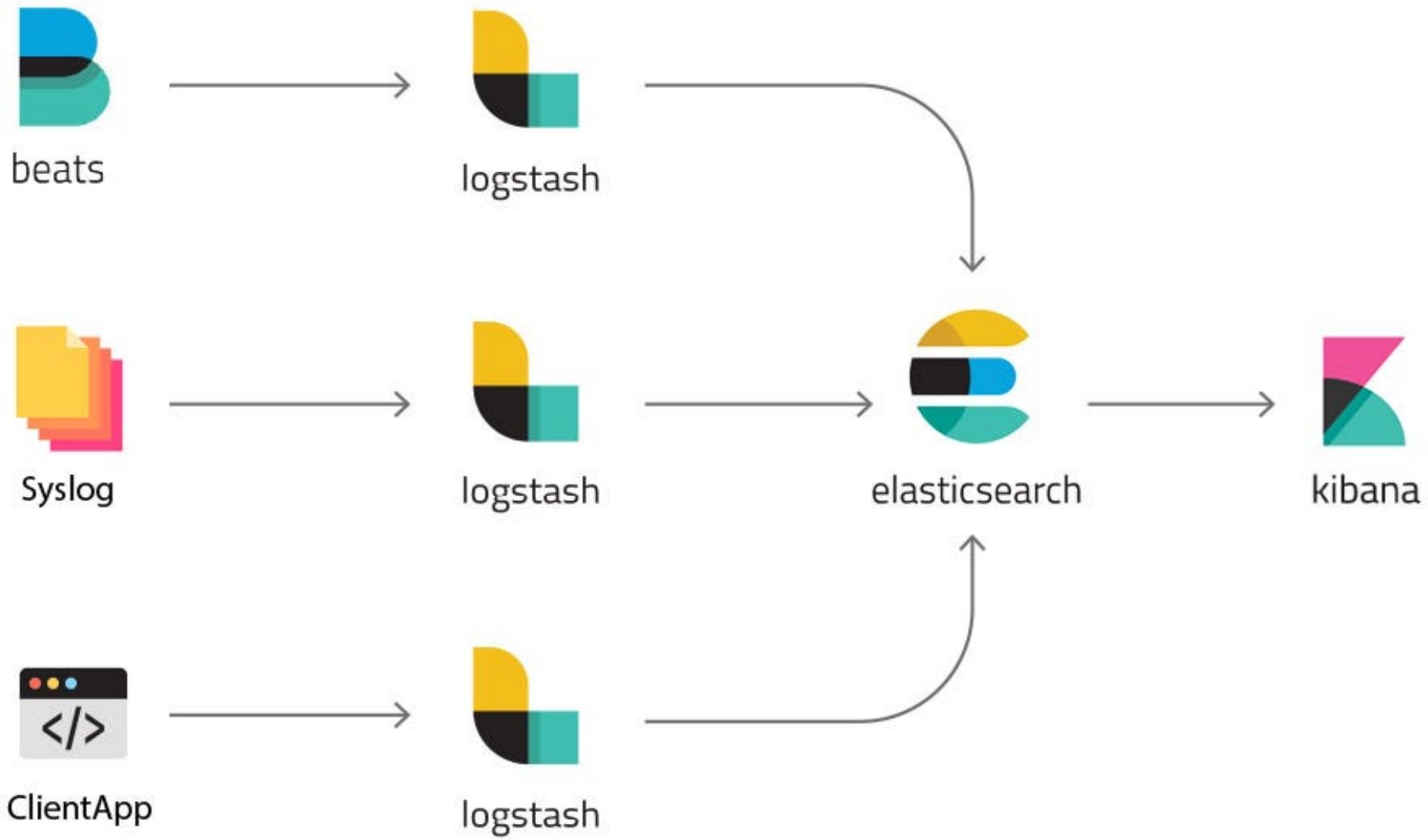
Components

Elasticsearch: Same search engine

Logstash: Data processing pipeline

Kibana: Same visualization platform

ELK Stack Alternative



Kubernetes Logging Architecture

Logging Levels

01

Node-level logging: Container logs stored on nodes

02

Cluster-level logging: Centralized log aggregation

03

Application-level logging: Direct logging to external systems

04

Sidecar pattern: Dedicated logging containers

Fluentd vs Fluent Bit

Feature	Fluentd	Fluent Bit
Resource Usage	Higher memory footprint	Lightweight, minimal resources
Plugin Ecosystem	Extensive Ruby plugins	Growing C/Go plugins
Use Case	Complex log processing	Edge/IoT, resource-constrained
Kubernetes Deployment	DaemonSet for log aggregation	DaemonSet for log collection

Log Collection Patterns

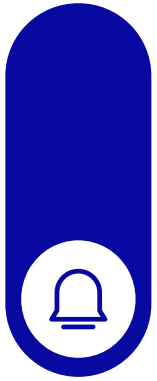
DaemonSet deployment: One pod per node for log collection

Sidecar containers: Application-specific log processors

Direct shipping: Applications send logs directly to Elasticsearch

Multi- stage pipeline: Fluent Bit → Fluentd → Elasticsearch

Configuration Examples



```yaml
# Fluent Bit ConfigMap
apiVersion: v1
kind: ConfigMap
metadata:
  name: fluent-bit-config
data:
  fluent-bit.conf: |
    [INPUT]
        Name tail
        Path /var/log/containers/*.log
        Parser docker
        Tag kube.*
```

03

GitOps with ArgoCD and Flux

GitOps Principles

- **Declarative configuration:** Infrastructure and applications as code
- **Version controlled:** Git as single source of truth
- **Automated deployment:** Continuous reconciliation
- **Observable:** Clear audit trail and rollback capabilities

ArgoCD Overview

Features

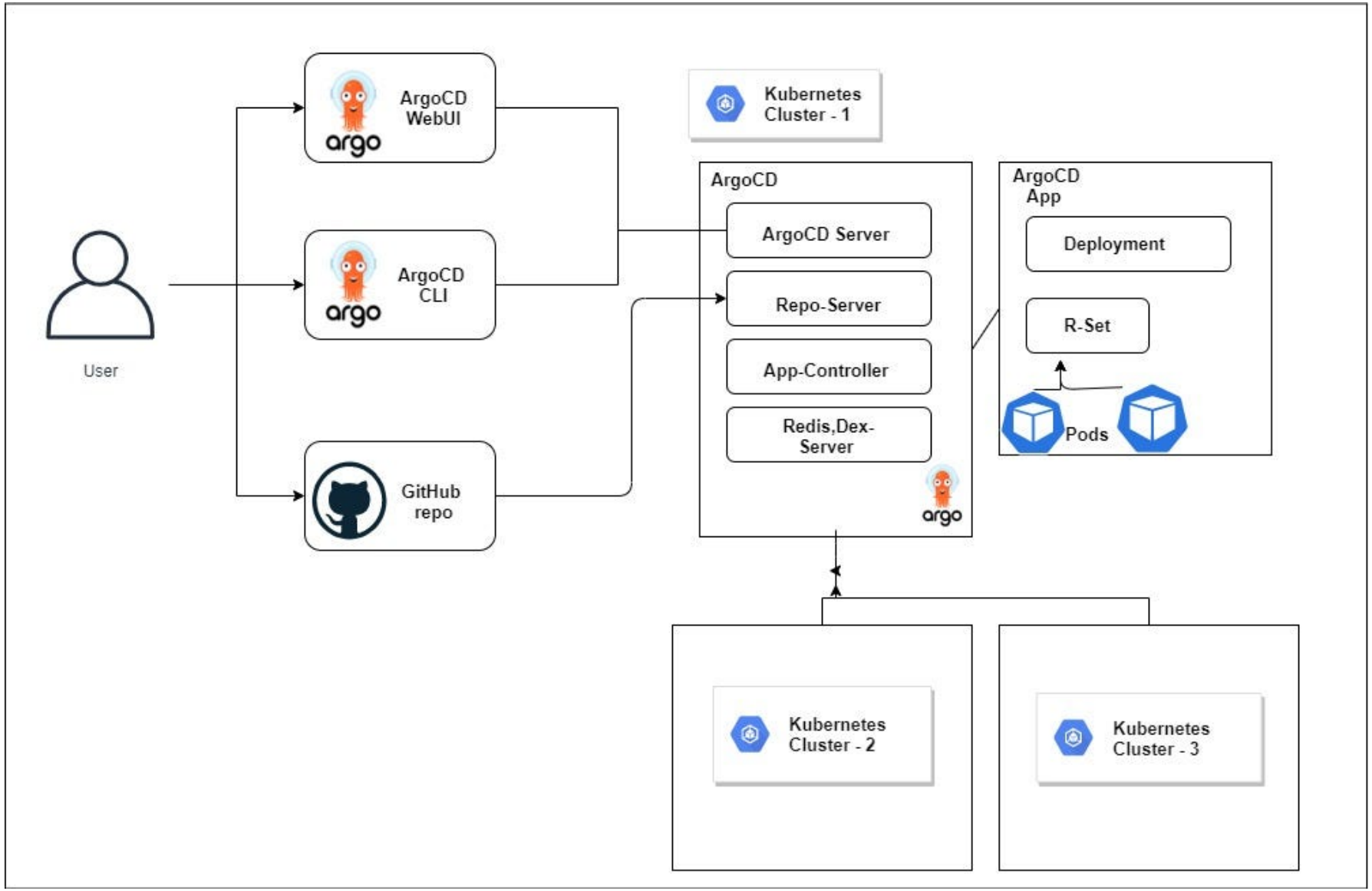
Kubernetes-native CD tool for GitOps workflows

Web UI and CLI for application management

Multi-cluster support with centralized management

RBAC integration with Kubernetes authentication

ArgoCD Architecture



ArgoCD Architecture

Application Controller: Monitors Git repositories and Kubernetes clusters

API Server: Exposes API for CLI and UI

Repository Server: Handles Git repository operations

Redis: Caching and session storage

ArgoCD Application Types

01

Helm Charts: Package manager for Kubernetes

02

Kustomize: Template- free configuration management

03

Plain YAML: Direct Kubernetes manifests

04

Jsonnet: Data templating language

Flux Overview

Toolkit Approach

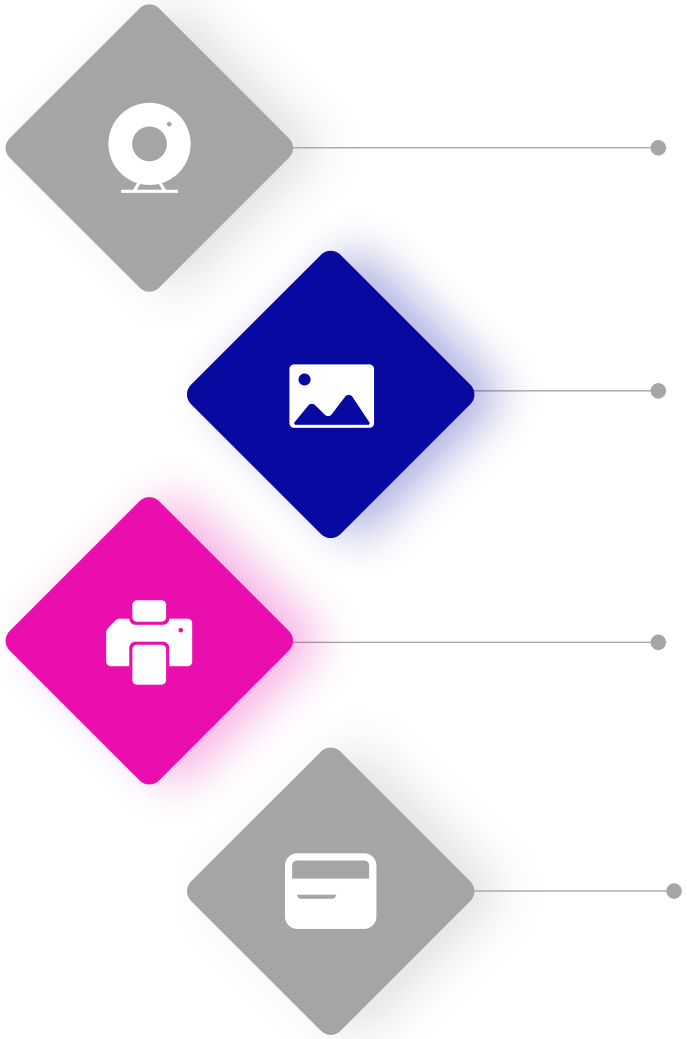
01 **CNCF graduated project** for GitOps

02 **Toolkit approach:** Modular components

03 **Multi- tenancy support** with namespace isolation

04 **Progressive delivery** with Flagger integration

Flux v2 Components



Source Controller: Handles Git and Helm repositories

Kustomize Controller: Applies Kustomize configurations

Helm Controller: Manages Helm releases

Notification Controller: Sends alerts and webhooks

ArgoCD vs Flux Comparison

Feature	ArgoCD	Flux
UI/UX	Rich web interface	CLI-focusted, optional UI
Multi-cluster	Native support	Requires additional setup
Helm SUpport	Built-in	Dedicated Helm controller
Learning Curve	Moderate	Steeper initial setup
Community	Large, active	CNCF backing, growing

GitOps Workflow

Workflow Steps

01

Developer commits code to Git repository

02

CI pipeline builds and pushes container images

03

GitOps tool detects configuration changes

04

Automatic synchronization applies changes to cluster

05

Monitoring and alerting track deployment status

Best Practices

Recommended Practices

01

Separate repositories: Application code vs. configuration

02

Environment branching: Different branches for dev/staging/prod

03

Secret management: External secret stores (Vault, AWS Secrets Manager)

04

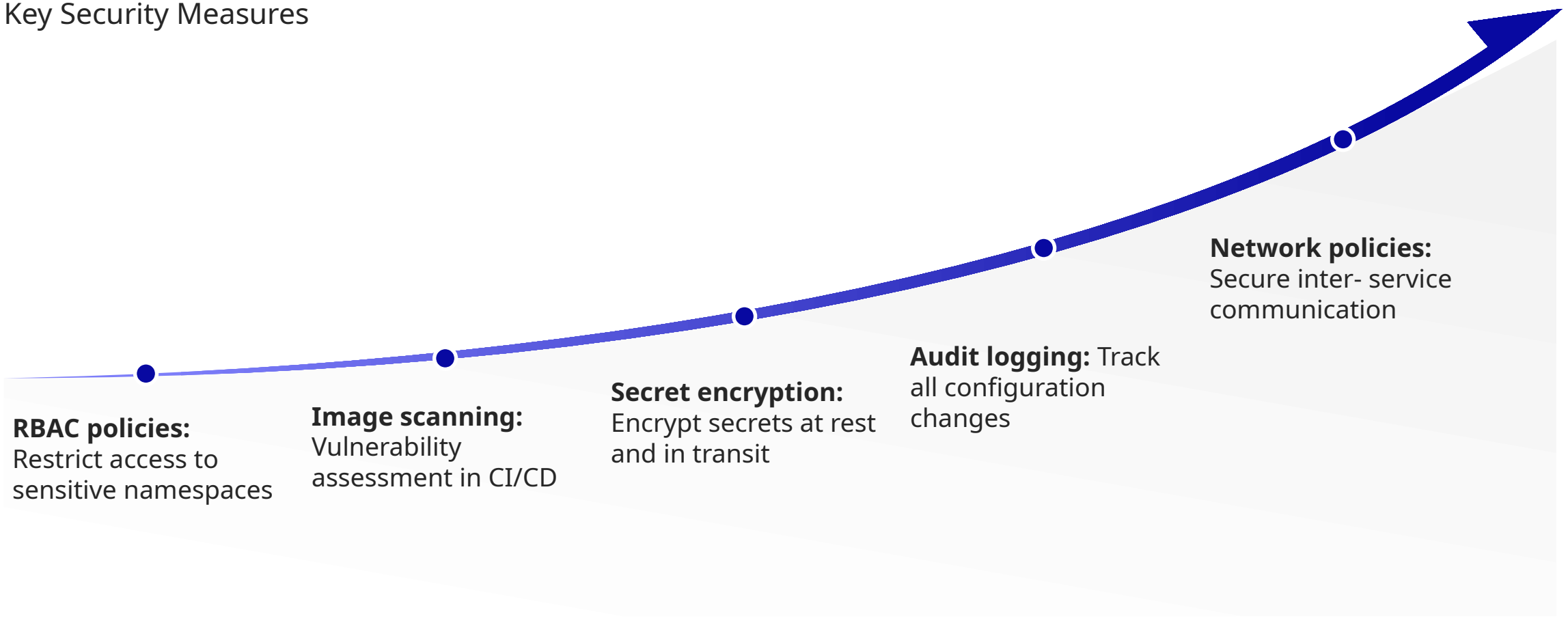
Progressive rollouts: Canary deployments and blue-green strategies

05

Monitoring integration: Combine with Prometheus for deployment metrics

Security Considerations

Key Security Measures



Thanks

Karthikeyan Vaiyapuri