Low-ops Kubernetes for a small on-premises footprint

Vincent Link
Mercedes-Benz AG
05/10/2022

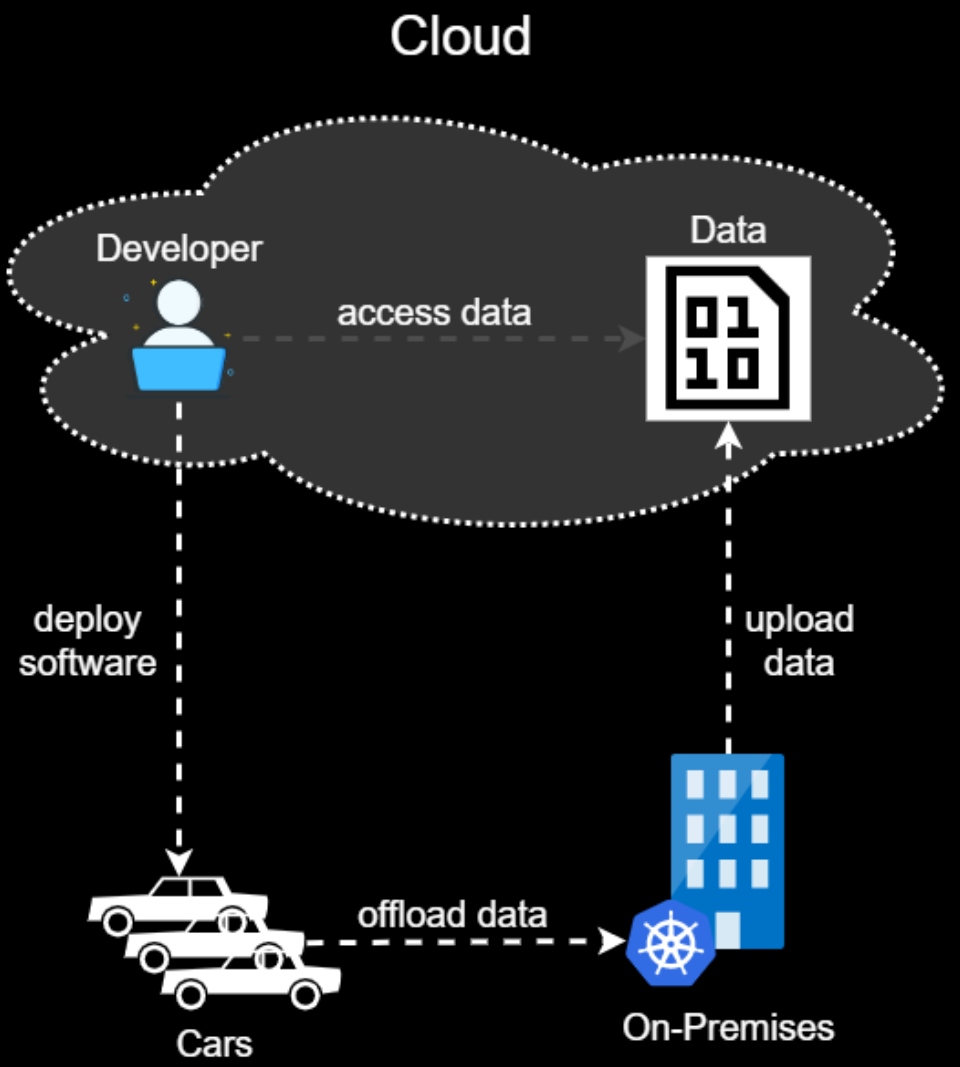
Mercedes-Benz

Background

IT-Infrastructure in the Automated Driving context

Use-Case

1. Data is collected by software running on cars
2. Data is offloaded at on-premises locations (globally)
3. Data is uploaded to the Cloud
4. Data is processed by developers in the Cloud
5. New software is developed and deployed on cars



About Me

- Since 2018 with Mercedes-Benz in the field of automated driving
- Working on Cloud- & on-prem IT-Infrastructure topics
- Background in Software Engineering
- Interests
  - Kubernetes & container technologies
  - Serverless software development
- Working Motto
  - YAGNI - You aren‘t gonna need it

Agenda

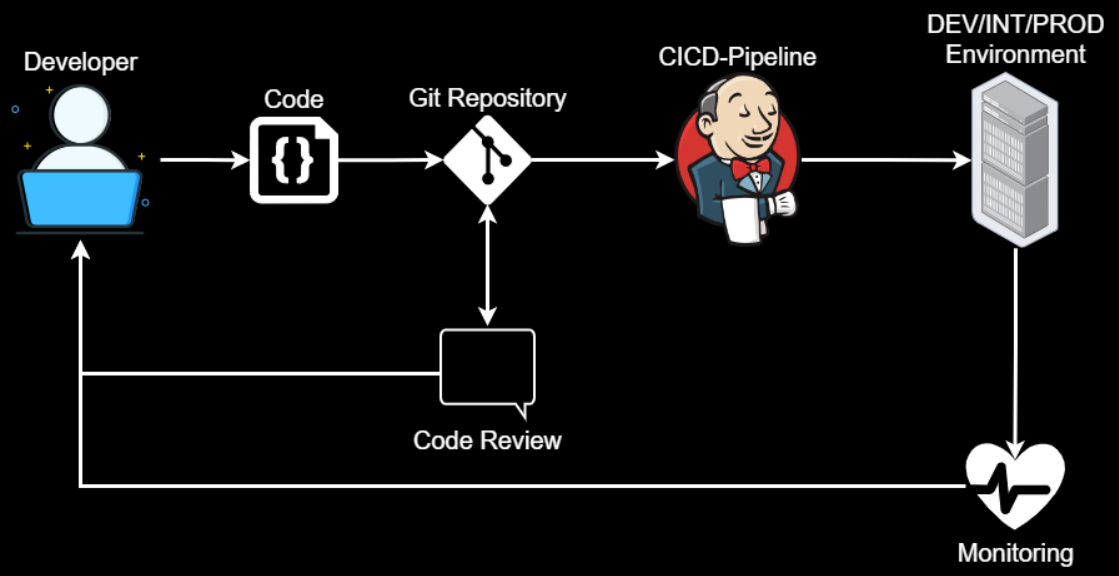
- Fundamentals
  - GitOps
  - Kubernetes
- High-Level Business Requirements
- Focused Topics
  - Kubernetes-Rollout
  - Application-Rollout
  - High-Availability
  - TLS-Certificates
  - Health Checks
  - Monitoring & Alerting
- Lessons Learned

What is GitOps?

Summary: Infrastructure Automation and Deployments with Version Control and CI/CD

Principles according to OpenGitOps

1. Declarative State
2. Versioned and Immutable State
3. State is Pulled Automatically
4. State is Continuously Reconciled



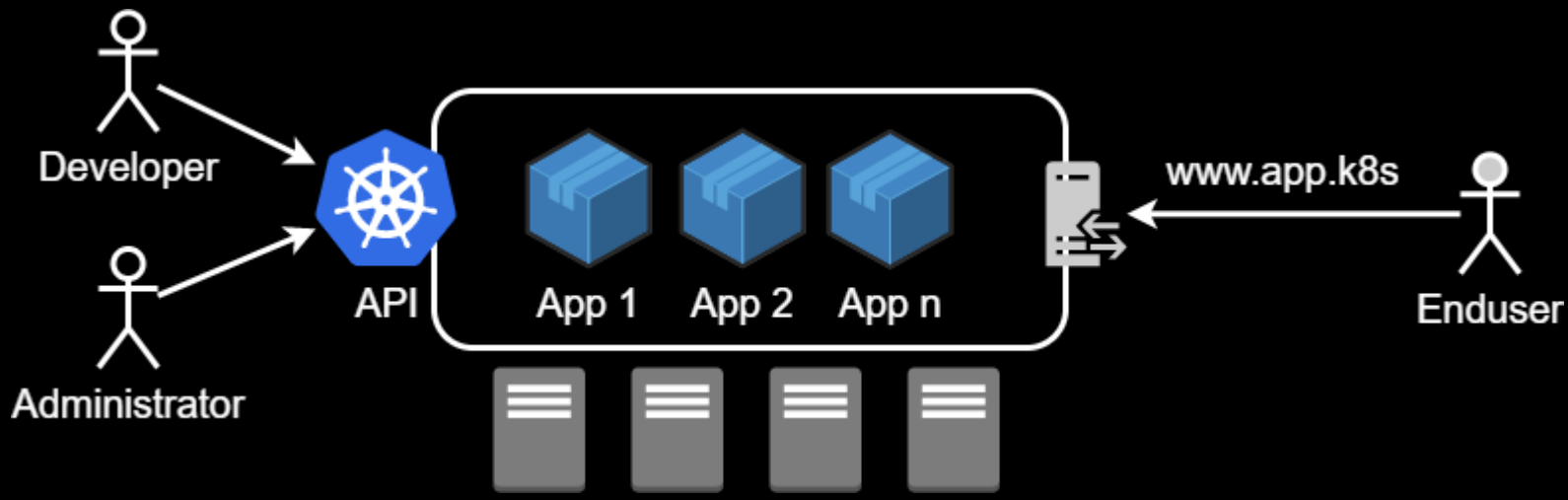
What is Kubernetes?

In summary, Kubernetes is …

- a container orchestration system
- able to manage IT infrastructure below the application level (storage, compute, network)

But most importantly

- Kubernetes creates the perfect developer self-service for IT infrastructure topics



Kubernetes gives you...

- Declarative management of all resources as code
- Automated rollouts and rollbacks of containers over multiple hosts
- Storage Orchestration of external systems
- Service Discovery and Load Balancing
- Configuration and Secret Managment
- Self-Healing capabilities (Healthchecks, „Restart“ on failure)
- A single API for your infrastructure

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.14.2
        ports:
        - containerPort: 80
```

High-Level Business Requirements (as understood by IT)

- Ingestion and upload of large amounts of data (xx TB per car and day)
- Environment to run applications
  - Ingest, Pre-/Post-Processing, Upload, …
- Redundant and Highly Available
- Scalable
- Secure Protocols
- High-Bandwidth (40Gbit/s+)
- Abstract IT infrastructure for developers

Where did we start

- Generic Servers
- Storage Appliance with NFS and S3 support
- Ubuntu Server as Operating System
  - managed via SaltStack
- Out of Scope
  - Housing
  - Power
  - Cooling
  - Network Infrastructure
  - OpenID Connect Provider (e.g. your companies Single Sign On)

Initial State

Developer

Administrator

Enduser

Kubernetes Rollout

Goal: Rollout and configure Kubernetes via Infrastructure as Code

Requirements

- Deploys a highly available Kubernetes cluster
- Support for Ubuntu
- Support for high bandwidth Kubernetes networking
- GitOps support

Try kubespray!

- fulfills all requirements
- based on ansible

kubespray

Additional Features

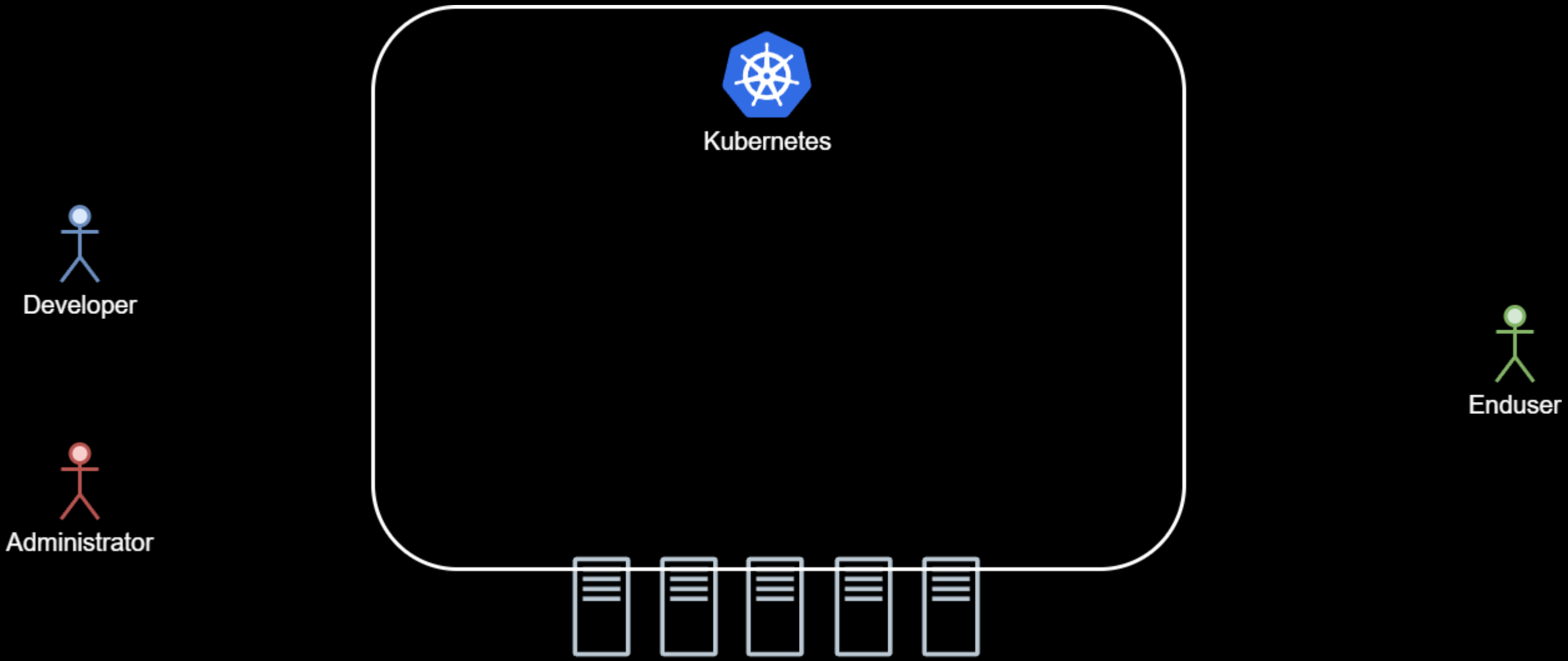
- HTTP Proxy Support
- Easily and extensively customizable
- Allows deployment of basic k8s applications

Usage

1. Clone the kubespray repository
2. Create your ansible inventory file
3. Customize the sane defaults
4. Deploy your Kubernetes Cluster
   - ansible-playbook -i hosts.ini cluster.yml

```
1  [kube_control_plane]
2  server1.local
3  server2.local
4  server3.local
5
6  [kube_node]
7  server4.local
8  server5.local
9
10 [etcd:children]
11 kube_control_plane
12
13 [k8s_cluster:children]
14 kube_control_plane
15 kube_node
```

State after kubespray



Application rollout to Kubernetes

Goal: Deploy apps running in Kubernetes the GitOps way

Requirements

- Fetch all Kubernetes ressources from git repos with drift detection
- Visibility of all deployments
- Cleanup resources deleted in the code!

Use ArgoCD!

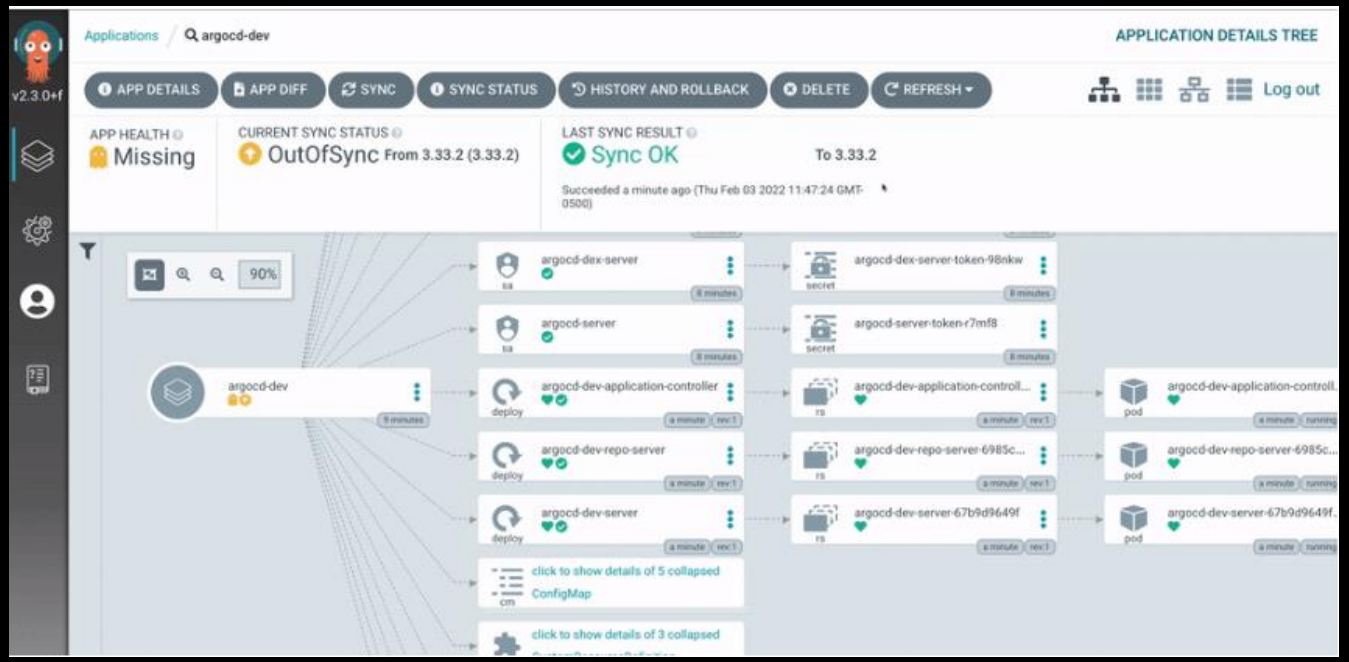
- Supports deployment from plain yaml, kustomize or also helm charts
- Granular permission management
- OIDC Support → connect your Single Sign On

ArgoCD

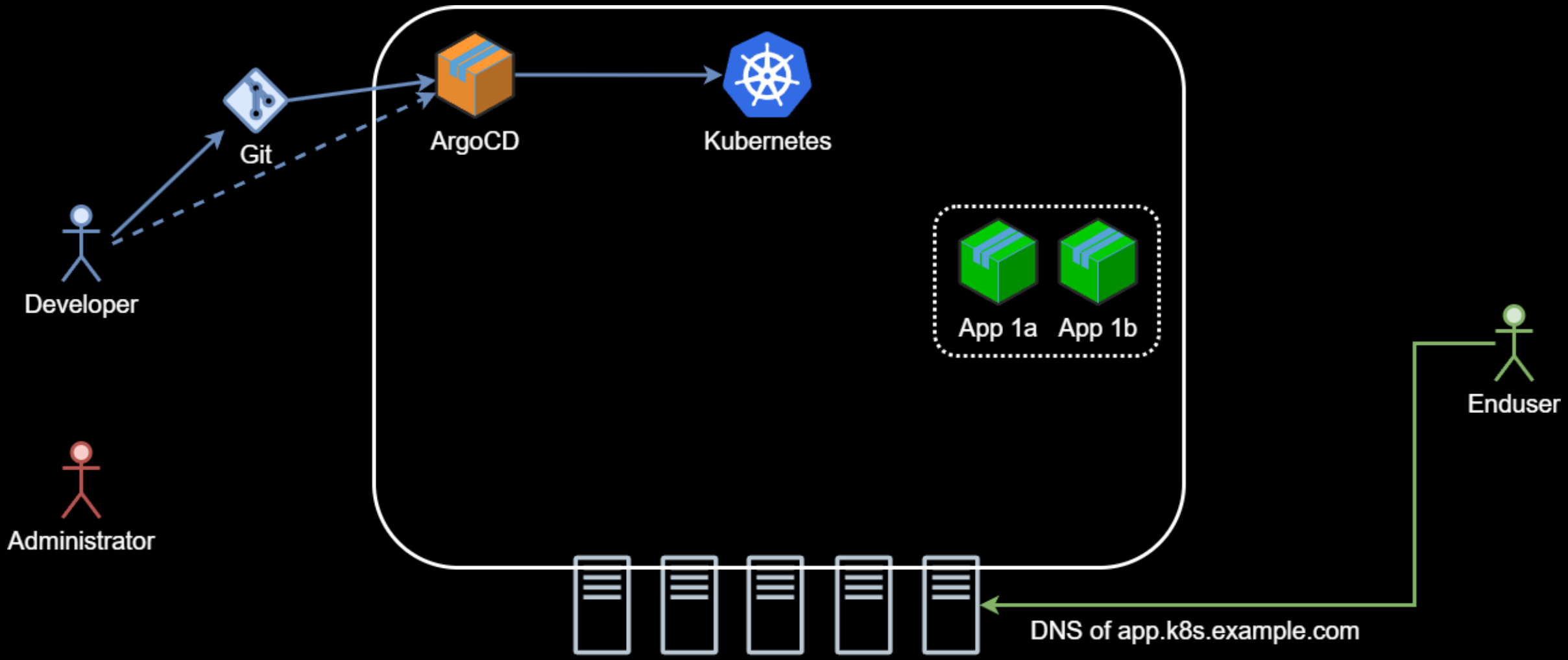
Deployment: „Manually“ apply YAMLs or via kubespray

Usage

1. Configure your repositories as YAMLs or in the UI
2. Configure synchronization options for this repo (auto sync, target branch, …)
3. Trigger a Sync



State after ArgoCD



Highly Available Load-Balancer for Kubernetes

Goal: Applications are reachable after individual failures of physical servers

Requirements

- Outages of single servers are handled
  - in case of hardware defects, reboots after OS or Kubernetes upgrades, …
- No physical appliance must be installed
- No manual interaction is needed

Prerequisites

- Kubernetes-Cluster has 3+ Control Plane nodes
- Developers understand stateless apps and externalized state

MetalLB - Load-Balancer for bare metal Kubernetes clusters

Usage

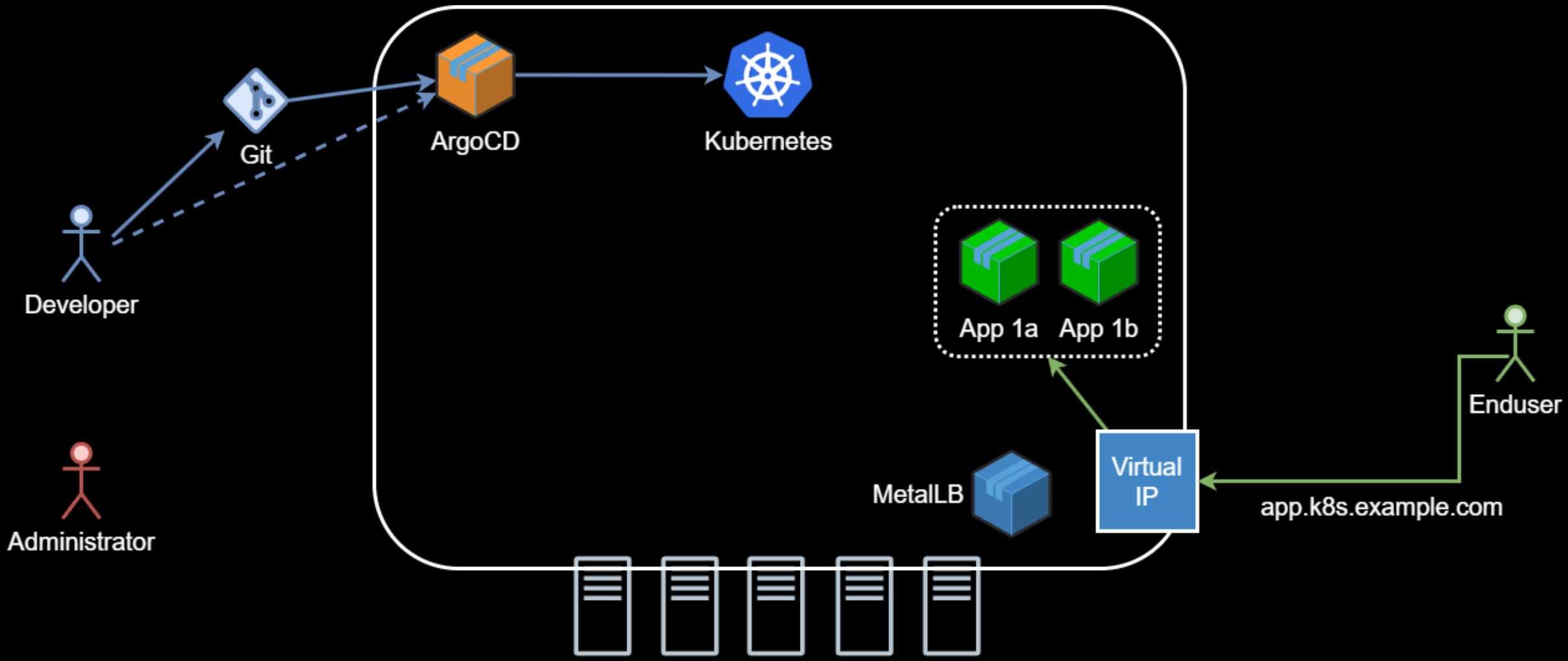
1. Deploy MetalLB (via kubespray or ArgoCD)
2. Reserve 1+ virtual IPs in your servers subnet
3. Configure MetalLB to attach this IP to your servers

```yaml
apiVersion: metallb.io/v1beta1
kind: IPAddressPool
metadata:
  name: address-pool
spec:
  addresses:
  - 192.168.9.1-192.168.9.5
```

How does it work?

- MetalLB agent runs on every node in Kubernetes
- Agent unavailability is detected
- IP is assigned to healthy node followed by ARP announcements
    - BGP also supported

State after MetalLB



Transparent TLS Certificates for Applications

Goal: HTTPS for every accessible service

Requirements

- Automated creation of SSL/TLS certificates
- Automated renewal
- No additional developer or admin interaction needed

Use cert-manager!

- Automatic Certificate Management Environment (ACME) for Mercedes internal Certificate Authority (CA) available

cert-manager - Automated X.509 certificate handling on Kubernetes

Support for self-signed, custom CA or ACME endpoints like Let‘s Encrypt

Usage

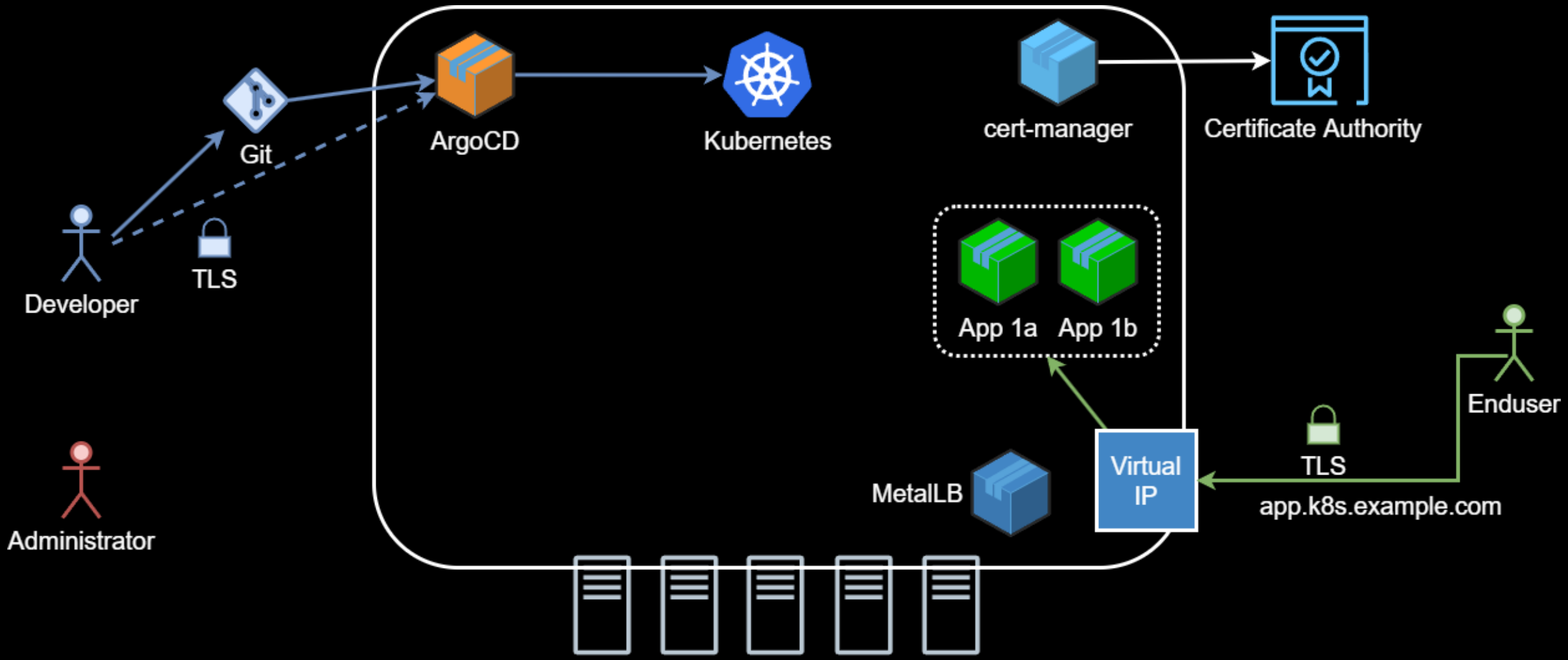
1. Deploy cert-manager (via kubespray or ArgoCD)
2. Configure certificate endpoints
3. Deploy Ingress Kubernetes resource

```yaml
apiVersion: cert-manager.io/v1
kind: ClusterIssuer
metadata:
  name: companies-acme
spec:
  acme:
    server: https://acme.intra.example.com/
    email: k8s-admins@example.com
    privateKeySecretRef:
      name: companies-acme-secret
    solvers:
      - http01:
          ingress:
            class: nginx
            serviceType: ClusterIP
```

Suggestion

- *.example.com points to ingress controller (~ reverse proxy)
- → Developers can choose desired domain names themselves

State after cert-manager



Monitoring and Alerting

Goal: Kubernetes and Application state is visible to developers and admins

Requirements

- Monitoring of infrastructure and end user applications
- Alerts on specific conditions
- Pretty Dashboards

Use kube-prometheus!

- Highly available Grafana, Prometheus and Alertmanager stack
- Highly customizable and extensible
- Predefined alerts for common issues

kube-prometheus

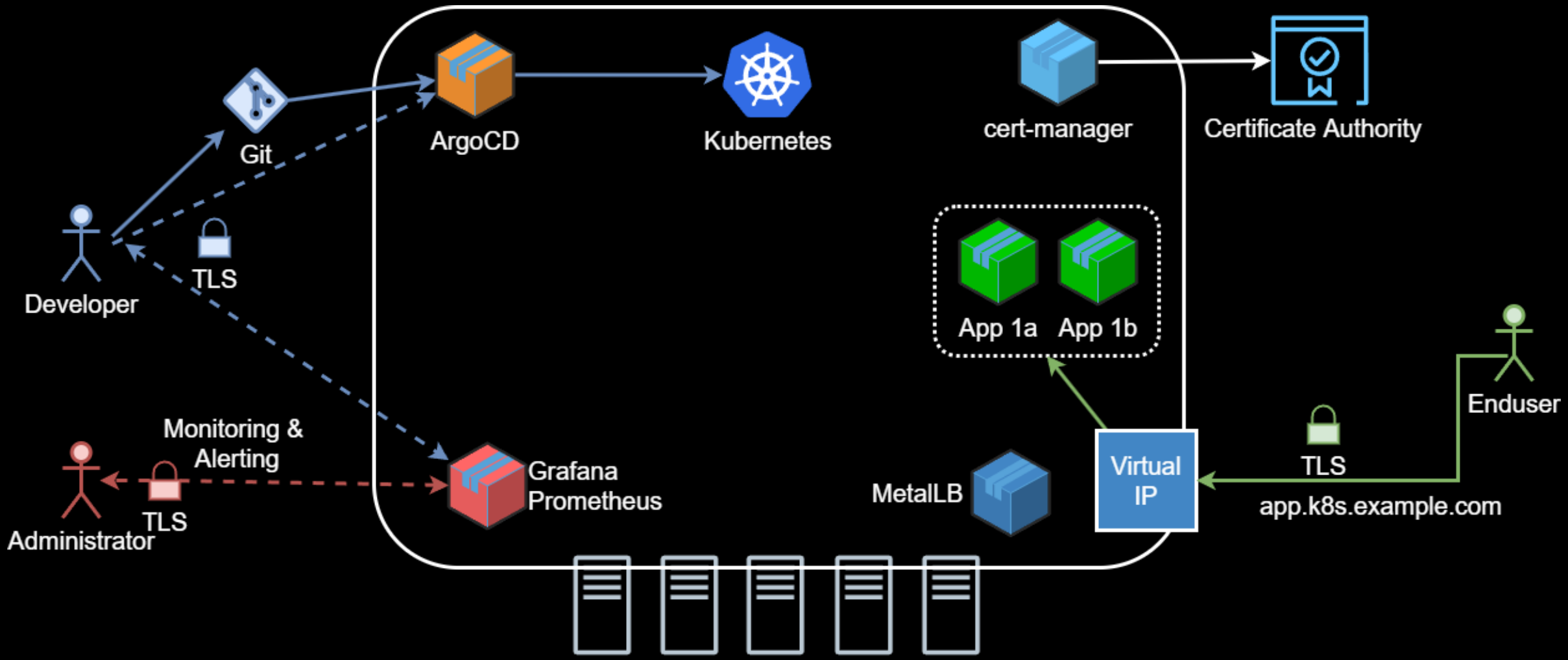
Usage

1. Understand the templating language jsonnet (tough!)
2. Setup a project and configure kube-prometheus
3. Generate the YAMLs for the full Grafana, Prometheus, Alertmanager stack
4. Deploy these YAMLs via ArgoCD

IMO

- Still worth the effort as it still reduces the complexity
- All discussed applications expose Prometheus metrics (e.g. MetalLB and ArgoCD)
    - Alert configurations are usually documented

State after kube-prometheus



Health Checks for Kubernetes

Goal: The cluster health is verified automatically

Requirements

- Developer and Enduser interactions with Kubernetes are regularly tested
- The underlying infrastructure is tested
- Prometheus Metrics are exposed and Alerts exist

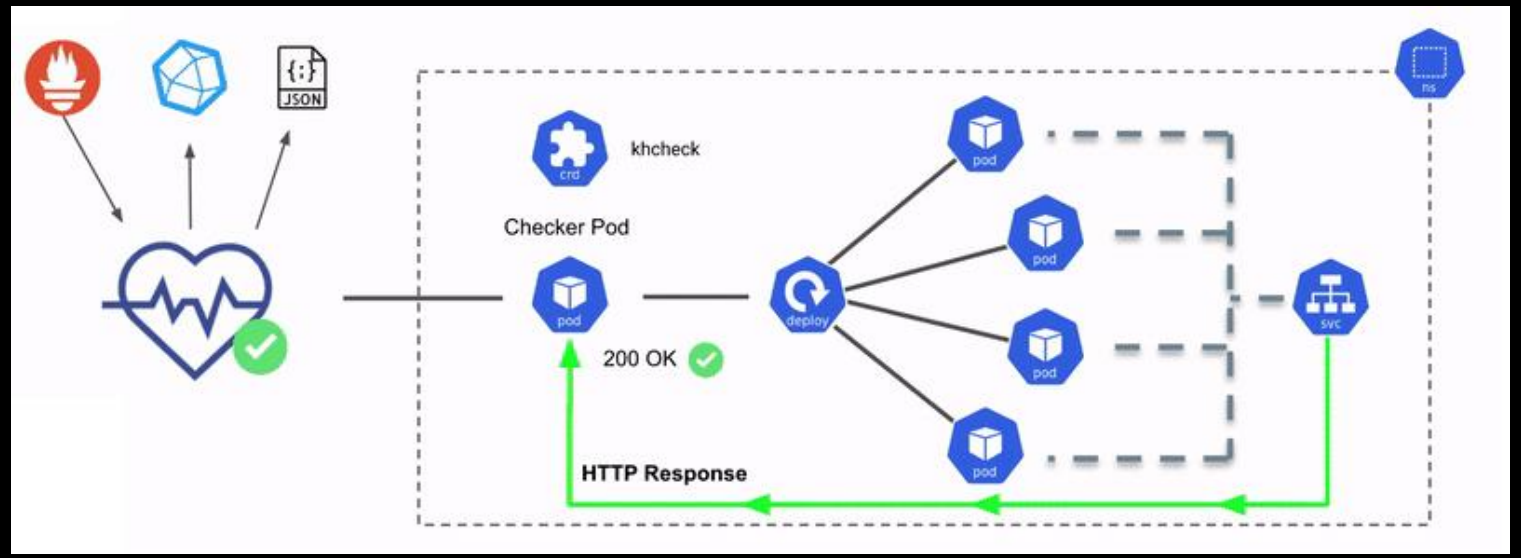
Use kuberhealthy!

- Application for continuous synthetic monitoring in Kubernetes

kuberhealthy

Regularly runs predefined and custom checks on Kubernetes

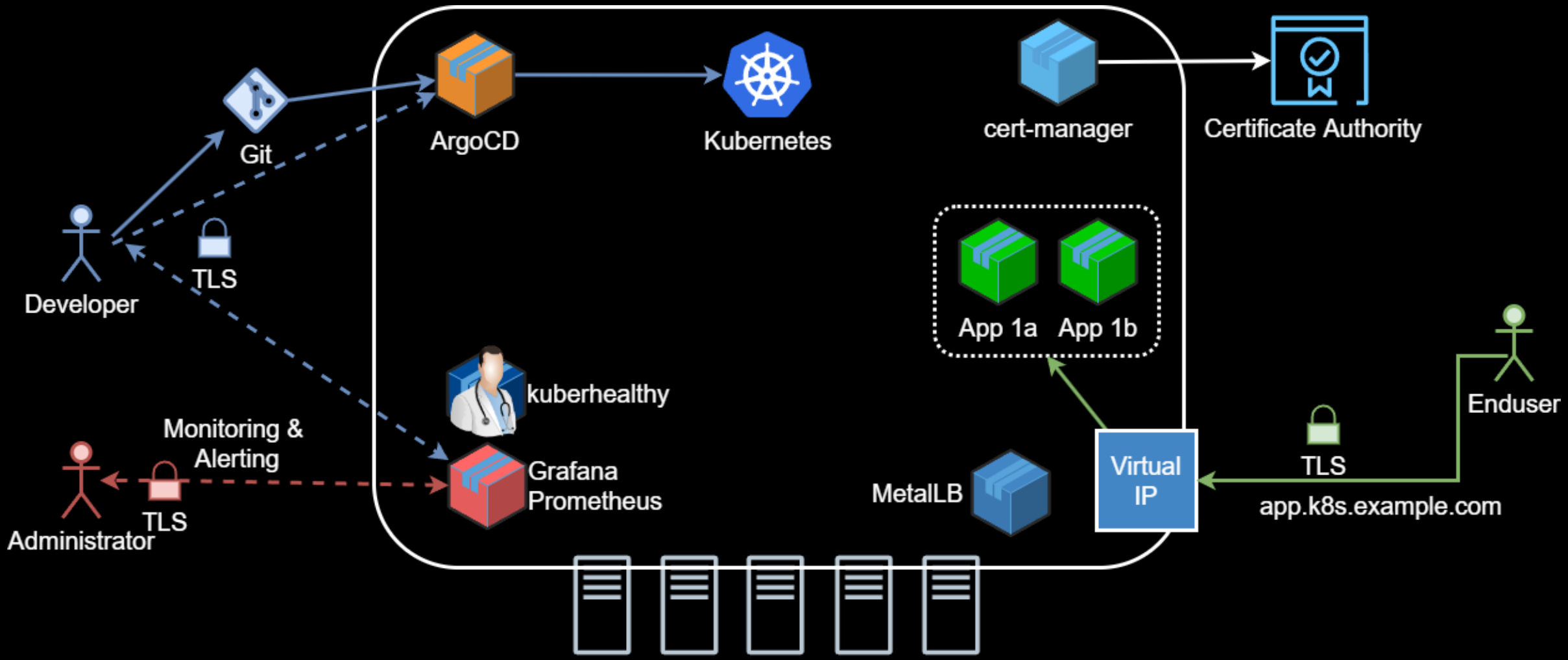
- Successful deployment of dummy applications
- Pulling of Container Images
- Network reachability checks
- …



Usage

1. Deploy kuberhealthy via ArgoCD (e.g. from helm)
2. Deploy KuberhealthyCheck ressource
3. Resolve alerts in case of problems

State after kuberhealthy



How did our IT infrastructure change?

Previous Setup

- Many non-containerized applications
- Containers running on individual hosts
- Hadoop File System distribution
- Mostly on-premises
- Closed-Source enterprise software
- Knowledge mostly at external contractor

Current Setup

- 100% containerized applications
- Kubernetes as container orchestration
- Storage appliance with NFS & S3 API
- Minimal on-premises setup
- FOSS all the way
- Knowledge 100% in-house

→ Low operational effort due to small footprint with high automation

Lessons Learned

- Remove IT infrastructure burdens from developers
- Embrace the FOSS mindset
  - Do not forget to contribute (report issues, fix bugs, add features, …)
- KISS, YAGNI, … - understand the demand, don’t overcomplicate things
- Run automated tests in all environments
- Scoped to our project
  - Buy managed storage, don’t DIY
  - Develop in-house, Automize expendable work, Scale externally

Questions?

Vincent Link (me)

- vincent.link@mercedes-benz.com →

Shirlei Sturm (my manager)

- shirlei.sturm@mercedes-benz.com