



KUBERNETES PATTERNS

Roland Huß, Red Hat, @ro14nd



"m" for menu, "?" for other shortcuts

AGENDA

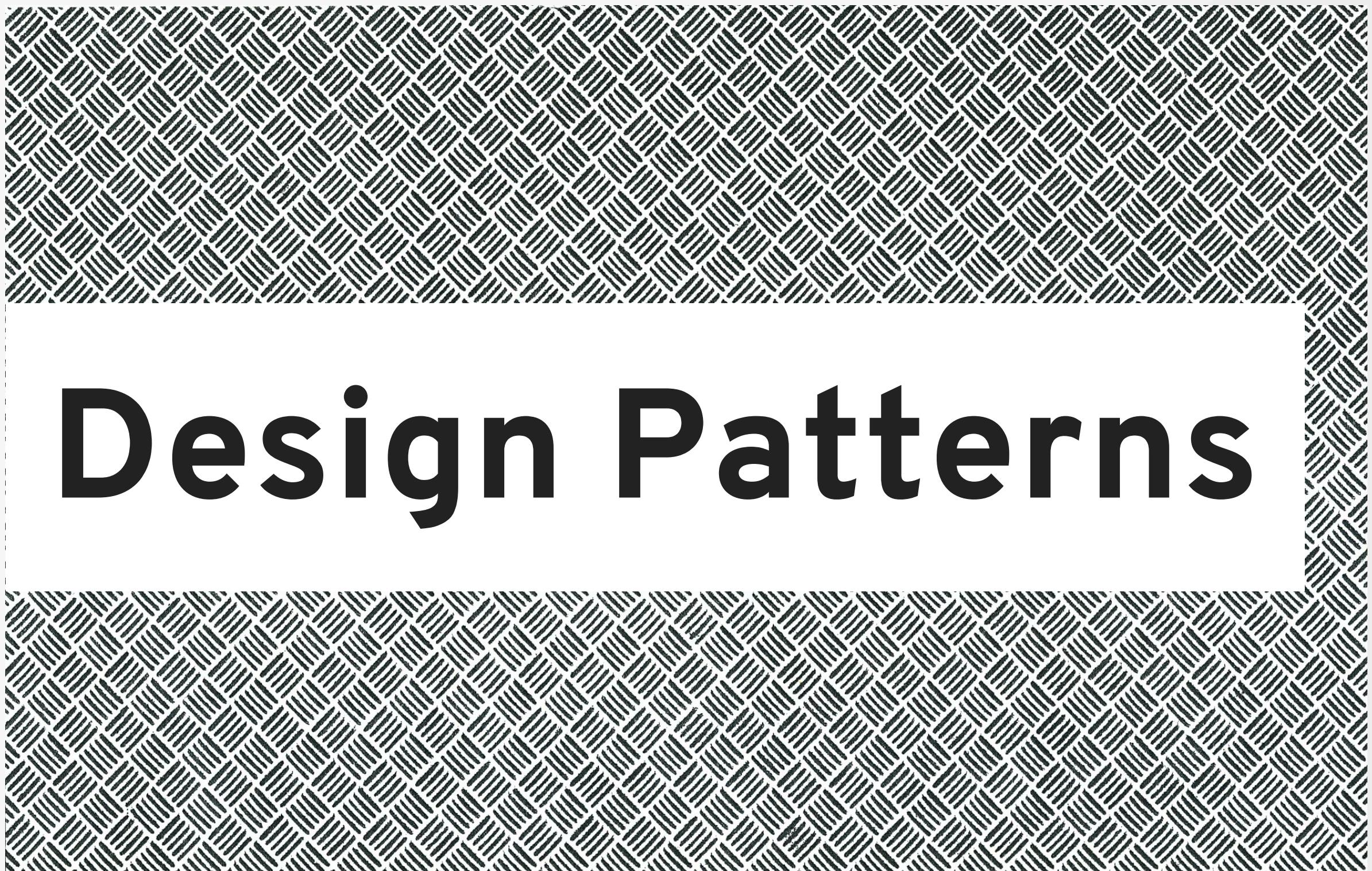
- Kubernetes
- Patterns
- Categories:
 - Foundational Patterns
 - Structural Patterns
 - Configurational Patterns
 - Advanced Patterns



KUBERNETES

- Open Source container orchestration system
 - Scheduling
 - Horizontal scaling
 - Self-healing
 - Service discovery
 - Rollout and Rollbacks
- Declarative, resource-centric REST API

Design Patterns



DESIGN PATTERN

A **Design Pattern** describes a
repeatable solution to a
software engineering **problem**.

A Pattern Language

Towns • Buildings • Construction



Christopher Alexander

Sara Ishikawa • Murray Silverstein

WITH

Max Jacobson • Ingrid Fiksdahl-King

Shlomo Angel

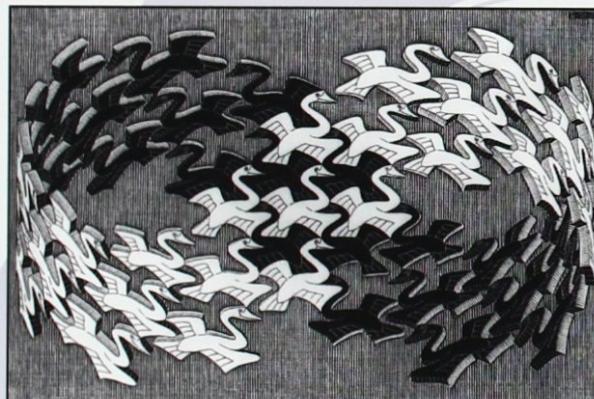


redhat

Design Patterns

Elements of Reusable
Object-Oriented Software

Erich Gamma
Richard Helm
Ralph Johnson
John Vlissides



Cover art © 1994 M.C. Escher / Cordon Art - Baarn - Holland. All rights reserved.

Foreword by Grady Booch

ADDISON-WESLEY PROFESSIONAL COMPUTING SERIES



The Addison-Wesley Signature Series

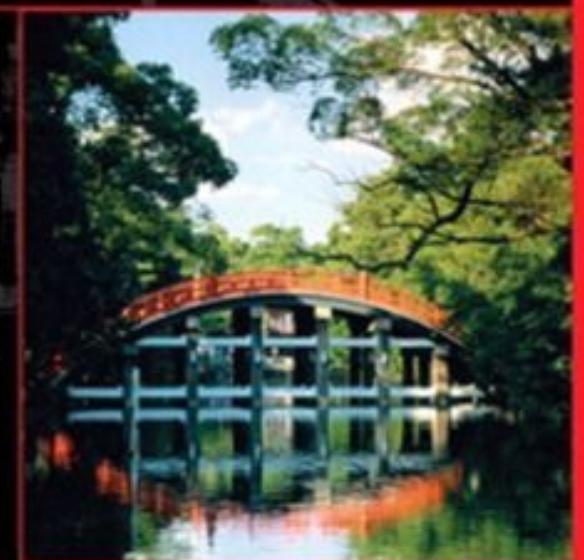
ENTERPRISE INTEGRATION PATTERNS

DESIGNING, BUILDING, AND
DEPLOYING MESSAGING SOLUTIONS

GREGOR HOHPE
BOBBY WOOLF

WITH CONTRIBUTIONS BY

KYLE BROWN
CONRAD F. D'CRUZ
MARTIN FOWLER
SEAN NEVILLE
MICHAEL J. RETTIG
JONATHAN SIMON

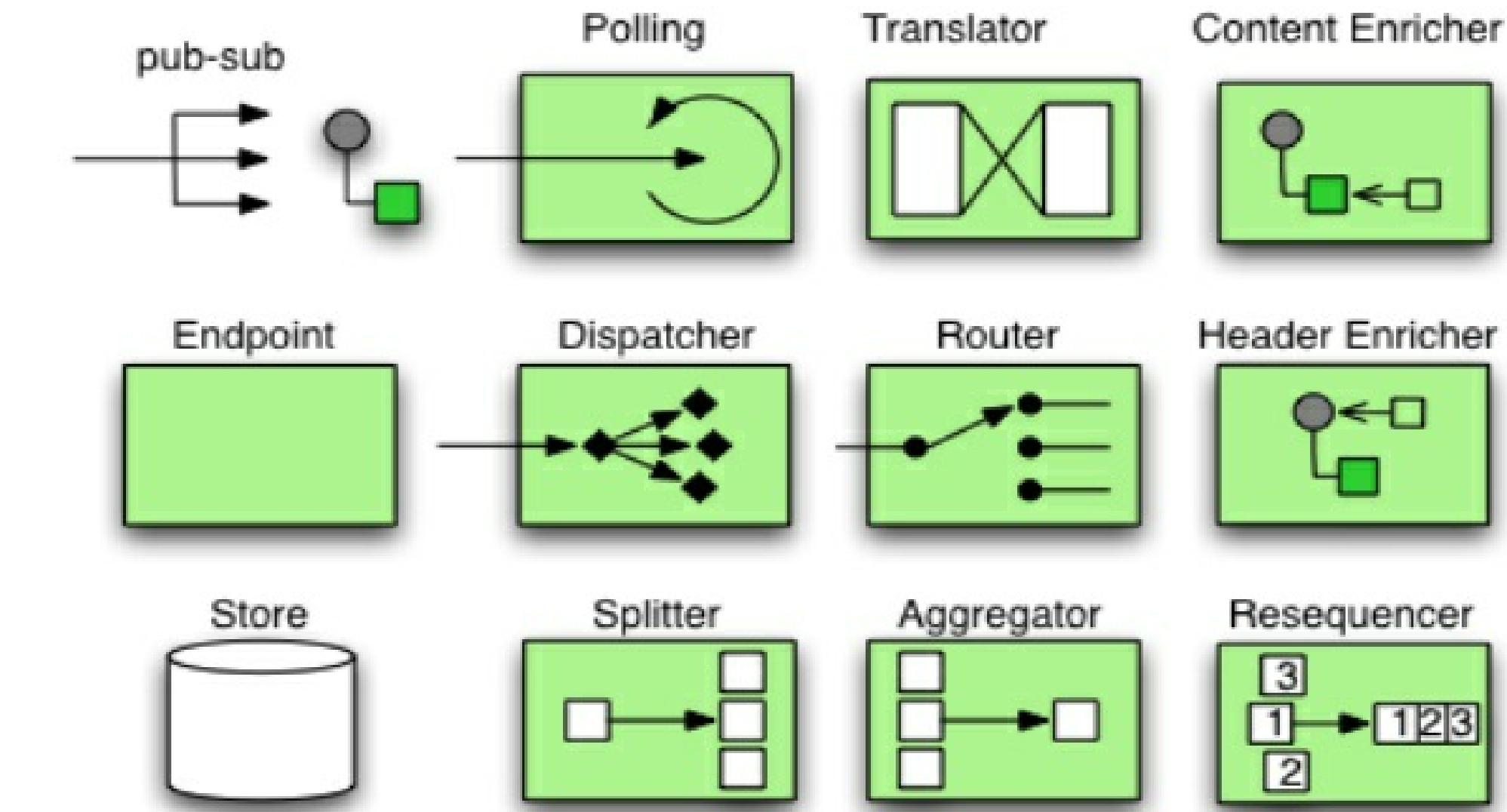


Forewords by John Crupi and Martin Fowler



A MARTIN FOWLER SIGNATURE
BOOK

Enterprise Integration Pattern



Kubernetes Patterns



Patterns, Principles, and Practices
for Designing Cloud Native Applications

Bilgin Ibryam & Roland Huss

<https://leanpub.com/k8spatterns>



STRUCTURE

- Problem
- Patterns:
 - Name
 - Solution
- <http://www.martinfowler.com/articles/writingPatterns.html>

FOUNDATIONAL PATTERNS

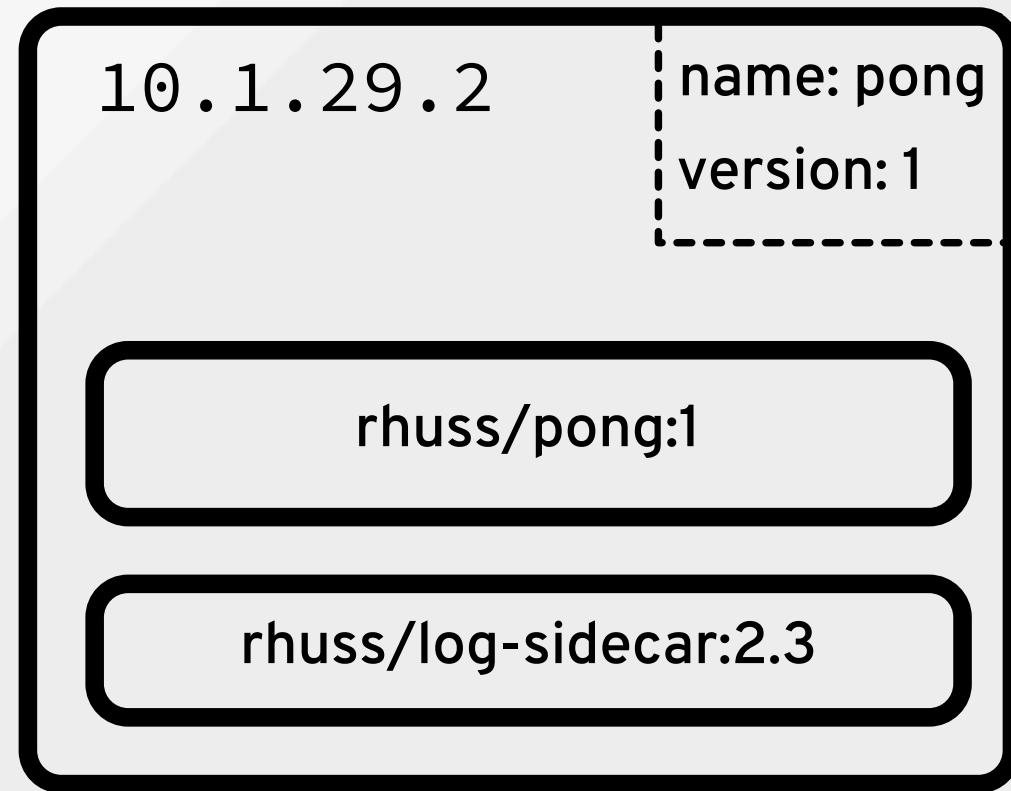
Automatable Unit

How can we create and manage applications with Kubernetes ?

- **Pods:** Atomic unit of containers
- **Services:** Entry point to pods
- Grouping via **Labels,**
Annotations, Namespaces

POD

- Kubernetes Atom
- One or more containers sharing:
 - IP and ports
 - Volumes
- Ephemeral IP address



POD DECLARATION

```
apiVersion: v1
kind: Pod
metadata:
  name: pong
  labels:
    name: pong
    version: "1"
spec:
  containers:
    - image: "rhuss/pong:1"
      name: pong
      ports:
        - containerPort: 8080
    - image: "rhuss/log-sidecar:2.3"
      name: log
```

REPLICA SET

- Responsible for managing **Pods**
- **replicas** : Number of **Pod** copies to keep
- Label selector chooses **Pods**
- Holds a template for creating new **Pods**

ReplicaSet

replicas: **3**

Selector:

name: pong
version: 1

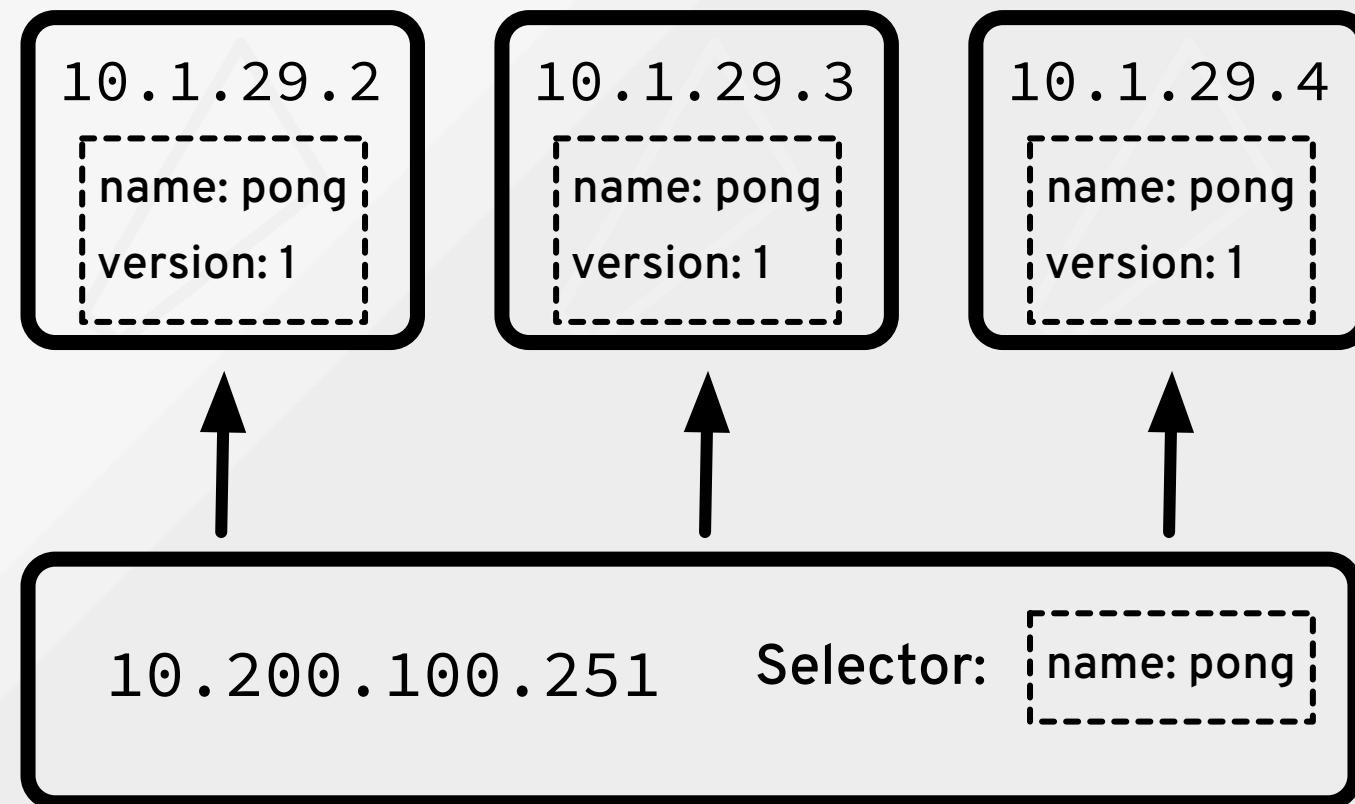
10.1.29.2
name: pong
version: 1

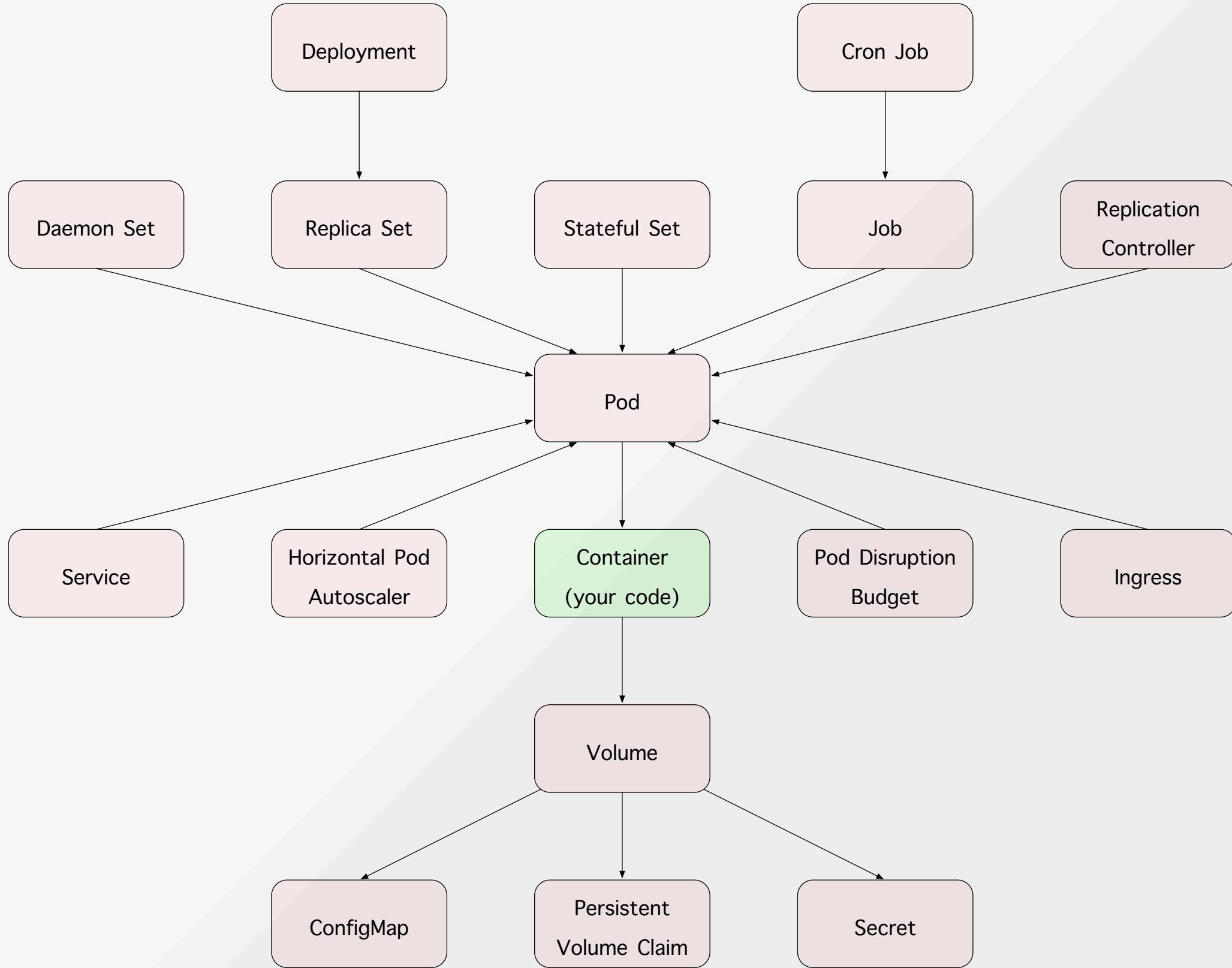
10.1.29.3
name: pong
version: 1

10.1.29.4
name: pong
version: 1

SERVICE

- Entrypoint for a set of **Pods**
- **Pods** chosen by **Label** selector
- Permanent IP address





Predictable Demands

How can we handle resource requirements deterministically ?

- Requirements should be declared to help in:
 - Matching infrastructure services
 - Scheduling decisions
 - Capacity planning

RUNTIME DEPENDENCIES

- Persistent Volumes
- Host ports
- Configuration via **ConfigMaps** and
Secrets

RESOURCE PROFILES

- Resources:
 - CPU, Network (compressible)
 - Memory (incompressible)
- App: Declaration of resource **requests and limits**
- Platform: Resource quotas and limit ranges

```
apiVersion: v1
kind: Pod
metadata:
  name: http-server
spec:
  containers:
  - image: nginx
    name: nginx
  resources:
    requests:
      cpu: 200m
      memory: 100Mi
    limits:
      cpu: 300m
      memory: 200Mi
```

QOS CLASSES

- **Best Effort**
 - No requests or limits
- **Burstable**
 - requests < limits
- **Guaranteed**
 - requests == limits

Declarative Deployment

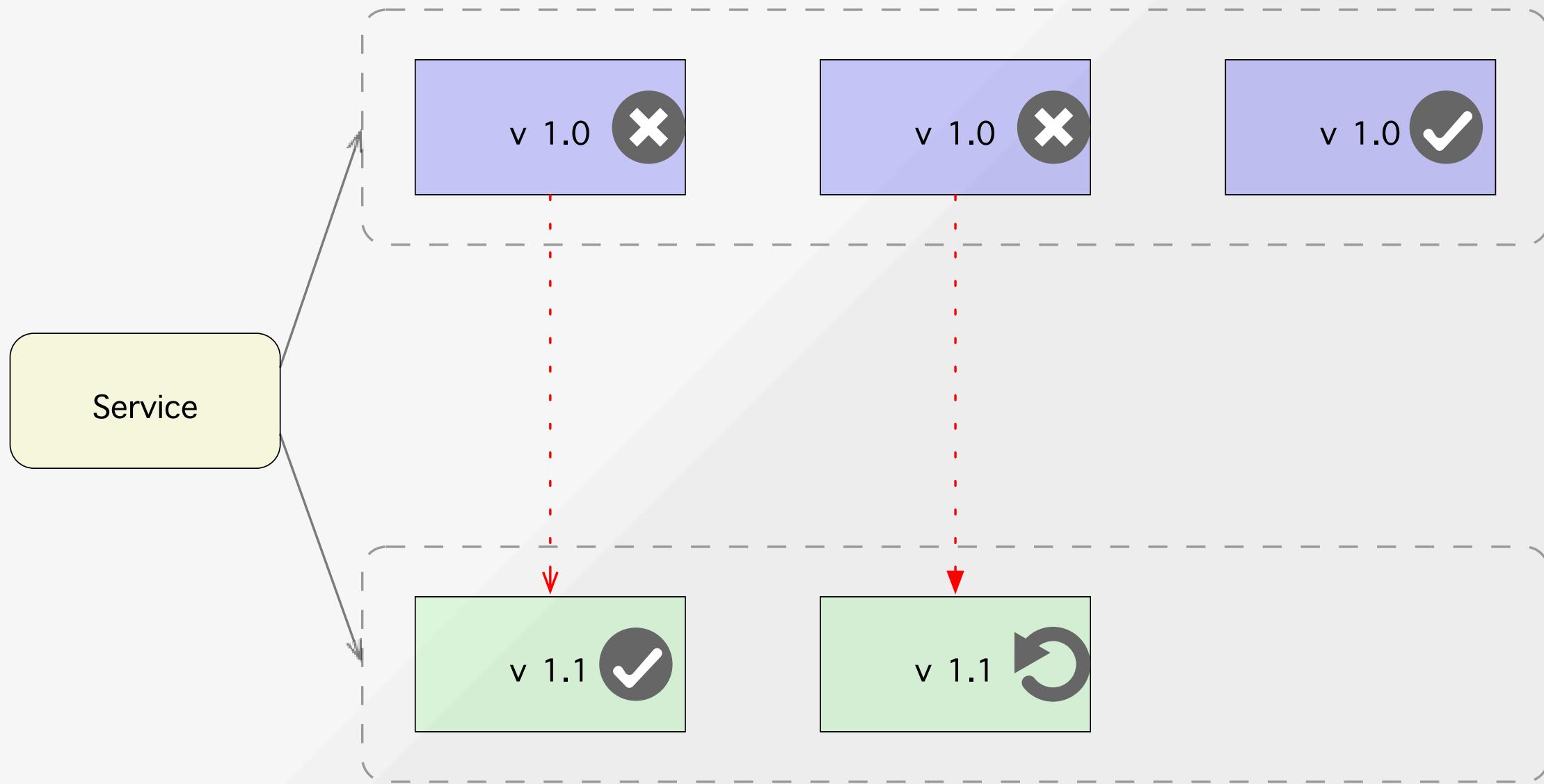
How can applications be deployed
and updated ?

- Declarative versus Imperative deployment
- Various update strategies

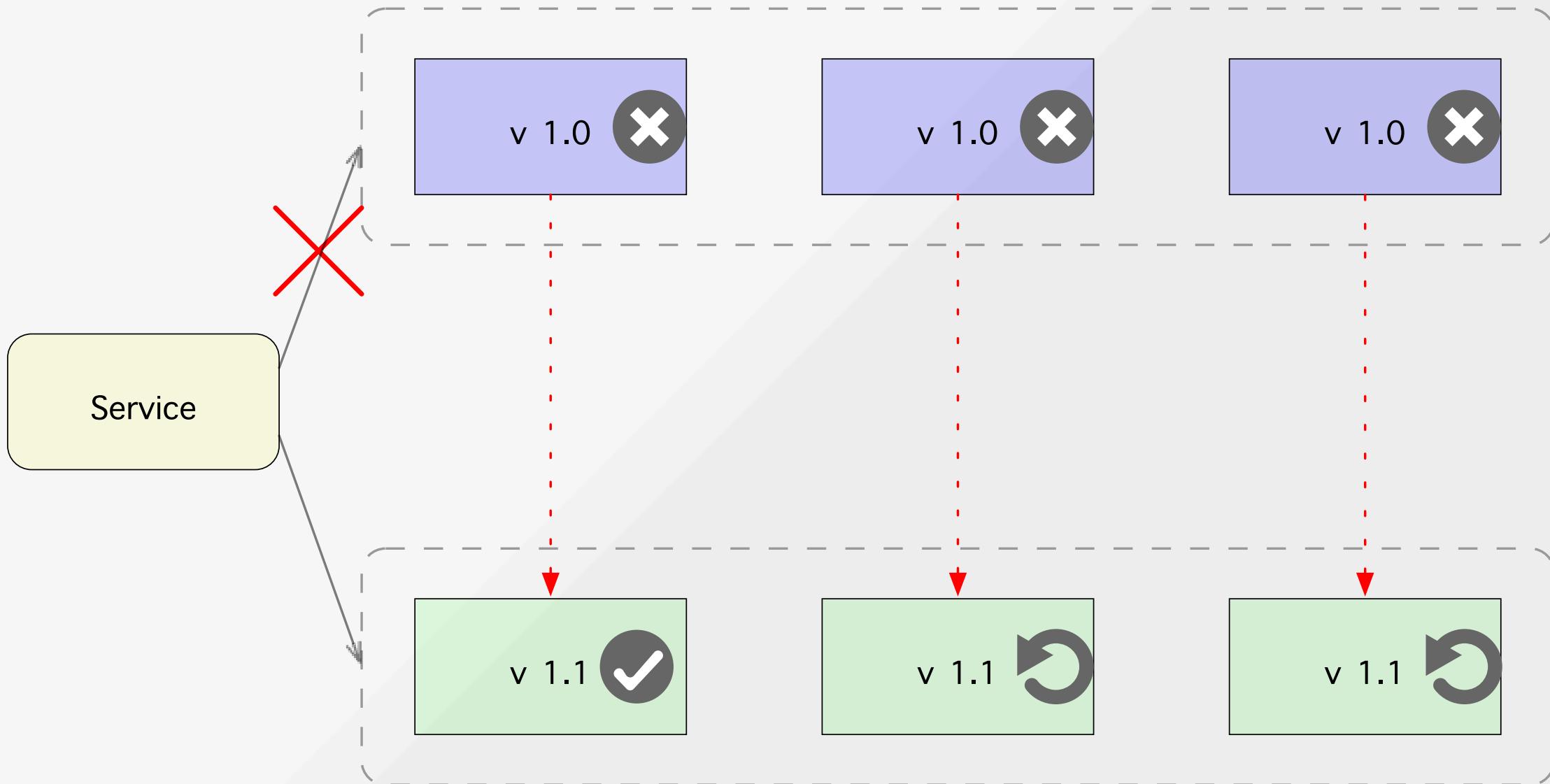
DEPLOYMENT

- Holds template for Pod
- Creates ReplicaSet on the fly
- Allows rollback
- Update strategies declarable
- Inspired by DeploymentConfig from OpenShift

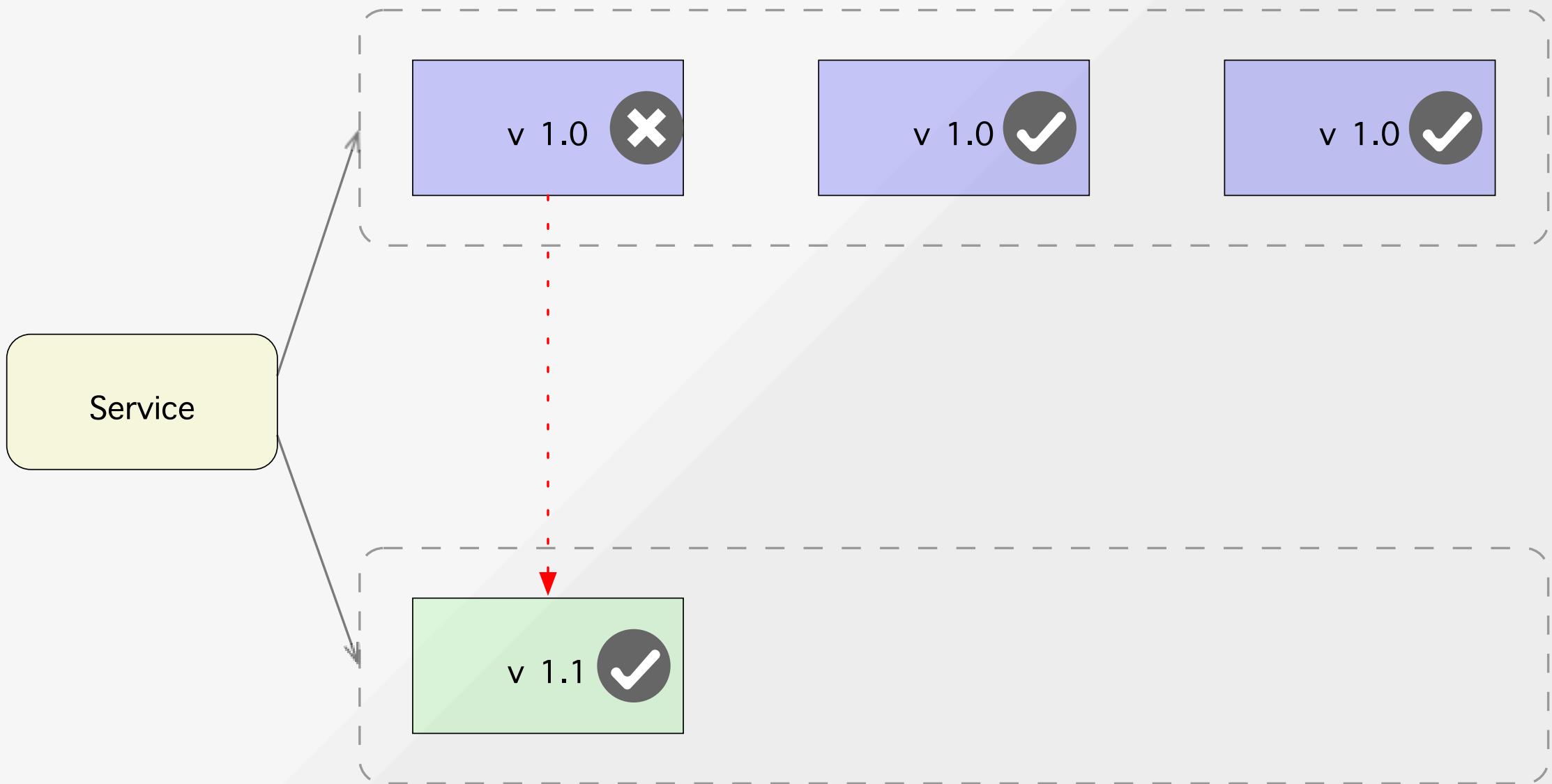
ROLLING



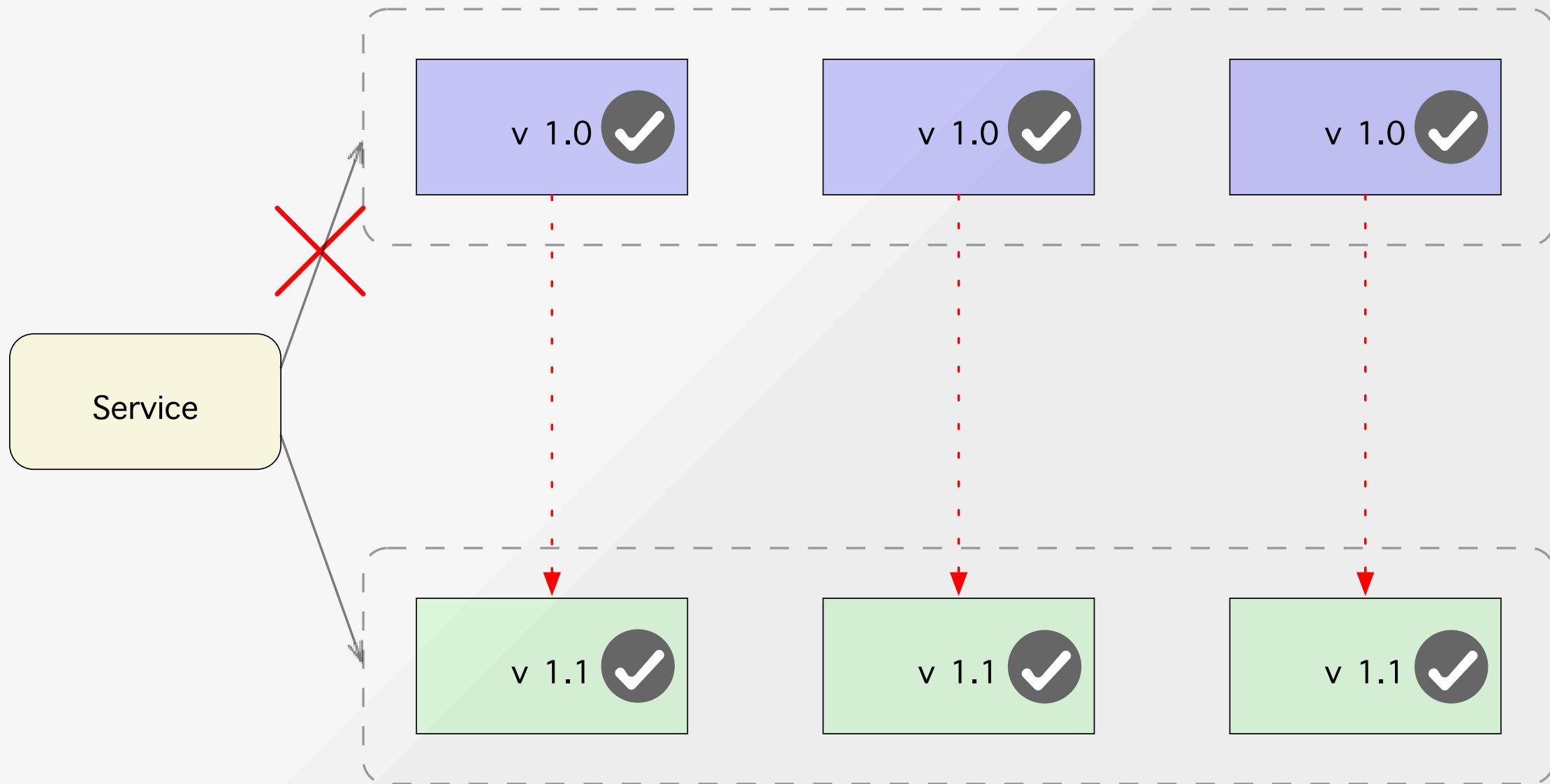
FIXED



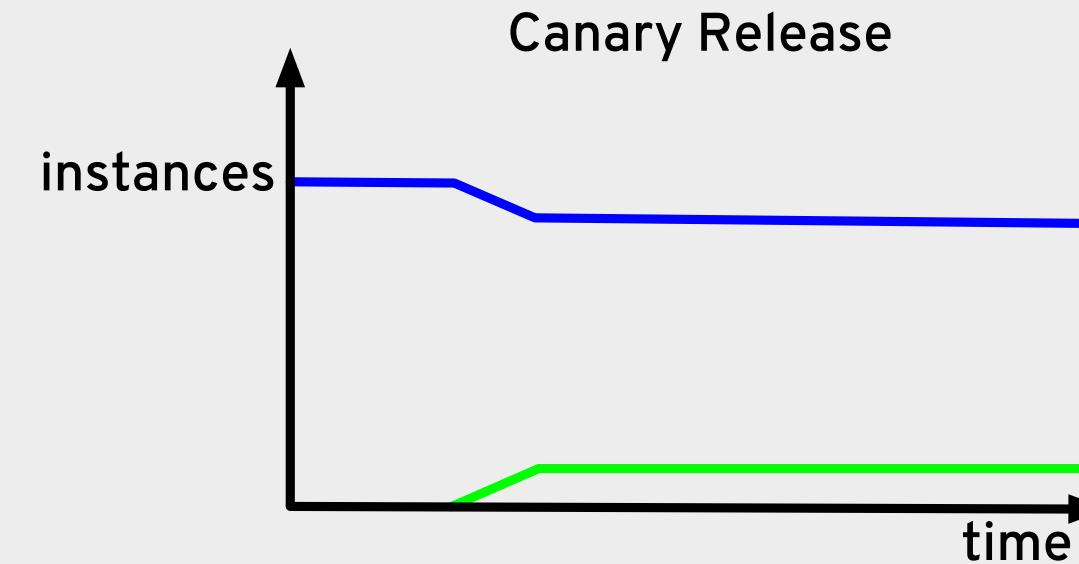
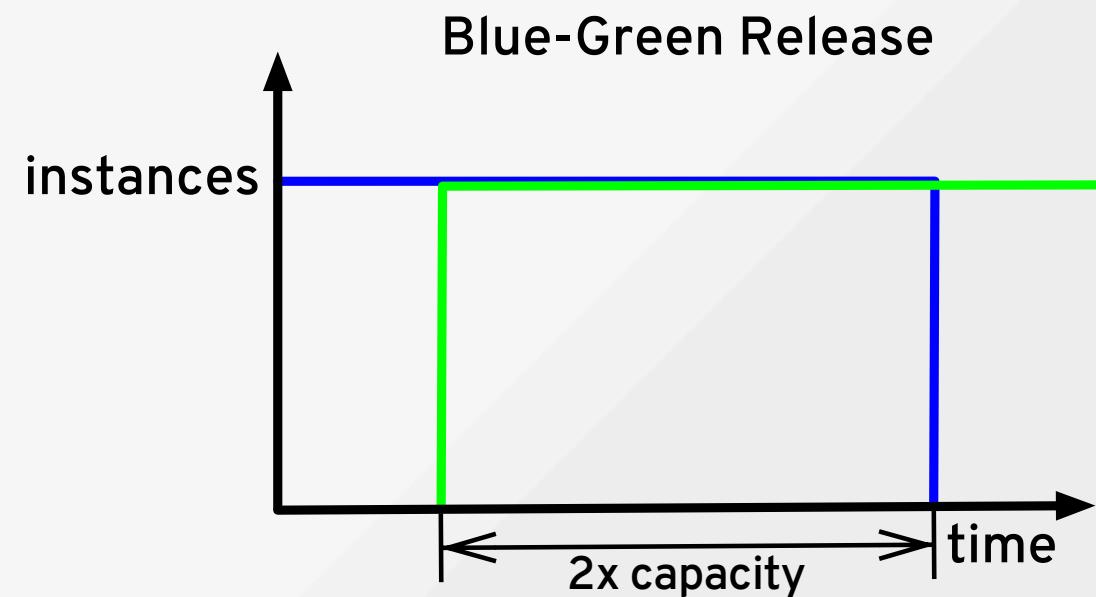
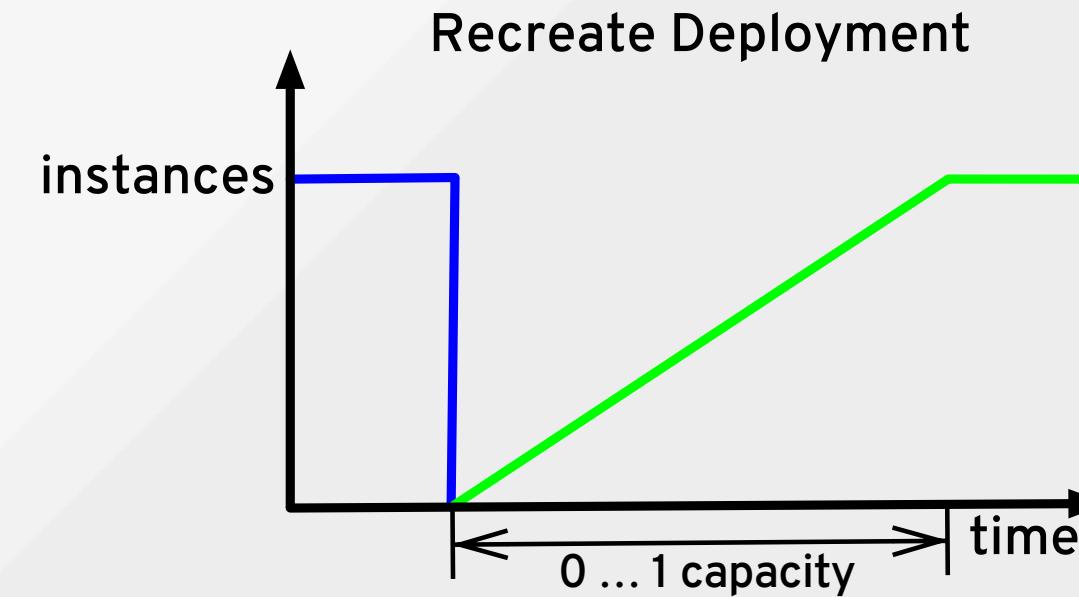
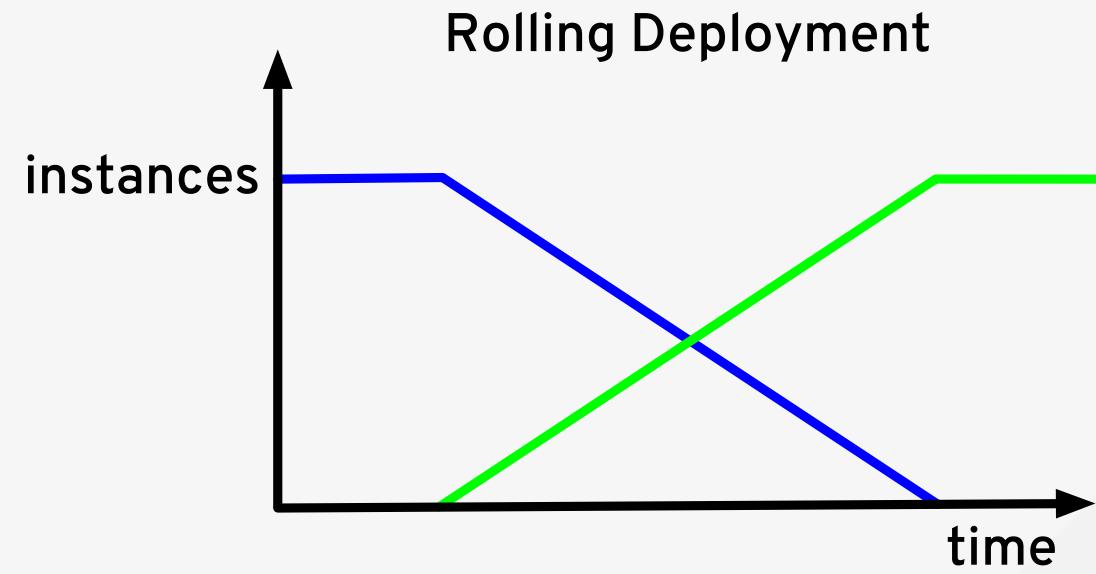
CANARY



BLUE-GREEN



SUMMARY

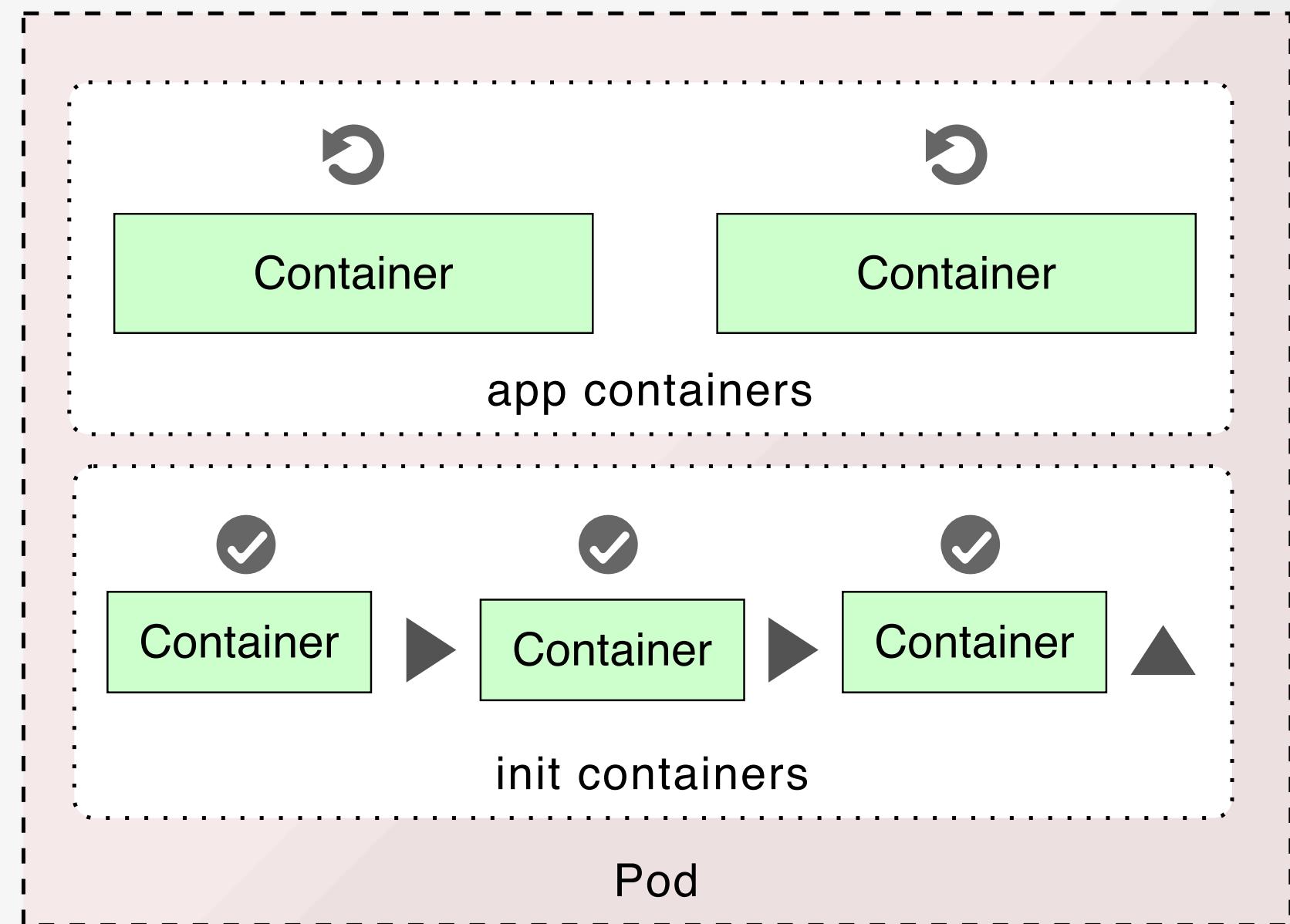


STRUCTURAL PATTERNS

Initializer

How can I initialize my containerized applications ?

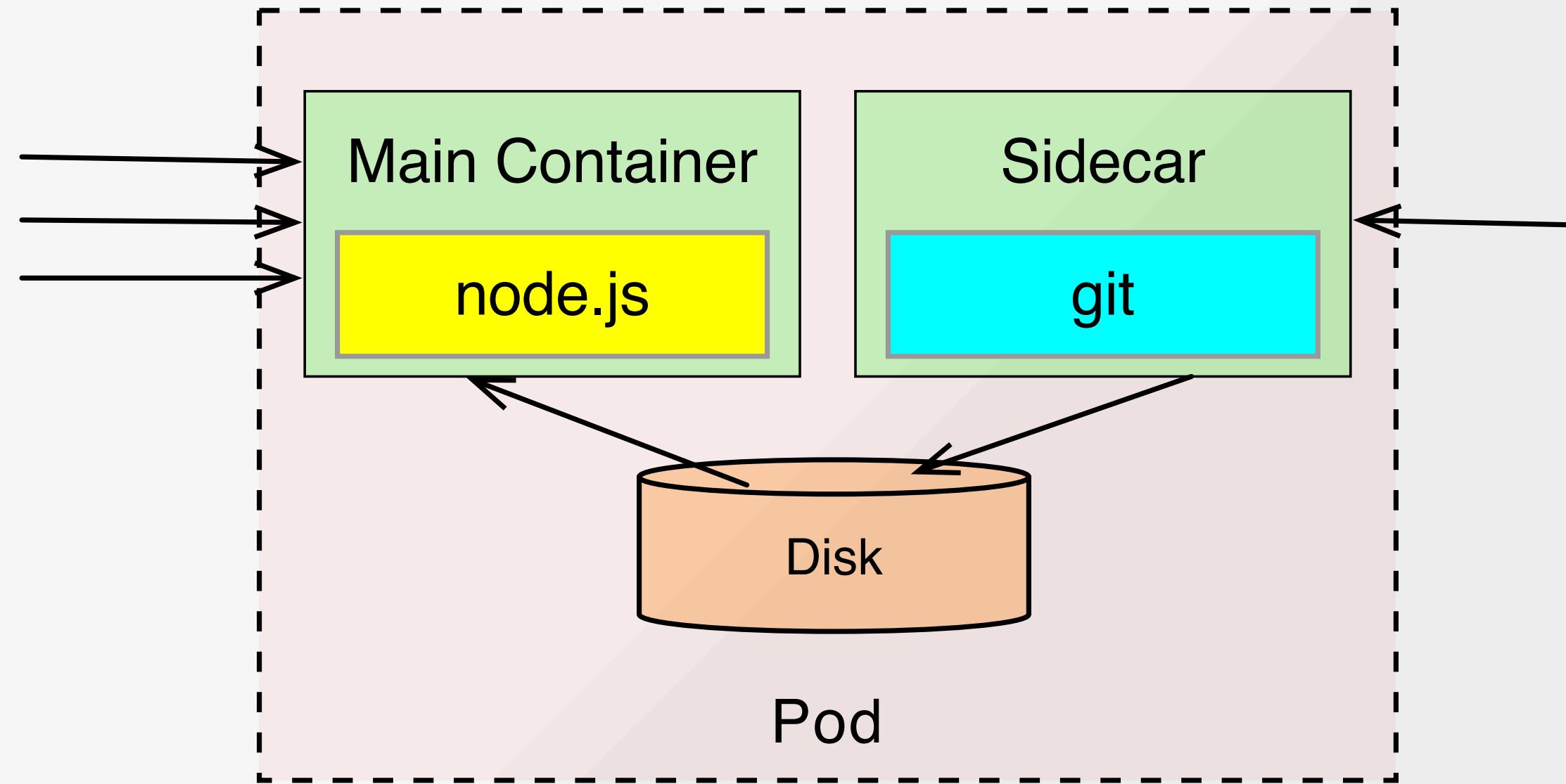
- Init container :
 - Part of a Pod
 - One shot action before Pod starts
 - Needs to be idempotent
 - Has own resource requirements



Sidecar

How can I extend the functionality of
an existing container ?

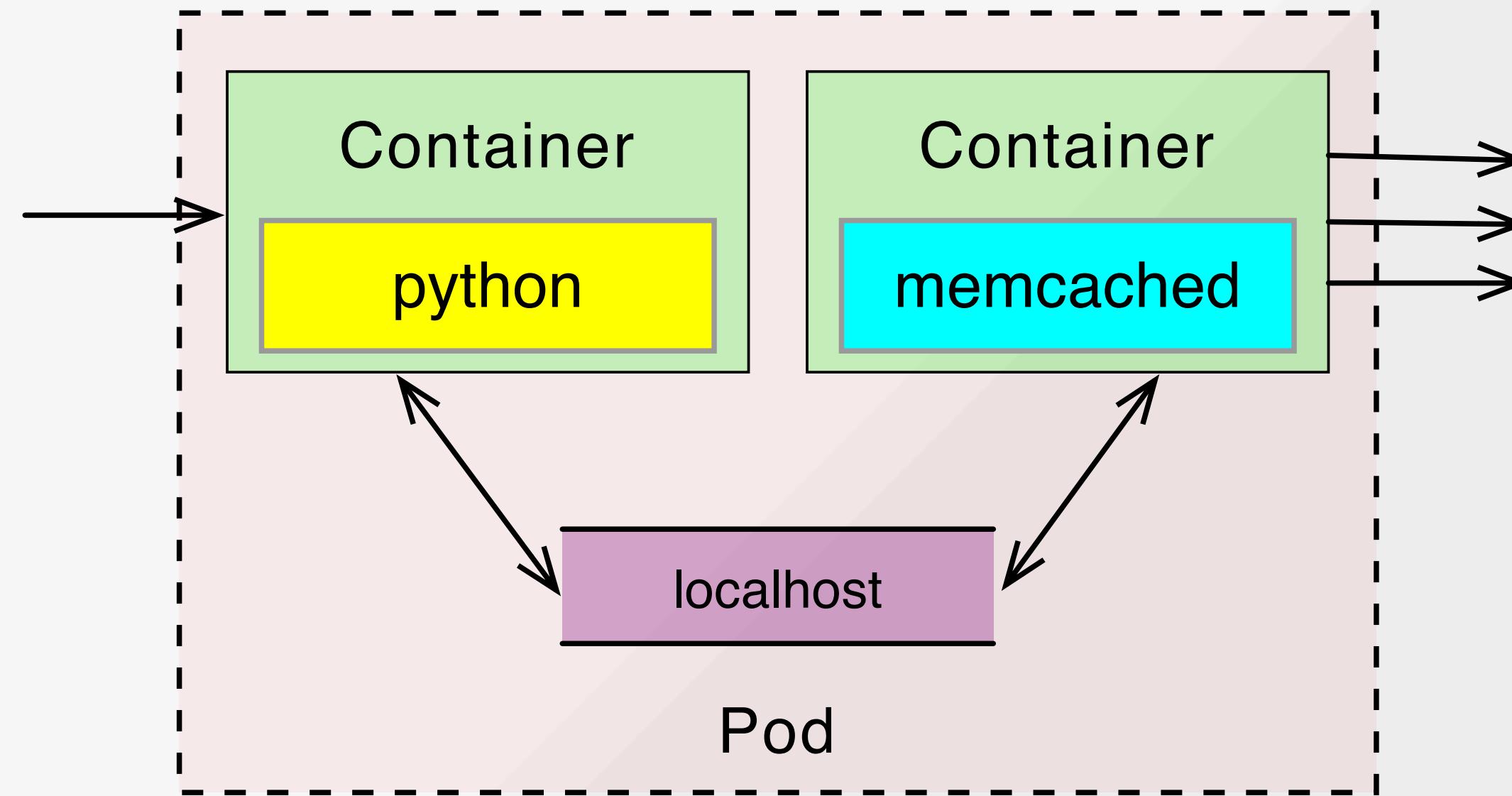
- Runtime collaboration of containers
- Connected via shared resources:
 - Network
 - Volumes



Ambassador

How to decouple a container's
access **to** the outside world ?

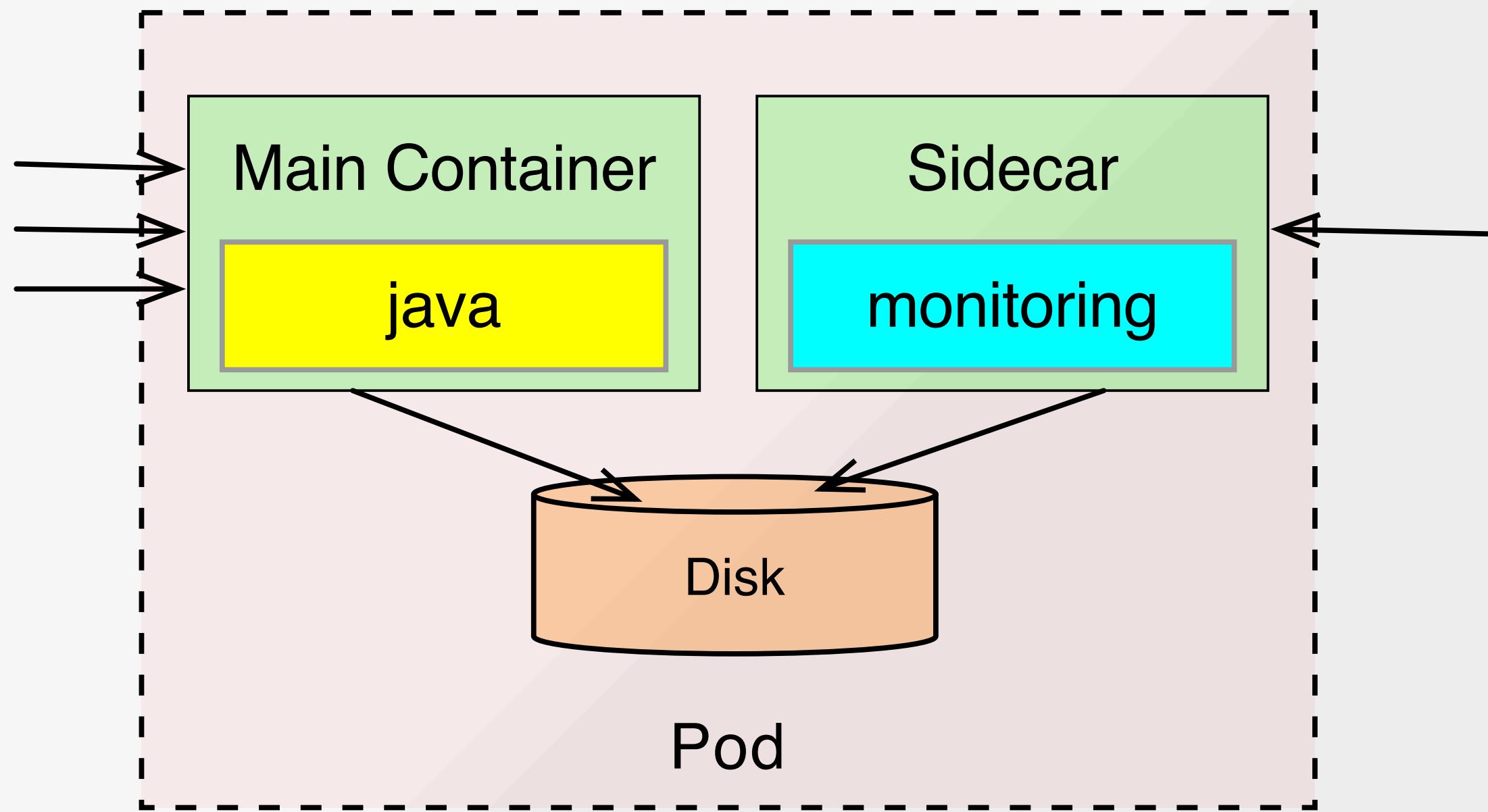
- Also known as **Proxy**
- Specialization of a Sidecar
- E.g. infrastructure services
 - Circuit breaker
 - Tracing



Adapter

How to decouple access to a container **from** the outside world ?

- Opposite of Ambassador
- Uniform access to application
- Examples:
 - Monitoring
 - Logging



CONFIGURATIONAL PATTERNS

How can applications be configured
for different environments ?

EnvVar Configuration

- Universal applicable
- Recommended by the Twelve Factor App manifesto
- Can be only set during startup of application

```
kind: Pod
spec:
  containers:
    - env:
        - name: DB_HOST
          value: "prod-database.prod.intranet"
        - name: DB_PASSWORD
          valueFrom:
            secretKeyRef:
              name: "db-passwords"
              key: "monogdb.password"
        - name: DB_USER
          valueFrom:
            configMapKeyRef:
              name: "db-users"
              key: "mongodb.user"
  image: acme/bookmark-service:1.0.4
```

Configuration Resource

- ConfigMap and Secret : Intrinsic K8s resources
- Can be used in two ways:
 - Reference for environment variables
 - Files mapped to a volume

```
kind: ConfigMap
metadata:
  name: spring-boot-config
data:
  JAVA_OPTIONS: "-Xmx512m"
  application.properties: |
    welcome.message=Hello !!!
    server.port=8080
```

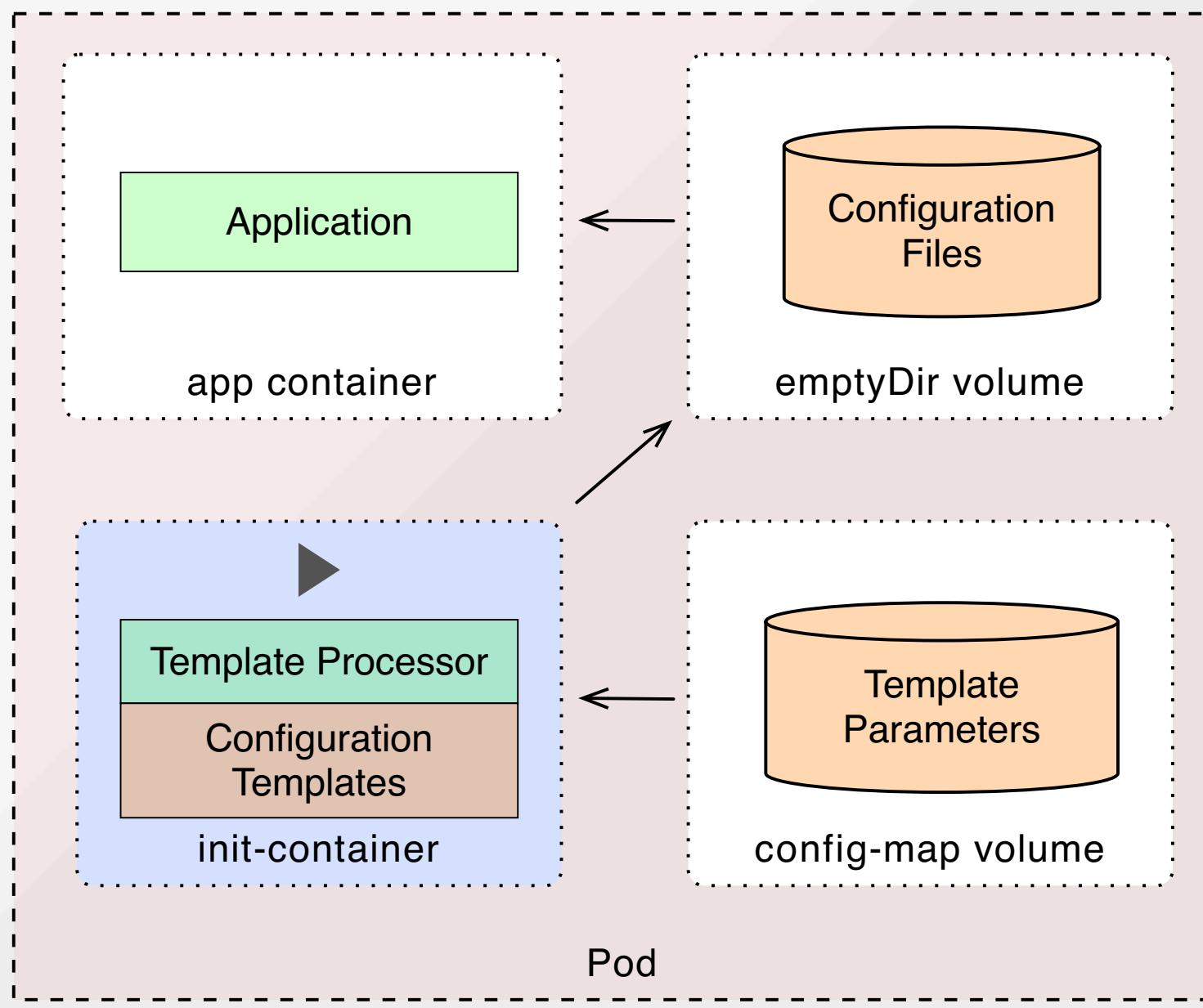
```
kind: Pod
spec:
  containers:
  - name: web
    volumeMounts:
    - name: config-volume
      mountPath: /etc/config
    # ...
  volumes:
  - name: config-volume
    configMap:
      name: spring-boot-config
```



Configuration Template

- ConfigMap not suitable for large configuration
- Managing similar configuration
- Ingredients:
 - Init-container with template processor and templates
 - Parameters from a ConfigMap Volume

CONFIGURATION TEMPLATE



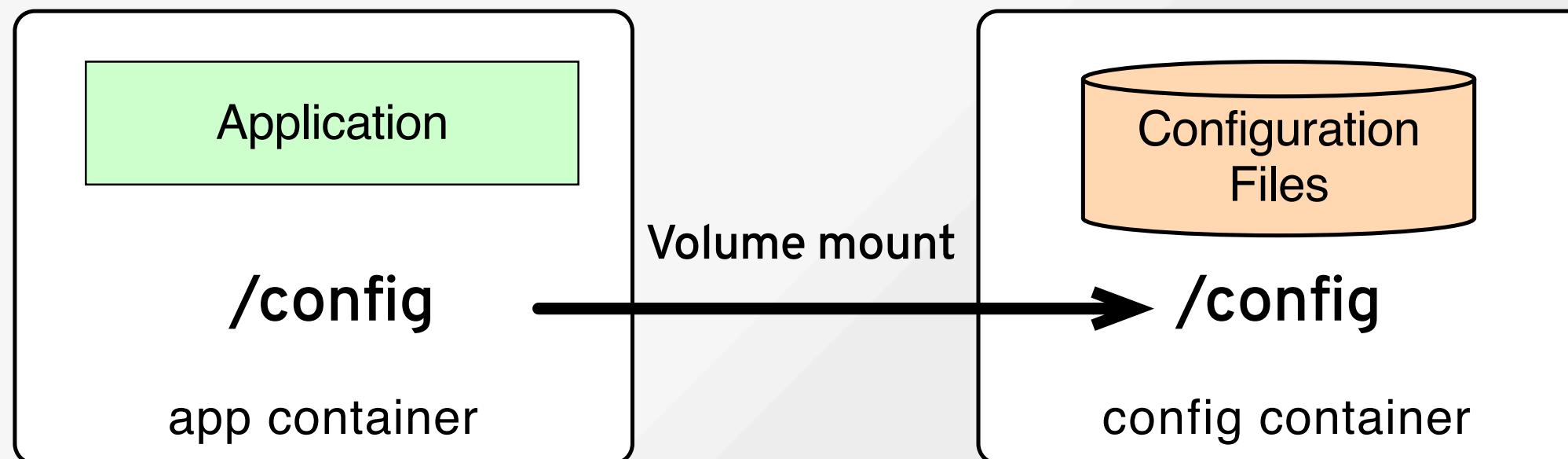
DISCUSSION

- Good for large, similar configuration sets per environment
- Parameterisation via **ConfigMaps** easy
- More complex

Immutable Configuration

- Configuration is put into a container itself
- Configuration container is linked to application container during runtime

CONTAINER VOLUMES

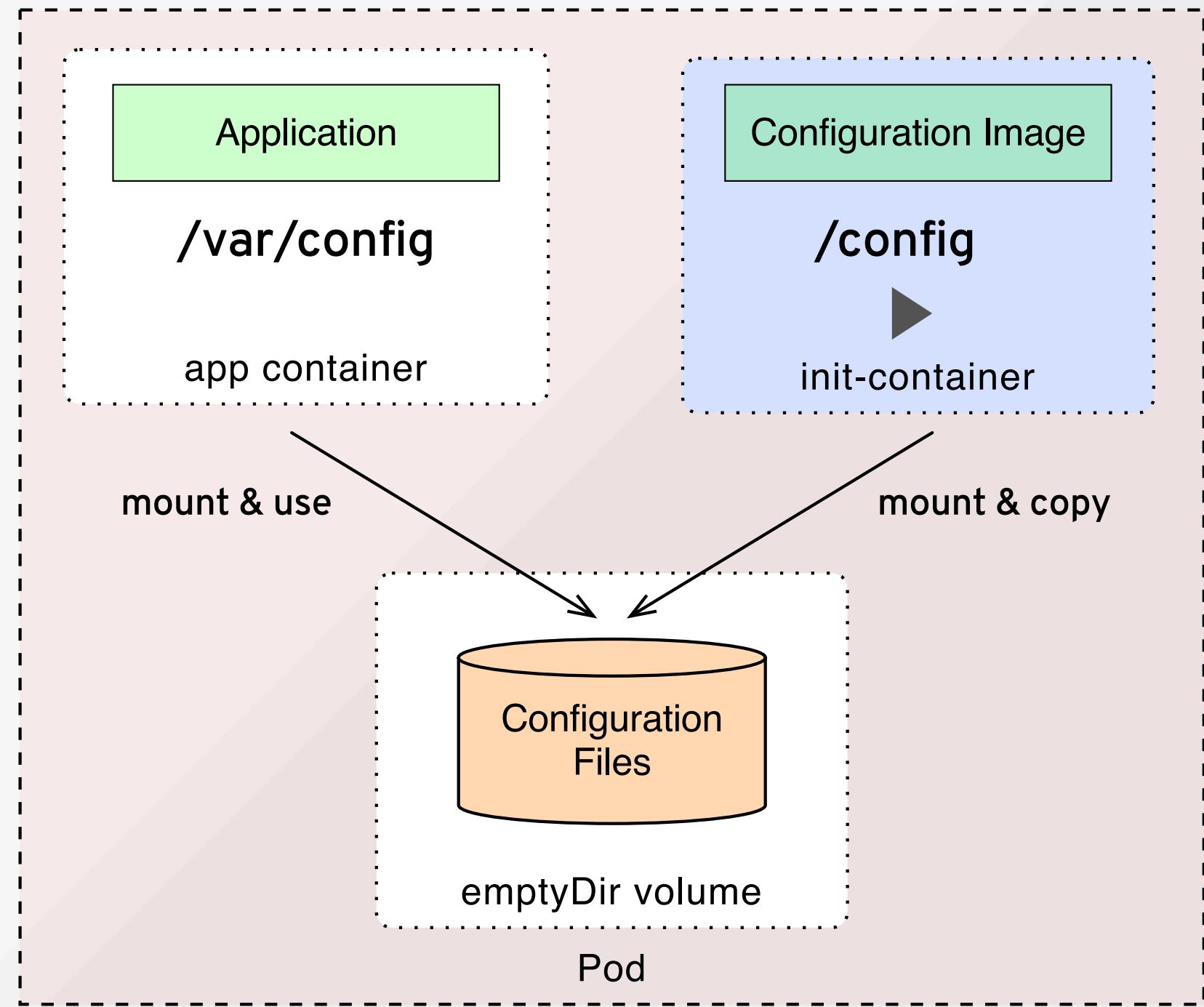


- Not directly supported by K8s
- docker-flexvol : K8s FlexVolume driver for Docker volumes

```
kind: Pod
metadata:
  name: nginx
spec:
  containers:
    - name: nginx
      image: nginx
    volumeMounts:
      - name: test
        mountPath: /data
    ports:
      - containerPort: 80
  volumes:
    - name: test
      flexVolume:
        driver: "dims.io/docker-flexvol"
        options:
          image: "my-container-image"
          name: "/data-store"
```



INITIALIZER



- Dockerfile for init container:

```
FROM busybox

ADD dev.properties /config-src/demo.properties
# ... add more to /config-src

# Using a shell here in order to resolve wildcards
ENTRYPOINT [ "sh", "-c", \
    "cp /config-src/* $1", "--" ]
```

- Build config image:

```
docker build -t k8spatterns/config-dev:1 .
```

```
spec:  
  initContainers:  
    - image: k8spatterns/config-dev:1  
      name: init  
      args:  
        - "/config"  
    volumeMounts:  
      - mountPath: "/config"  
        name: config-directory  
  
  containers:  
    - image: k8spatterns/demo:1  
      name: demo  
    volumeMounts:  
      - mountPath: "/config"  
        name: config-directory  
  
  volumes:  
    - name: config-directory  
      emptyDir: {}
```

DISCUSSION

- Immutable Configuration ...
 - can be versioned
 - can be distributed via a registry
 - is immutable
 - can be arbitrary large
- Parameterisation via OpenShift Templates

ADVANCED PATTERNS

Custom Controller

How can I extend the platform itself
without changing it ?

- Watching resources by registering for Kubernetes events
- Reacting on changes in resource declarations

CONTROLLER

- Managed pod listening for Kubernetes API events
- **State Reconciliation** : Make the current state like the declared desired state
- Often used in combination with **CustomResources**

CATEGORIES

- **Extension Controller**: Extend the Kubernetes platform itself
- **Application Controller**: Combine Kubernetes with an application specific domain

Custom Resource

How can I manage custom domain specific resources ?

- **Custom Resource Definition** (CRD) managed by Kubernetes
- Accessible via the Kubernetes API
- Watched by Custom Controllers

EXAMPLE CRD

```
apiVersion: apiextensions.k8s.io/v1beta1
kind: CustomResourceDefinition
metadata:
  name: prometheuses.monitoring.coreos.com
spec:
  group: monitoring.coreos.com
  names:
    kind: Prometheus
    plural: prometheuses
  scope: Namespaced
  version: v1
  validation: ....
```

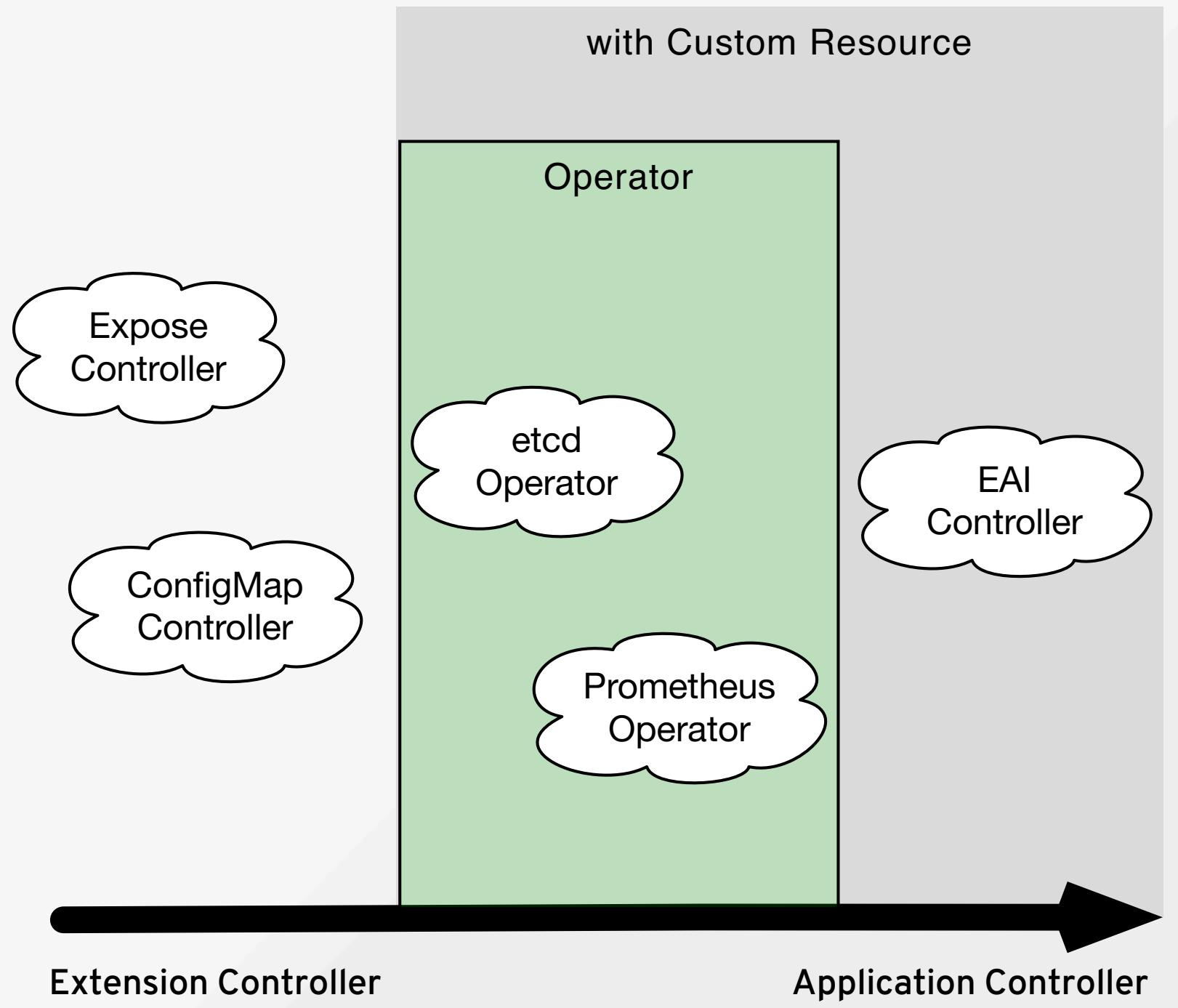
EXAMPLE CRD

```
apiVersion: monitoring.coreos.com/v1
kind: Prometheus
metadata:
  name: prometheus
spec:
  serviceMonitorSelector:
    matchLabels:
      team: frontend
  resources:
    requests:
      memory: 400Mi
```

OPERATORS

- Combine Custom Controller and Custom Resource
- Manages and deploys custom Kubernetes application
- Operator Framework by CoreOS:
 - Operator SDK
 - Operator Lifecycle Manager
 - Operator Metering

SPECTRUM



WRAP UP

- Kubernetes offers a rich feature set to manage containerised applications.
- Patterns can help in solving recurring Kubernetes challenges.
- More and more patterns are emerging.



QUESTIONS ?

Twitter ro14nd

Book <https://leanpub.com/k8spatterns>

Slides <https://github.com/ro14nd-talks/kubernetes-patterns>