



# KUBERNETES PATTERNS

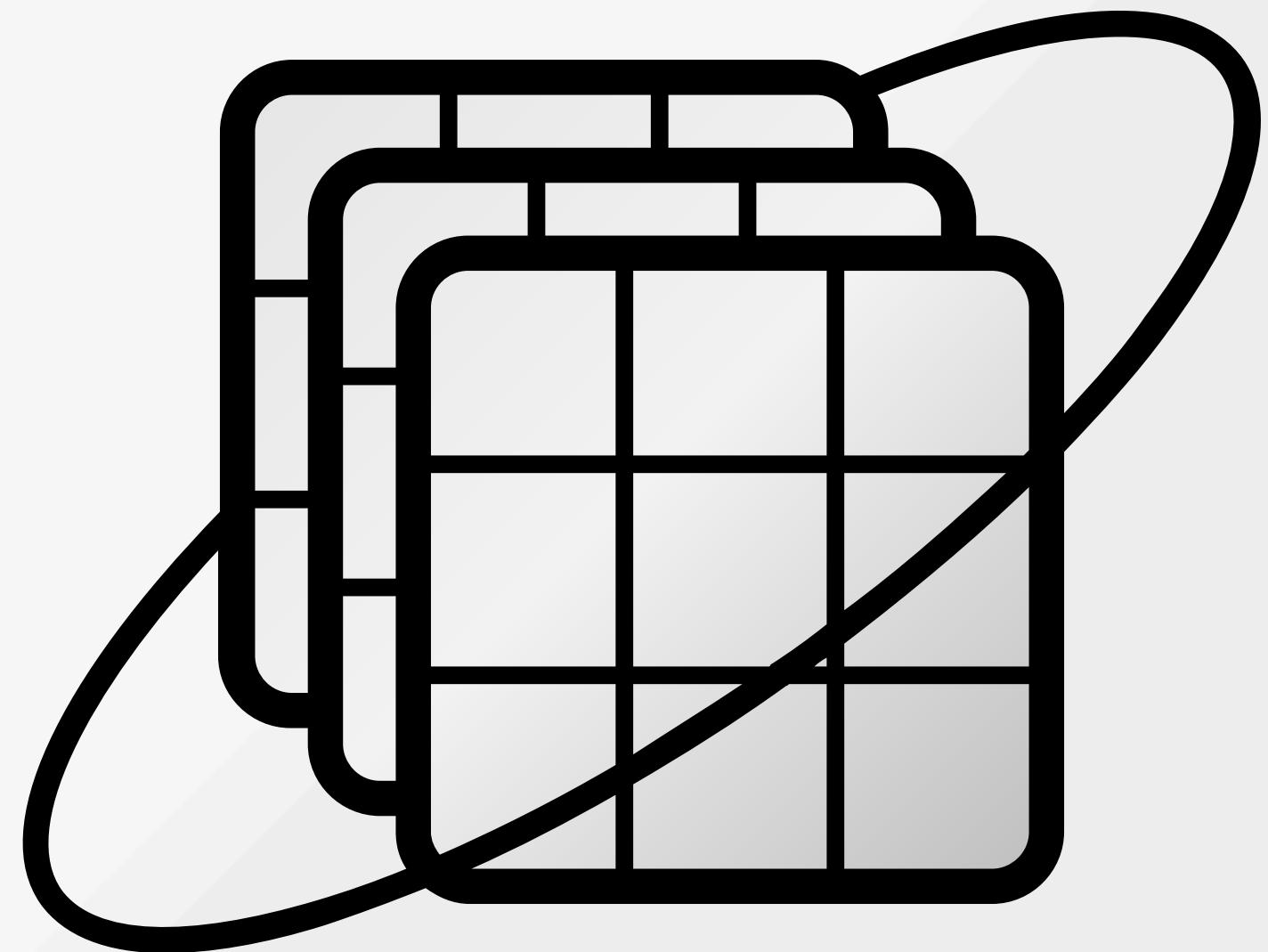
Dr. Roland Huß, Red Hat, @ro14nd

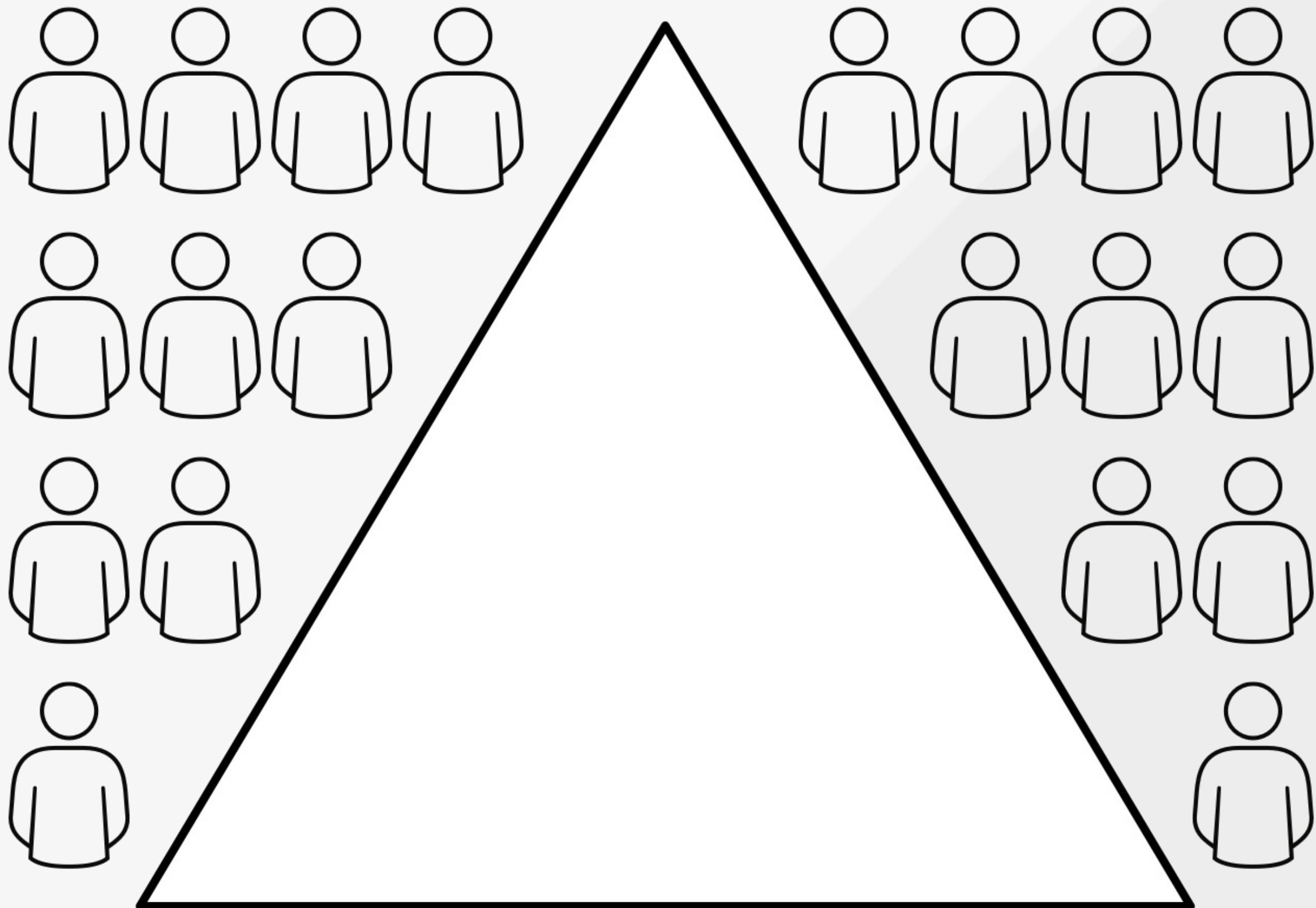


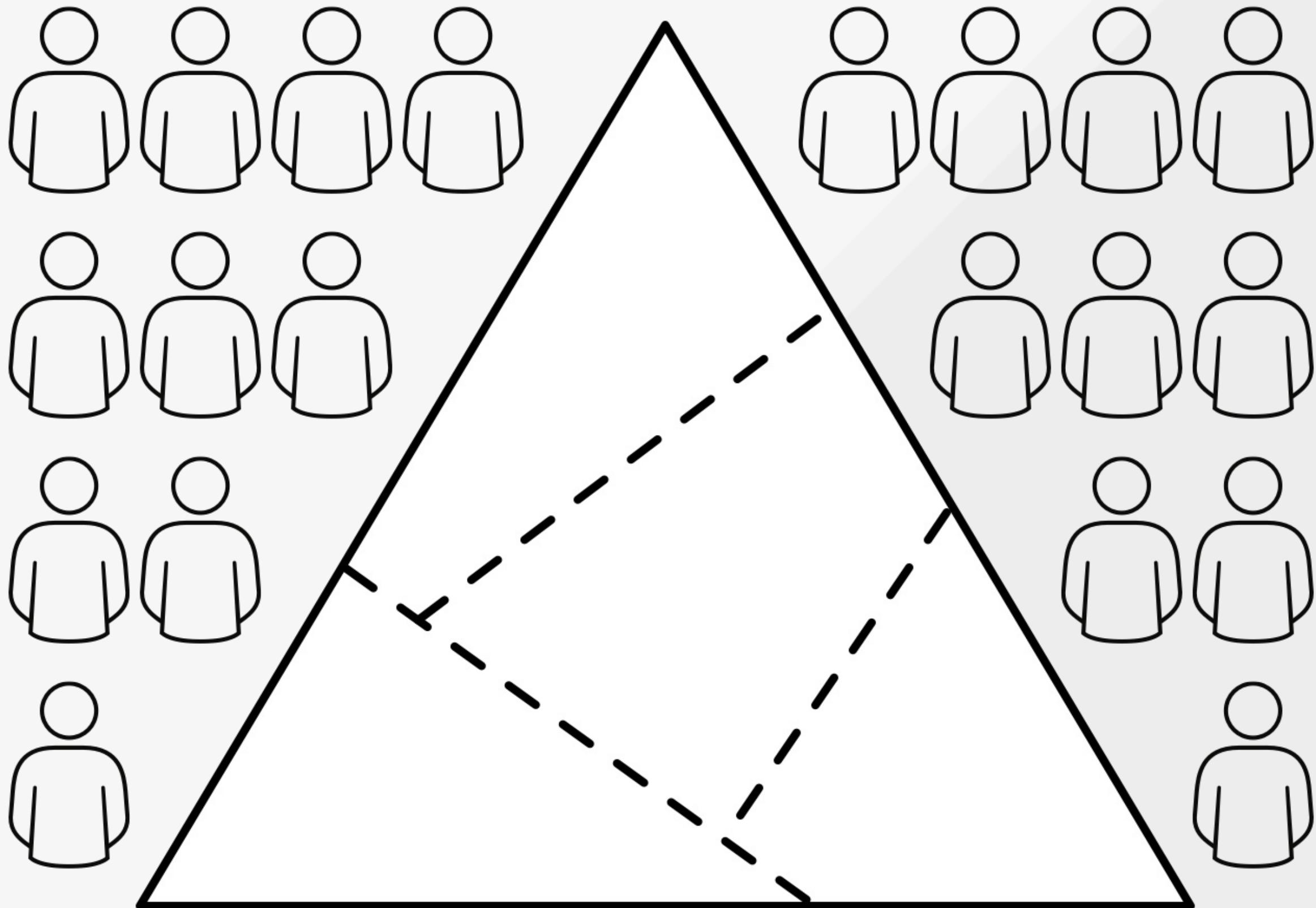
"m" for menu, "?" for other shortcuts

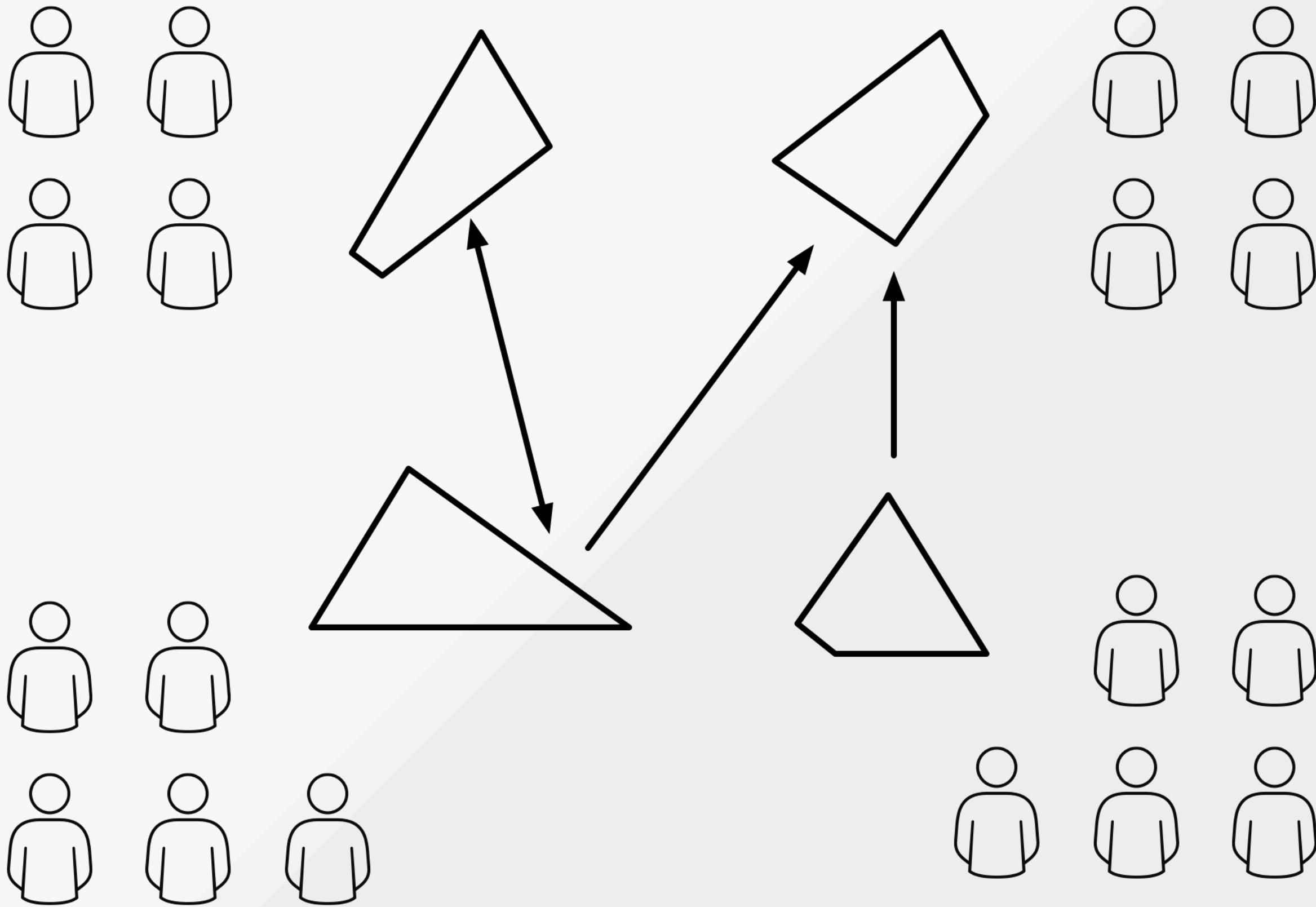
# AGENDA

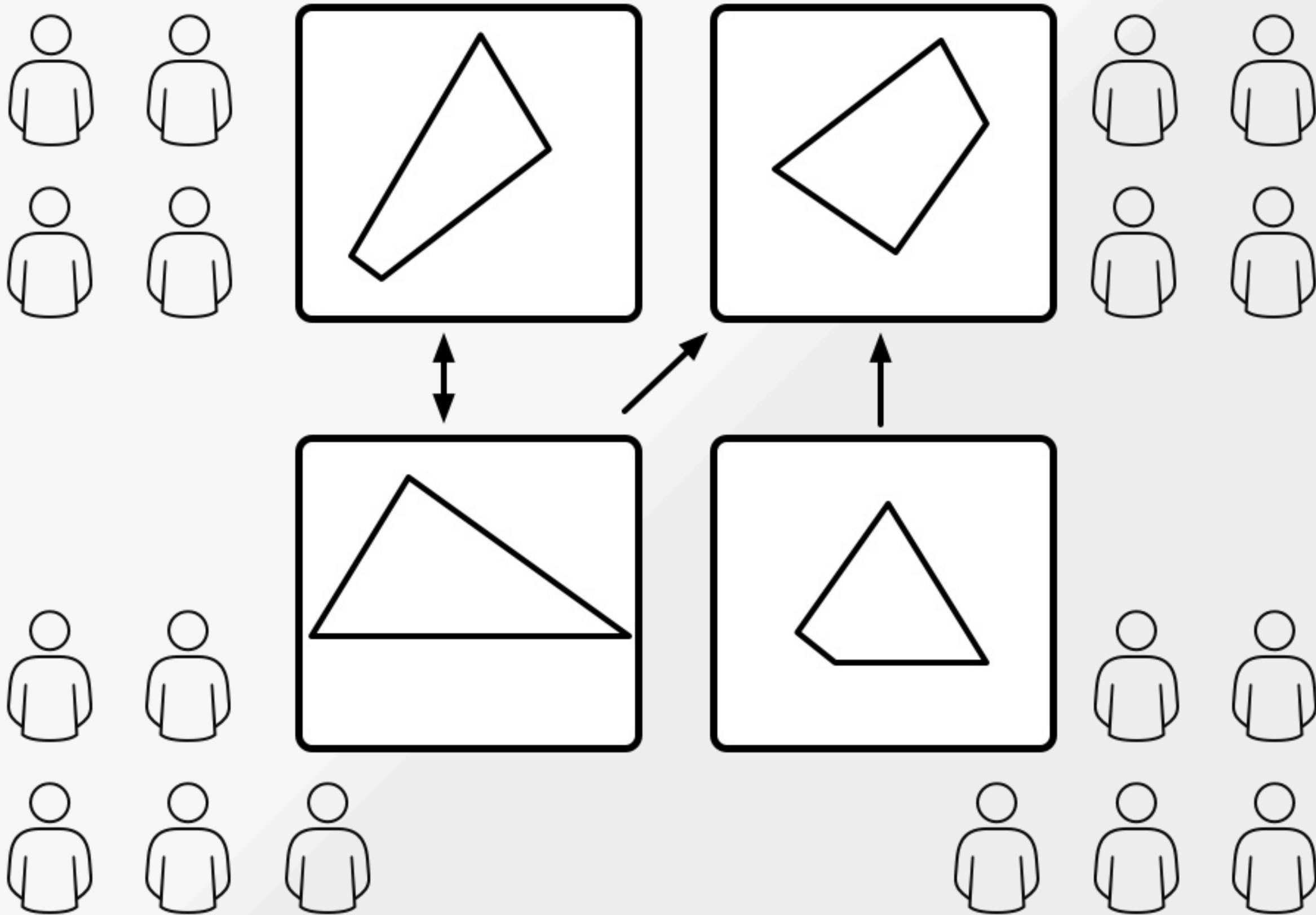
- Kubernetes
- Patterns
- Categories:
  - Foundational Patterns
  - Structural Patterns
  - Configurational Patterns

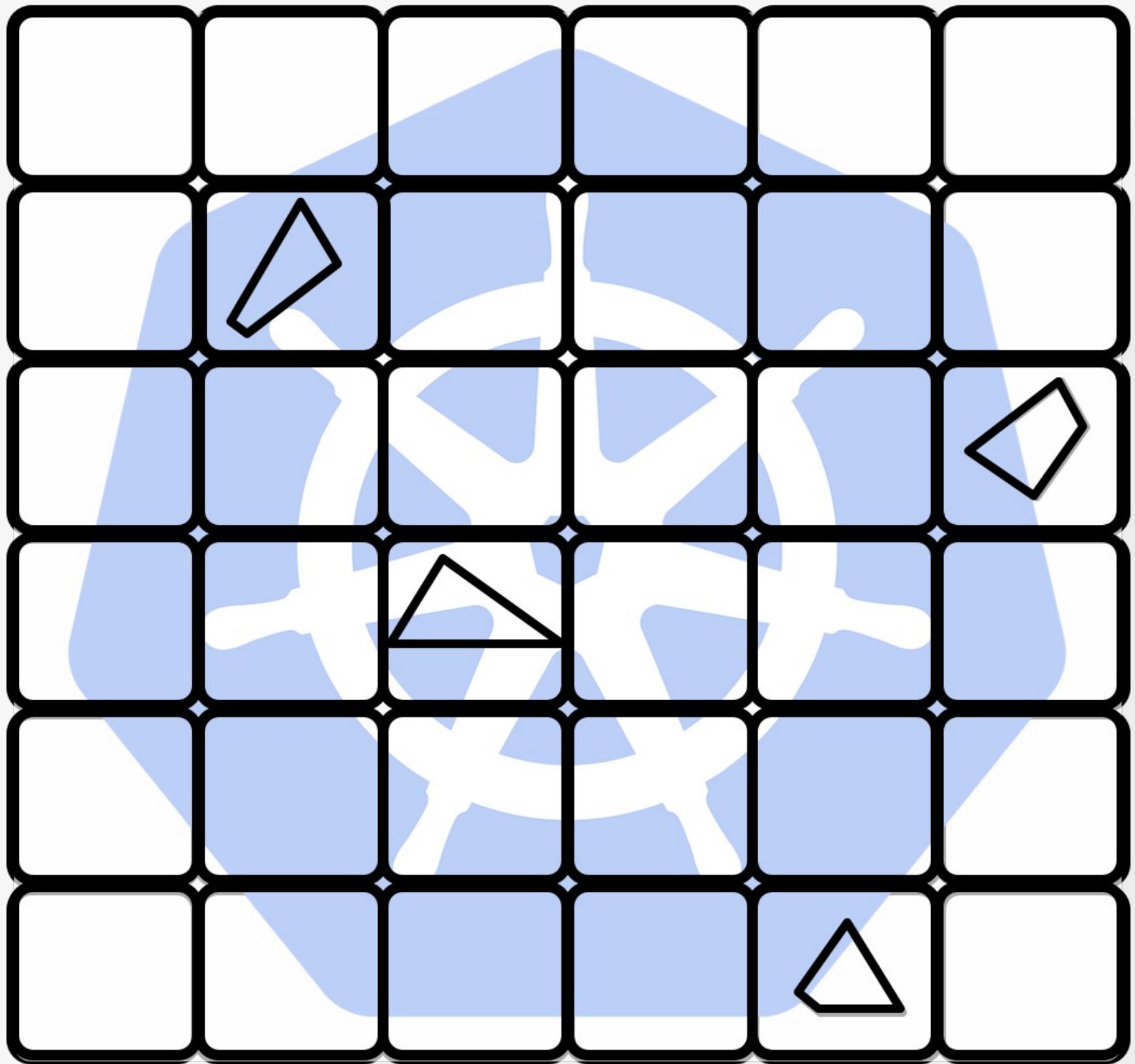










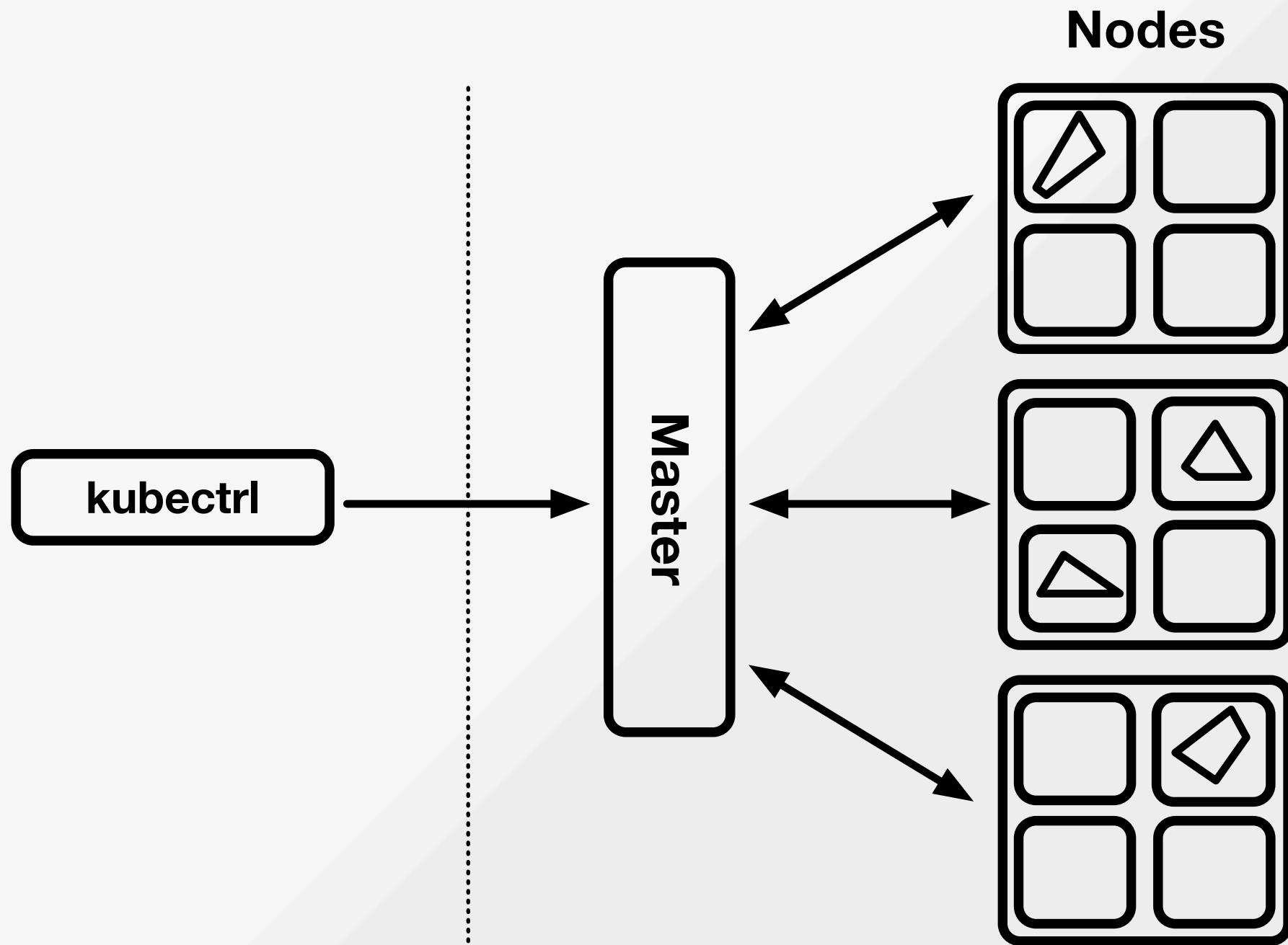




# KUBERNETES

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

# ARCHITECTURE



# 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

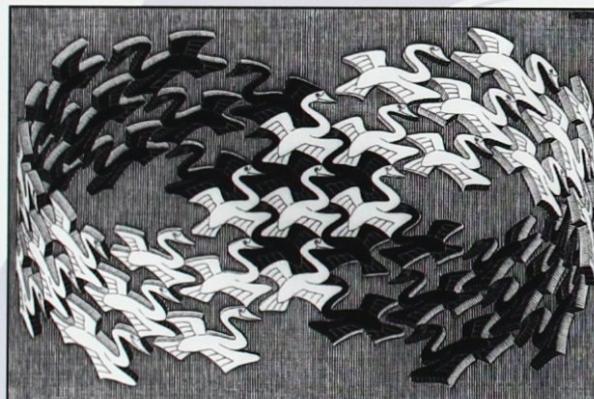


redhat

# 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



# 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

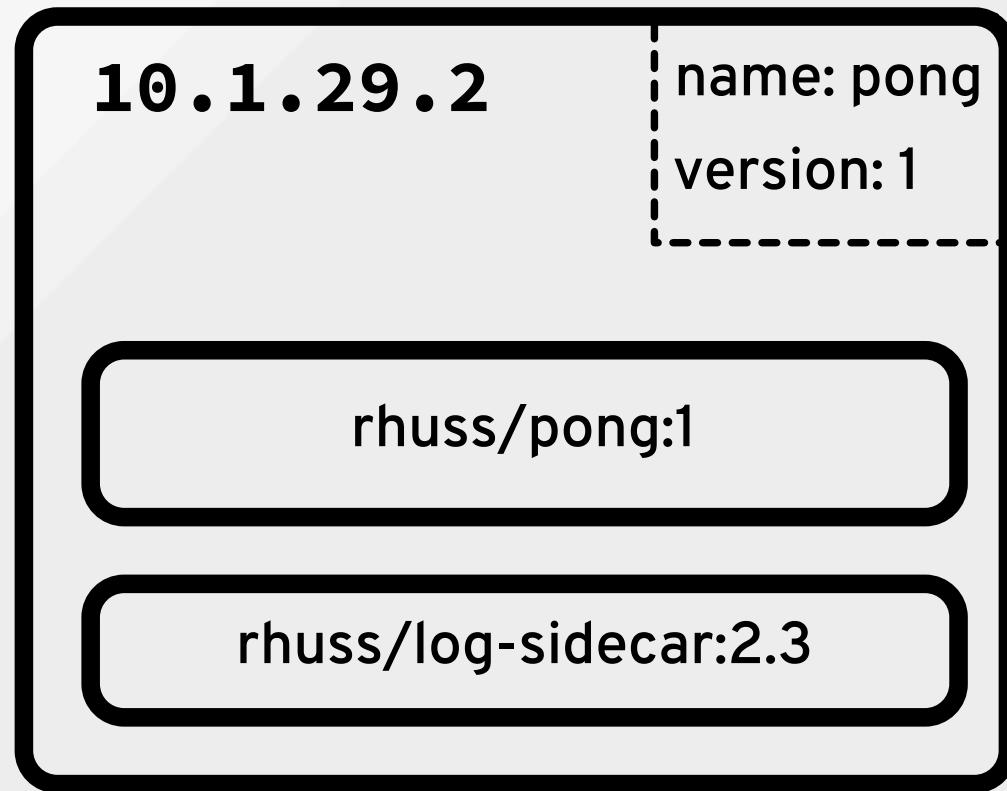
# Automatable Unit

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**

e x@|vt f x"

replicas: F

Selector:

name: pong  
version: 1

10.1.29.2  
name: pong  
version: 1

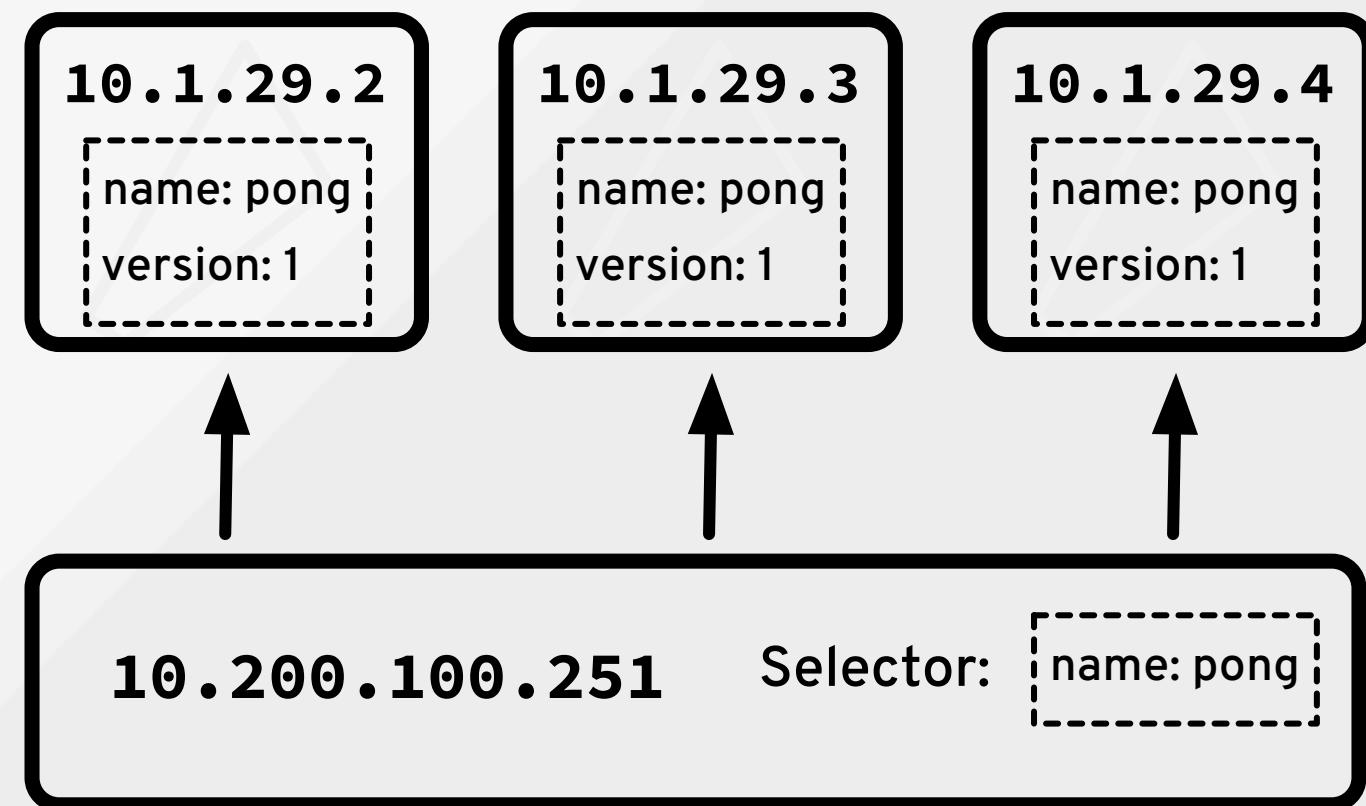
10.1.29.3  
name: pong  
version: 1

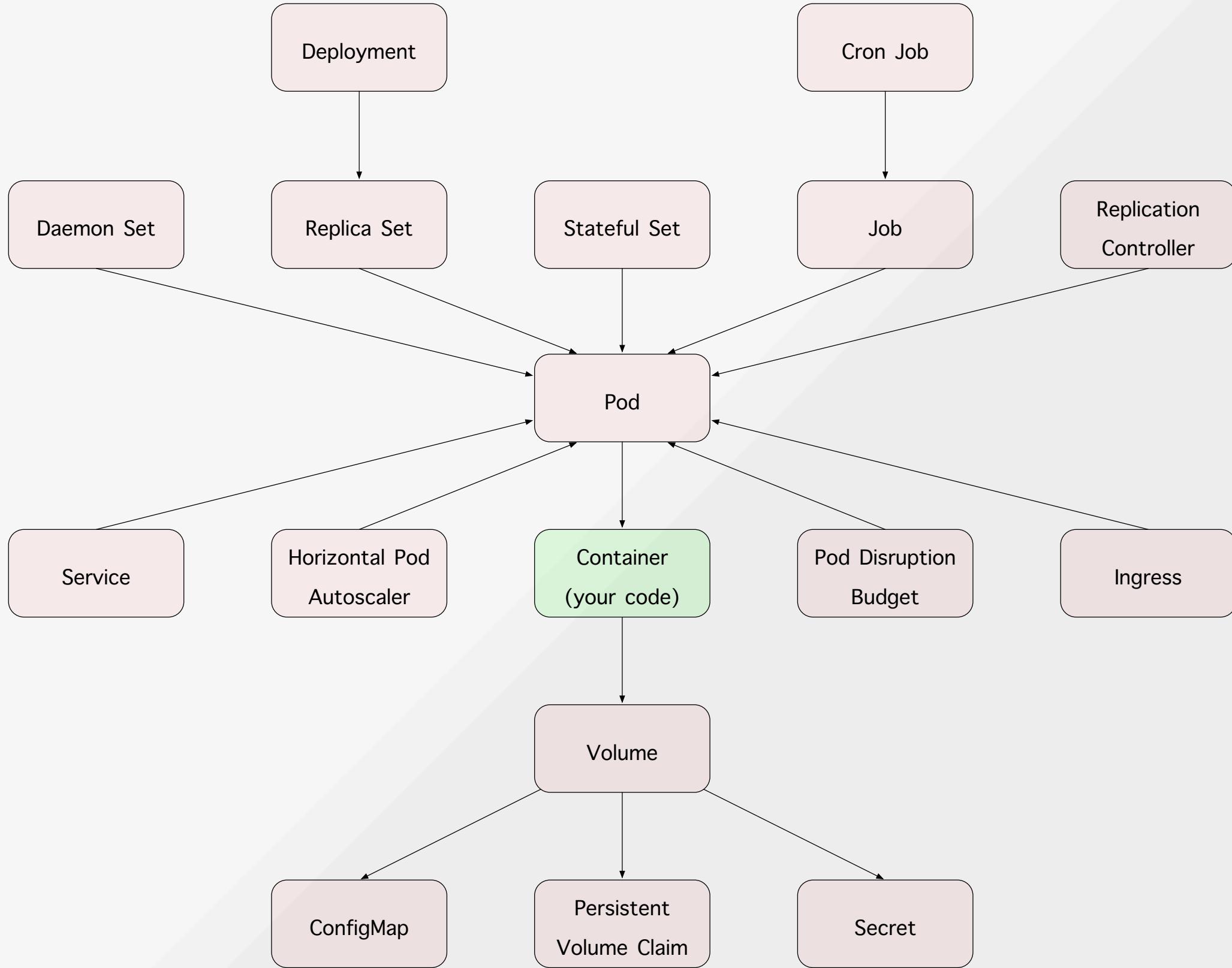
10.1.29.4  
name: pong  
version: 1



# 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

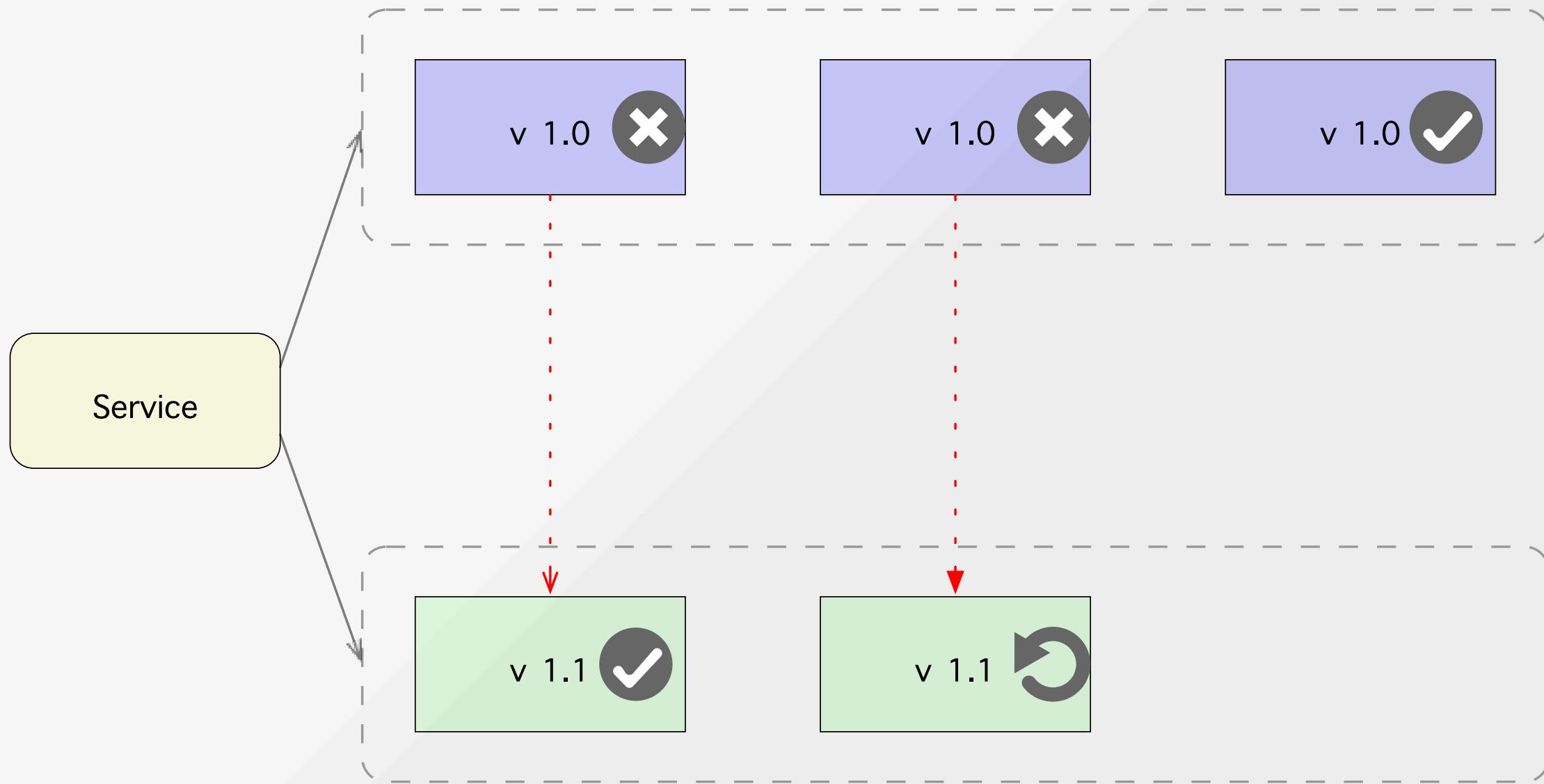
How can applications be deployed  
and updated ?

- Declarative versus Imperative deployment
- Various update strategies

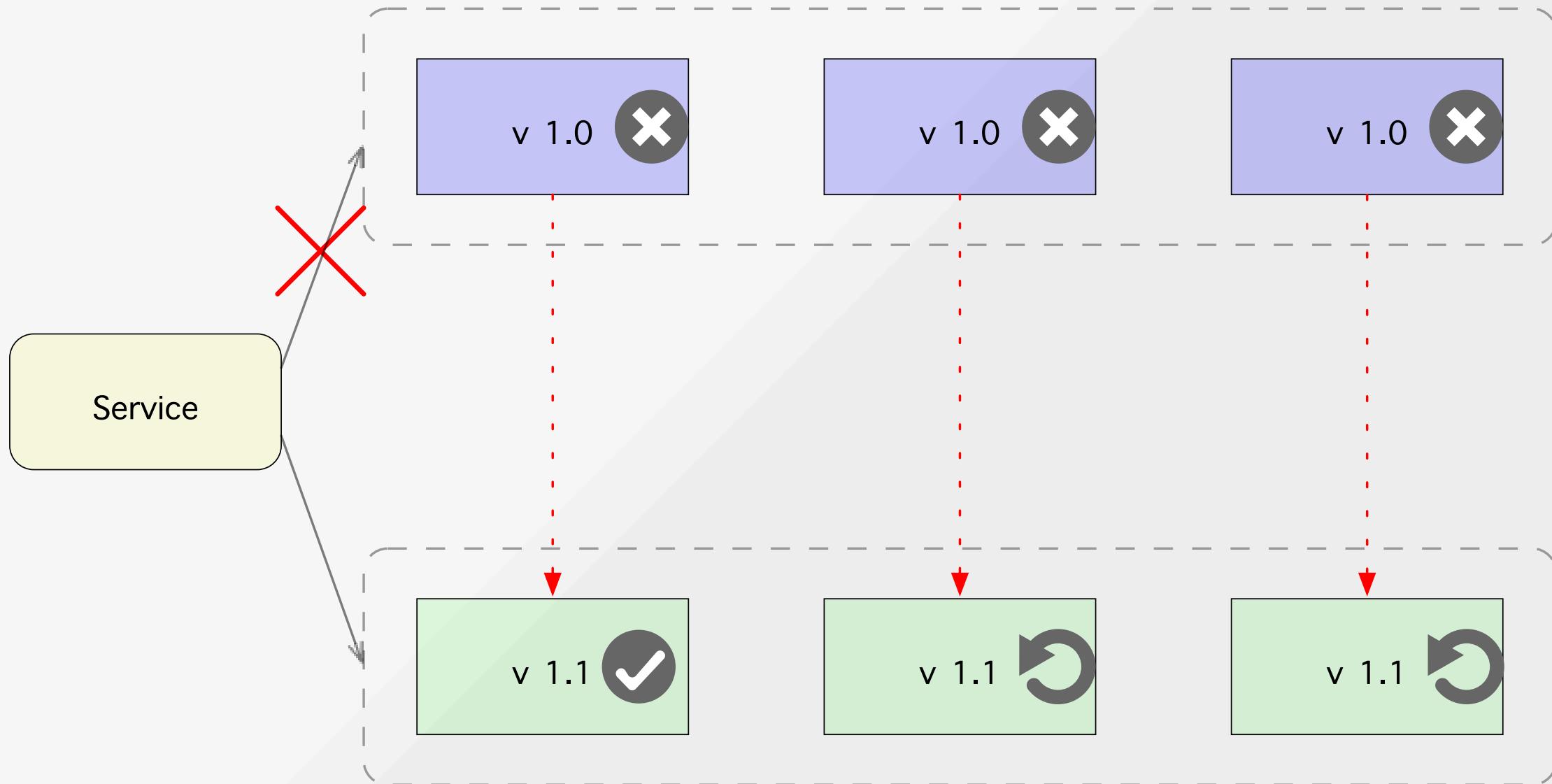
# DEPLOYMENT

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

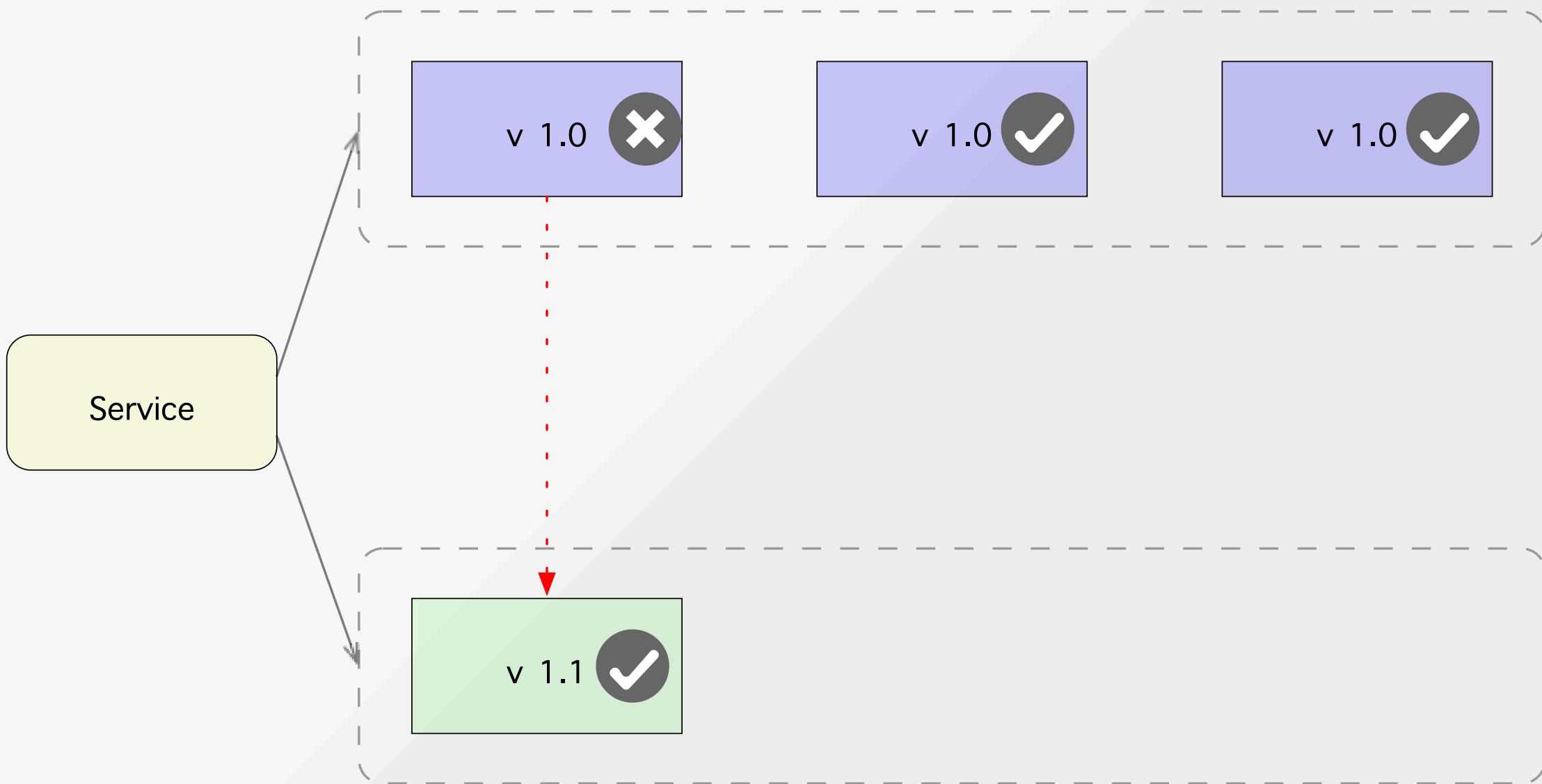
# ROLLING



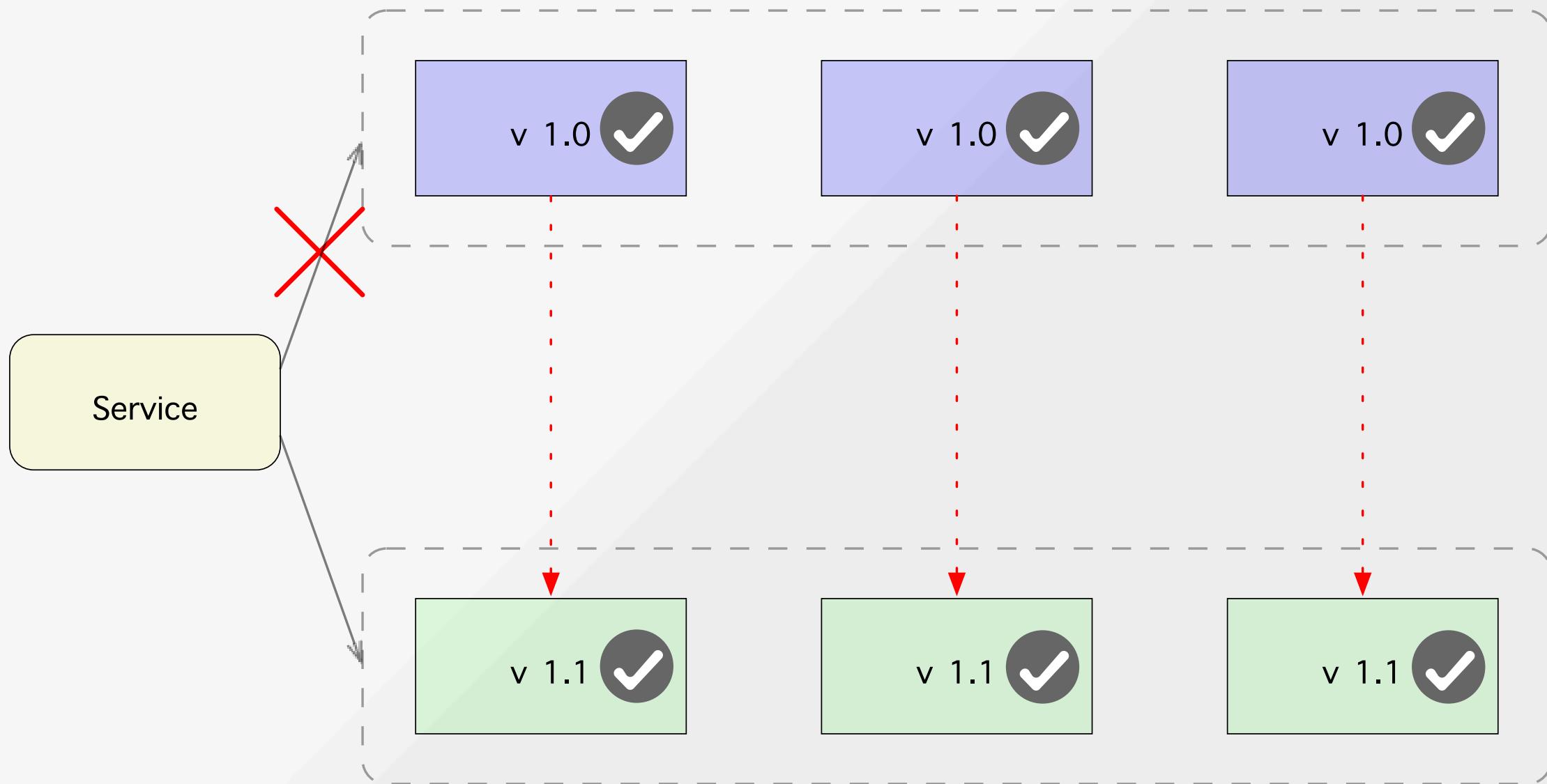
# FIXED



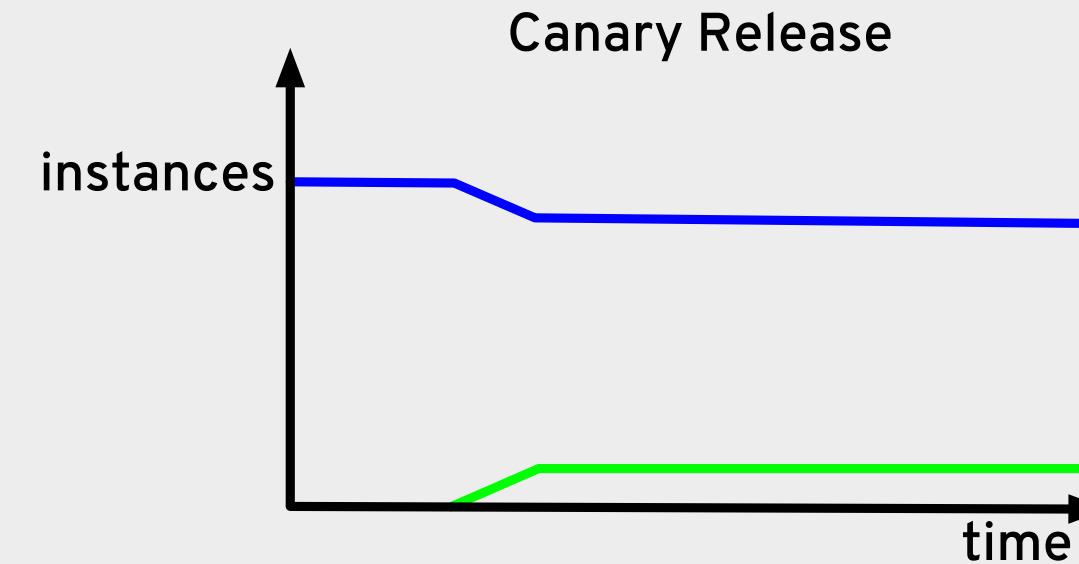
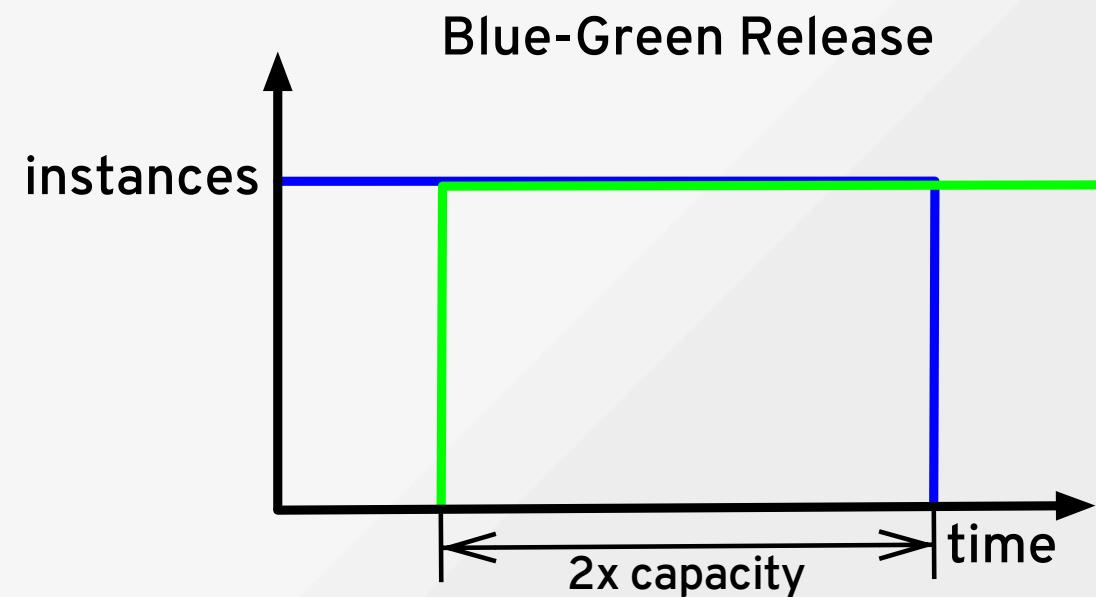
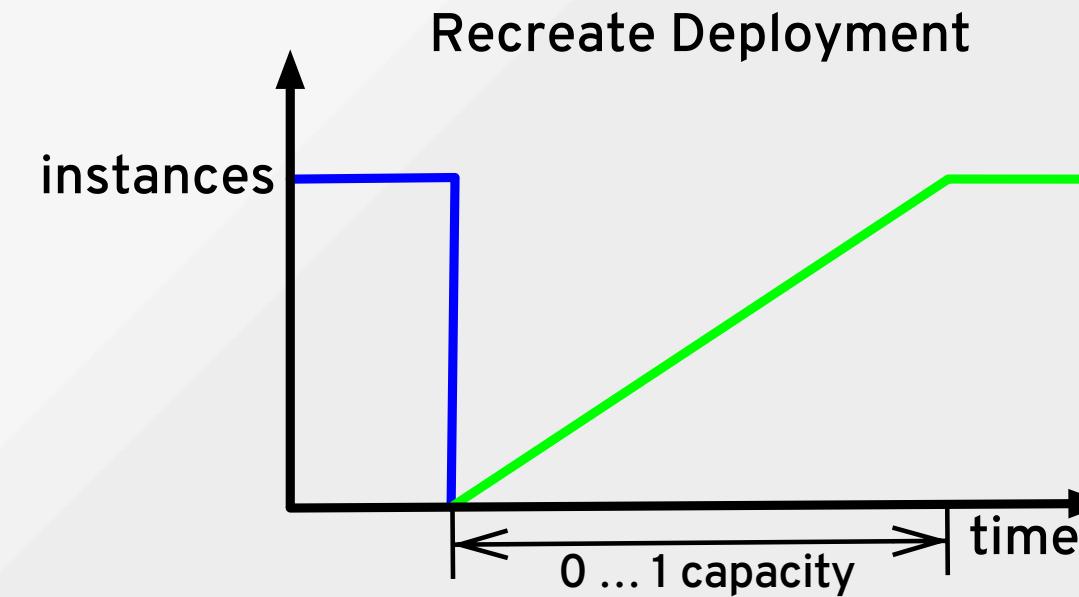
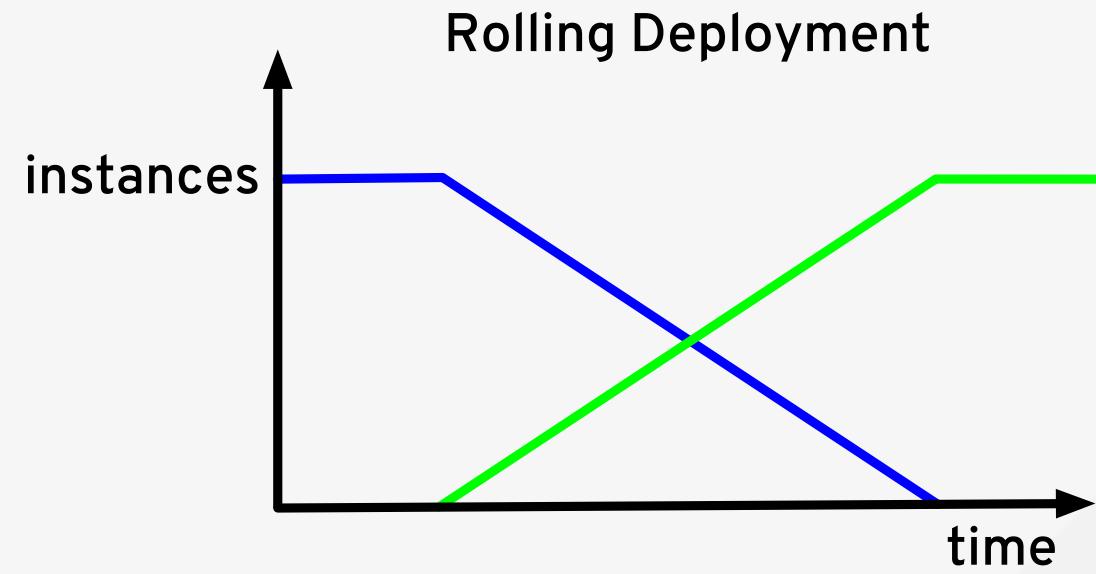
# CANARY



# BLUE-GREEN



# SUMMARY

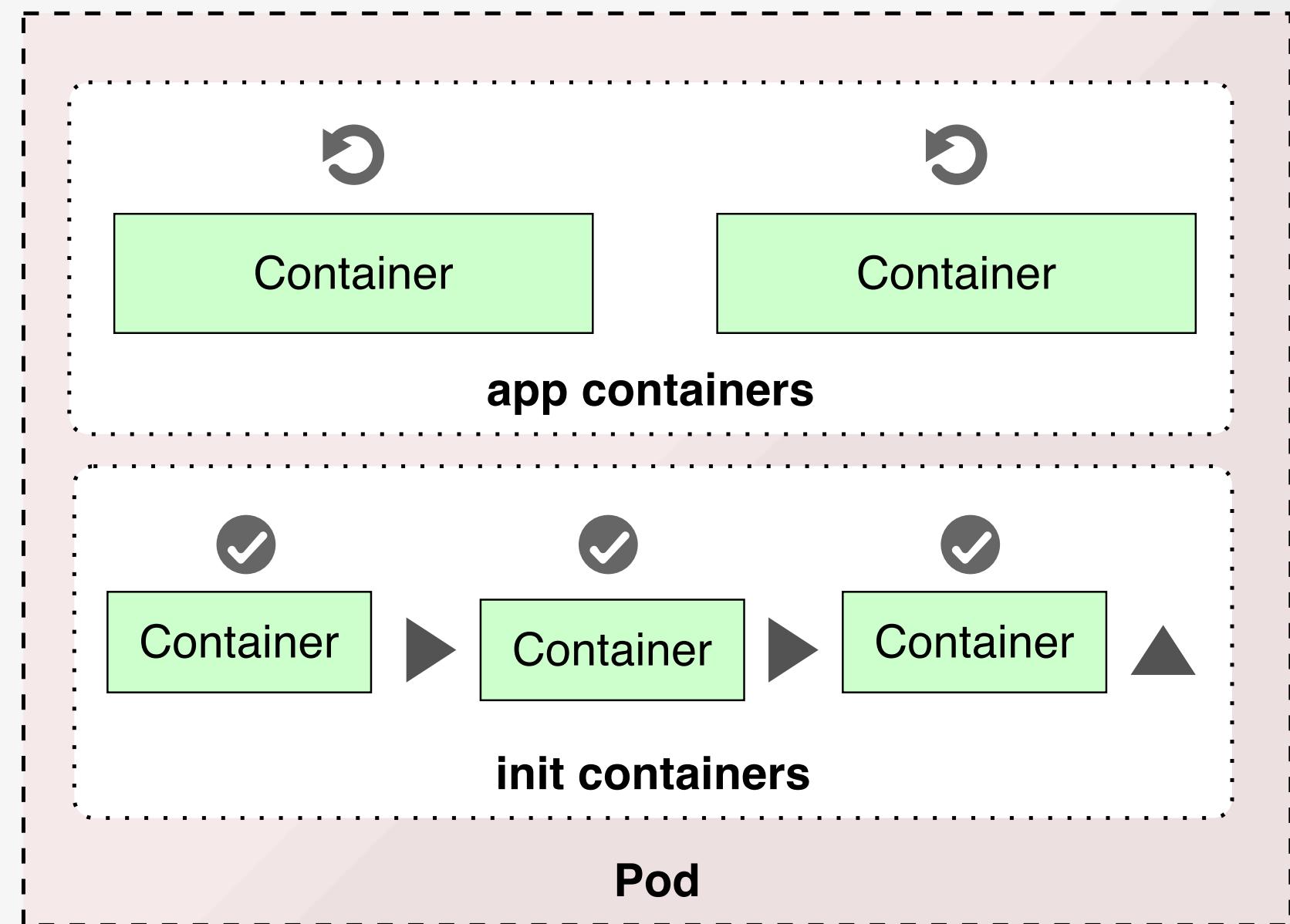


# **STRUCTURAL PATTERNS**

# Initializer

How can I initialize my containerized applications ?

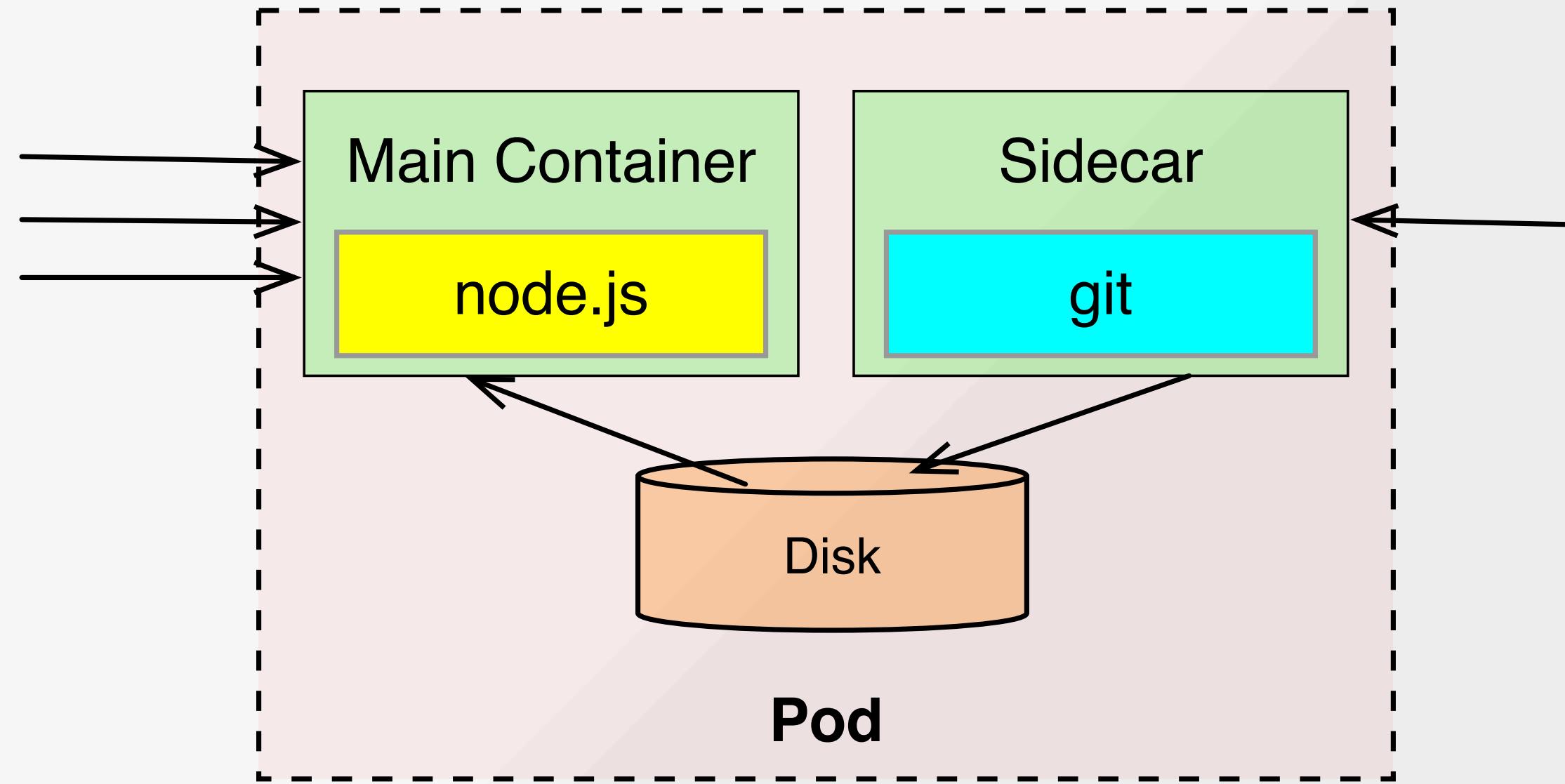
- Init containers :
  - Part of an Pod
  - One shot action before Pod starts
  - Need to be idempotent
  - Have own resource requirements



# Sidecar

How can I extend the functionality of  
an existing container ?

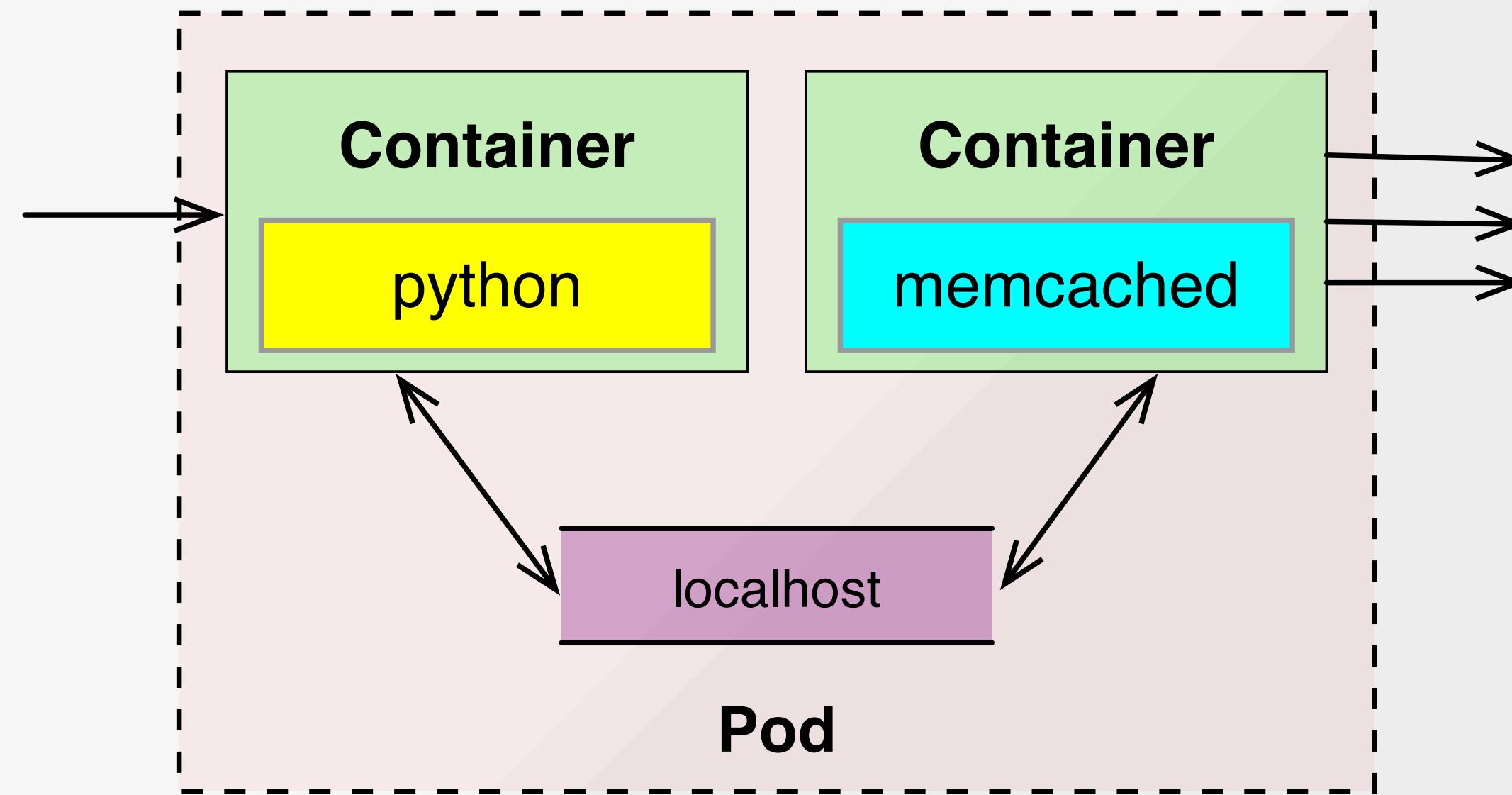
- Runtime collaboration of containers
- Connected via shared resources:
  - Network
  - Volumes



# Ambassador

How can a container access **to** the outside world be decoupled ?

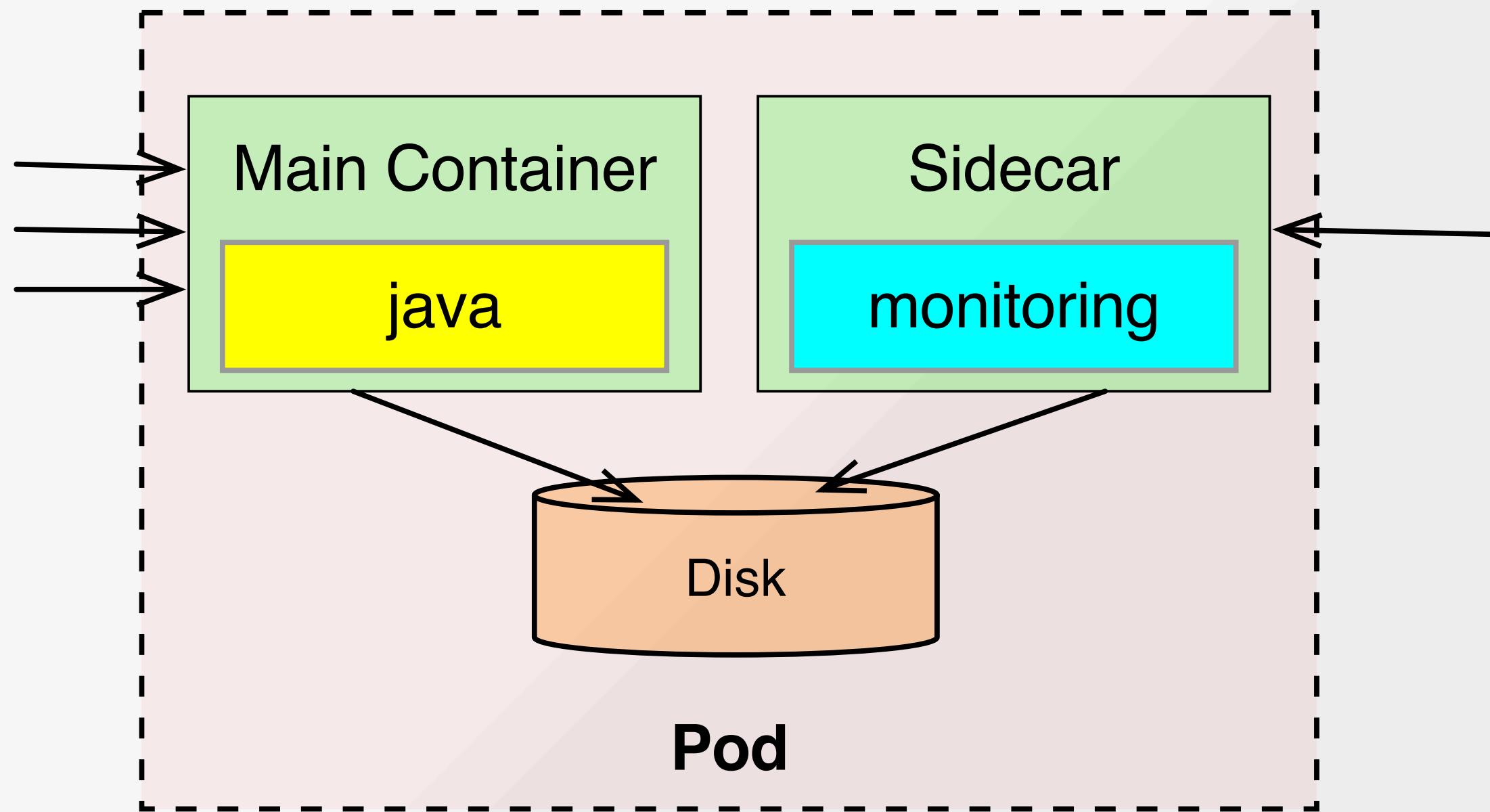
- Also known as **Proxy**
- Specialization of a Sidecar
- E.g. infrastructure services
  - Circuit breaker
  - Tracing



# Adapter

How can a container access **from** the outside world be decoupled ?

- 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
```

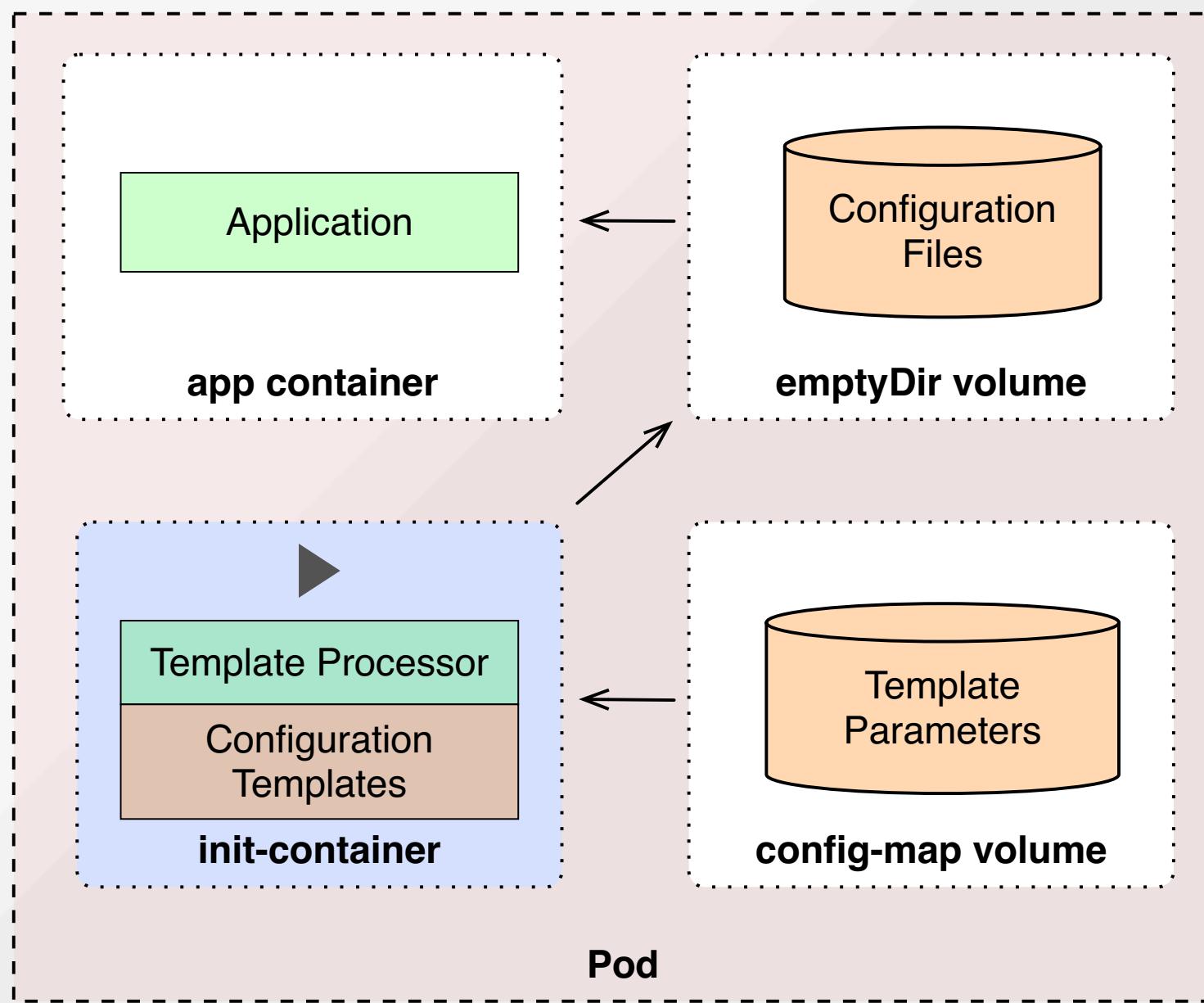
```
kind: Pod
spec:
  containers:
  - name: web
    volumeMounts:
    - name: config-volume
      mountPath: /etc/config
    # ...
  volumes:
  - name: config-volume
    configMap:
      name: spring-boot-config
```



# 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



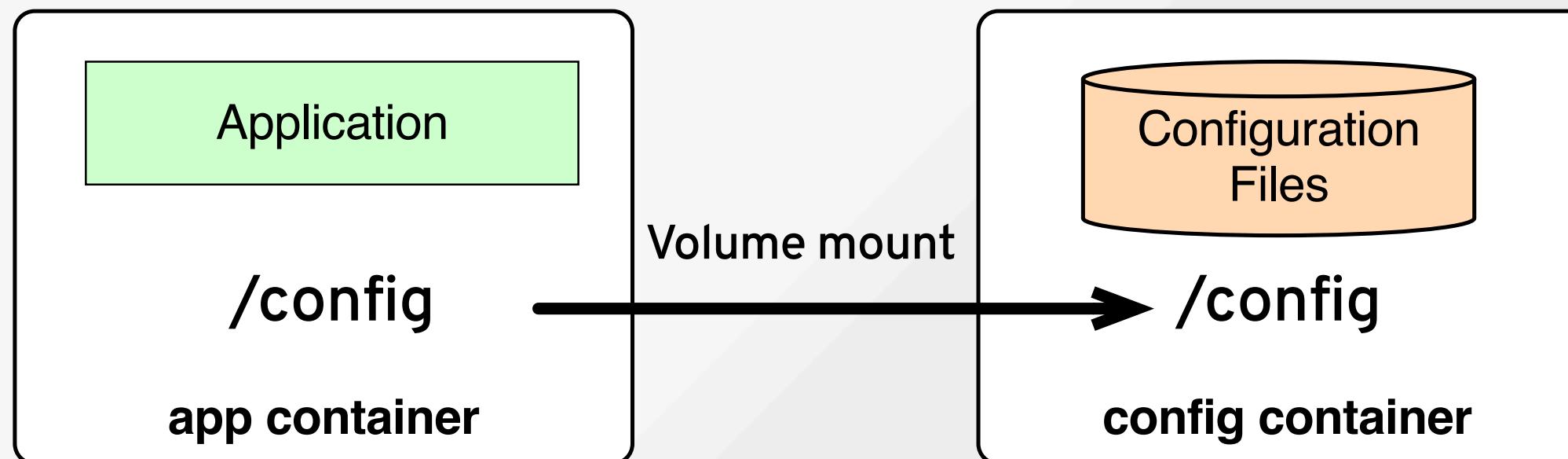
# DISCUSSION

- 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

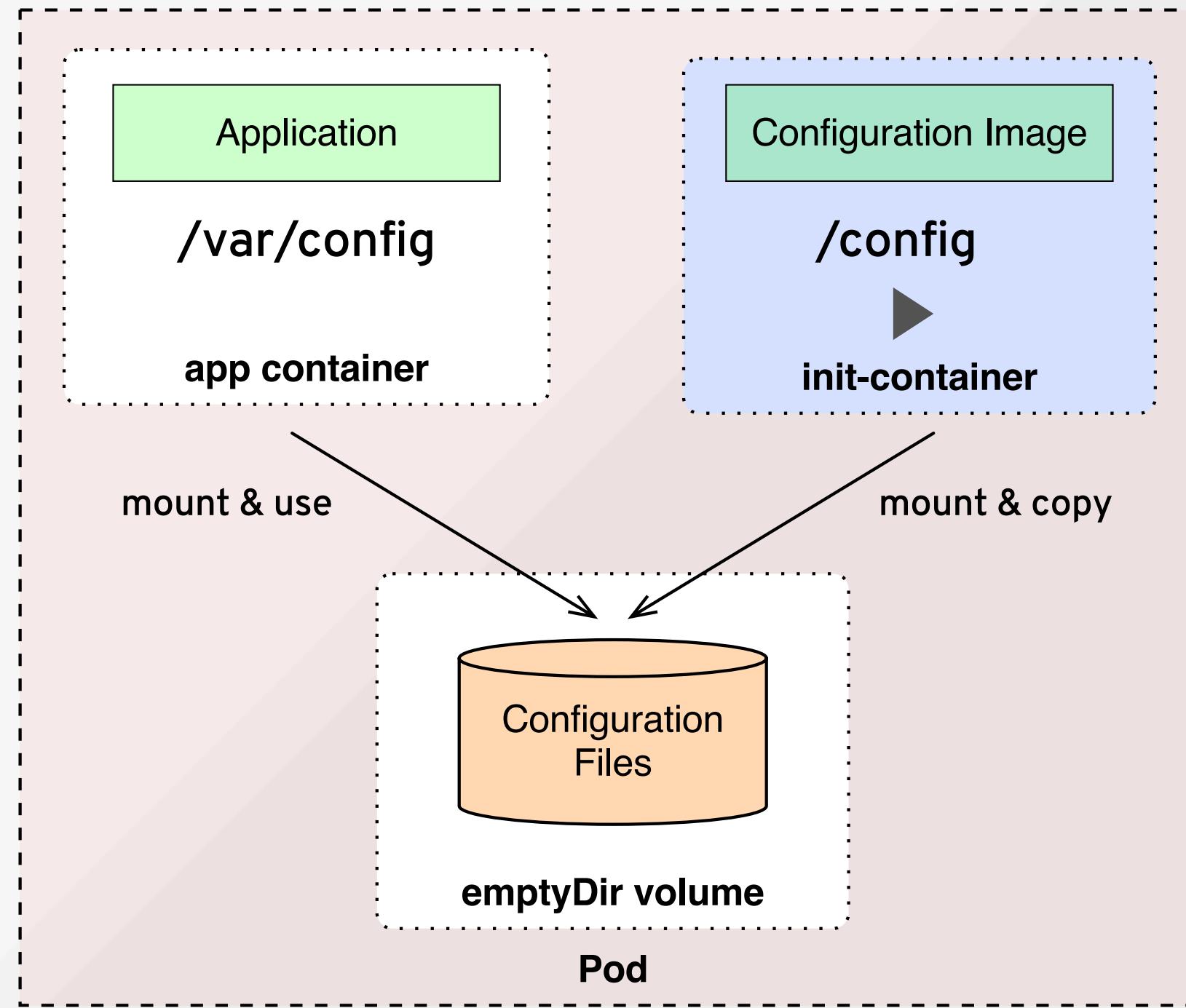


- Not directly supported by K8s
- docker-flexvol : K8s FlexVolume driver for Docker volumes

```
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"
```



# INIT CONTAINER



- 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

# 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.



redhat.<sup>®</sup>

# QUESTIONS ?

Twitter ro14nd

---

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

---

Slides <http://bit.ly/kubernetes-patterns-javazone-2017>