redhat.

Leveraging the power of Kubernetes with patterns

For fun and profit

Roel Hodzelmans & Wian Vos

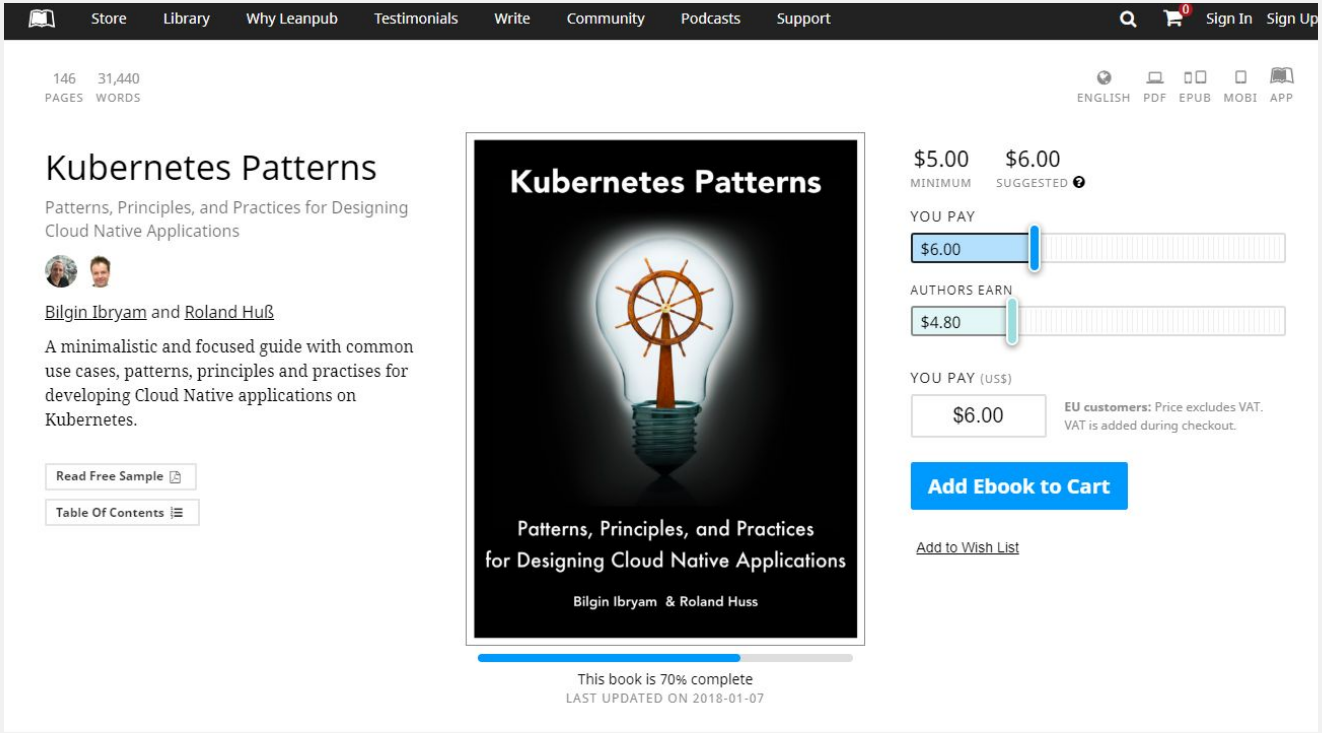
#WHOAREWE

Roel Hodzelmans
Senior Solution Architect

Wian Vos
Solution Sales Professional*

* recent convert to the dark side

#BasedOnTheWorkOff



https://leanpub.com/k8spatterns

AGENDA

Steps between you and free food.

Design Patterns

Origin/Why/Structure

K8S Introduction

Again? Yes.. but short

K8S Patterns

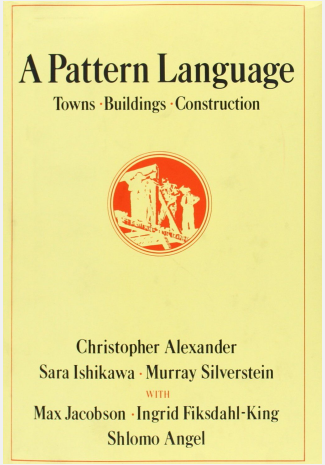
- Foundational
- Structural
- Configurational
- Advanced??

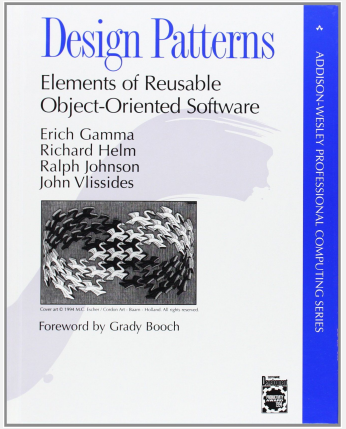
Design Patterns

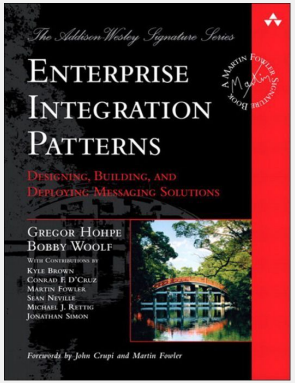
Design Patterns

Where did they come from

- Originating from Architecture
- Widely used in most fields of engineering
- Check:
  https://www.martinfowler.com/articles/writingPatterns.html





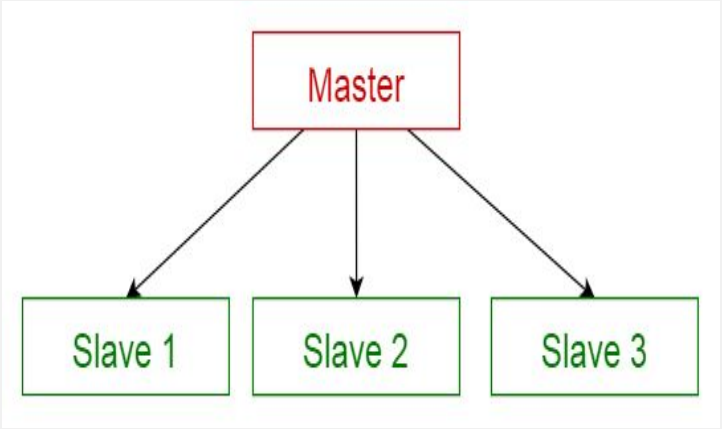


redhat.

Design Patterns

They are everywhere

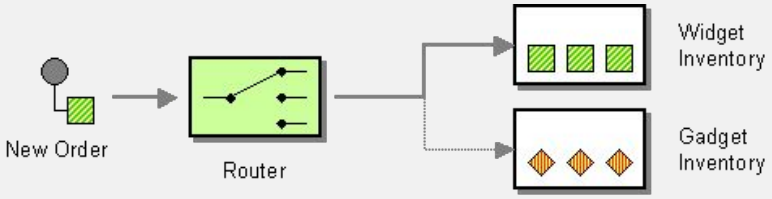
Computer engineering:
Master slave Pattern

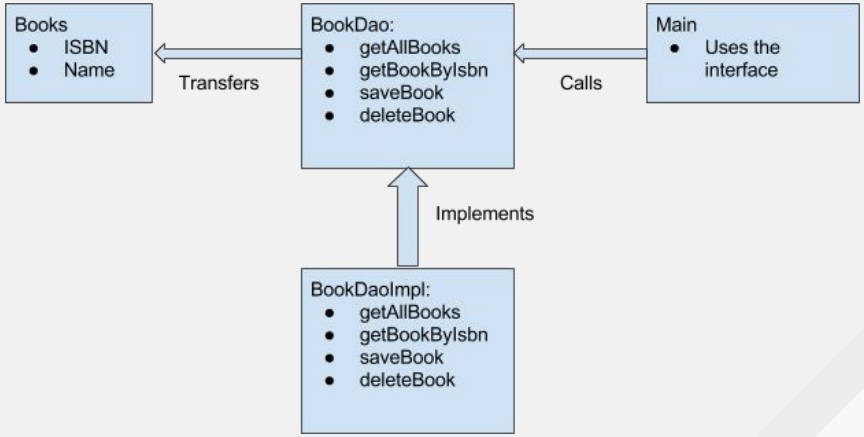


Design Patterns

They are everywhere

Development examples:
Content Based Routing and Data Access Object

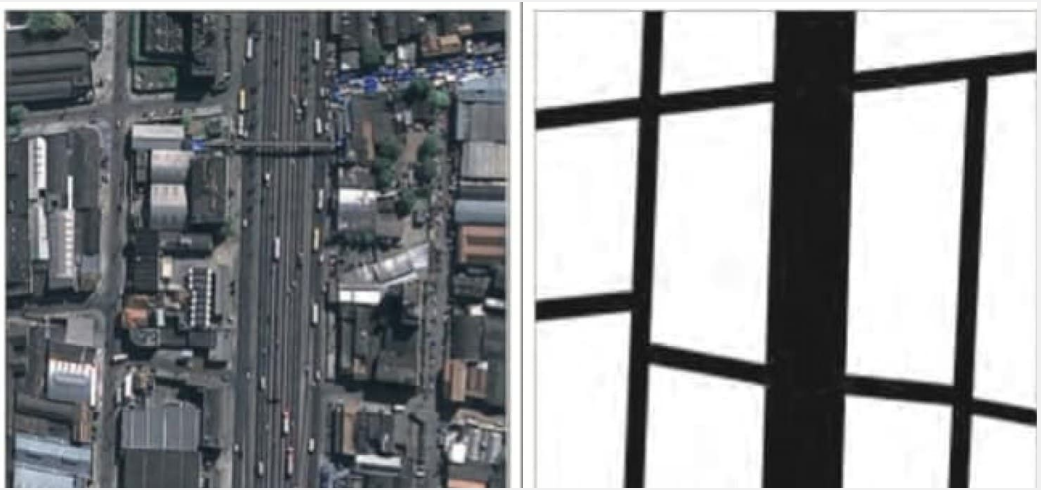




Design Patterns

They are everywhere

City Planning:
Workplace boxes



Design Patterns

They are everywhere

Automotive Industry:
V8

Design Patterns

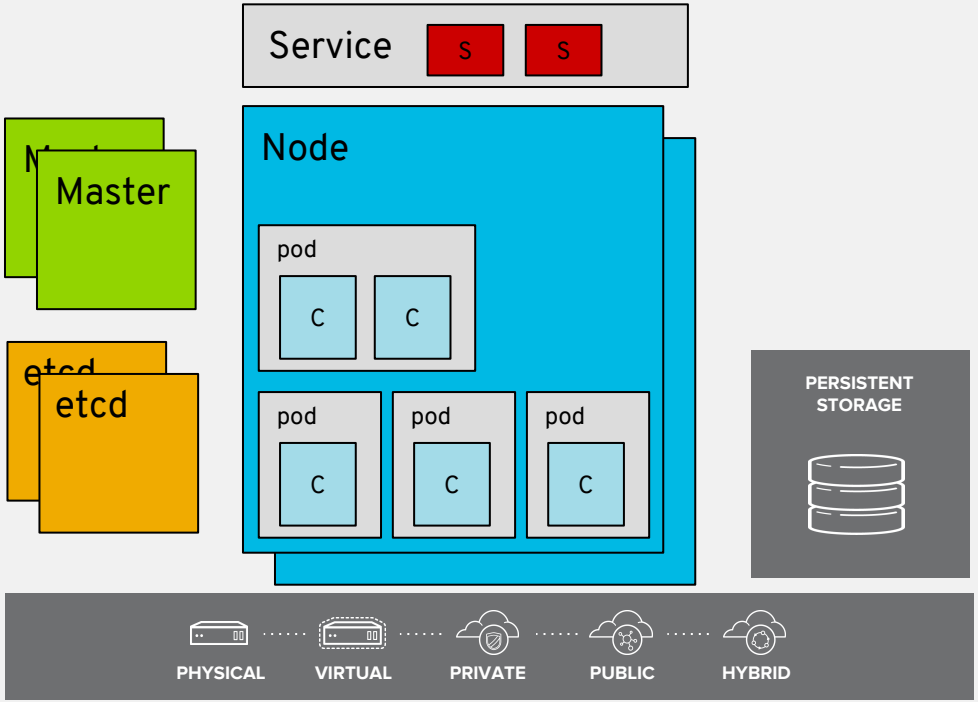
They are everywhere

Patterns give us a common reference

K8S Introduction

K8S Introduction

The short version



#SoWhatAboutKubernetesPatterns

FOUNDATIONAL PATTERNS

Automatable Unit

Foundational patterns

How can we create and manage applications with Kubernetes ?

- **Pods:** Atomic unit of containers
- **Services:** Entry point to pods
- **Deployments:** Collection of deployable units

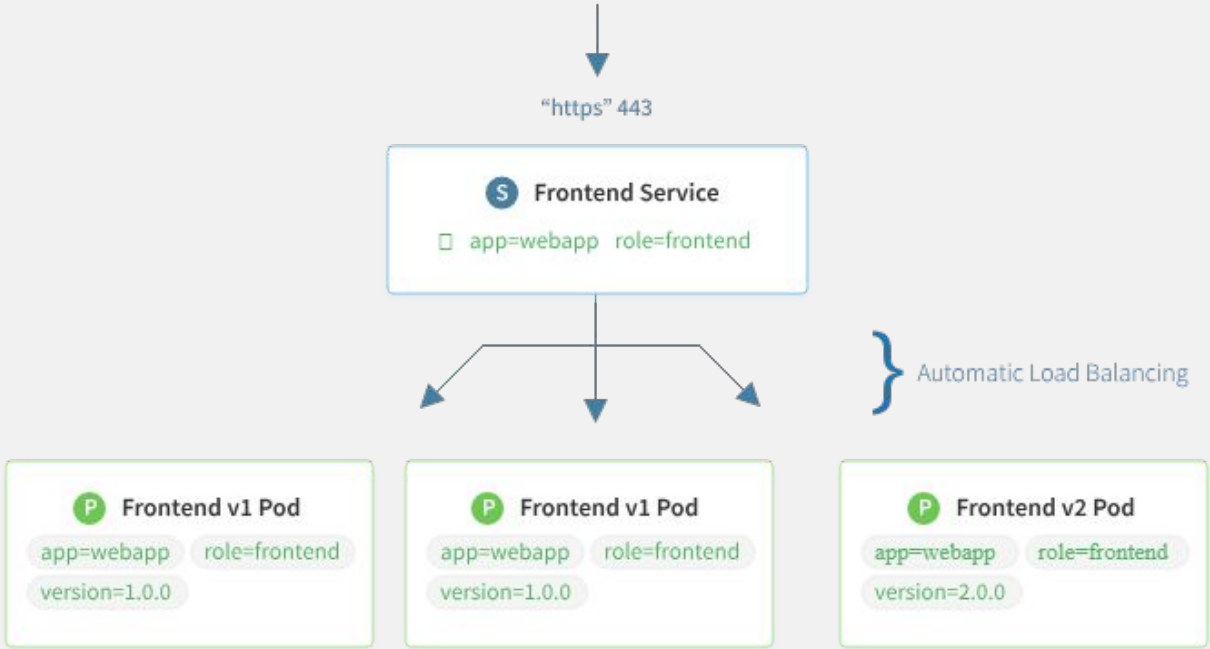
Pod

Foundational patterns - Automatable unit

```
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
```

Services

Foundational patterns - Automatable unit



Deployment

Foundational patterns - Declarative deployments

- Holds template for Pods/Containers/Services and (ingress)
- Allows rollback and upgrade
- Update strategies declarable

Predictable demands

Foundational patterns

How can we handle resource requirements deterministically?

- Runtime dependencies
- Resource profiles
- Quality of Service

Runtime dependencies

Foundational patterns - Predictable demands

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

Resource profiles

Foundational patterns - Predictable demands

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

Resource profiles

Foundational patterns - Predictable demands

```yaml
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
```

Quality of Service

Foundational patterns - Predictable demands

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

Declarative deployments

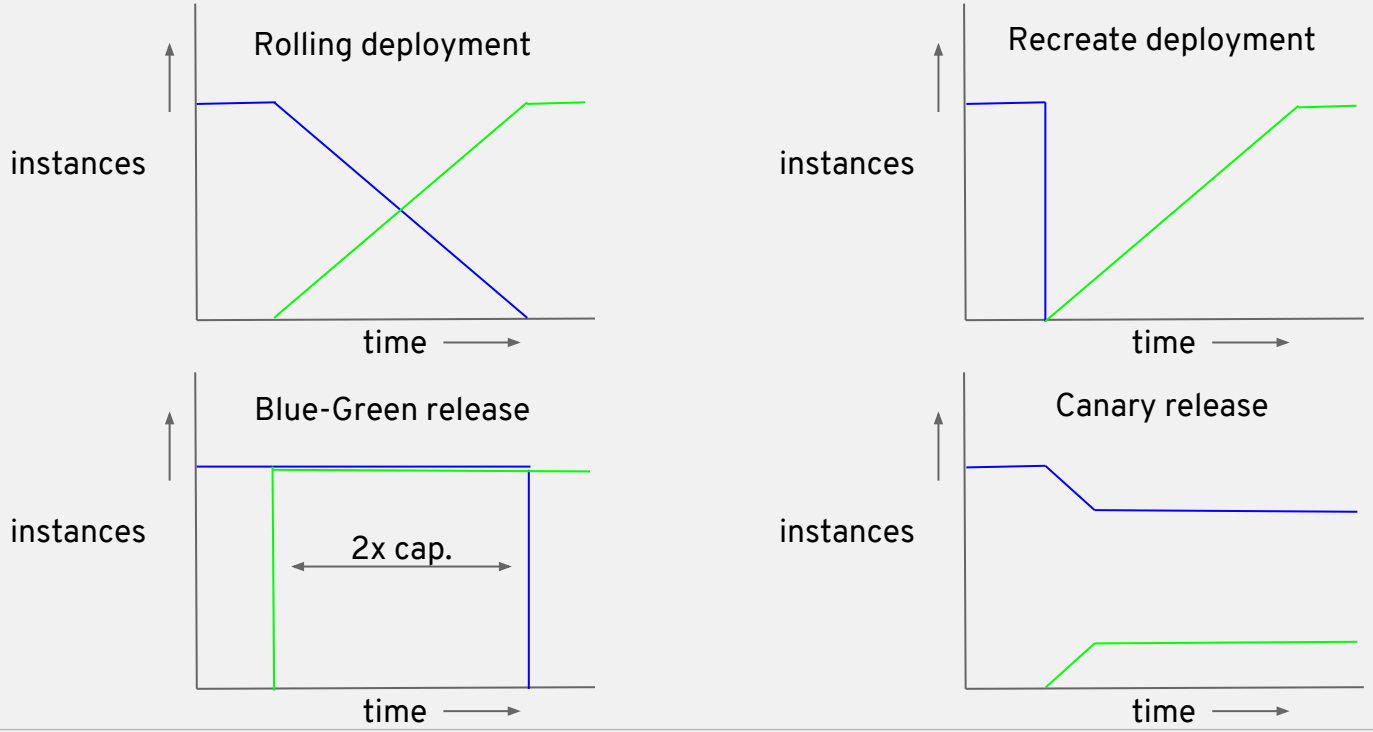
Foundational patterns

How can applications be deployed and updated?

- Rolling update
- Canary deploy
- Blue-green
- Recreate deployment

Deployment strategies

Foundational patterns - Declarative deployments



STRUCTURAL PATTERNS

Initializer

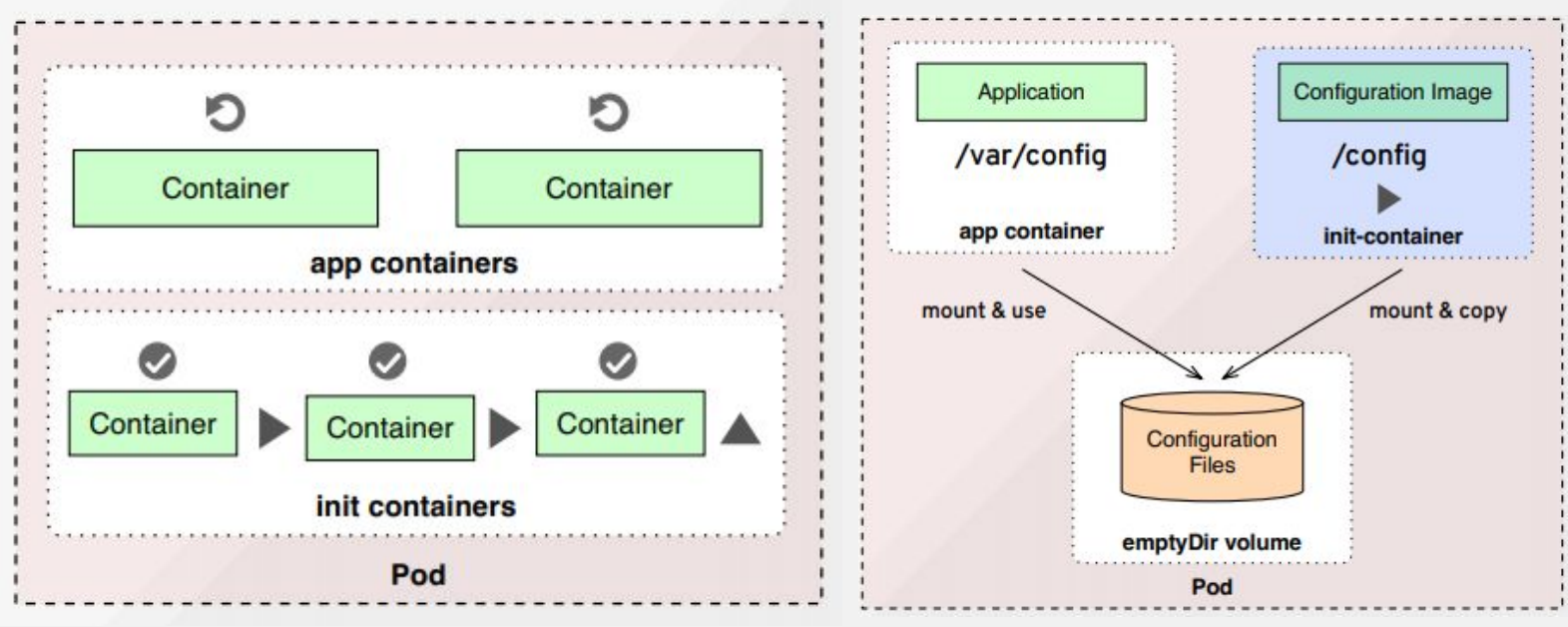
Structural patterns

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

Initializer

Structural patterns - init container



Sidecar

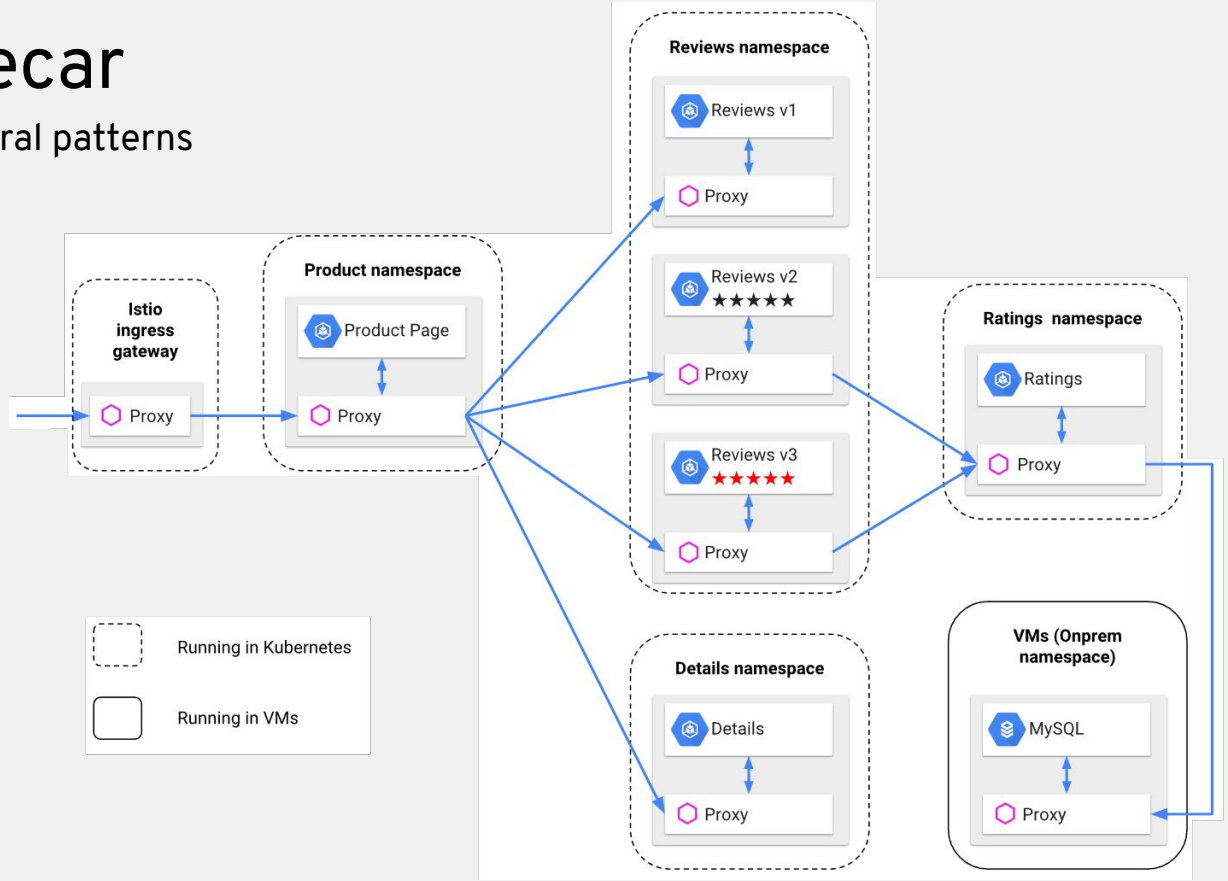
Structural patterns

How can I extend the functionality of an existing container ?

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

Sidecar

Structural patterns



Ambassador

Structural patterns

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

Structural patterns

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

- Opposite of Ambassador
- Uniform access to Utilities from pod
- Examples:
  - Monitoring
  - Logging

CONFIGURATIONAL PATTERNS

Deployment configuration

Configurational patterns

How can apps be configured for different environments?

- EnvVar
- Configuration Resource
- Configuration Template

EnvVar

Configurational patterns

```
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

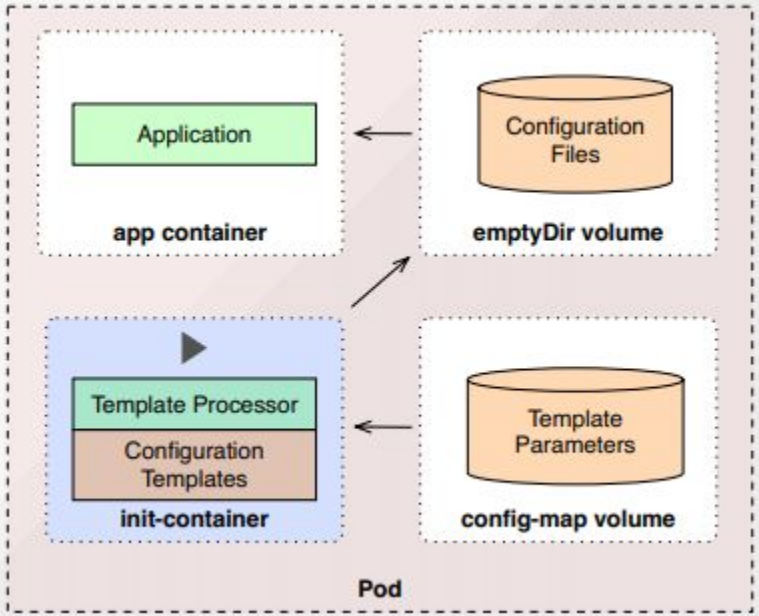
Configurational patterns

```
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

Configurational patterns



ADVANCED PATTERNS

Custom controller

Advanced patterns

How can I extend the platform itself without changing it?

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

Custom controller - Categories

Advanced patterns

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

Custom controller

Advanced patterns

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

Custom resources

Advanced patterns

How can I manage custom domain specific resources?

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

Custom resources

Advanced patterns

```
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: ....
```

Custom resources

Advanced patterns

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

Operator

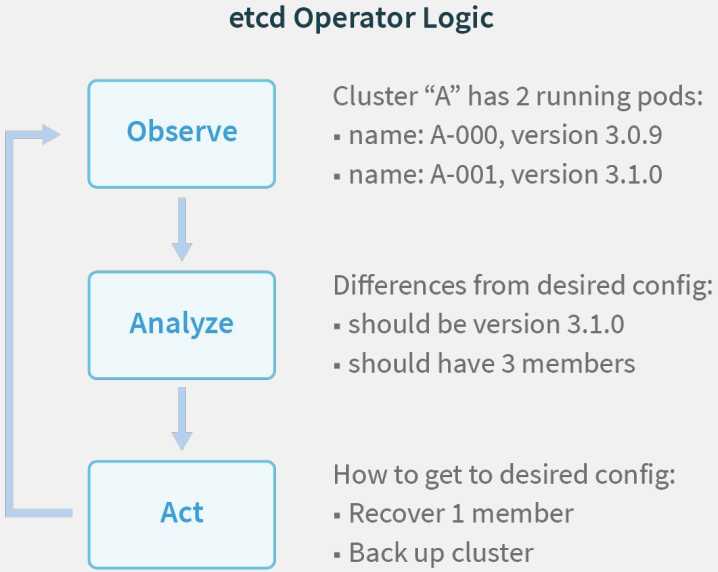
Advanced patterns

How do I package, deploy and manage a Kubernetes application?

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

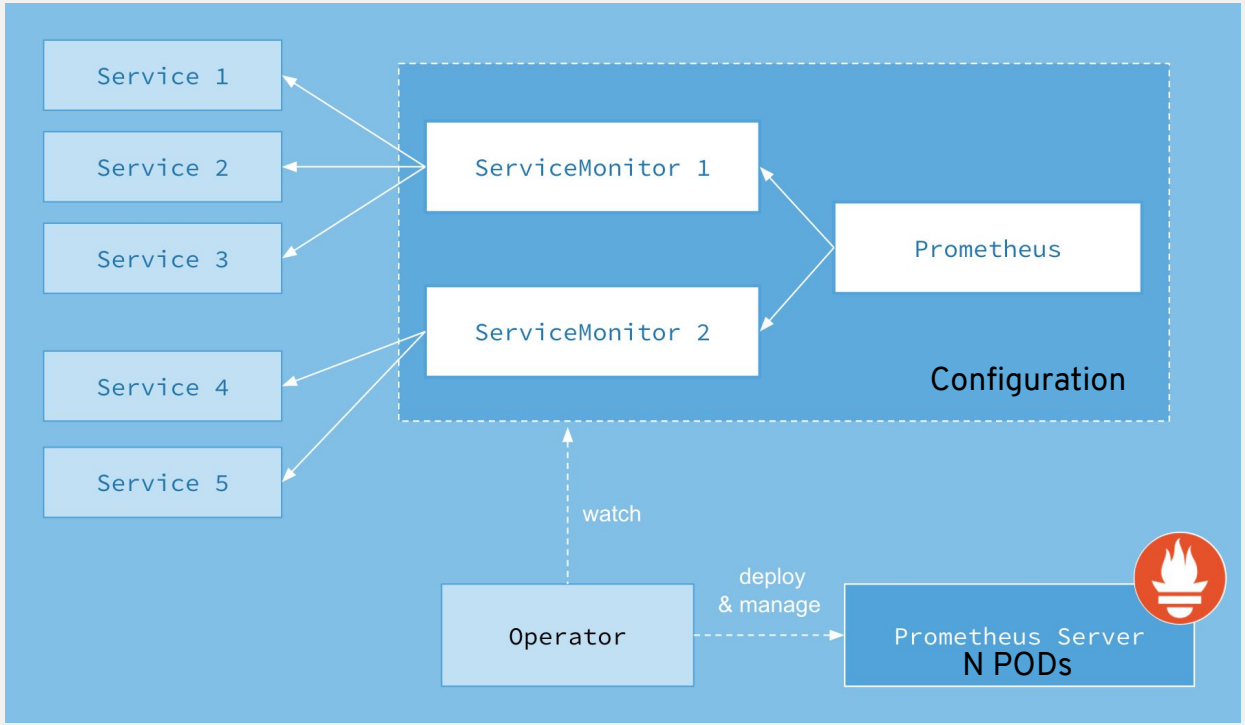
Operator

Advanced patterns



Operator

Advanced patterns



redhat.

THANK YOU

plus.google.com/+RedHat

facebook.com/redhatinc

linkedin.com/company/red-hat

twitter.com/RedHat

youtube.com/user/RedHatVideos