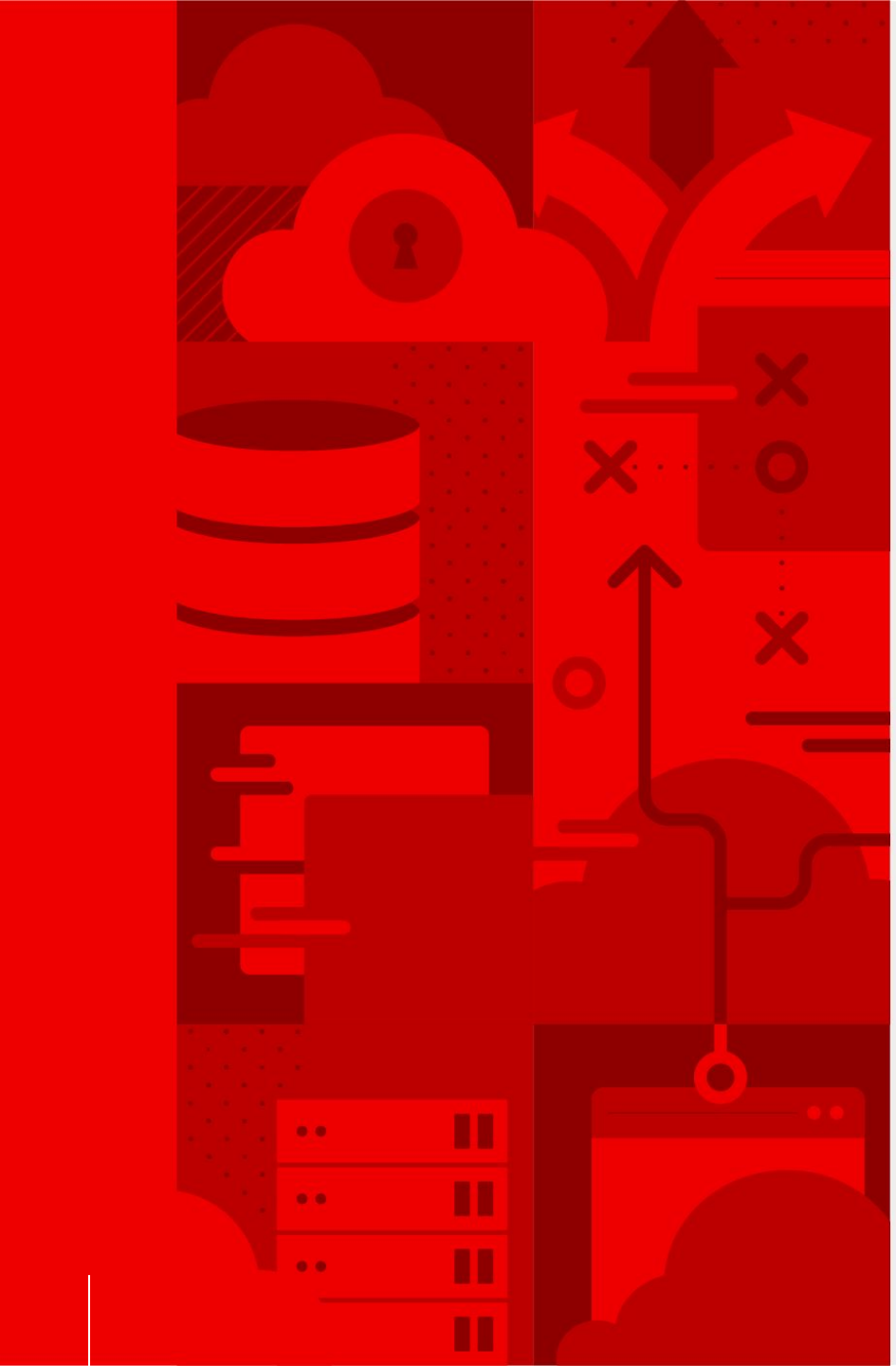


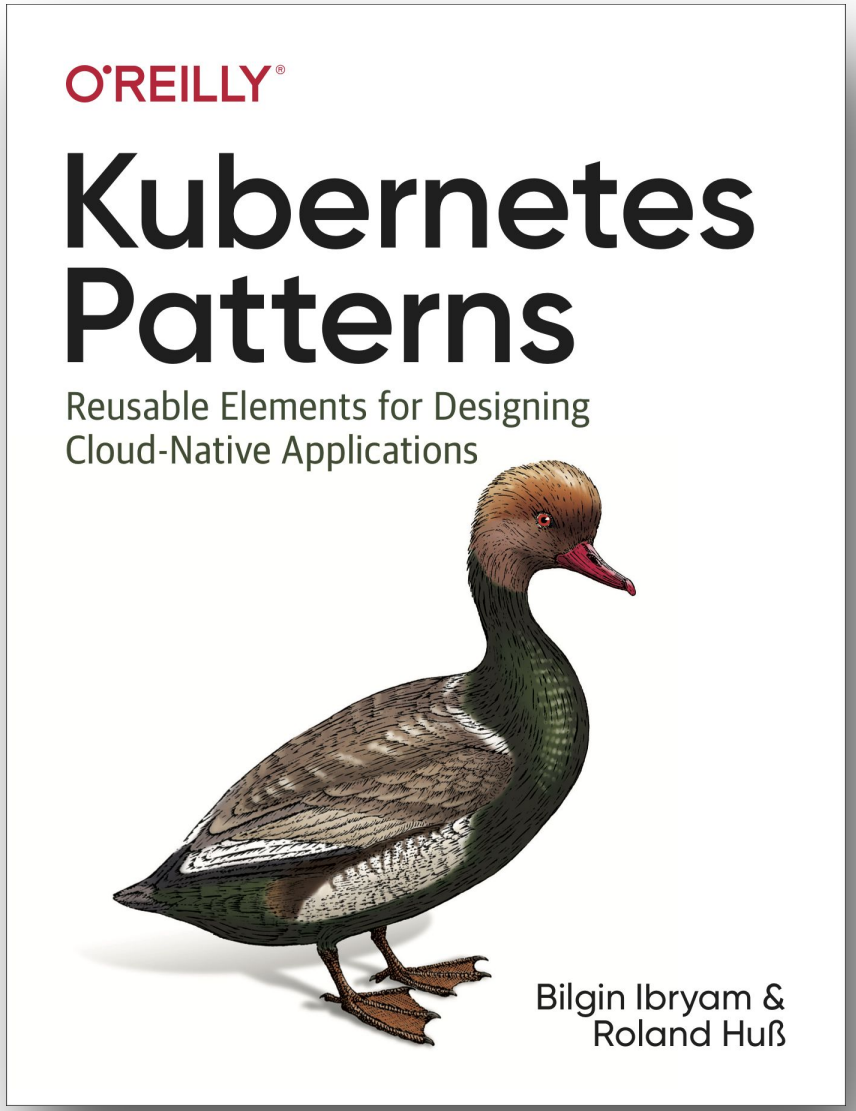
Kubernetes Patterns

Reusable Elements for Designing Cloud-Native Applications

Dr. Roland Huß
Principal Software Engineer
@ro14nd
https://k8spatterns.io

Bilgin Ibryam
Principal Product Manager
@bibryam
https://k8spatterns.io

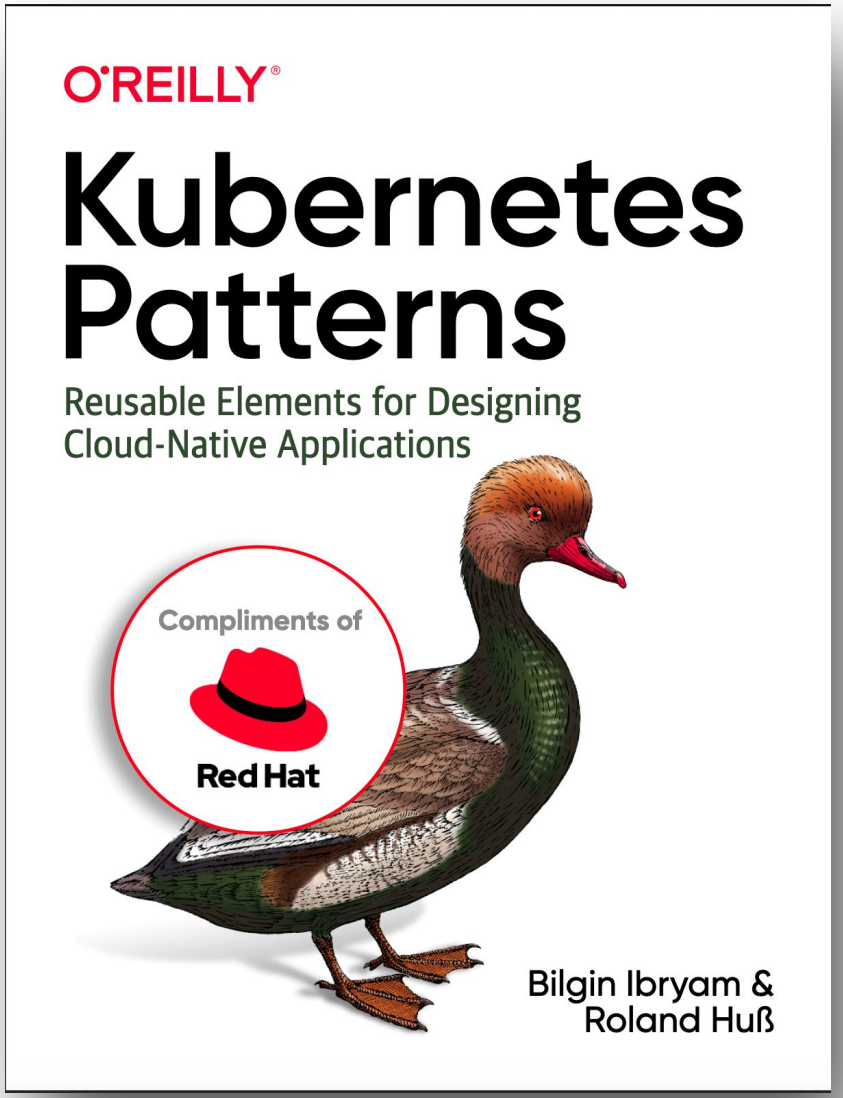
Red Hat



https://k8spatterns.io

Red Hat

https://twitter.com/Grady_Booch/status/1252674791376449536



http://bit.ly/kubernetes-patterns-red-hat

Red Hat

Examples

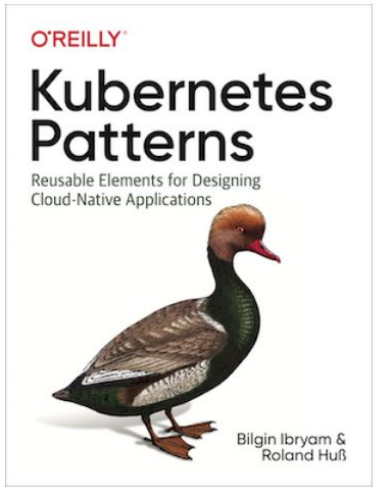
README.adoc

Kubernetes Patterns - Examples

This GitHub project contains the examples from *Kubernetes Patterns - Reusable Elements for Designing Cloud-Native Applications* book by Bilgin Ibryam and Roland Huß

Installation instructions for the example prerequisites are summarised in INSTALL. By default, you need access to a vanilla Kubernetes installation, like Minikube. If addons are required, the example description explains this additional requirement.

For feedback, issues or questions in general, please use the issue tracker to open issues. Also, we love contributions like spelling fixes, bug fixes, improvements, ... Please open Pull Requests, we are happy to review them!



Patterns

All example are categorised according to the Book's patterns category. Each of the examples is contained in an extra directory per pattern and is self-contained. [1]

Foundational Patterns

Predictable Demands

Our sample random generator dealing with hard requirements on ConfigMap and PersistentVolumeClaims as well as with resource limits.

Declarative Deployment

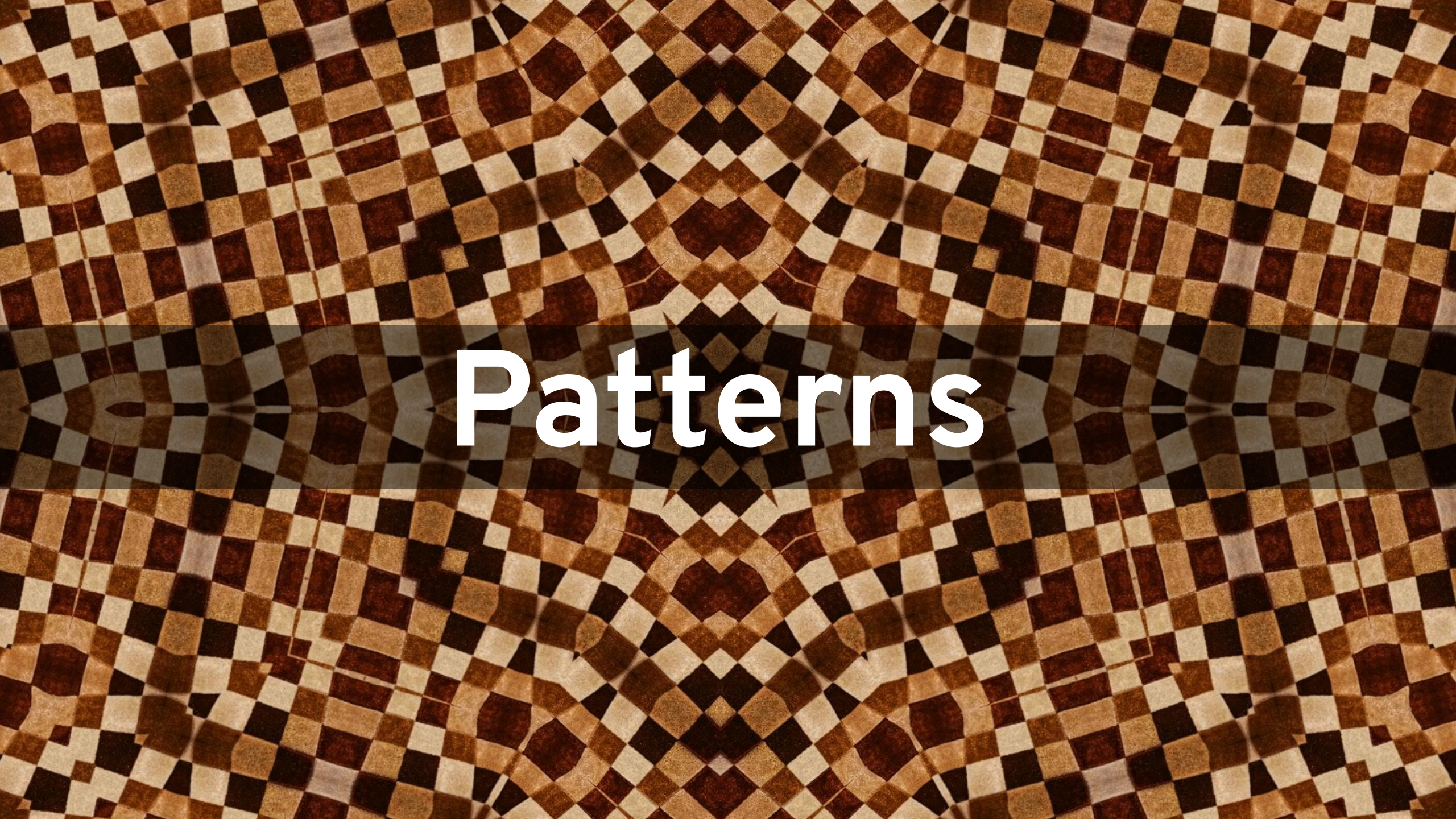
Rolling and fixed update of the random generator Deployment from version 1.0 to 2.0.

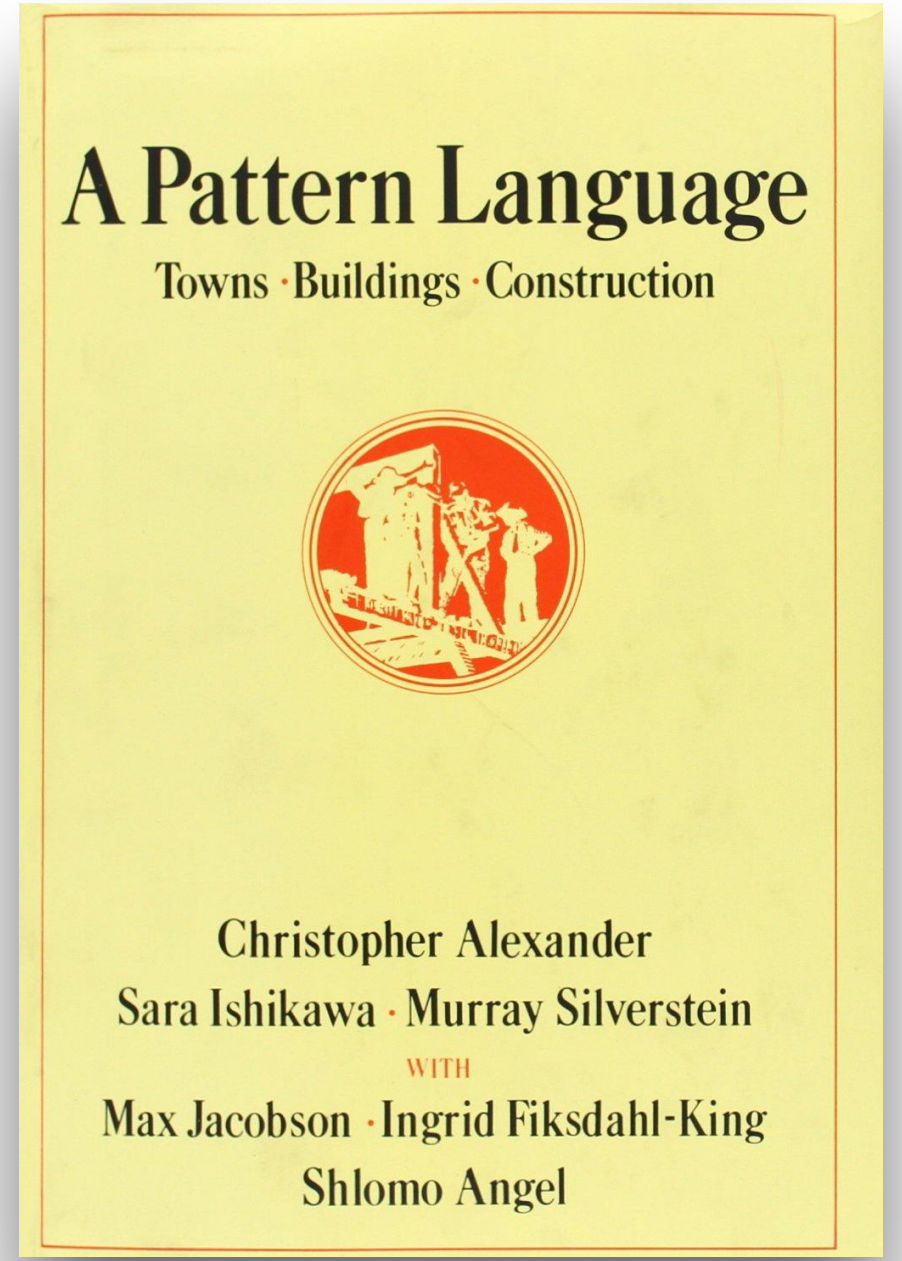
Health Probe

Liveness and Readiness probes for the random generator.

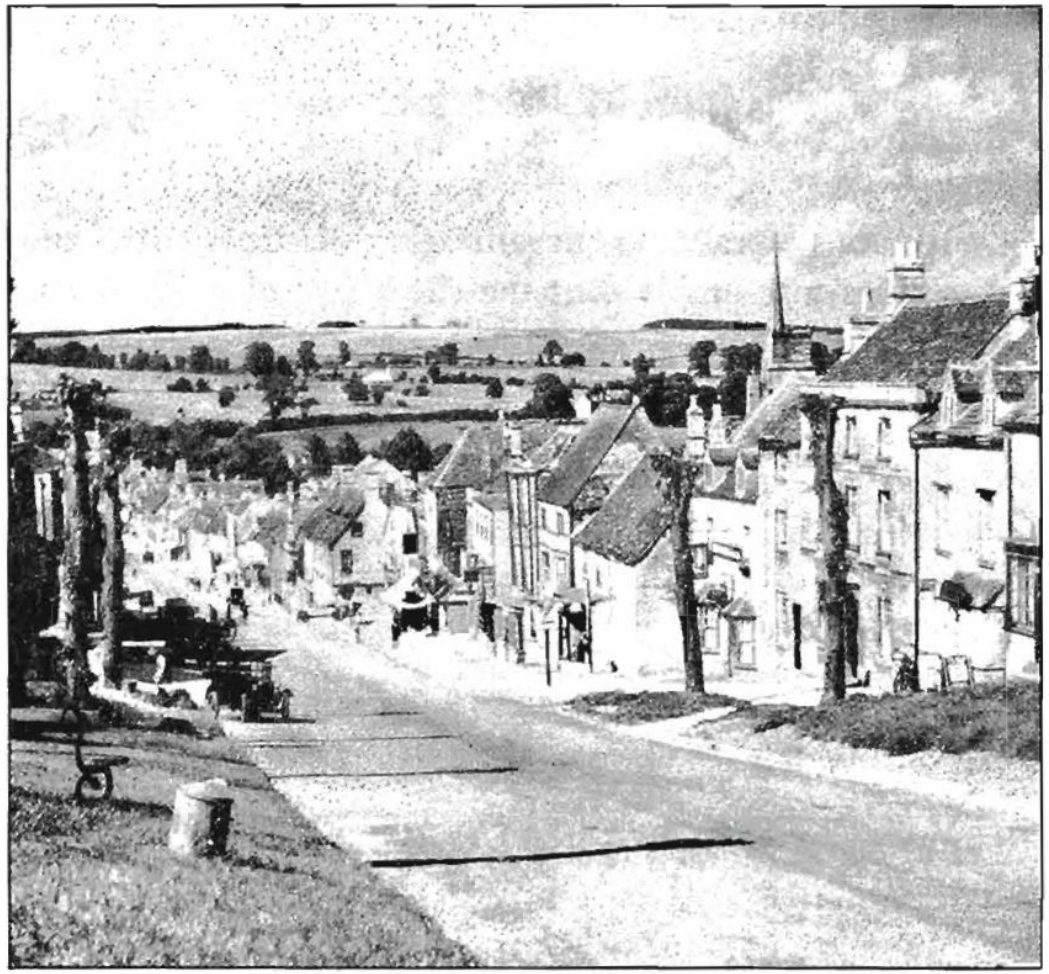
https://github.com/k8spatterns/examples

Red Hat

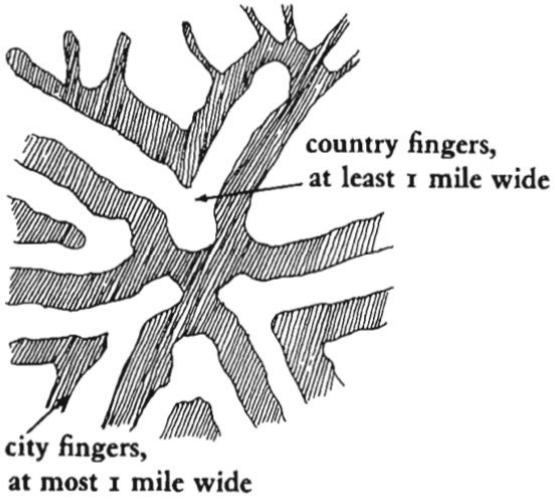

Patterns


A Pattern Language
Towns · Buildings · Construction
Christopher Alexander
Sara Ishikawa · Murray Silverstein
WITH
Max Jacobson · Ingrid Fiksdahl-King
Shlomo Angel

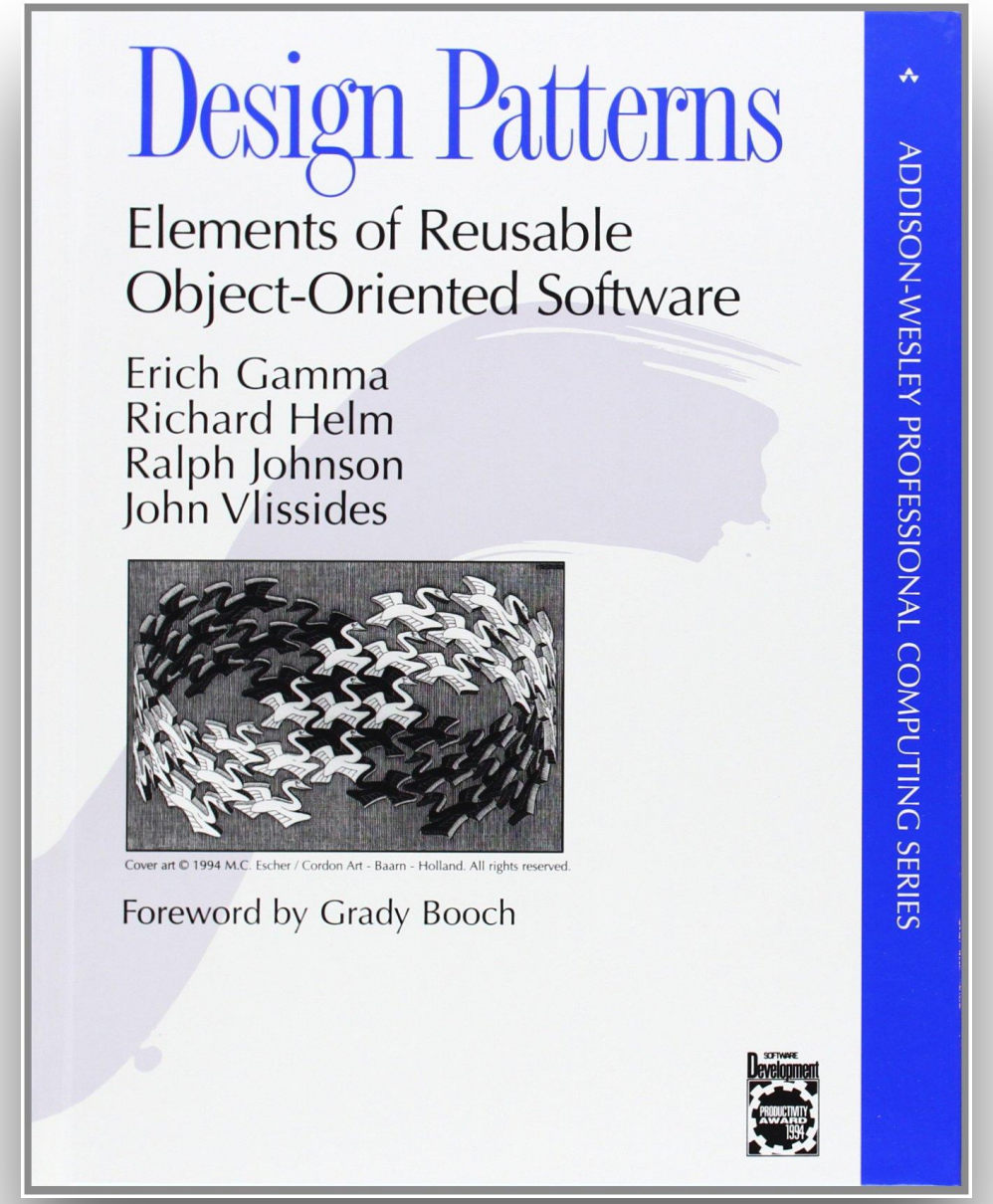
3 CITY COUNTRY FINGERS**



When the countryside is far away
the city becomes a prison.



A Pattern Language, Christopher Alexander et. al, 1977, pp. 21


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

Red Hat

Patterns

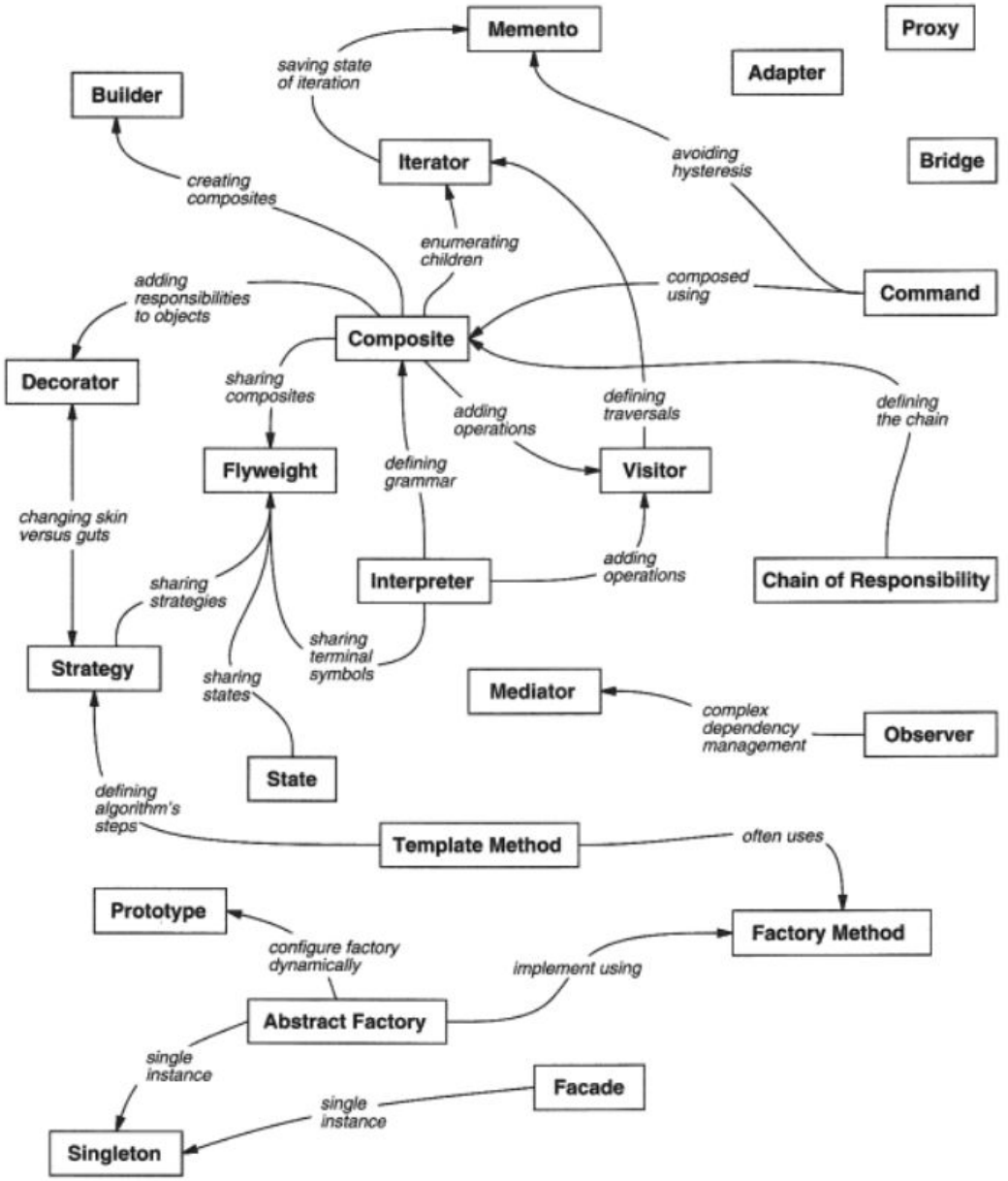


Figure 1.1: Design pattern relationships

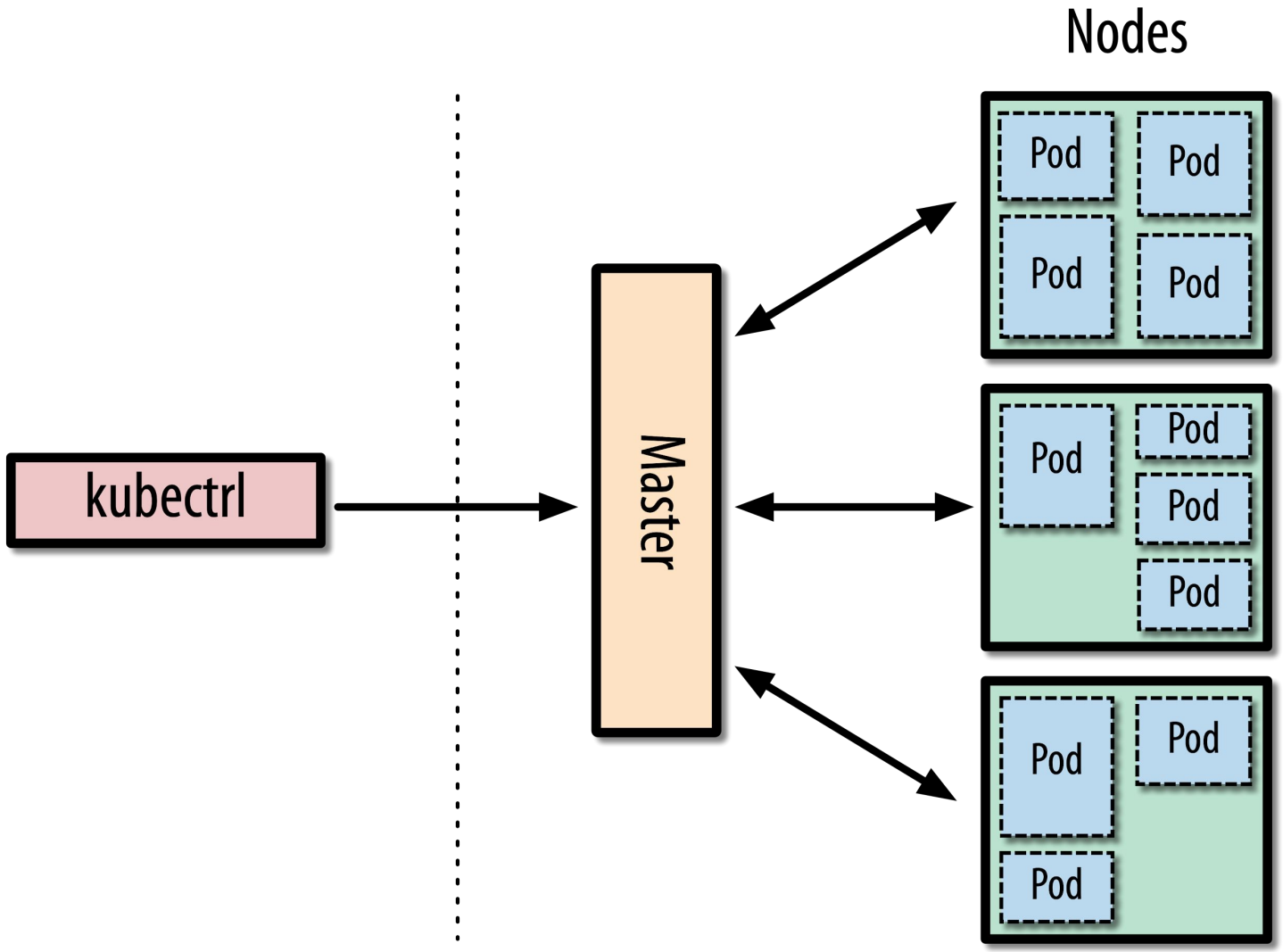
Design Patterns, Elements of Reusable Object-Oriented Software, E. Gamma et. al., 1994, p. 12


Kubernetes

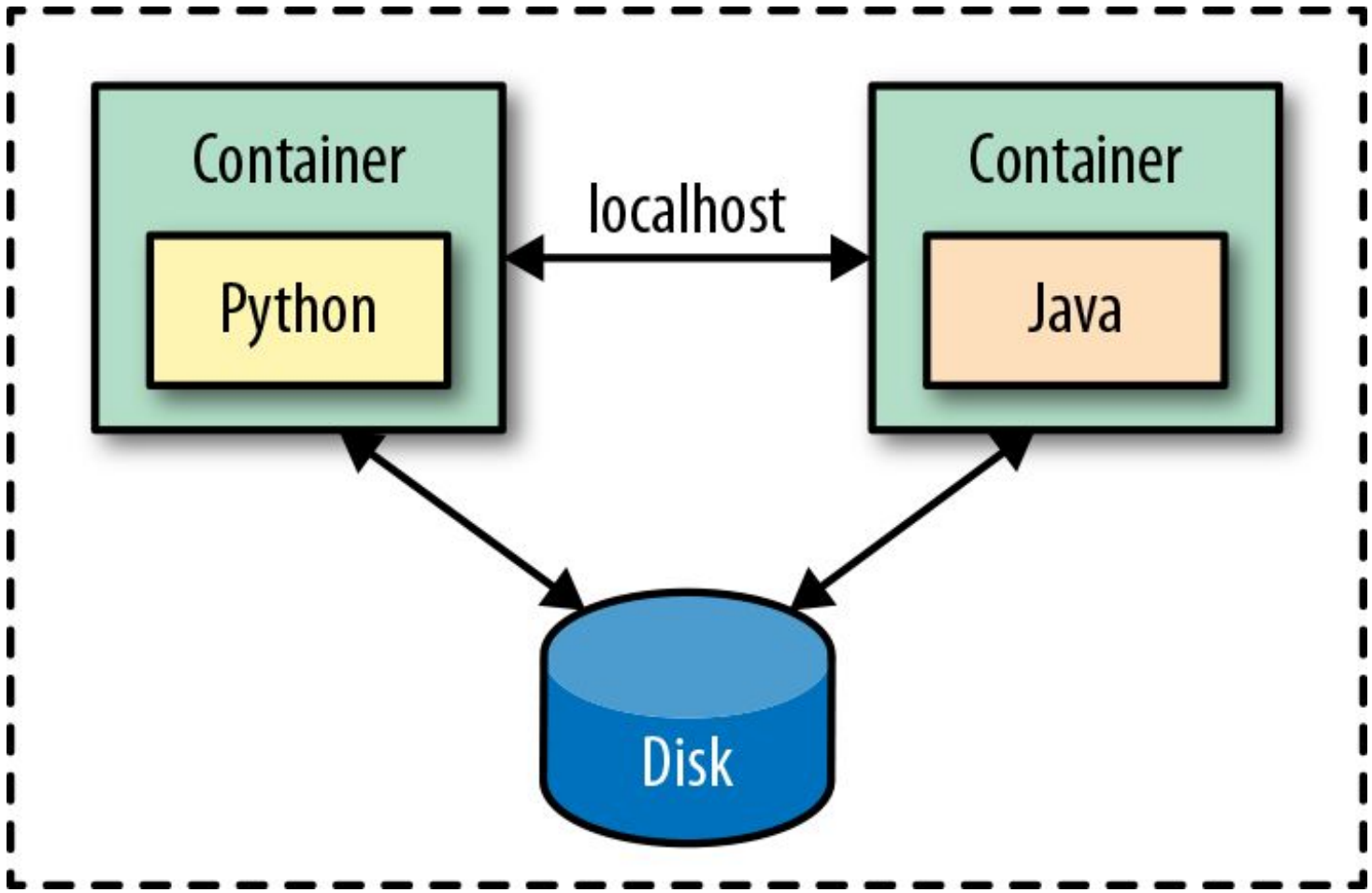
Kubernetes

- Open Source container orchestration system started by Google in 2014
  - Scheduling
  - Self-healing
  - Horizontal and vertical scaling
  - Service discovery
  - Rollout and Rollbacks
- Declarative resource-centric REST API

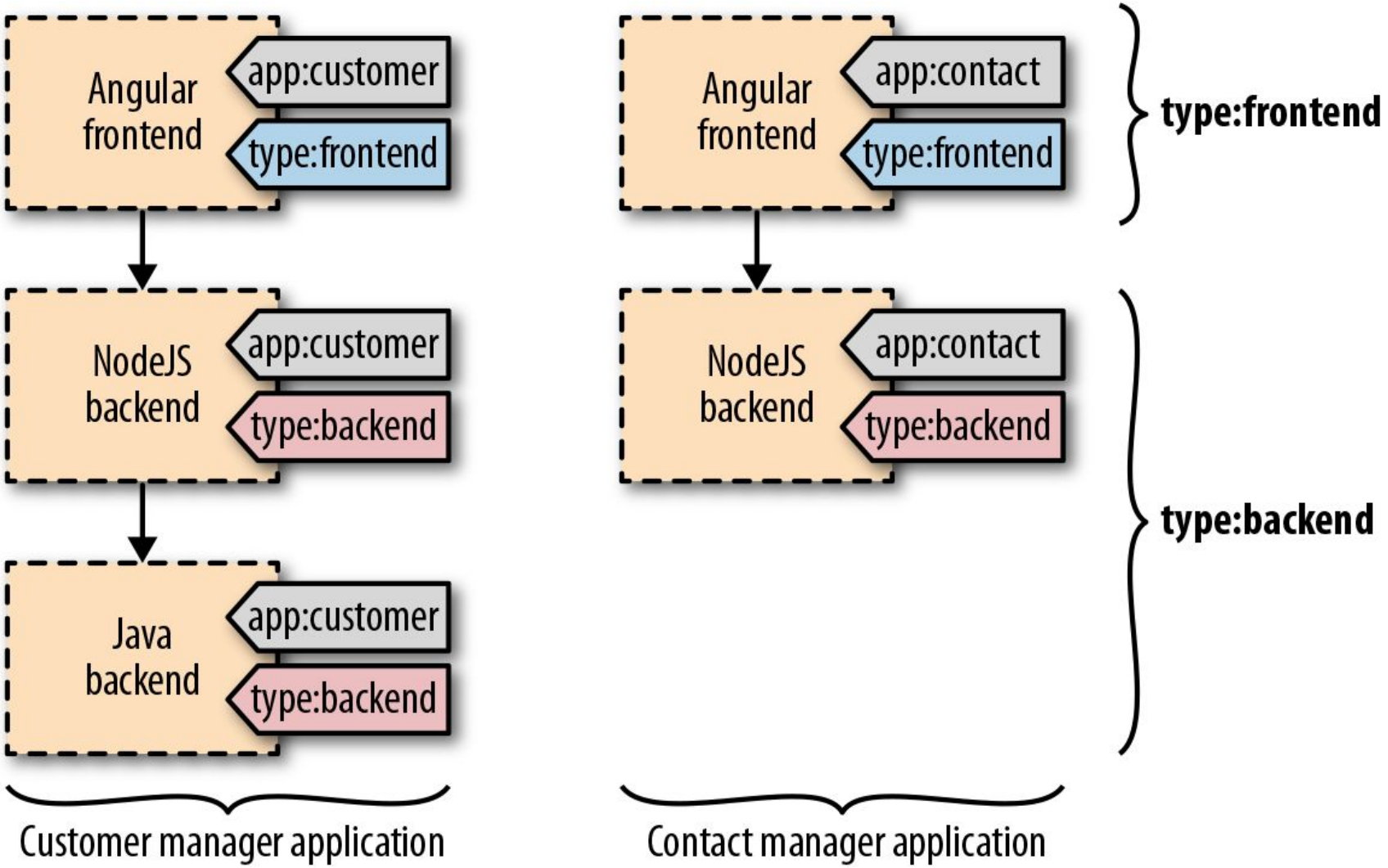
Architecture

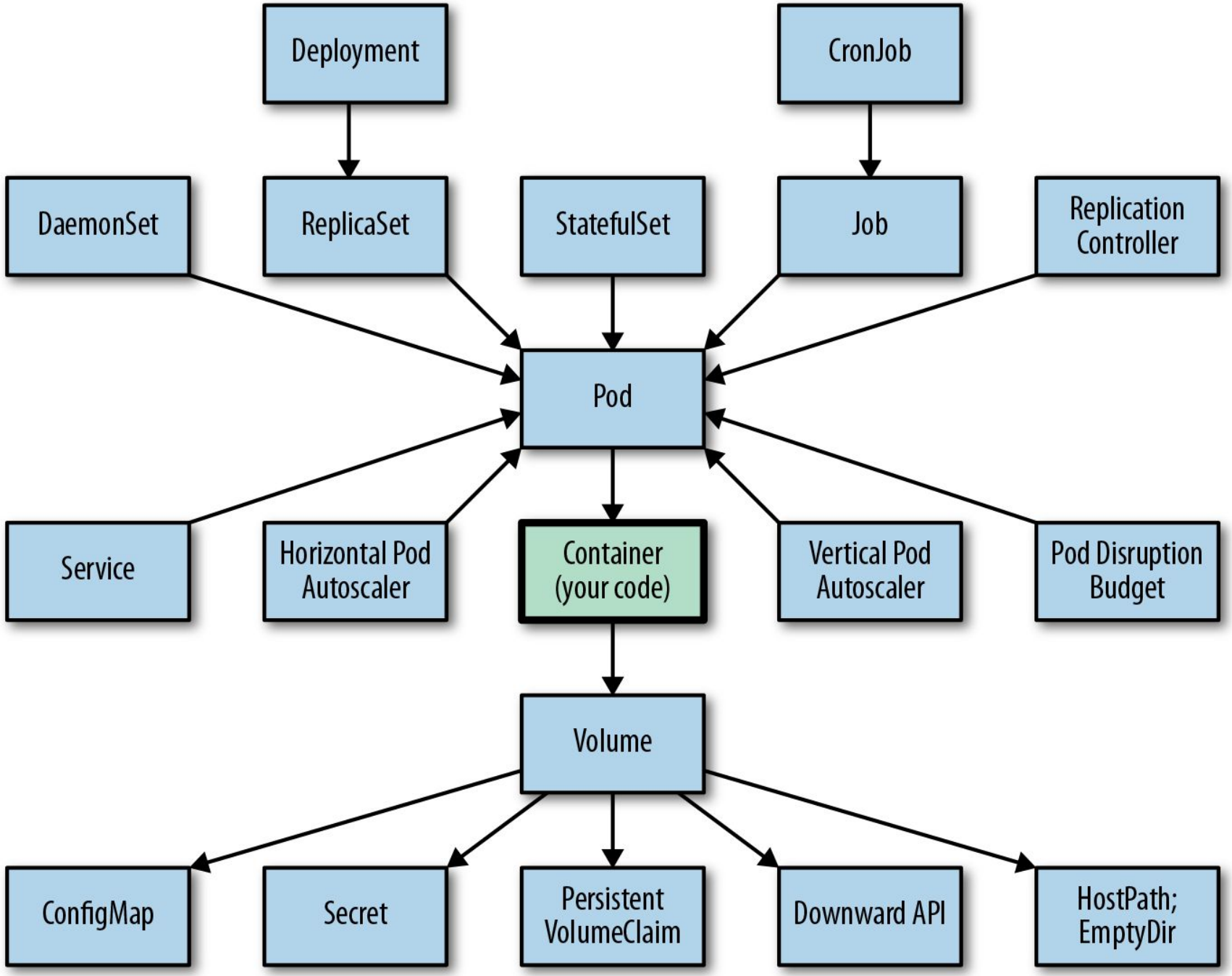


Pod



Labels

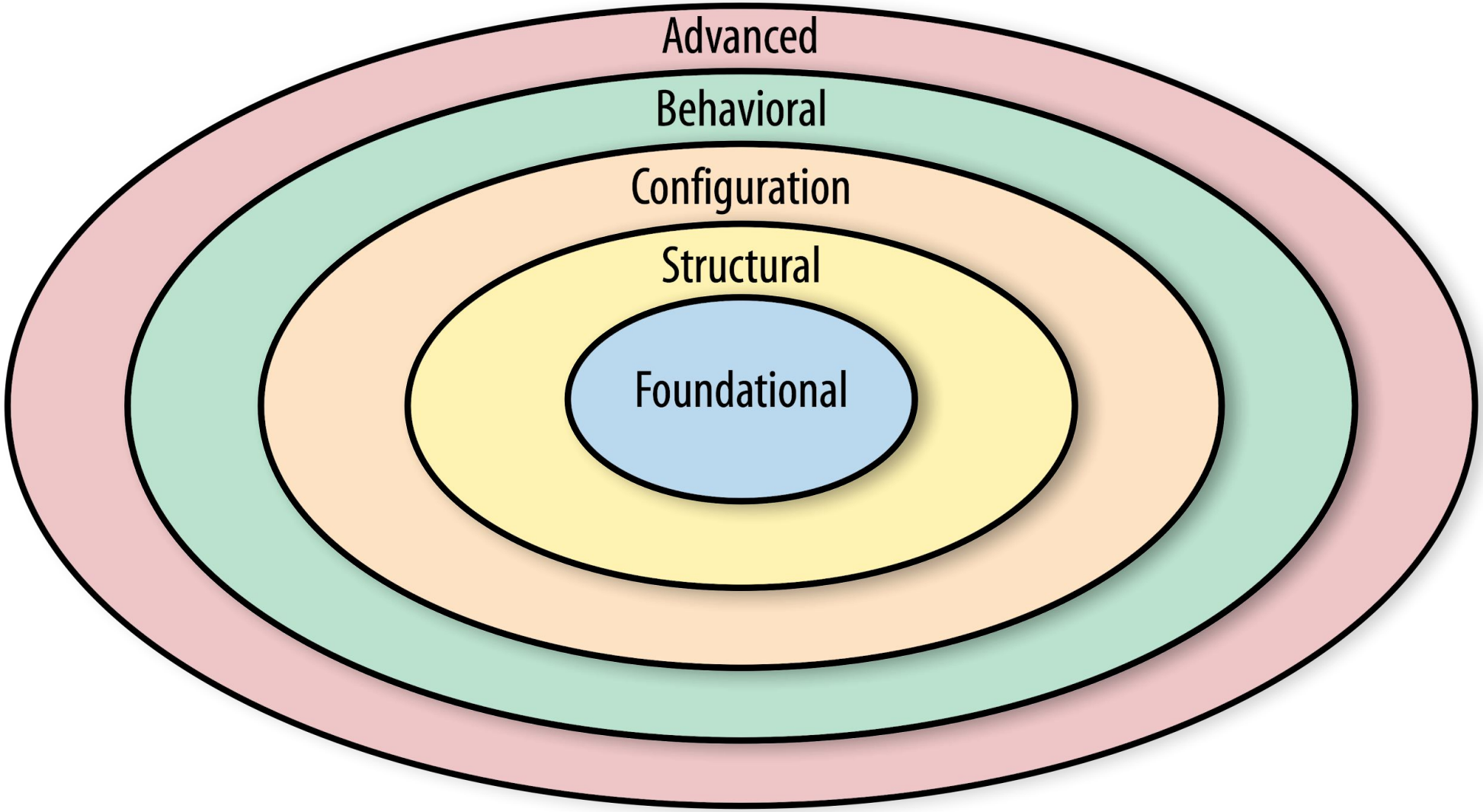



Deployment
CronJob
DaemonSet
ReplicaSet
StatefulSet
Job
Replication Controller
Pod
Service
Horizontal Pod Autoscaler
Container (your code)
Vertical Pod Autoscaler
Pod Disruption Budget
Volume
ConfigMap
Secret
Persistent VolumeClaim
Downward API
HostPath; EmptyDir

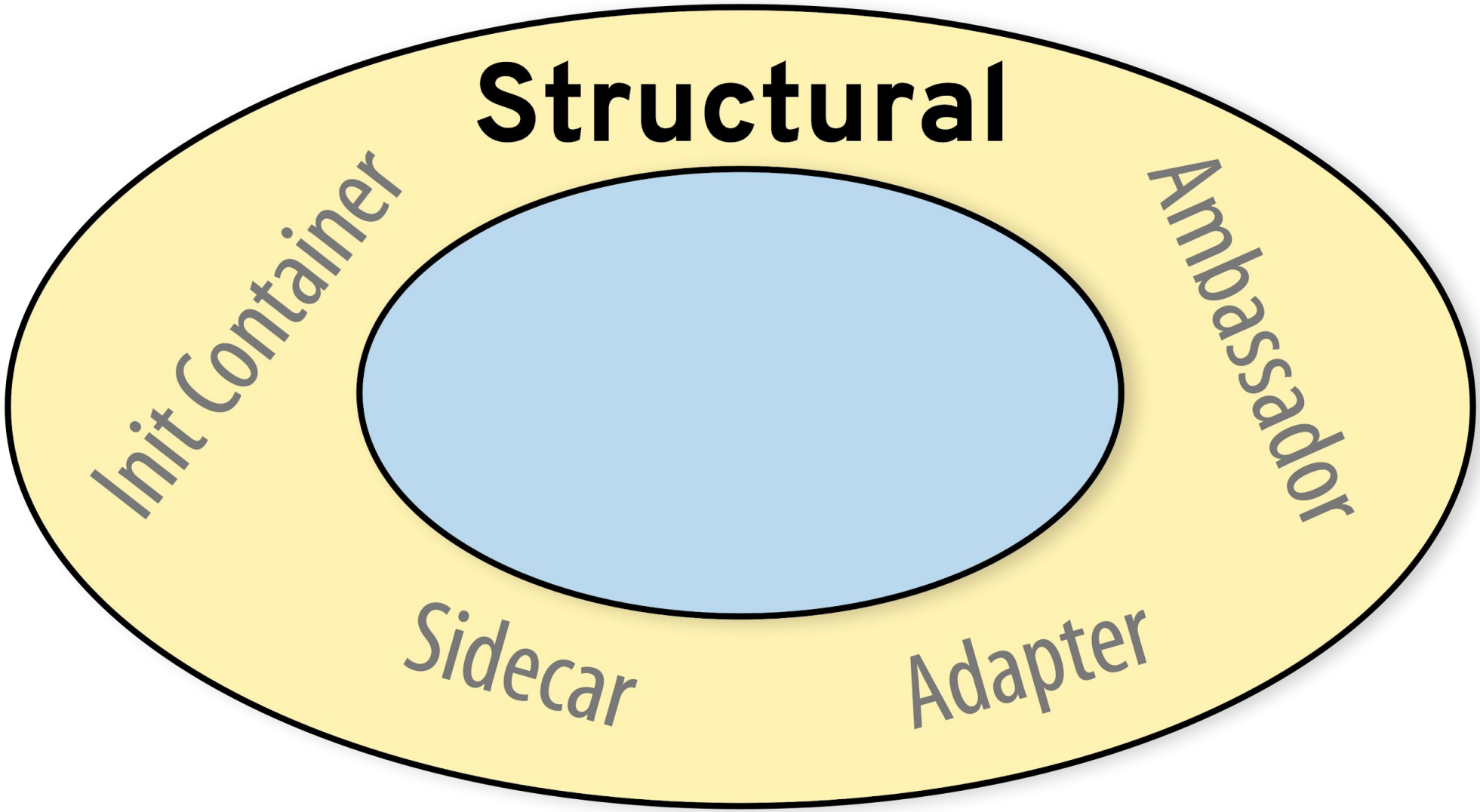
Local vs Distributed Primitives

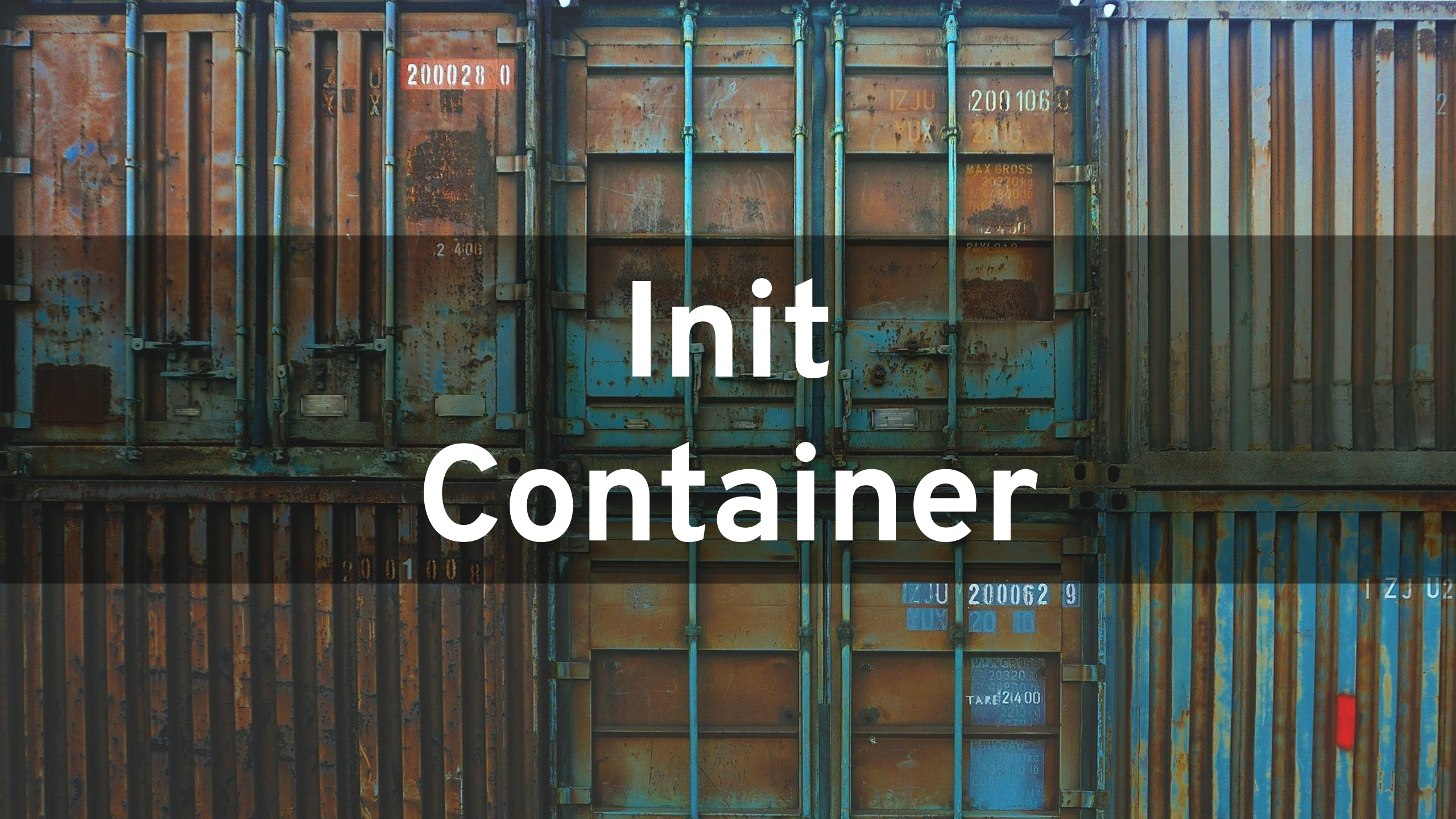
Concept	Local primitive	Distributed primitive
Behavior encapsulation	Class	Container image
Behavior instance	Object	Container
Unit of reuse	*.jar*	Container image
Composition	Class A contains Class B	Sidecar pattern
Inheritance	Class A extends Class B	A container's `FROM` parent image
Deployment unit	*.jar*/*.war*/*.ear*	Pod
Buildtime/Runtime isolation	Module, Package, Class	Namespace, Pod, container
Initialization preconditions	Constructor	Init container
Postinitialization trigger	Init-method	`postStart`
Predestroy trigger	Destroy-method	`preStop`
Cleanup procedure	`finalize()`, shutdown hook	Defer container[a]
Asynchronous & parallel execution	`ThreadPoolExecutor`, `ForkJoinPool`	Job
Periodic task	`Timer`, `ScheduledExecutorService`	CronJob
Background task	Daemon thread	DaemonSet
Configuration management	`System.getenv()`, `Properties`	ConfigMap, Secret

Pattern Categories



Red Hat



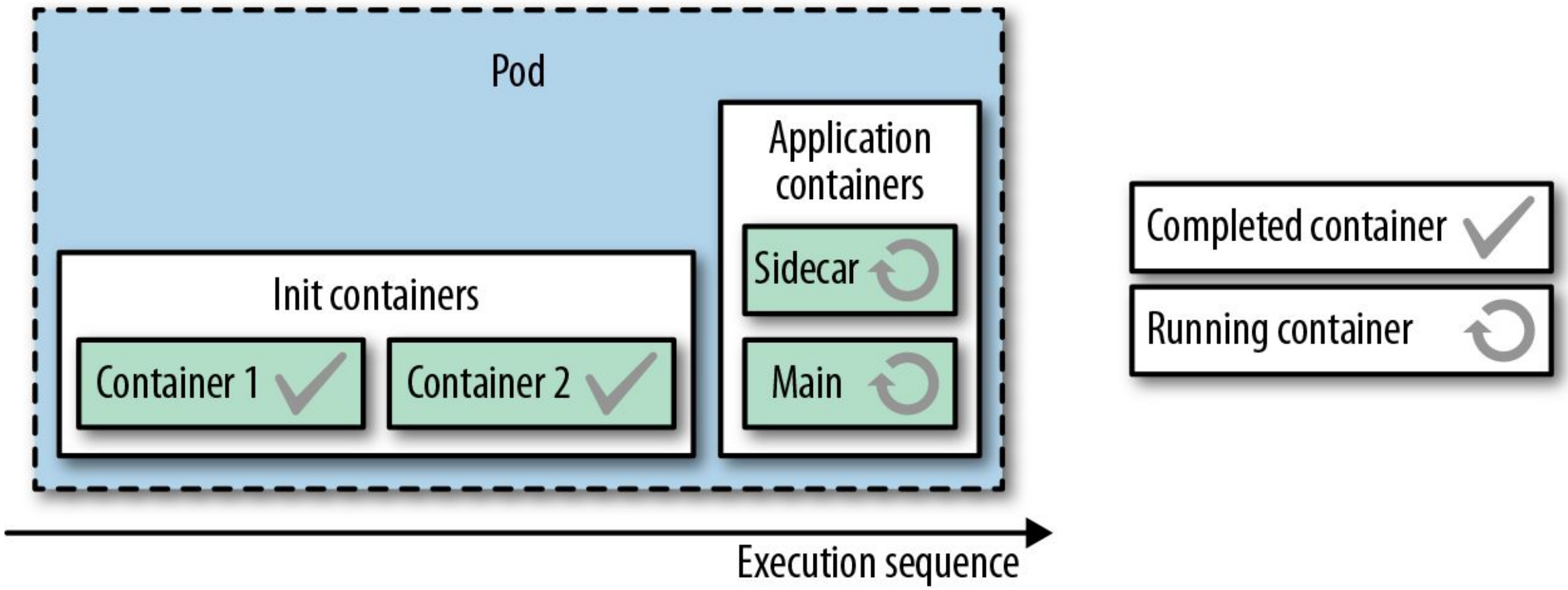

Init
Container

How to separate initialization steps from the main application

Init Container

- Part of a Pod
- One shot actions before application starts
- Needs to be idempotent
- Has own resource requirements

Init Container



Red Hat

Init Container

```
apiVersion: v1
kind: Pod
....
spec:
  initContainers:
  - name: download
    image: axeclbr/git
    command: [ "git","clone","https://github.com/myrepo","/data"]
    volumeMounts:
    - mountPath: /var/lib/data
      name: source
  containers:
  - name: run
    image: docker.io/centos/httpd
    volumeMounts:
    - mountPath: /var/www/html
      name: source
  volumes:
  - emptyDir: {}
    name: source
```

Demo

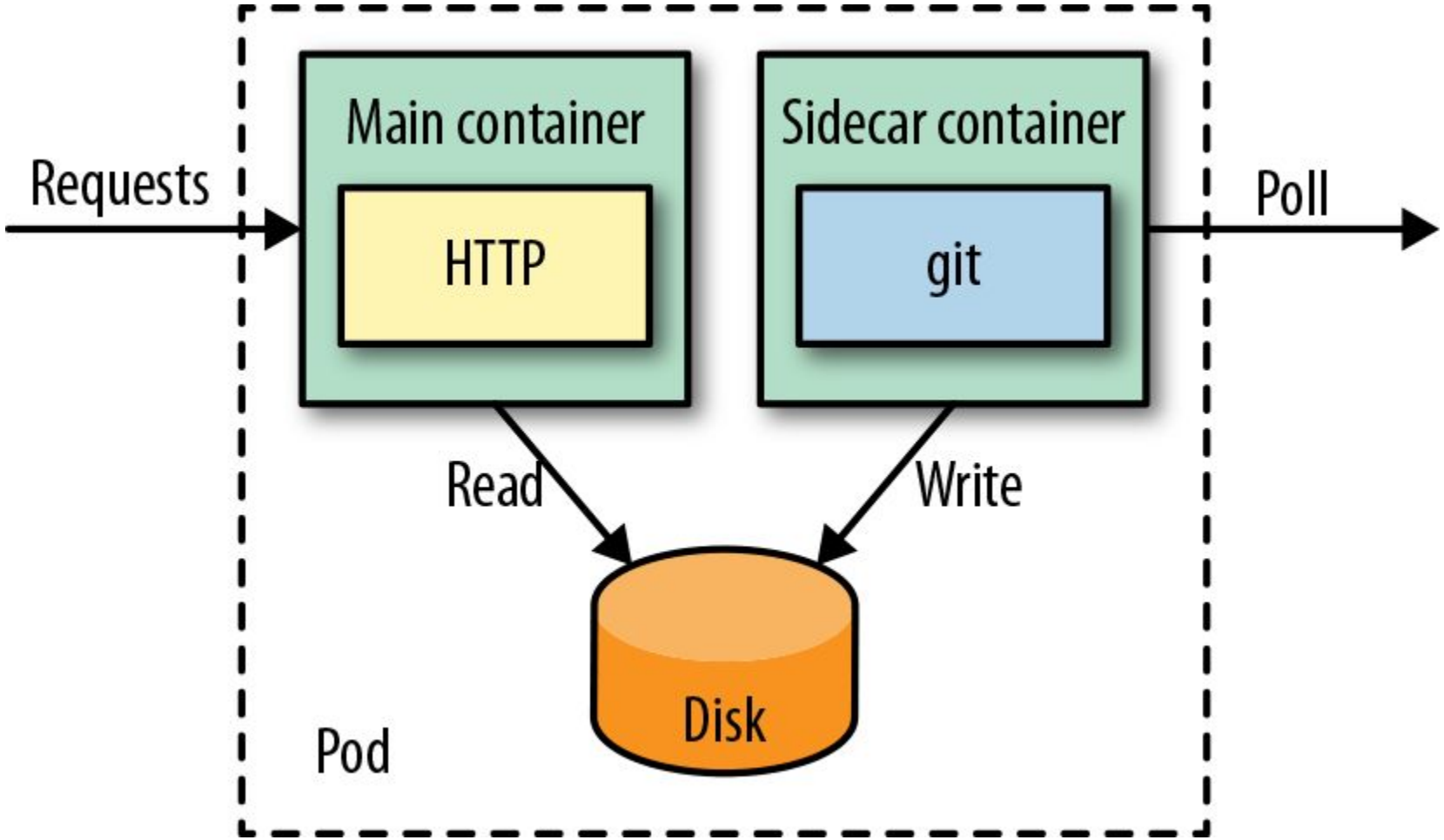
Sidecar

How to enhance the functionality of an application without changing it

Sidecar

- Runtime collaboration of containers
- Connected via shared resources:
  - Network
  - Volumes
- Similar what AOP is for programming
- Separation of concerns

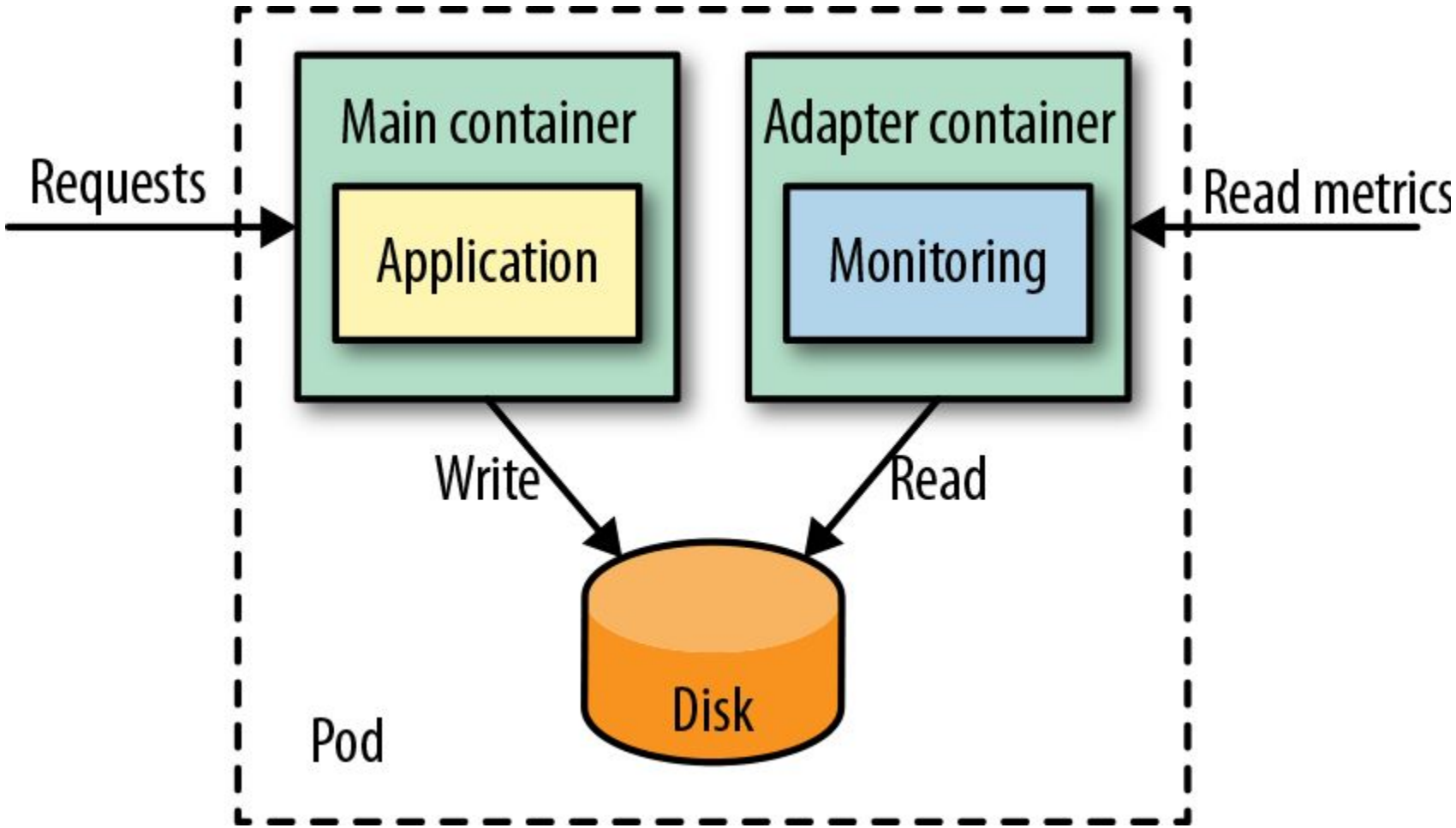
Sidecar



Adapter

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

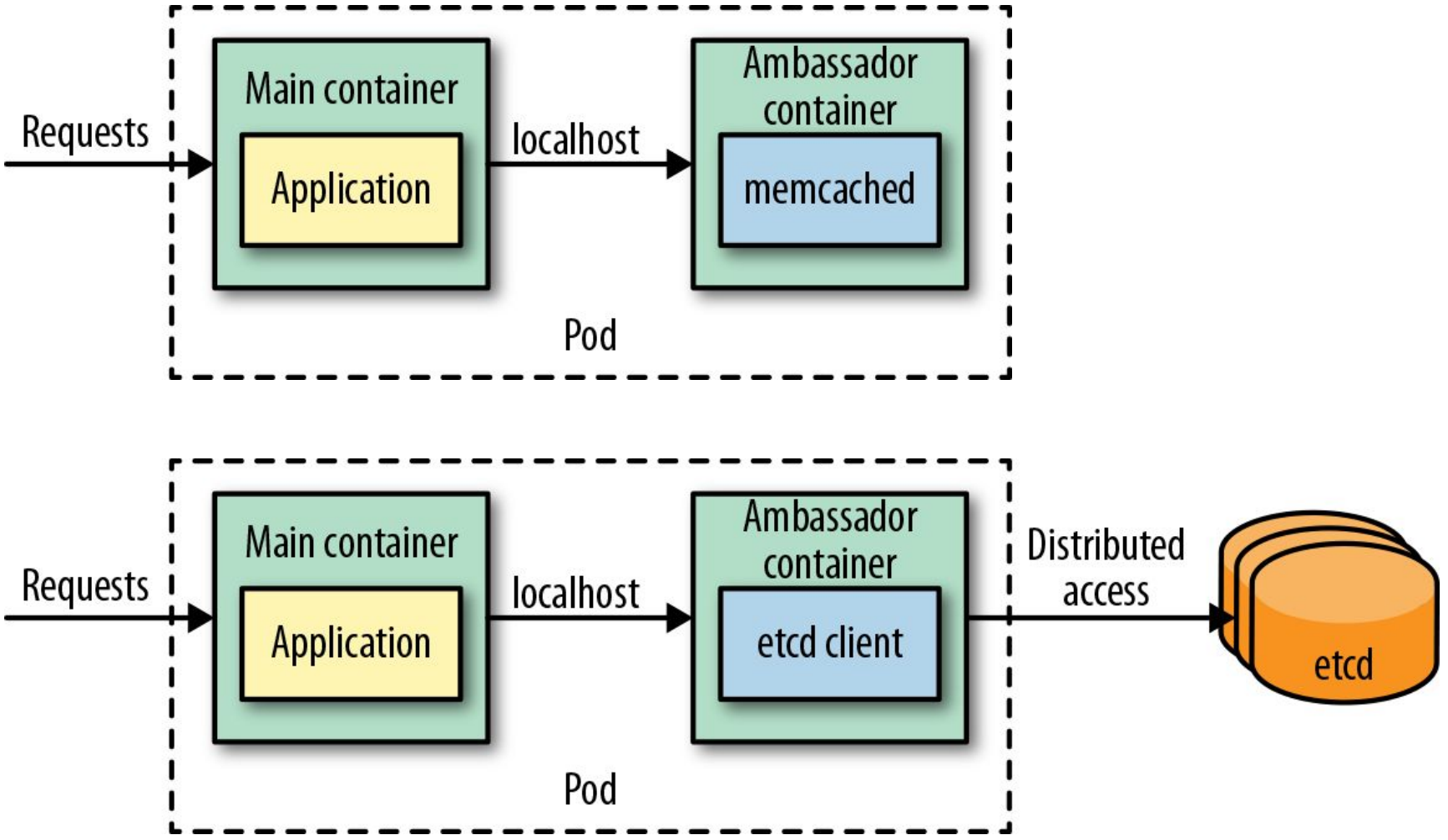
Adapter




Ambassador

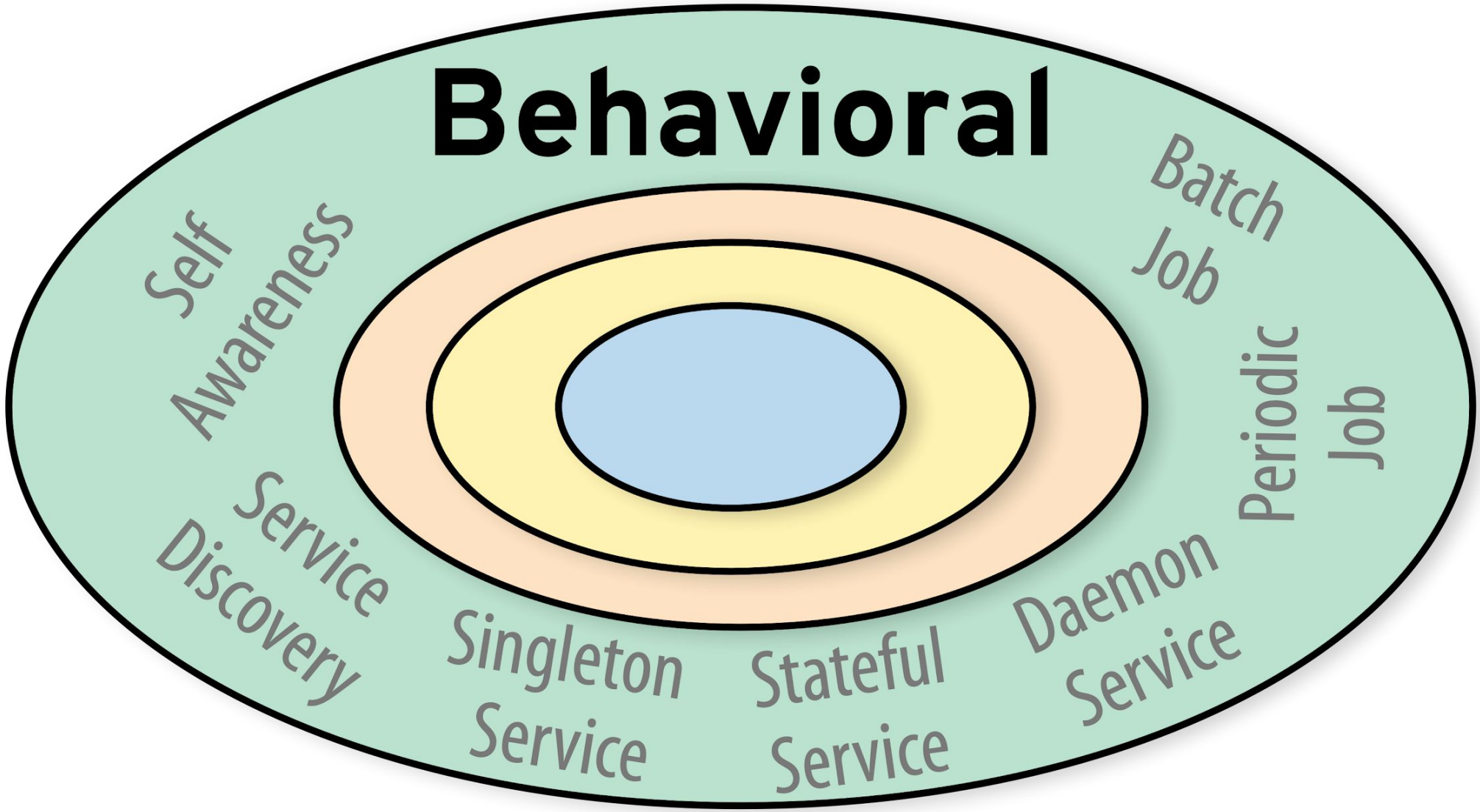
How to decouple a container's access **to** the outside world

Ambassador



Demo

Behavioral Patterns



Red Hat


Stateful
Service

How to manage stateful workloads

Distributed Stateful Services

- Non-shared persistent Storage
- Unique and stable network address
- Unique identity
- Defined instance order
- Minimal availability

StatefulSet

- Similar to ReplicaSet
- Additional Elements
  - serviceName - reference to headless Service
  - volumeClaimTemplates - template for creating instance unique PVCs
- Assigned PVs are **not** automically deleted
- Headless service for creating DNS entries for each instance's Pod

Headless Service

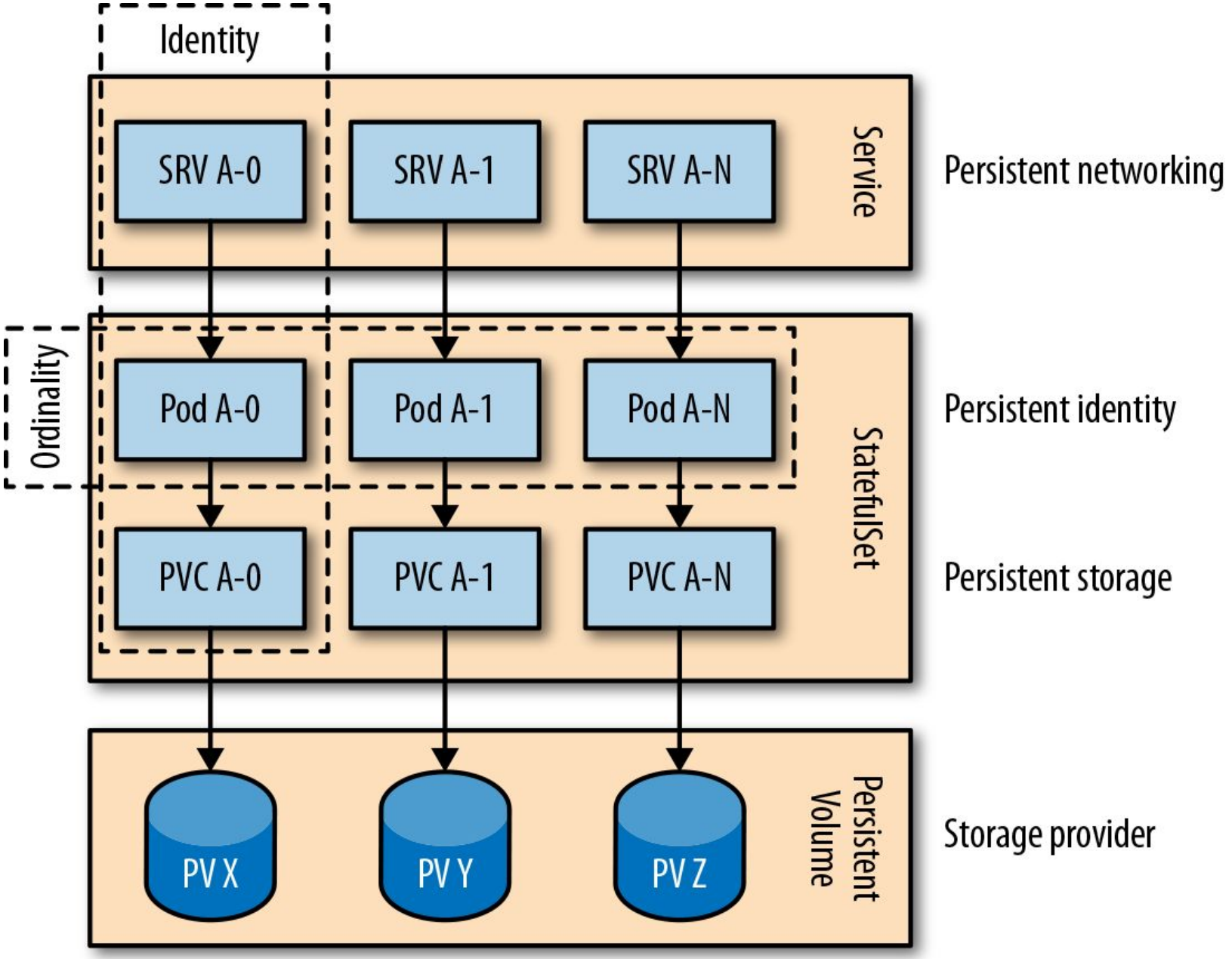
```
apiVersion: v1
kind: Service
metadata:
  name: random-generator
spec:
  clusterIP: None
  selector:
    app: random-generator
  ports:
  - name: http
    port: 8080
```

rg-0.random-generator.default.svc.cluster.local
rg-1.random-generator.default.svc.cluster.local
...

Stateful Service

```
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: rg
spec:
  serviceName: random-generator
  replicas: 2
  selector:
    matchLabels:
      app: random-generator
  template:
    metadata:
      labels:
        app: random-generator
    spec:
      containers:
      - image: k8spatterns/random-generator:1.0
        name: random-generator
        name: http
        volumeMounts:
        - name: logs
          mountPath: /logs
  volumeClaimTemplates:
  - metadata:
      name: logs
    spec:
      resources:
        requests:
          storage: 10Mi
```

Stateful Service



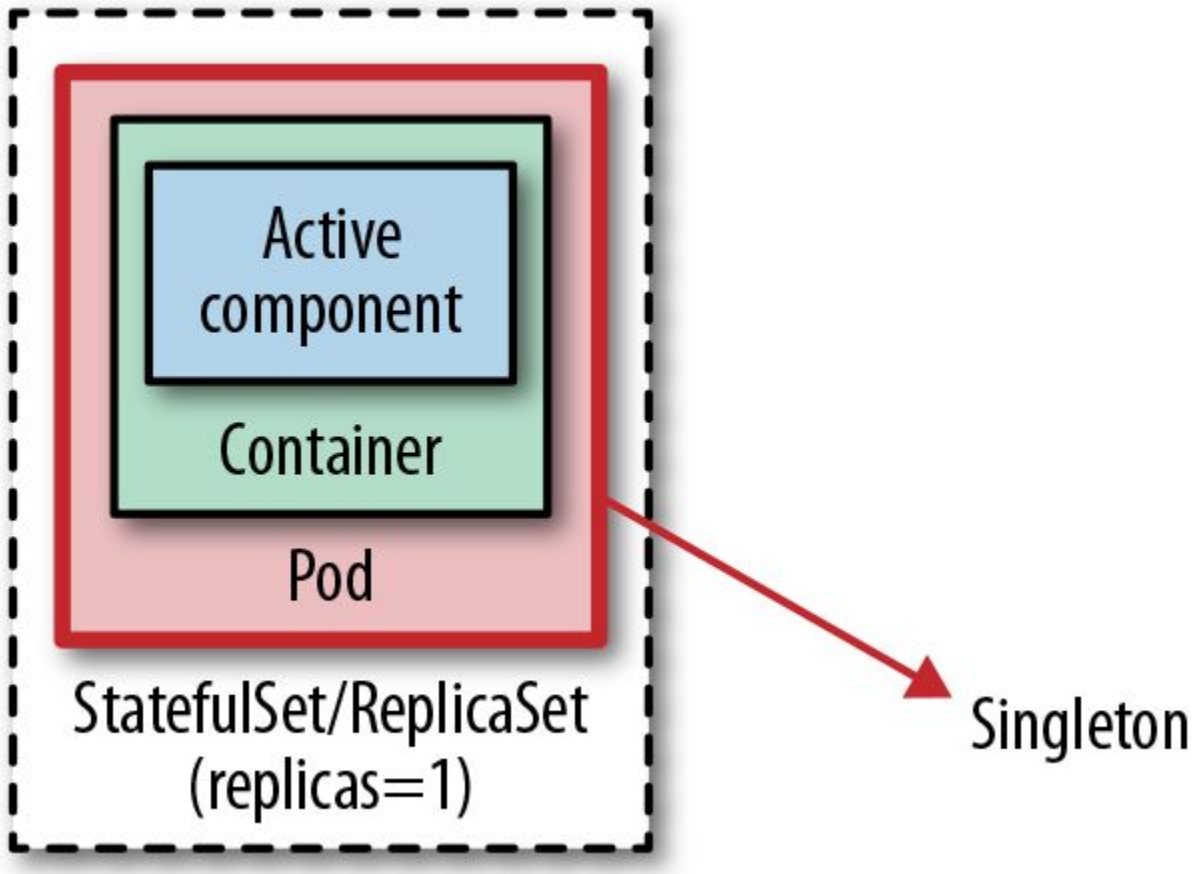
Red Hat

Demo

Singleton Service

How to ensure that only one application instance is active

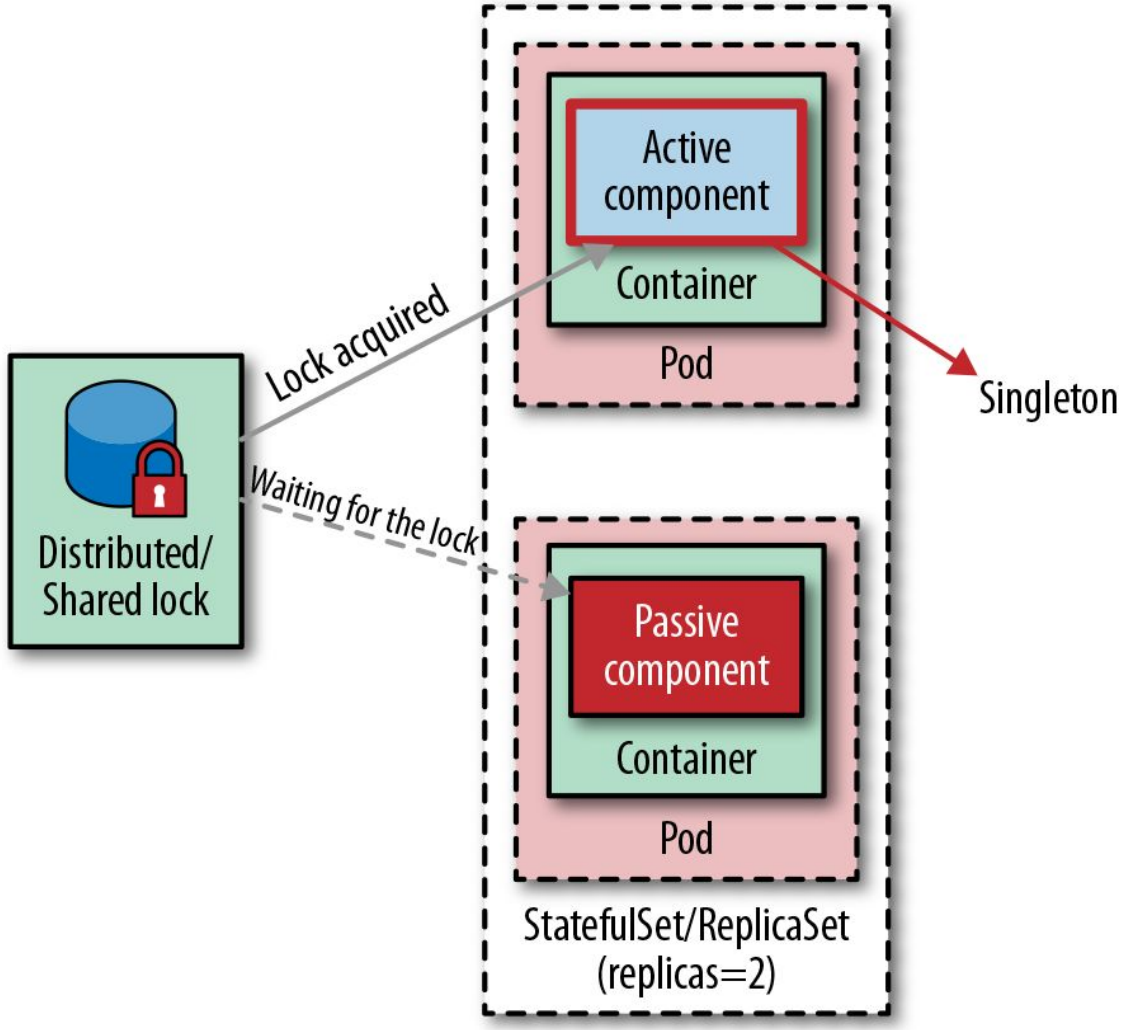
Out-of-Application Locking



Out-of-Application Locking

- ReplicaSet with 1 replica
- Highly available Pod which is monitored and restarted in case of failures
- ReplicaSet favors availability over consistency
  → more than one Pod can exists temporarily
- Alternative: StatefulSet with 1 replica

In-Application Locking



In-Application Locking

- Distributed lock shared by simultaneously running applications
- *Active-Passive* topology
- Distributed lock implementations e.g.
  - Zookeeper
  - Consul
  - Redis
  - etcd

Pod Disruption Budget

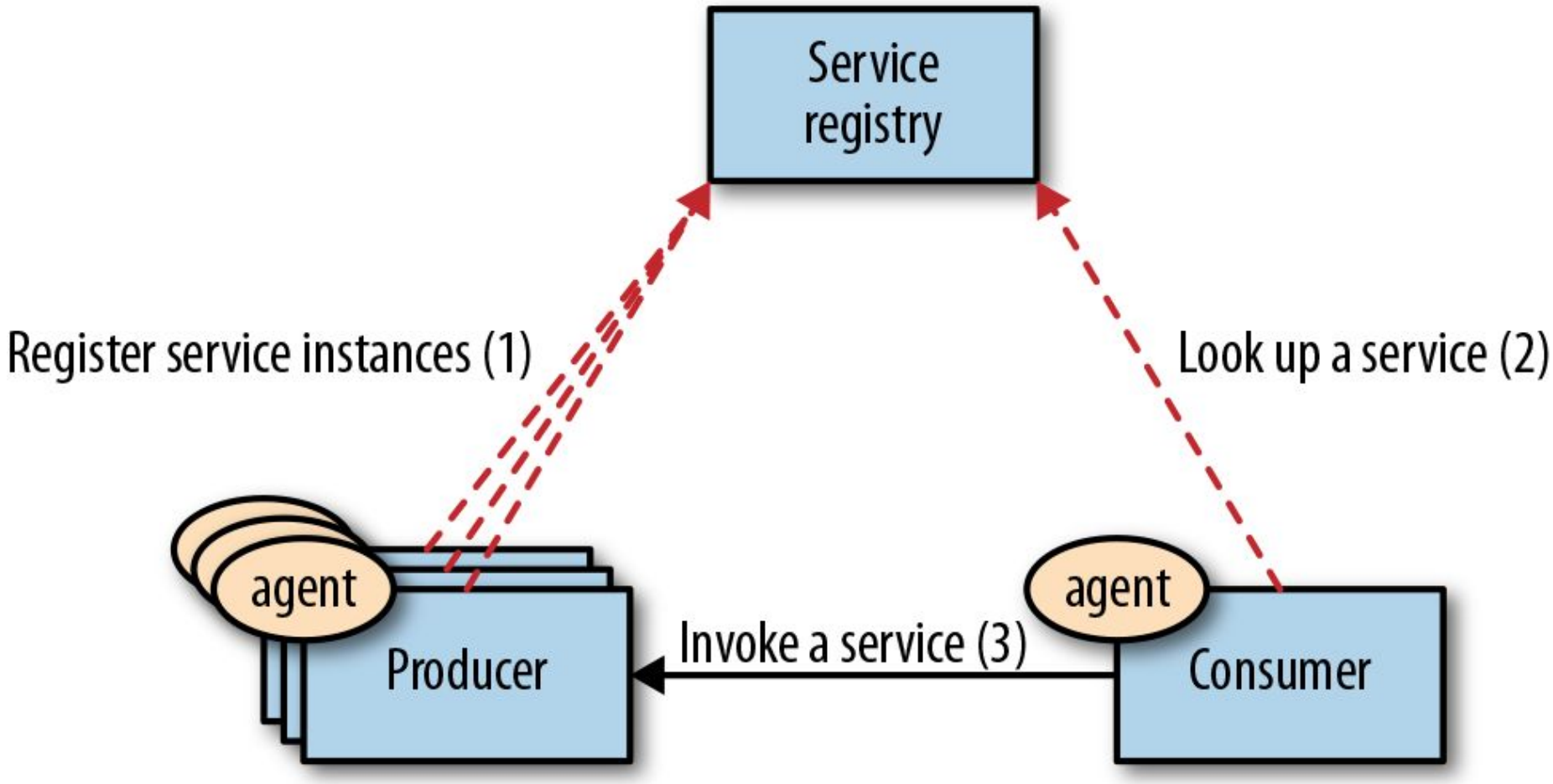
Ensures a certain number of Pods will not *voluntarily* be evicted from a node

```
apiVersion: policy/v1beta1
kind: PodDisruptionBudget
metadata:
  name: random-generator-pdb
spec:
  selector:
    matchLabels:
      app: random-generator
  minAvailable: 2
```

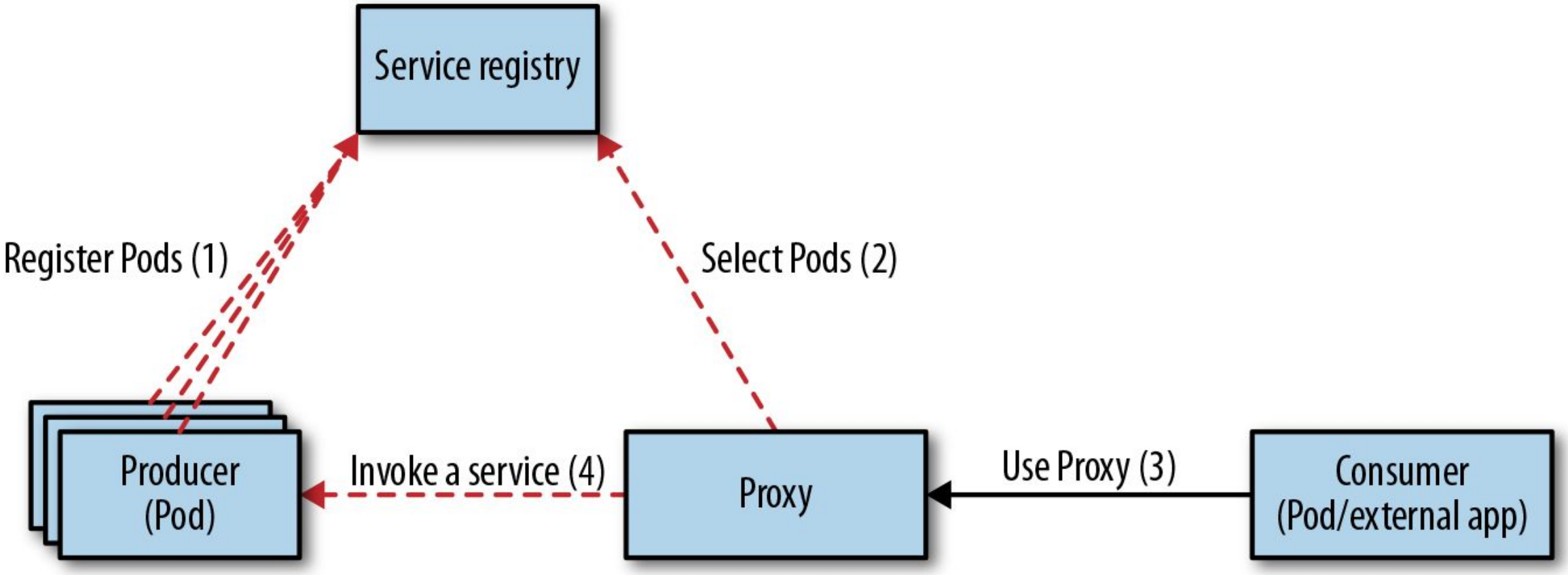

Service
Discovery

How to discover and use services

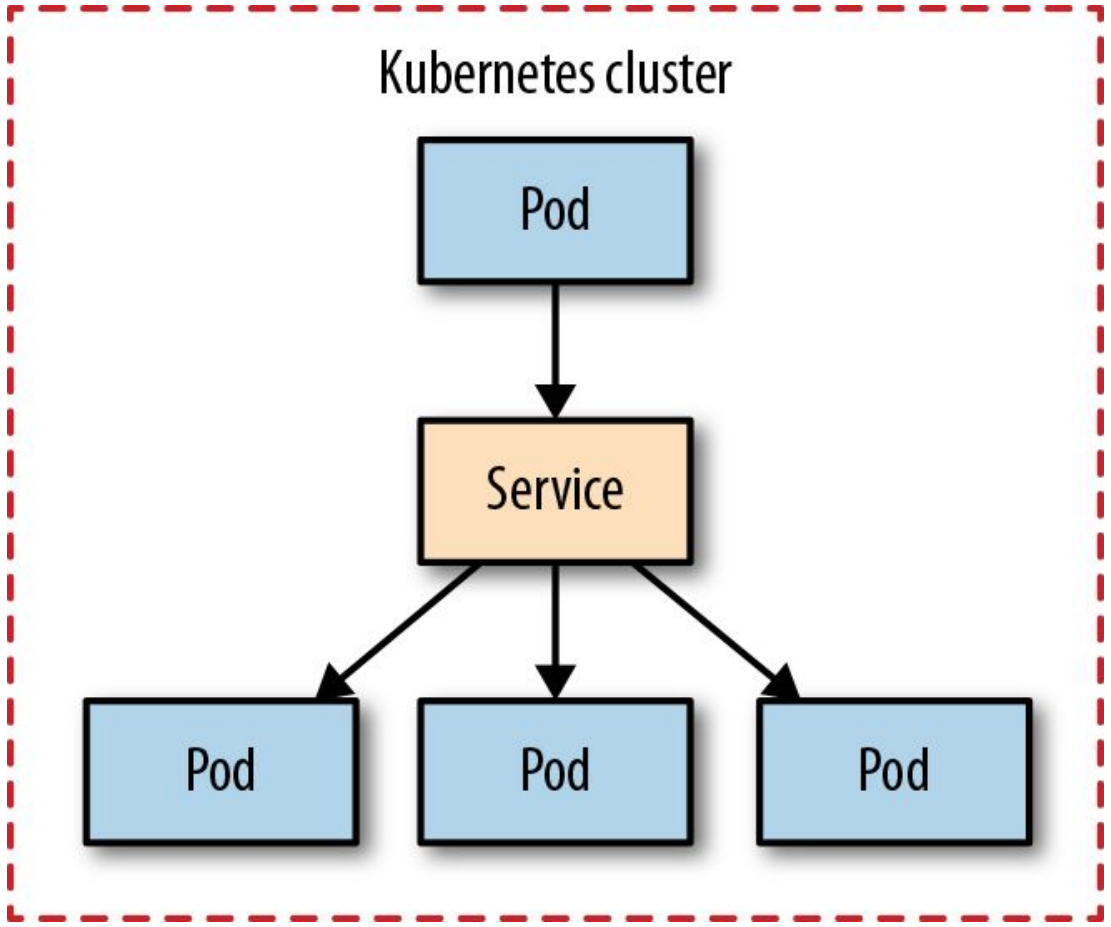
Client-side Service Discovery (non Kubernetes)



Server-side Service Discovery (Kubernetes)

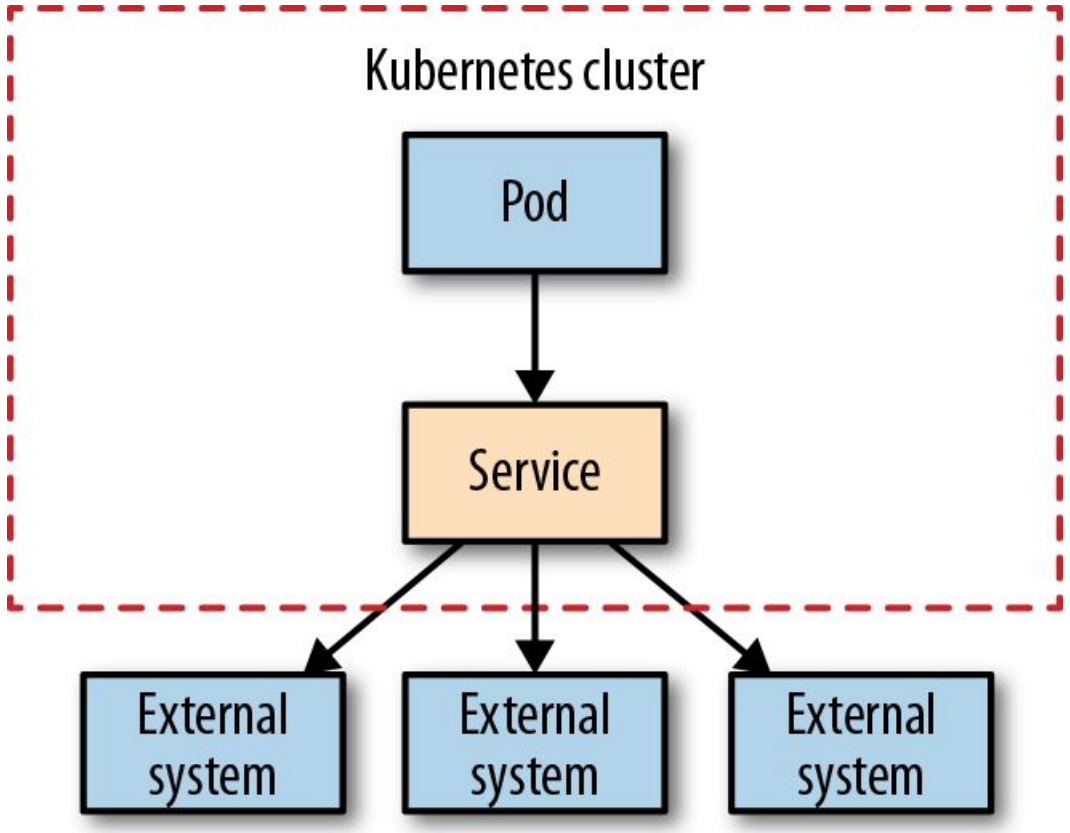


Internal Service Discovery



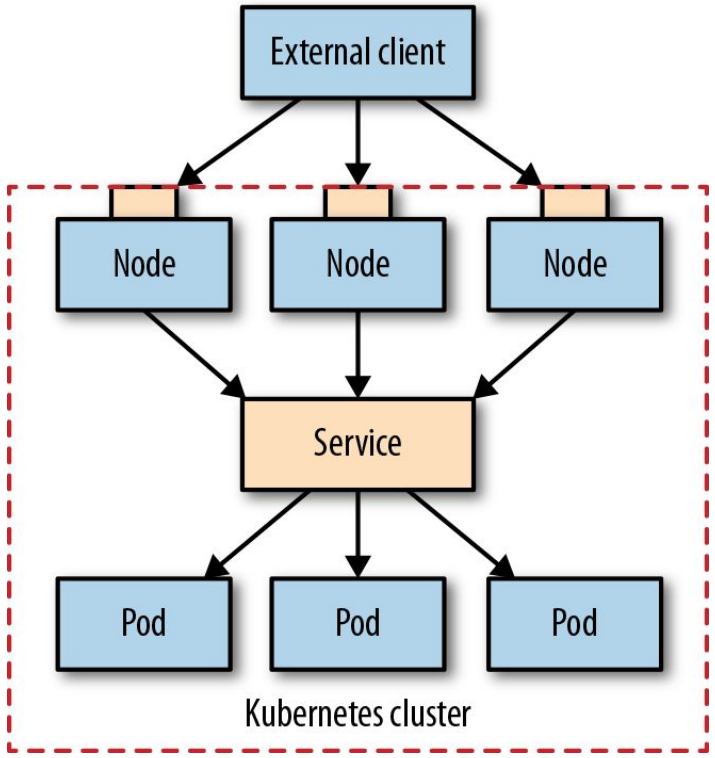
- Discovery through DNS lookups
- Pods picked by label selector
- Multiple ports per Service
- Session affinity on IP address
- Successful readiness probes required for routing
- Virtual IP address for each Service

Manual Service Discovery

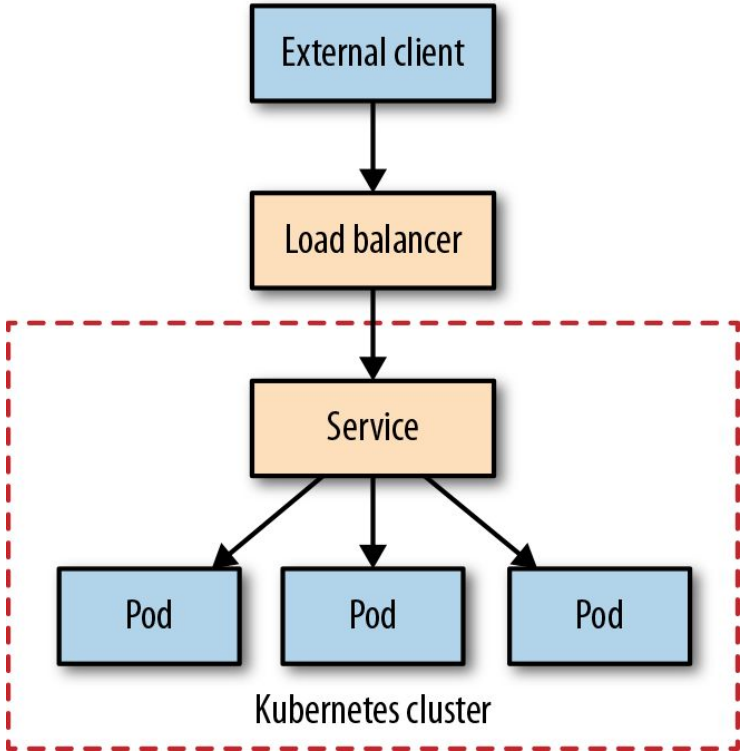


- Service without selector
- Manually creating Endpoint resource with the same name as the Service
- Service of type **`ExternalName`** map are registered as DNS CNAMEs

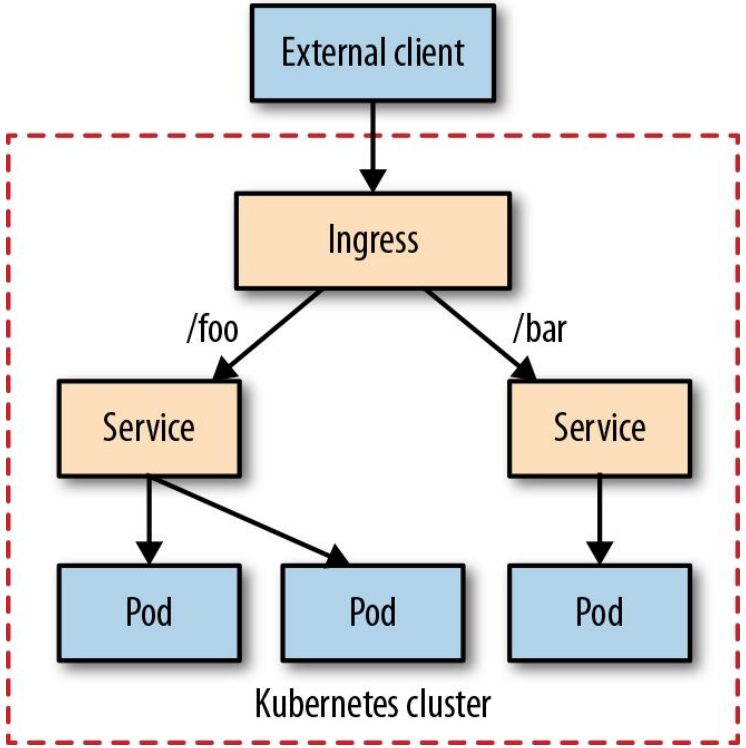
Node Port



Load Balancer



Ingress

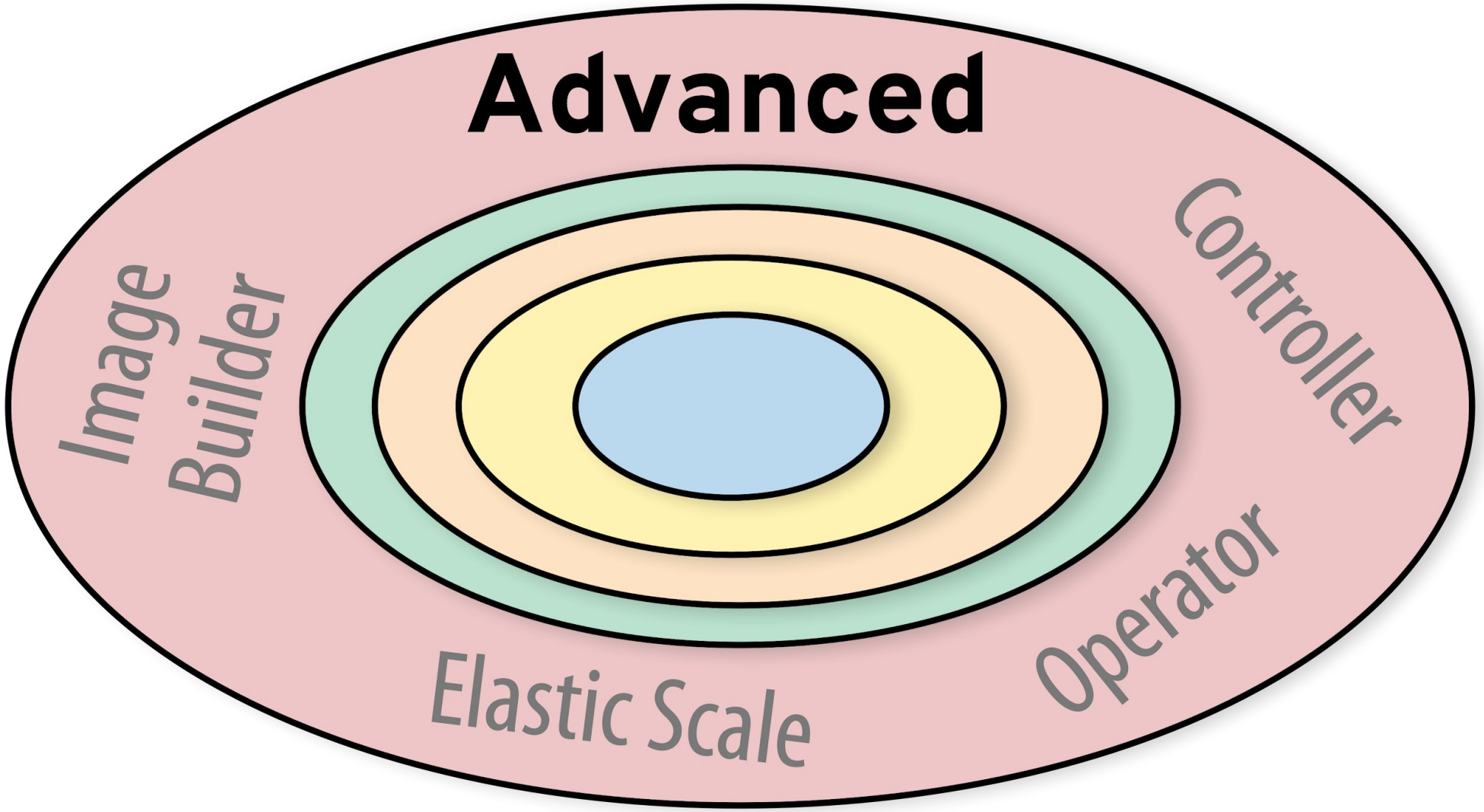


Ingress

```
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: random-generator
spec:
  rules:
  - http:
      paths:
      - path: /
        backend:
          serviceName: random-generator
          servicePort: 8080
      - path: /cluster-status
        backend:
          serviceName: cluster-status
          servicePort: 80
```

Service Discovery

Name	Configuration	Client type	Summary
ClusterIP	`type: ClusterIP` `.spec.selector`	Internal	The most common internal discovery mechanism
Manual IP	`type: ClusterIP` `kind: Endpoints`	Internal	External IP discovery
Manual FQDN	`type: ExternalName` `.spec.externalName`	Internal	External FQDN discovery
Headless Service	`type: ClusterIP` `.spec.clusterIP: None`	Internal	DNS-based discovery without a virtual IP
NodePort	`type: NodePort`	External	Preferred for non-HTTP traffic
LoadBalancer	`type: LoadBalancer`	External	Requires supporting cloud infrastructure
Ingress	`kind: Ingress`	External	L7/HTTP-based smart routing mechanism


Advanced
Image Builder
Controller
Operator
Elastic Scale

Controller

How to get from the current state to a declared target state

State Reconciliation

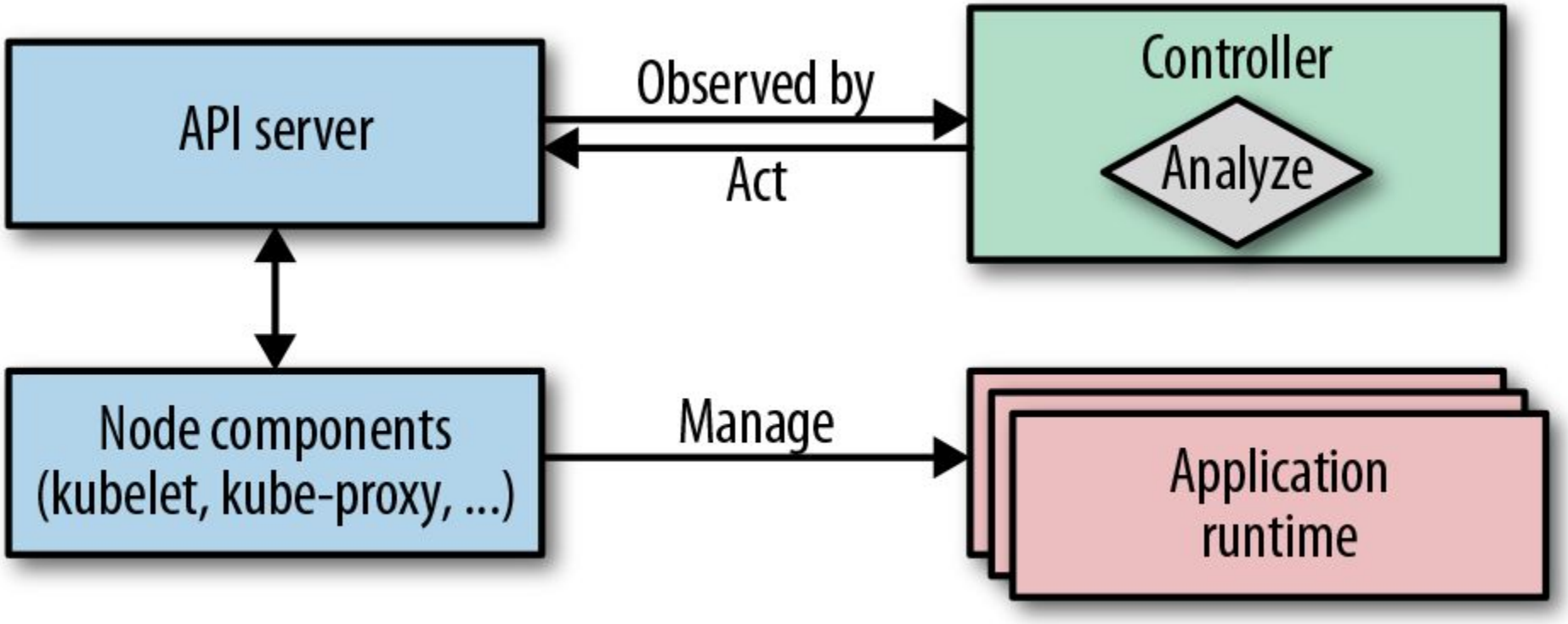
- Kubernetes as distributed state manager
- Make the **actual** state more like the declared **target** state.

🔎 ***Observe*** - Discover the actual state

🤔 ***Analyze*** - Determine difference to target state

🔨 ***Act-*** Perform actions to drive the actual to the desired state

Observe - Analyze - Act



Common Triggers

- Labels
  - Indexed by backend
  - Suitable for selector-like functionality
  - Limitation on characterset for names and values
- Annotations
  - No syntax restrictions
  - Not indexed
- ConfigMaps
  - Good for complex structured state declarations
  - Simple alternative to CustomResourceDefinitions

ConfigMap Watch Controller

```
namespace=${WATCH_NAMESPACE:-default}
base=http://localhost:8001
ns=namespaces/$namespace

curl -N -s $base/api/v1/${ns}/configmaps?watch=true | \
while read -r event
do
  type=$(echo "$event" | jq -r '.type')
  config_map=$(echo "$event" | jq -r '.object.metadata.name')
  annotations=$(echo "$event" | jq -r '.object.metadata.annotations')

  if [ $type = "MODIFIED" ]; then
    # Restart Pods using this ConfigMap
    # ...
  fi
done
```


Operator

How to encapsulate operational knowledge into executable software

Definition

> An **operator** is a Kubernetes **controller** that understands two domains: Kubernetes and *something else*. By combining knowledge of both areas, it can **automate** tasks that usually require a human operator that understands both domains.
>
> **Jimmy Zelinskie**
> http://bit.ly/2Fjlx1h

Technical:

Operator = Controller + CustomResourceDefinition

CustomResourceDefinition

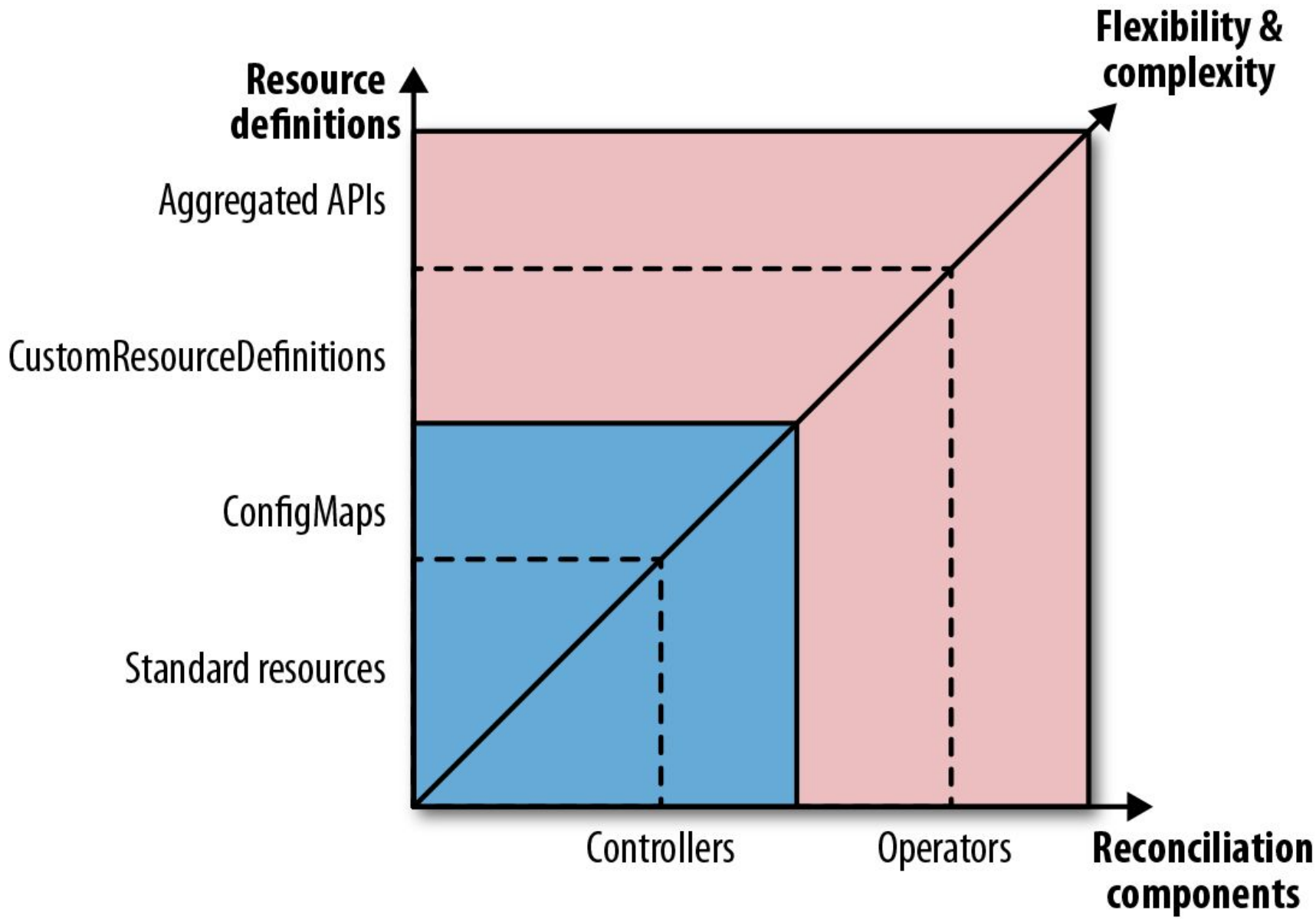
Custom resource is modelling a custom domain and managed through the Kubernetes API

```
apiVersion: apiextensions.k8s.io/v1beta1
kind: CustomResourceDefinition
metadata:
  name: configwatchers.k8spatterns.io
spec:
  scope: Namespaced
  group: k8spatterns.io
  version: v1
  names:
    kind: ConfigWatcher
    plural: configwatchers
  validation:
    openAPIV3Schema:
    ...
```

Custom Resource

Operator

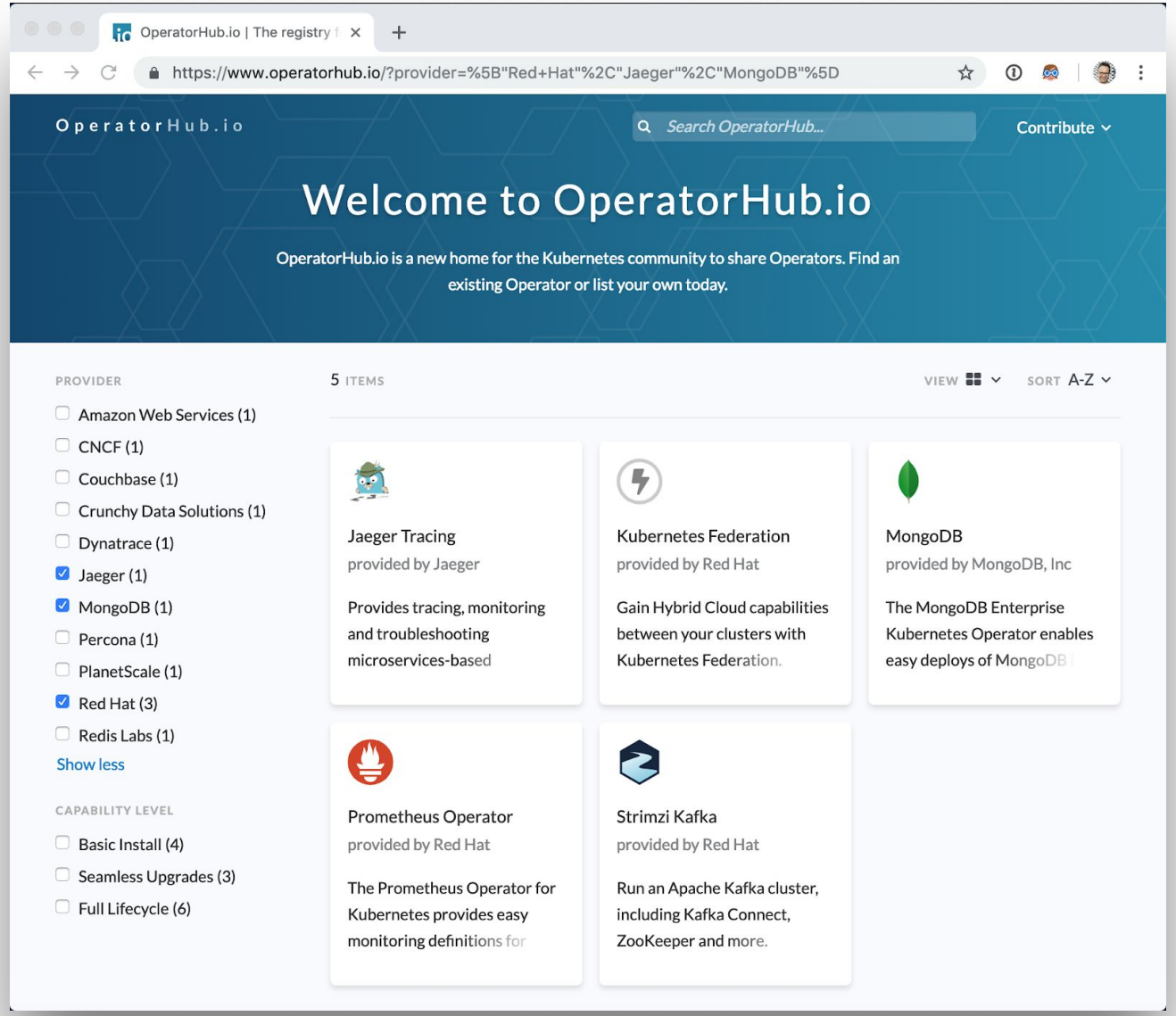
```
kind: ConfigWatcher
apiVersion: k8spatterns.io/v1
metadata:
  name: webapp-config-watcher
spec:
  configMap: webapp-config
  podSelector:
    app: webapp
```


Flexibility & complexity
Resource definitions
Aggregated APIs
CustomResourceDefinitions
ConfigMaps
Standard resources
Controllers
Operators
Reconciliation components

CRD Classification

- Installation CRDs
  - Installing and operating applications
  - Backup and Restore
  - Monitoring and self-healing
  - Example: Prometheus for installing Prometheus & components
- Application CRDs
  - Application specific domain concepts
  - Example: ServiceMonitor for registering Kubernetes service to be scraped by Prometheus

Operator Hub



Red Hat

Operator Development

- Operator can be implemented in any language
- Frameworks:
  - Operator Framework (Golang, Helm, Ansible) https://github.com/operator-framework
  - Kubebuilder (Golang) https://github.com/kubernetes-sigs/kubebuilder
  - Metacontroller (Language agnostic) https://metacontroller.app/
  - jvm-operators (Java, Groovy, Kotlin, ...) https://github.com/jvm-operators

Demo

Examples

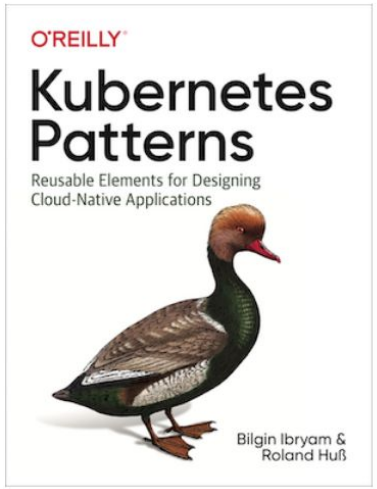
README.adoc

Kubernetes Patterns - Examples

This GitHub project contains the examples from *Kubernetes Patterns - Reusable Elements for Designing Cloud-Native Applications* book by Bilgin Ibryam and Roland Huß

Installation instructions for the example prerequisites are summarised in INSTALL. By default, you need access to a vanilla Kubernetes installation, like Minikube. If addons are required, the example description explains this additional requirement.

For feedback, issues or questions in general, please use the issue tracker to open issues. Also, we love contributions like spelling fixes, bug fixes, improvements, ... Please open Pull Requests, we are happy to review them!



Patterns

All example are categorised according to the Book's patterns category. Each of the examples is contained in an extra directory per pattern and is self-contained. [1]

Foundational Patterns

Predictable Demands

Our sample random generator dealing with hard requirements on ConfigMap and PersistentVolumeClaims as well as with resource limits.

Declarative Deployment

Rolling and fixed update of the random generator Deployment from version 1.0 to 2.0.

Health Probe

Liveness and Readiness probes for the random generator.

https://github.com/k8spatterns/examples

Red Hat

Thank you

https://k8spatterns.io

@ro14nd

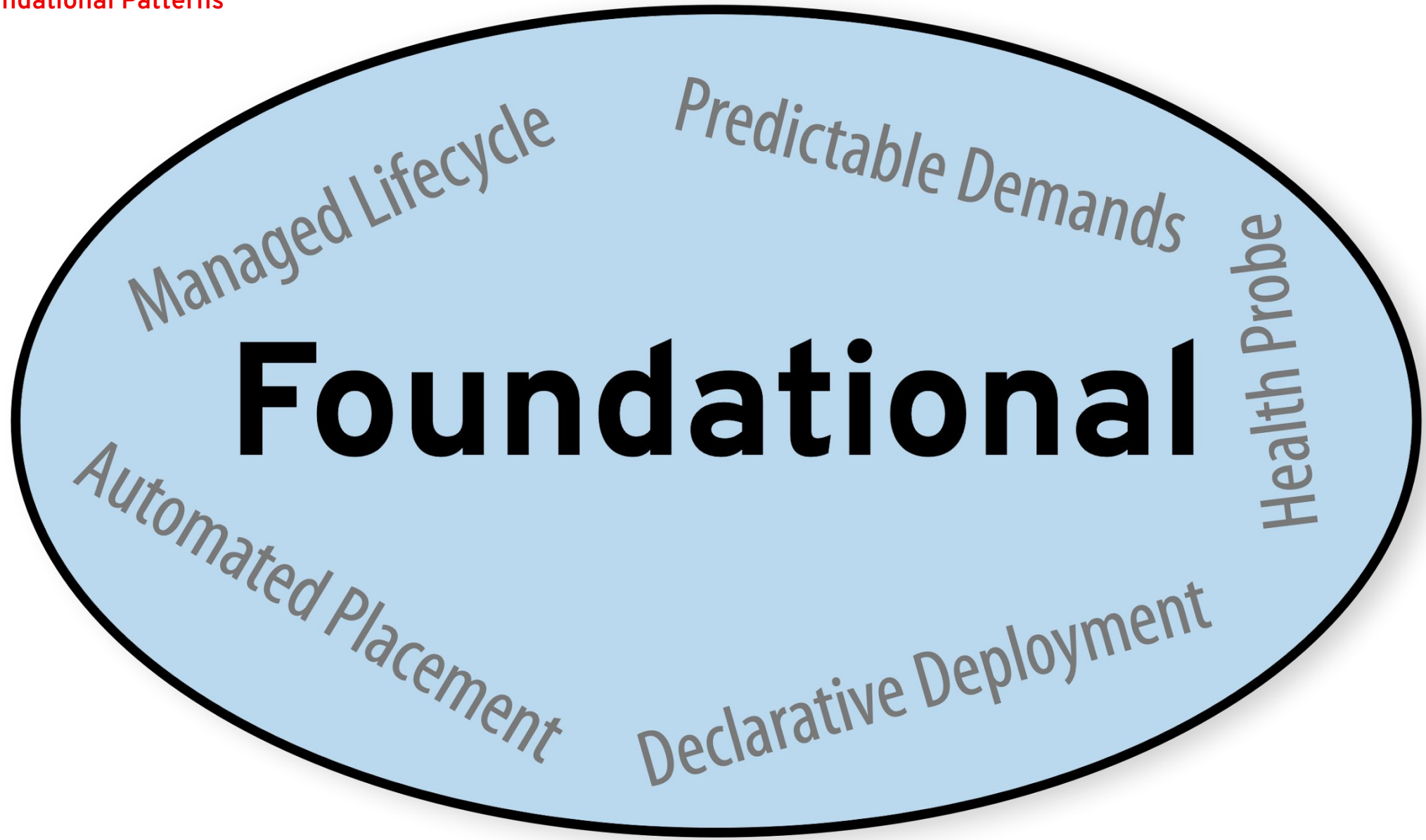
@bibryam

@k8spatterns

Red Hat

Picture Credits

https://www.pexels.com/photo/brown-and-black-pattern-2158386/
https://pixabay.com/photos/ship-helm-sunset-cutter-coast-guard-759954/
https://unsplash.com/photos/yo01Z-9HQAw
https://www.pexels.com/photo/turned-on-light-crane-1117211/
https://www.pexels.com/photo/shallow-focus-photography-of-black-ship-1095814/
https://pixabay.com/photos/containers-storage-rusted-rusty-1209079/
https://pixabay.com/photos/motocross-sidecar-race-motorsport-1045661/
https://www.pexels.com/photo/golden-gate-bridge-san-francisco-california-1141853/
https://unsplash.com/photos/M_I-crkDO-k
https://www.pexels.com/photo/reflection-playstation-pad-gaming-18174/
https://unsplash.com/photos/UJP_QpCKj7M
https://www.pexels.com/photo/grayscale-photo-of-person-holding-chess-piece-1498958/
https://pixabay.com/photos/cans-manufacturing-business-2888650/
https://unsplash.com/photos/IRoX0shwjUQ
https://www.freeimages.com/photo/poppy-in-wheat-1344010
https://pixabay.com/photos/telescope-field-glass-spyglass-1966366/
https://unsplash.com/photos/UHDx3BHIFvY
https://pixabay.com/photos/bake-advent-christmas-cookie-1786926/
https://pixabay.com/photos/balloons-colors-party-celebration-1869790/
https://unsplash.com/photos/ymf4_9Y9S_A
https://unsplash.com/photos/ZUabNmumOcA
https://pixabay.com/photos/lost-places-machines-old-factory-3991951/


Managed Lifecycle
Predictable Demands
Health Probe
Foundational
Automated Placement
Declarative Deployment


Predictable
Demands

How to declare application requirements

Application Requirements

- Declared requirements help in
  - Scheduling decisions
  - Capacity planning
  - Matching infrastructure services
- Hard runtime dependencies
  - Persistent Volumes
  - Host ports
  - Dependencies on ConfigMaps and Secrets

Resource Profile

```
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 Classes

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

Exercise


Health
Probe

How to communicate an application's health state to Kubernetes

Monitoring Health

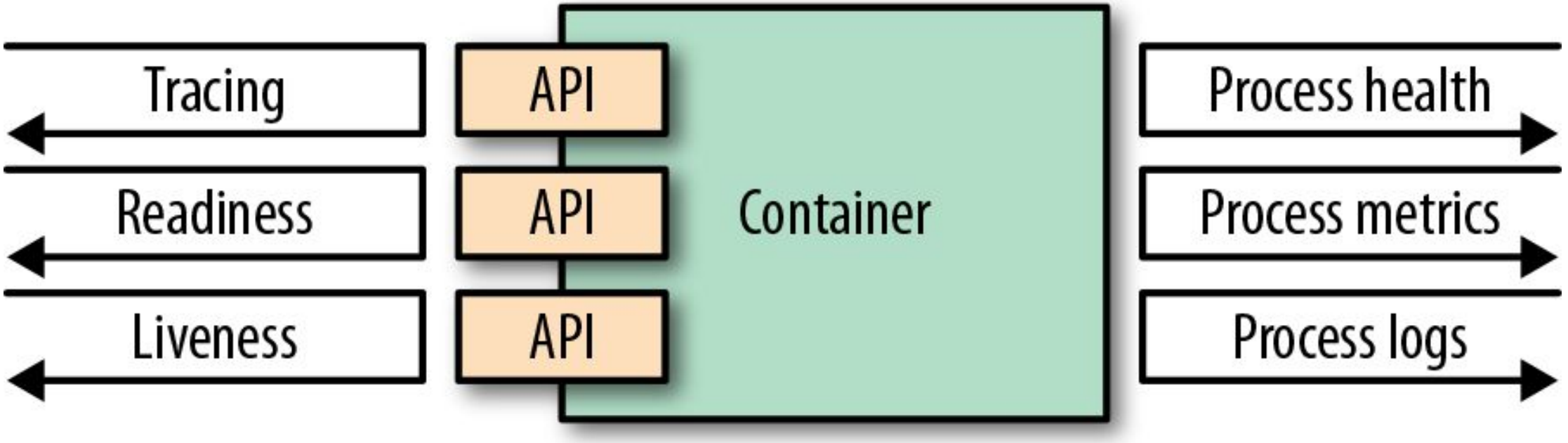
- Process health checks
  - Checks running process
  - Restarts container if container process died
- Application provided Health Probes
  - **Liveness Probe**: Check application health
  - **Readiness Probe**: Check readiness to process requests

Liveness & Readiness

- Liveness Probe
  - Restarting containers if liveness probes fail
- Readiness Probe
  - Removing from service endpoint if readiness probe fails
- Probe methods
  - HTTP endpoint
  - TCP socket endpoint
  - Unix command's return value

```
apiVersion: v1
kind: Pod
metadata:
  name: pod-with-readiness-check
spec:
  containers:
  - image: k8spatterns/random-generator:1.0
    name: random-generator
    livenessProbe:
      httpGet:
        path: /actuator/health
        port: 8080
      initialDelaySeconds: 30
    readinessProbe:
      exec:
        command: [ "stat", "/var/run/random-generator-ready" ]
```

Container Observability Options



Exercise

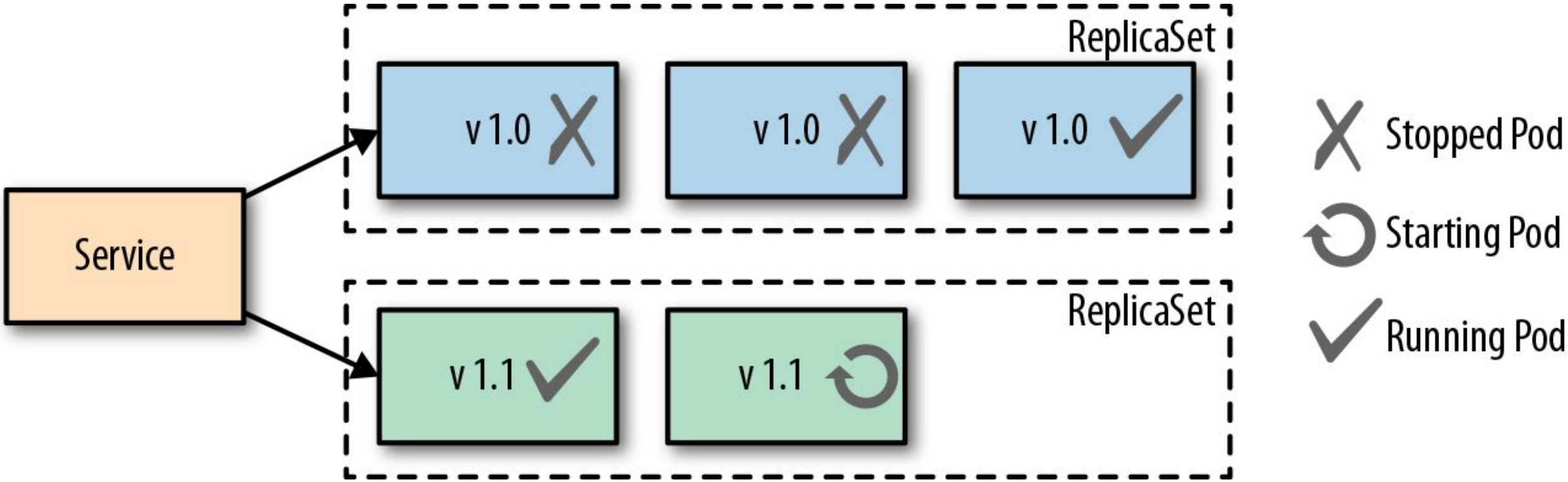
Declarative Deployment

How to perform application installations and upgrades by configuration.

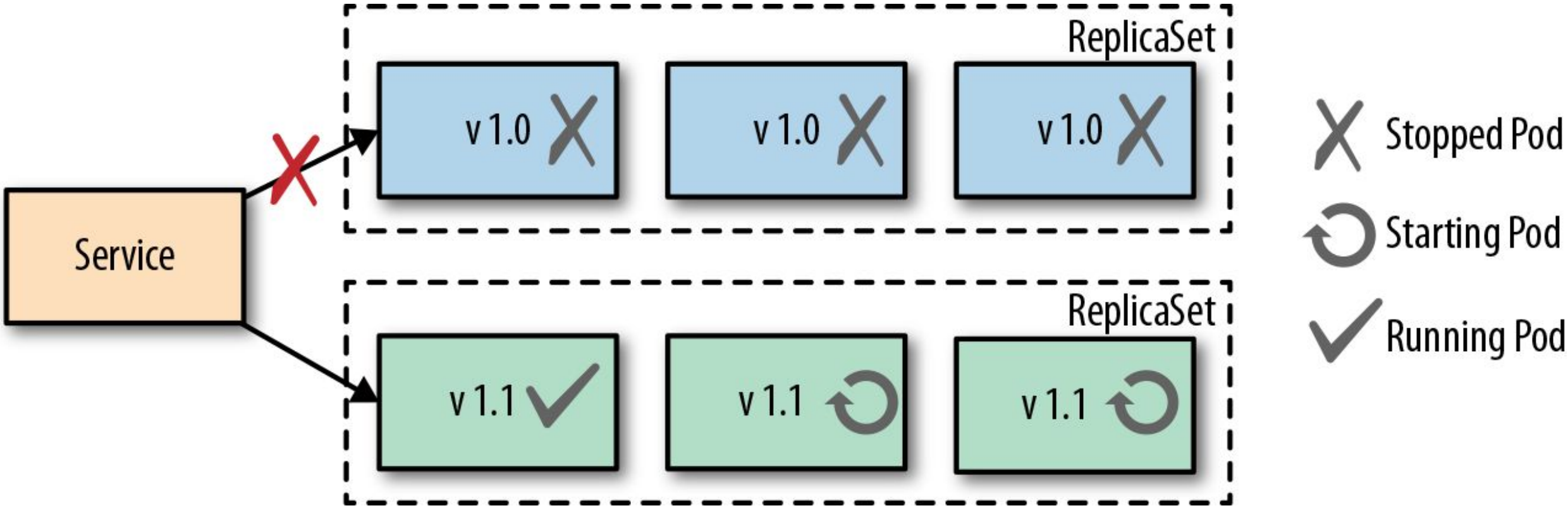
Deployment

- **Declarative** vs. **Imperative** deployment
- Deployment Kubernetes Resource:
  - Holds template for Pod
  - Creates ReplicaSet on the fly
  - Allows rollback
  - Update strategies are declarable

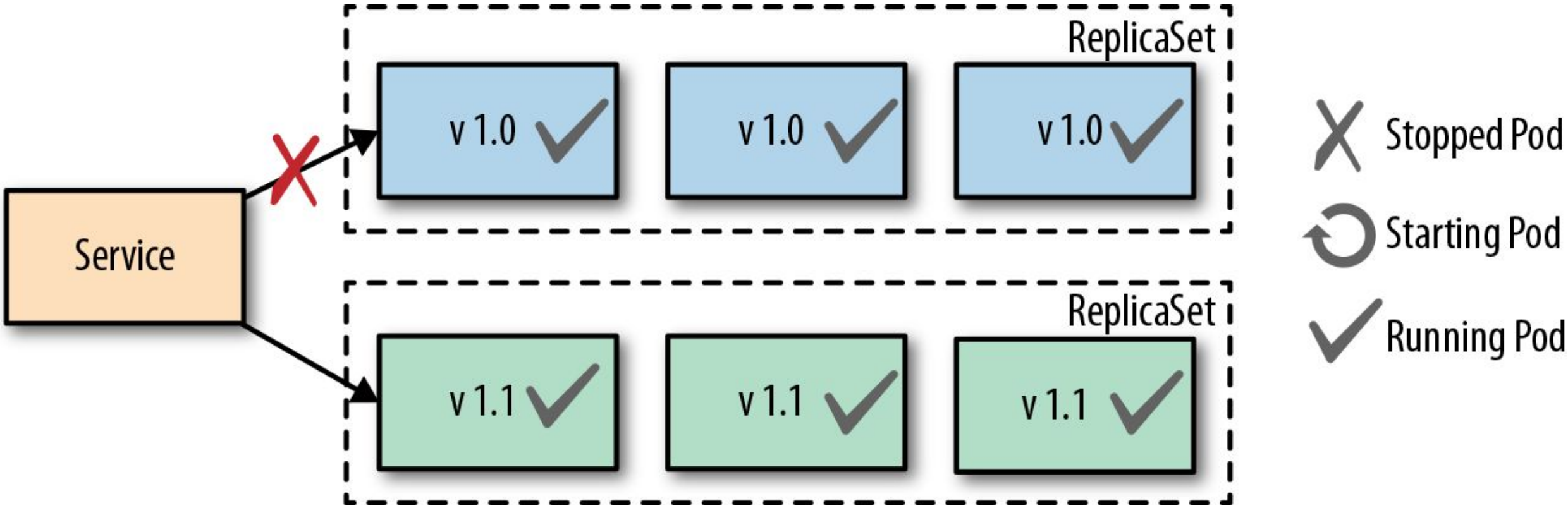
Rolling Deployment



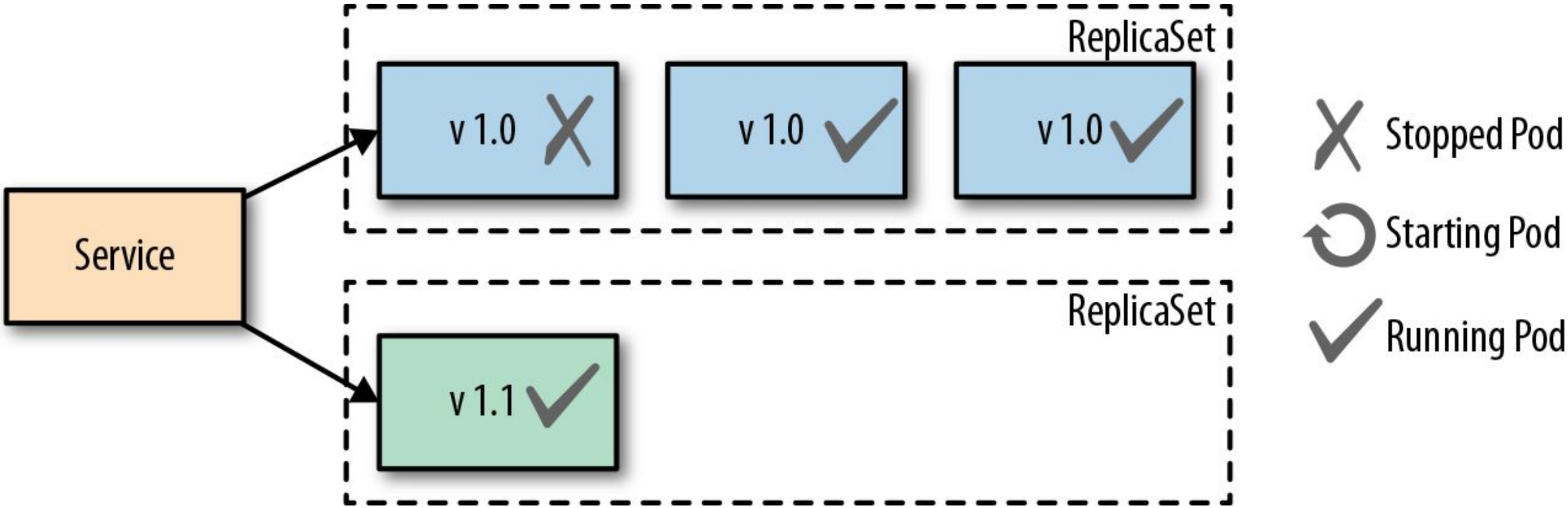
Fixed Deployment

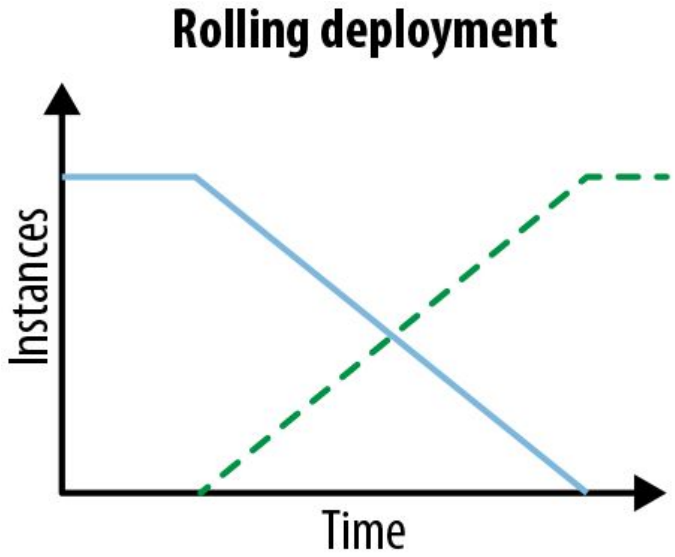


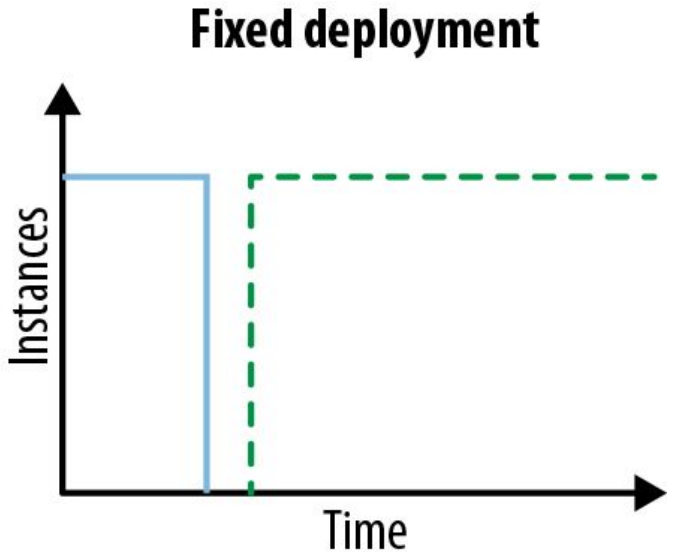
Blue-Green Release

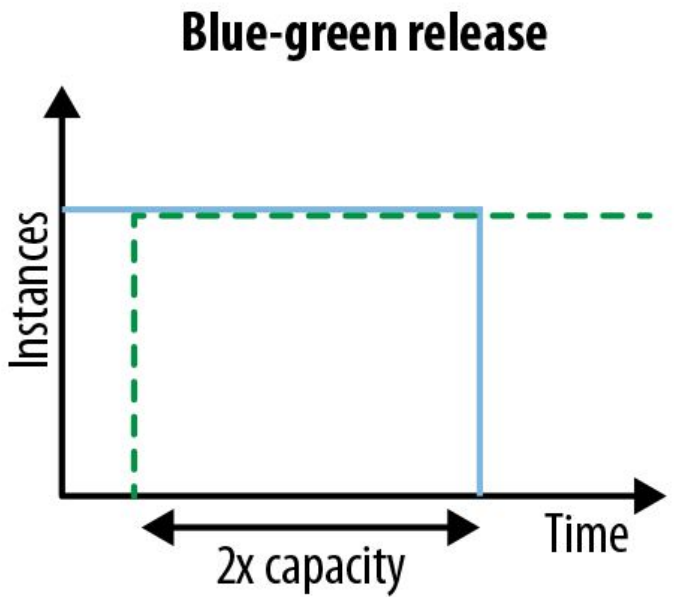


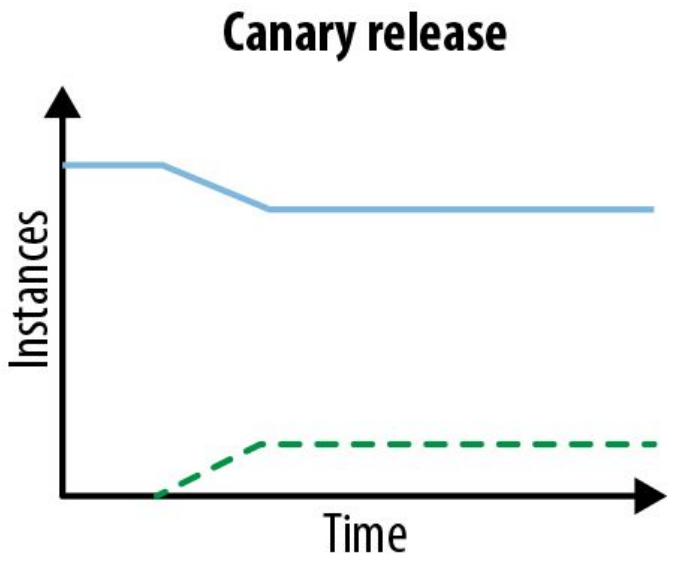
Canary Release




Rolling deployment
Instances
Time


Fixed deployment
Instances
Time


Blue-green release
Instances
Time
2x capacity

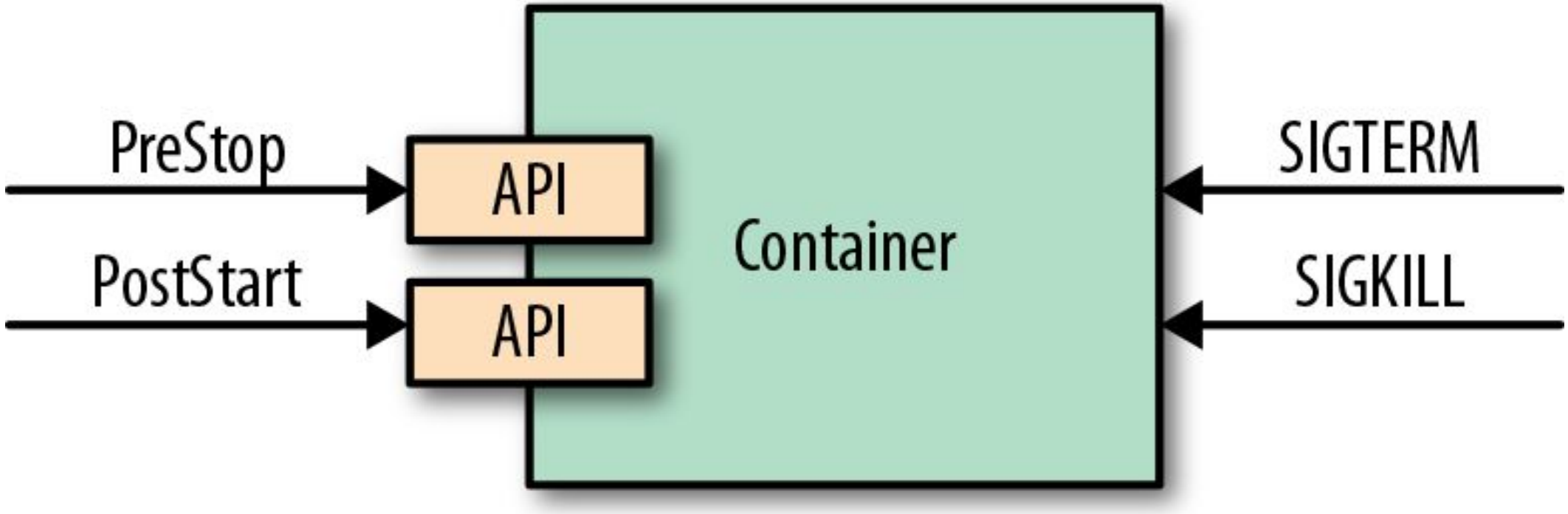

Canary release
Instances
Time

Exercise


Managed
Lifecycle

How applications should react on lifecycle events

Managed Container Lifecycle



Lifecycle Events

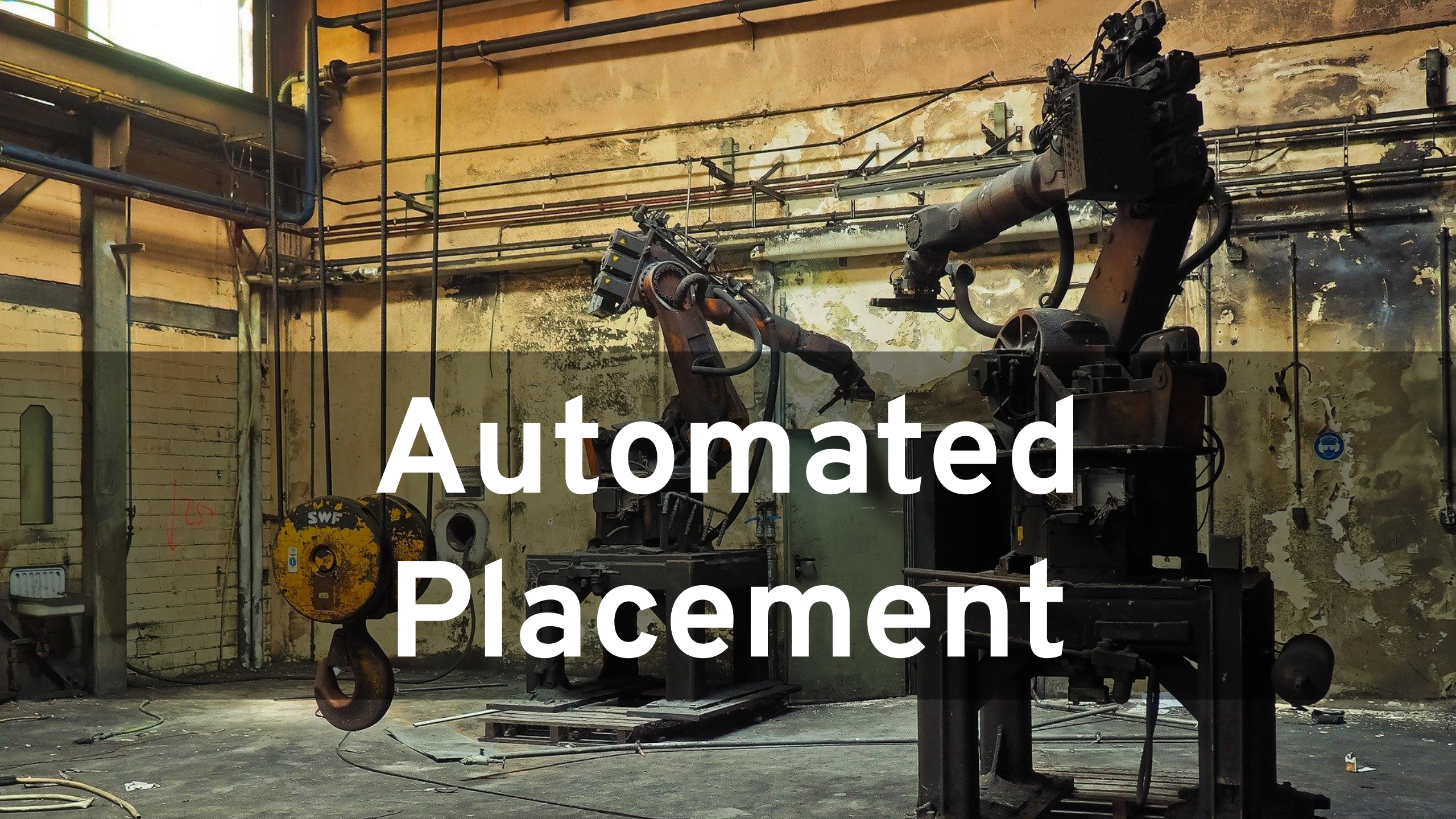
- SIGTERM
  - Initial event issued when a container is going to shutdown
  - Application should listen to this event to cleanup properly and then exit
- SIGKILL
  - Final signal sent after a grace period which can't be catched
  - `terminationGracePeriodSeconds`: Time to wait after SIGTERM (default: 30s)

Lifecycle Hooks

- `postStart`
  - Called after container is created
  - Runs in parallel to the main container
  - Keeps Pod in status *Pending* until exited successfully
  - **exec** or **httpGet** handler types (like *Health Probe*)
- preStop
  - Called before container is stopped
  - Same purpose & semantics as for SIGTERM

```
apiVersion: v1
kind: Pod
metadata:
  name: pre-stop-hook
spec:
  containers:
  - image: k8spatterns/random-generator:1.0
    name: random-generator
    lifecycle:
      postStart:
        exec:
          command:
          - sh
          - -c
          - sleep 30 && echo "Wake up!" > /tmp/postStart_done
      preStop:
        httpGet:
          port: 8080
          path: /shutdown
```

Exercise



Automated Placement

How Kubernetes schedules containers

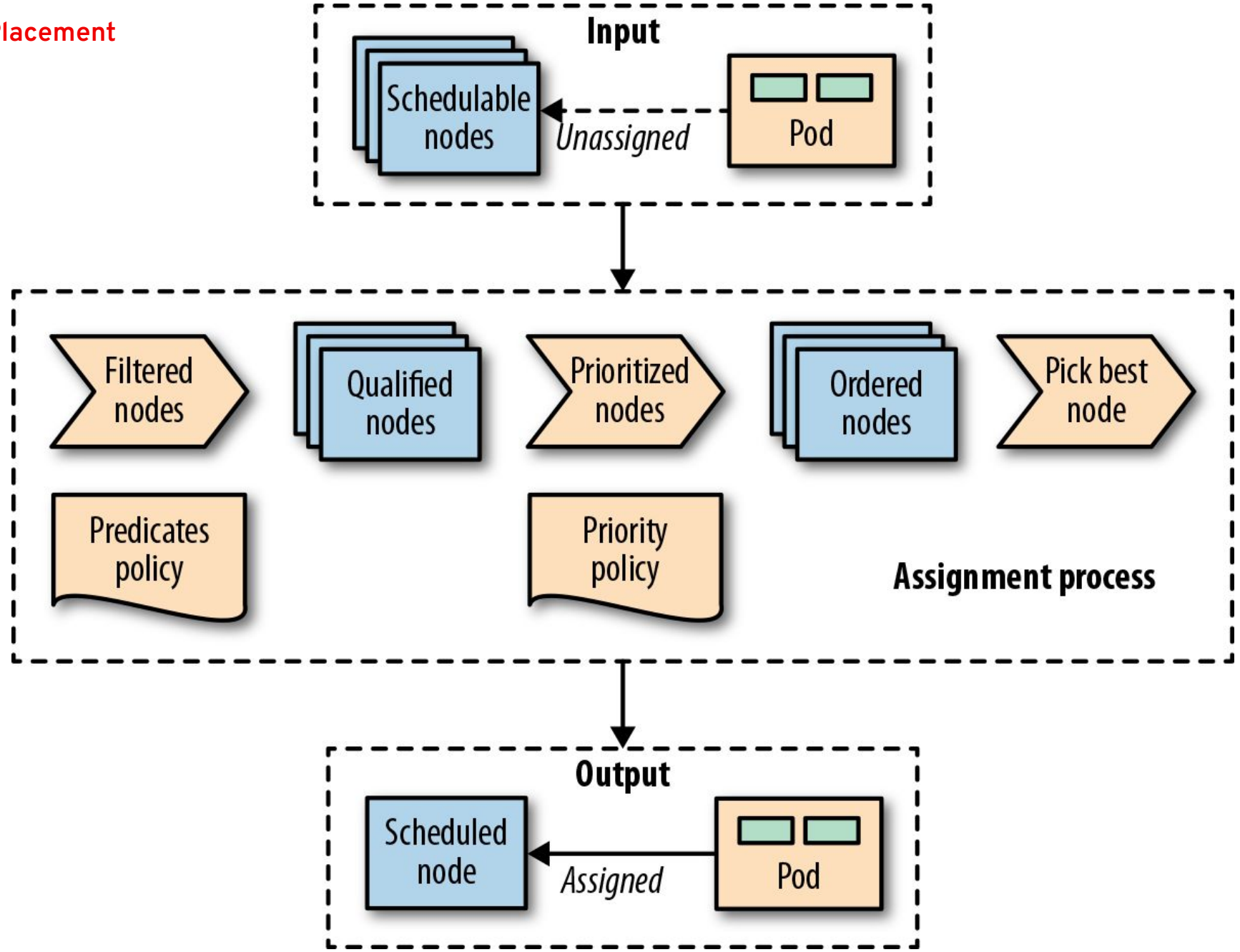
Scheduler

- Responsible for assigning Pods to nodes
- **Predicates**: Filter of suitable nodes
- **Priorities**: Ordering of nodes according to preference
- Predicates + Priorities = Scheduler Policy
- Multiple schedulers with different policies can be defined
- Pod can overwrite with `.spec.schedulerName`

Scheduler Policy

```
{
  "kind": "Policy",
  "apiVersion": "v1",
  "predicates": [
    {
      "name": "PodFitsHostPorts"
    },
    {
      "name": "PodFitsResources"
    },
    ...
  ],
  "priorities": [
    {
      "name": "LeastRequestedPriority",
      "weight": 2
    },
   ..,
  ]
}
```

Automated Placement

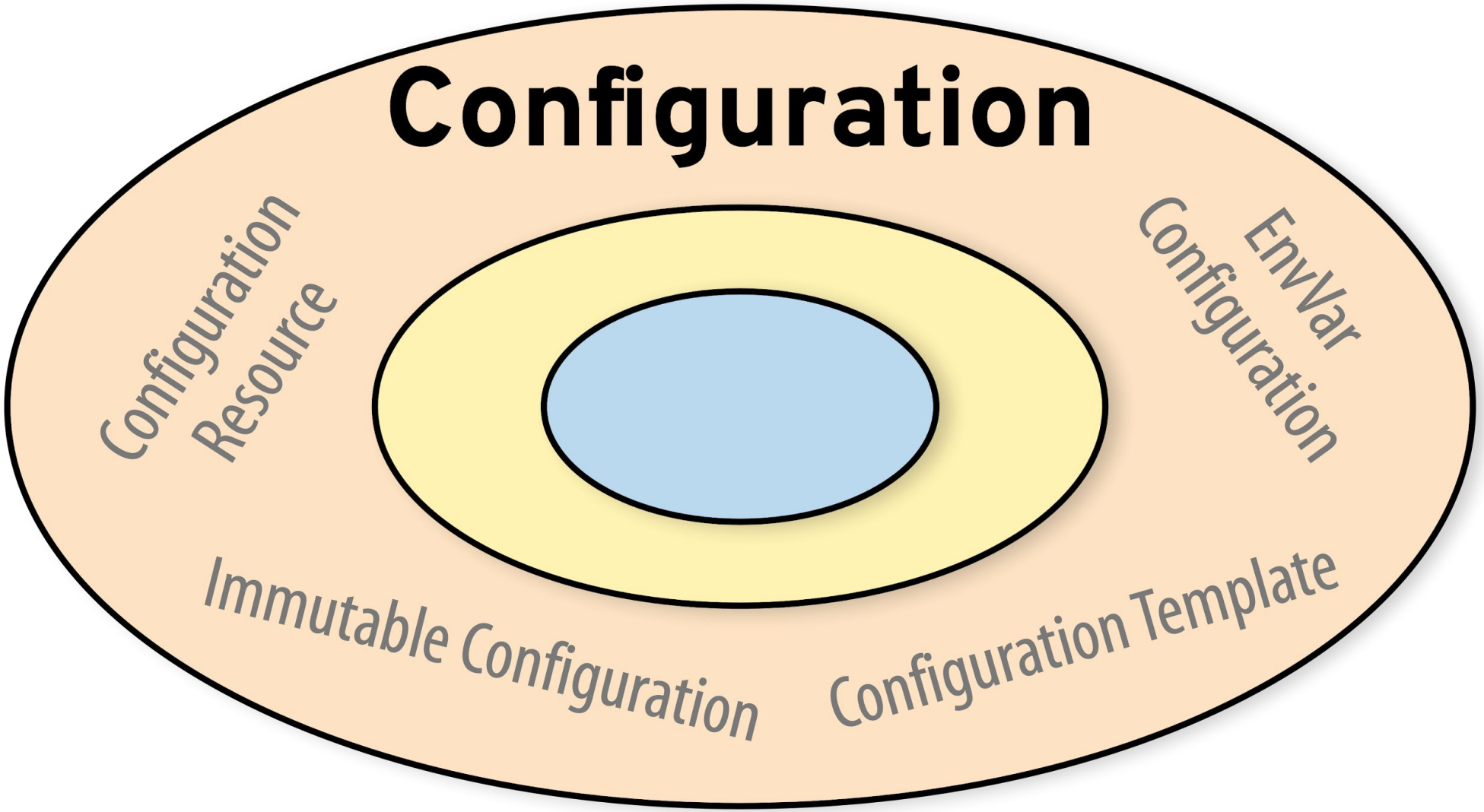


Red Hat

How to influence the Scheduler

- General rule: Let the schedule do its job
- **Node Selector**: Label selector to pick a node
- **Node Affinity**: Required and preferred scheduling rules on nodes (label and field matching)
- **Pod Affinity** and **Antiaffinity**: Specify rules how pods are placed with respect to other Pods
- **Taints** and **Tolerations**: Lets node specify which pods might be scheduled on them (opposite of node selector)

Exercise


Configuration
Configuration Resource
EnvVar Configuration
Immutable Configuration
Configuration Template


EnvVar
Configuration

How to use environment variables to configure your applications

Environment Variables

- Simplest way to externalize configuration for containerized applications
- 12factor.net: Rule 3 - “Store config in the environment”
- When specified in a Pod:
  - Literally in the container spec
  - Imported from a ConfigMap or Secret
- Limitations
  - Immutable for a running process
  - Not suitable for a large configuration set

Exercise

Configuration Resource

How to manage configuration with Kubernetes resources

ConfigMap

- Key-Value Map
- Use in Pods as:
  - environment variables
  - volumes with keys as file names and values as file content

```
kubectl create cm spring-boot-config \
   --from-literal=JAVA_OPTIONS=-Djava.security.egd=file:/dev/urandom \
   --from-file=application.properties
```

Secret

- Like ConfigMap but content Base64 encoded
- Secrets are …
  - … only distributed to nodes running Pods that need it
  - … only stored in memory in a `tmpfs` and never written to physical storage
  - … stored encrypted in the backend store (etcd)
- Access can be restricted with RBAC rules
- But: For high security requirements application based encryption is needed

Exercise

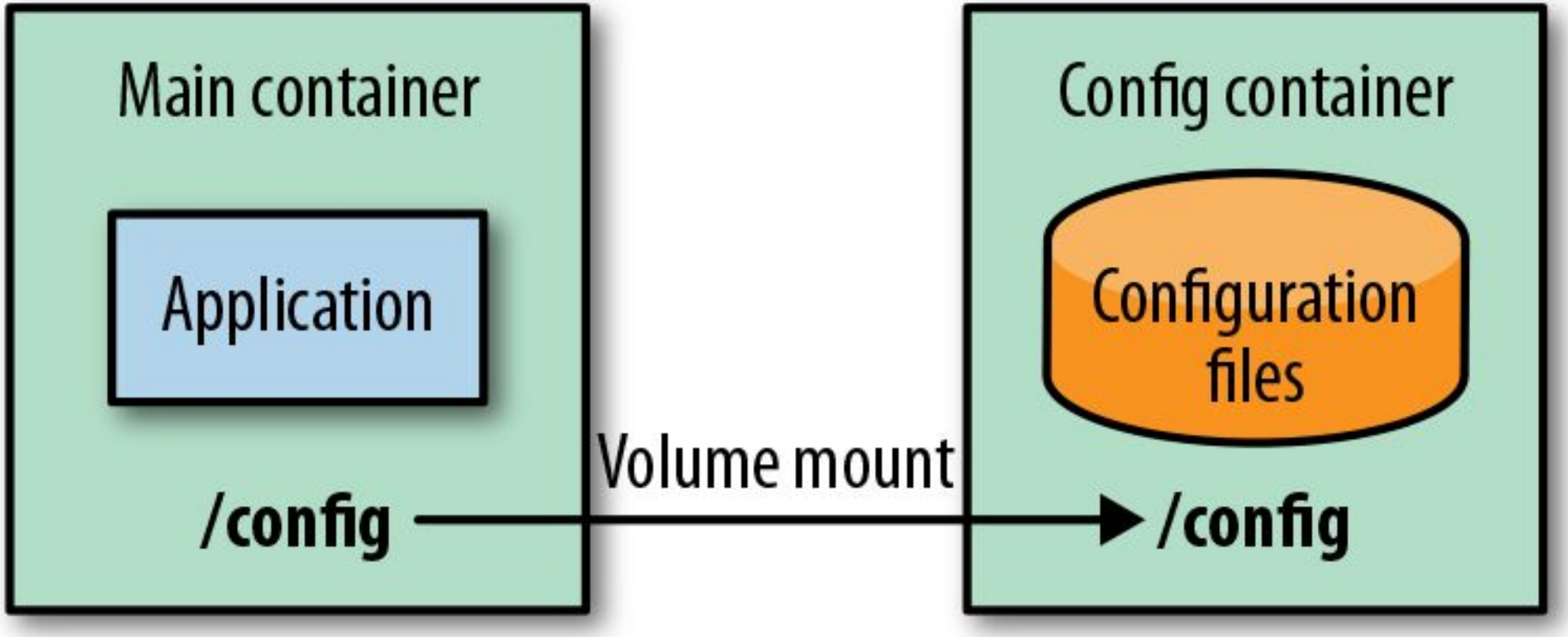

Immutable
Configuration

How to configure your application with immutable container images

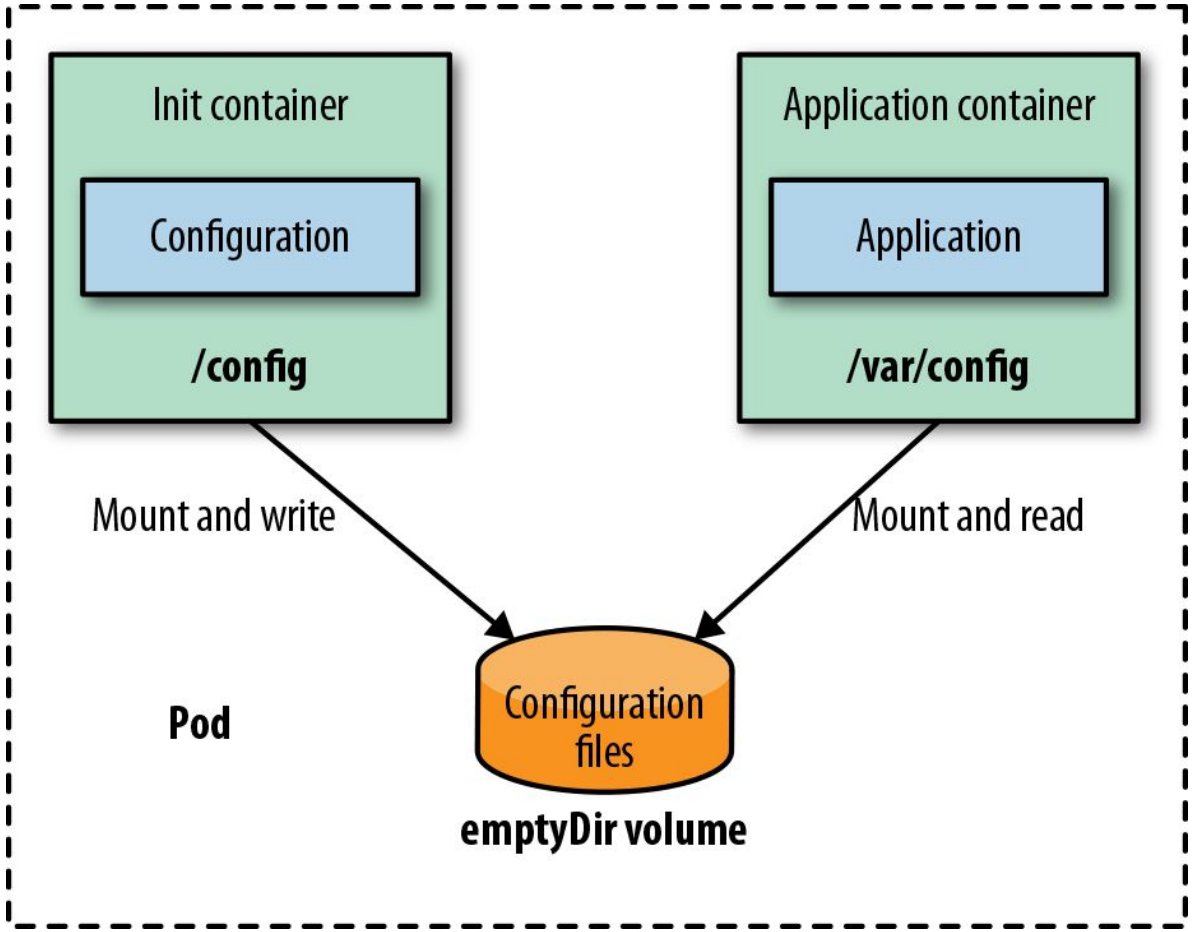
Immutable Configuration

- Configuration that can not be changed
- Every configuration change results in a new *Immutable Configuration* set
- Different sets of configuration for different environments (dev/prod)
- Revisioning and auditing for configuration
- Solution: Configuration stored in container images
  - Versioned via tags and digests
  - Distributed via image registry

Docker volume mount

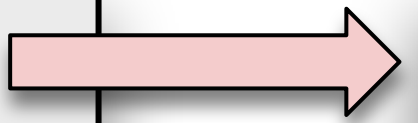


Configuration provided by Init Container



Stateful Service

```
initContainers:
- image: k8spatterns/config-dev:1
  name: init
  args:
  - "/config"
  volumeMounts:
  - mountPath: "/config"
    name: config-directory
containers:
- image: k8spatterns/demo:1
  name: demo
  ports:
  - containerPort: 8080
name: http
    protocol: TCP
  volumeMounts:
  - mountPath: "/config"
    name: config-directory
volumes:
  - name: config-directory
    emptyDir: {}
```



```
FROM busybox
ADD dev.properties /config-src/demo.properties
ENTRYPOINT \
   [ "sh", "-c", "cp /config-src/* $1", "--" ]
```

Init Containers to the rescue

- Kubernetes: No container image volume mounts
- Instead: Copy over configuration from image to a shared volume in an *Init Container*
- Exchanging the configuration image for different environments is challenging
- Templating of Deployments to the rescue
  - OpenShift Templates, Helm Charts, Operator with CRD …
- Good for large to huge configuration data (like a Wildfly server configuration)

OpenShift Template

```
apiVersion: v1
kind: Template
metadata:
  name: demo
parameters:
  - name: CONFIG_IMAGE
    description: Name of configuration image
    value: k8spatterns/config-dev:1
objects:
- apiVersion: v1
  kind: DeploymentConfig
    // ....
    spec:
      template:
          metadata:
            // ....
            spec:
              initContainers:
              - name: init
                image: ${CONFIG_IMAGE}
                ...
```

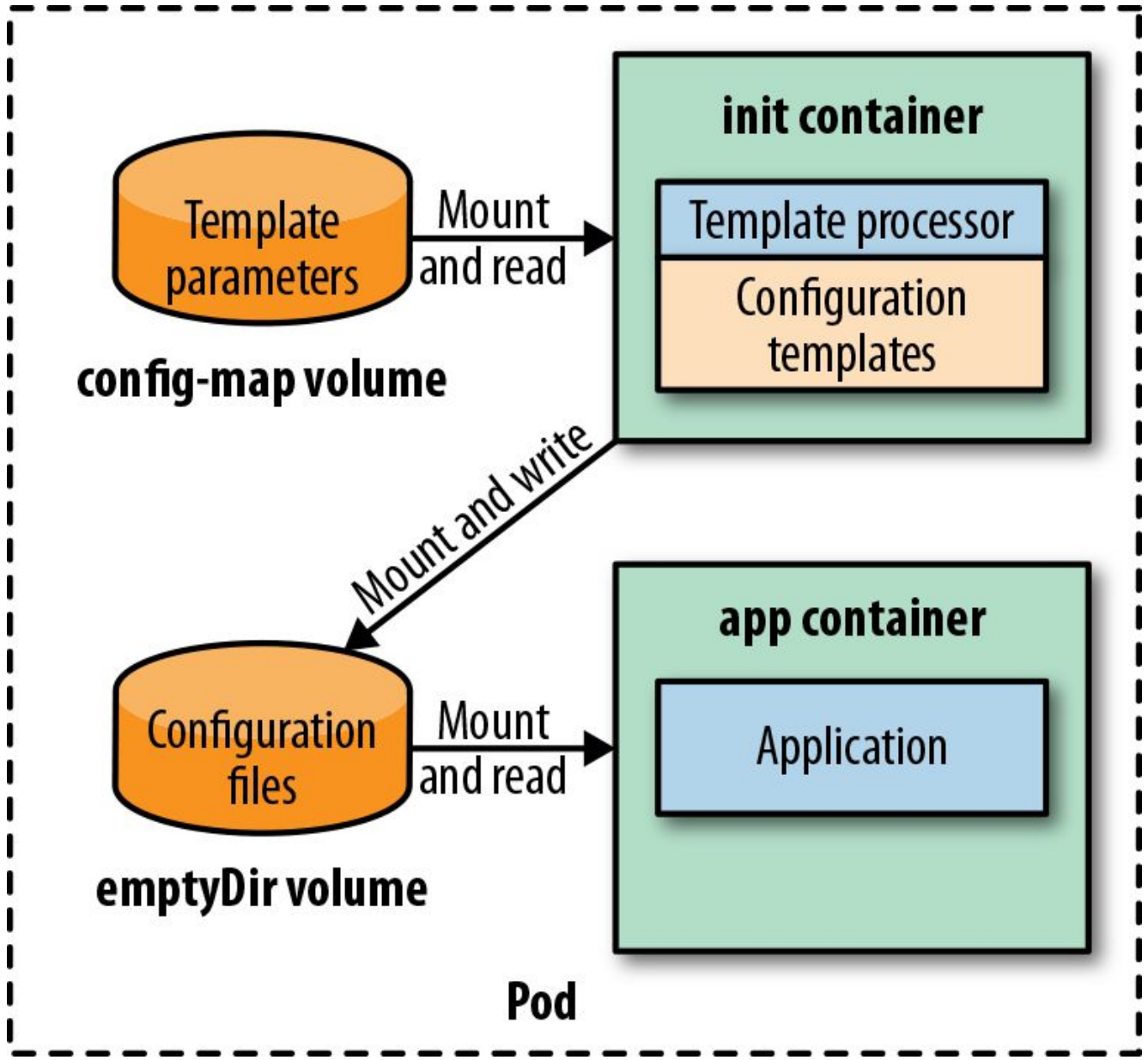
Exercise

Configuration Template

How to manage large and complex similar configuration data

Preparing Configuration during Startup

- Init Container ...
  - ... contains a template processor
  - ... holds the configuration template
  - ... picks up template parameter from a ConfigMap
  - ... stores final configuration on a shared volume
- Main Container ....
  - ... accesses created configuration from shared volume


init container
Template processor
Configuration templates
Template parameters
Mount and read
config-map volume
Mount and write
app container
Application
Configuration files
Mount and read
emptyDir volume
Pod

Configuration Template

```
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: wildfly-cm-template
spec:
  replicas: 1
  template:
    spec:
      initContainers:
      - image: k8spatterns/config-init
        name: init
        volumeMounts:
        - mountPath: "/params"
          name: wildfly-parameters
        - mountPath: "/out"
          name: wildfly-config
```

```
containers:
- image: jboss/wildfly:10.1.0.Final
  name: server
  volumeMounts:
  - mountPath: "/config"
    name: wildfly-config
volumes:
- name: wildfly-parameters
  configMap:
    name: wildfly-params-cm
- name: wildfly-config
  emptyDir: {}
```

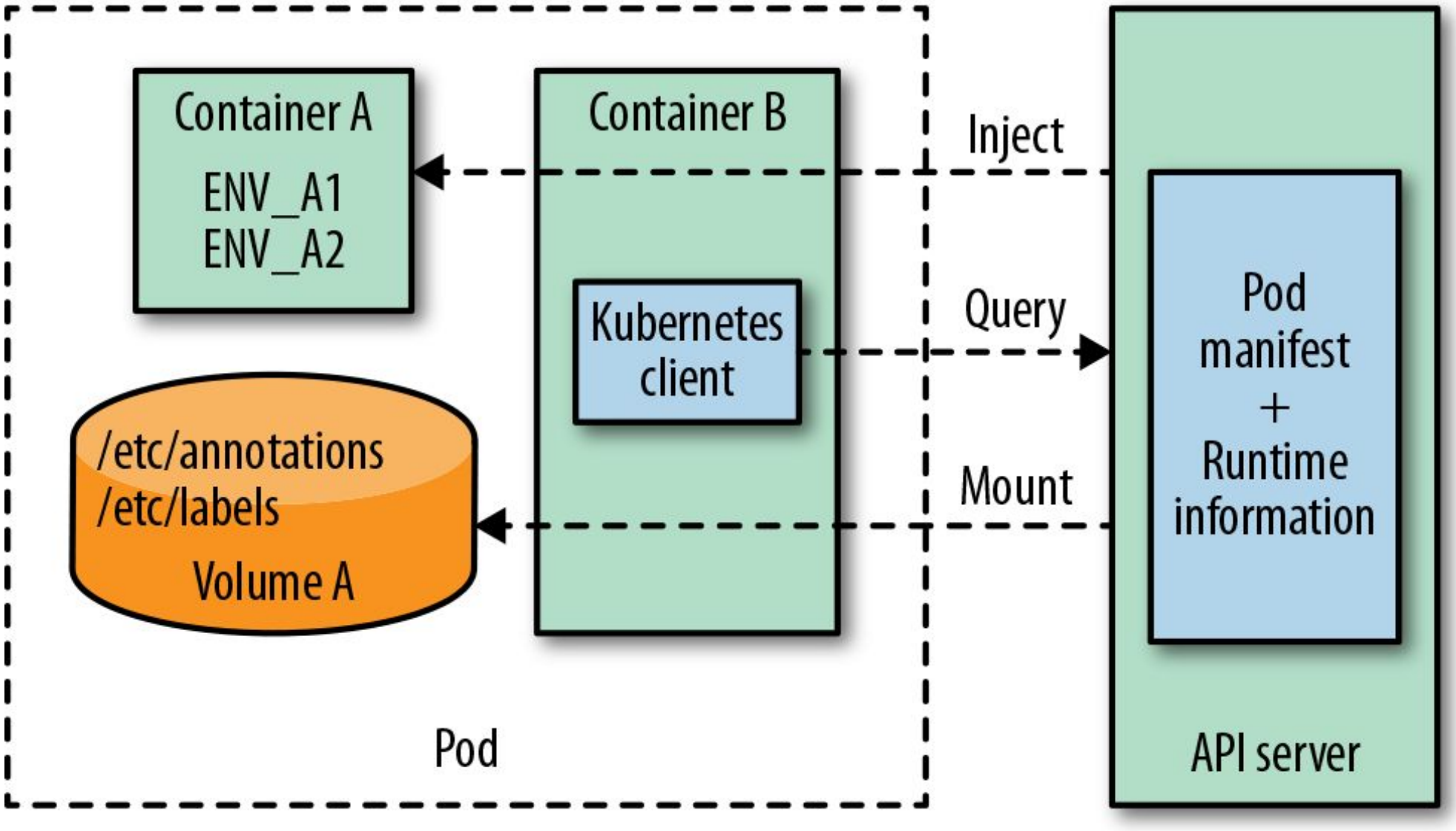
Exercise


Self
Awareness

How an application can access cluster context information

Downward API

- Passing Pod metadata to containers
  - … via environment variables
  - … via files
- “Injection” instead of active lookup (Inversion-of-control)


Container A
ENV_A1
ENV_A2
Container B
Kubernetes client
/etc/annotations
/etc/labels
Volume A
Pod
Inject
Query
Mount
Pod manifest + Runtime information
API server

```
apiVersion: v1
kind: Pod
metadata:
  name: random-generator
spec:
  containers:
  - image: k8spatterns/random-generator:1.0
    name: random-generator
    env:
    - name: POD_IP
      valueFrom:
        fieldRef:
          fieldPath: status.podIP
    - name: MEMORY_LIMIT
      valueFrom:
        resourceFieldRef:
          container: random-generator
          resource: limits.memory
```

```
apiVersion: v1
kind: Pod
metadata:
  name: random-generator
spec:
  containers:
  - image: k8spatterns/random-generator:1.0
    name: random-generator
    volumeMounts:
    - name: pod-info
      mountPath: /pod-info
  volumes:
  - name: pod-info
    downwardAPI:
      items:
      - path: labels
        fieldRef:
          fieldPath: metadata.labels
```

fieldRef.fieldPath

Name	Description
`spec.nodeName`	Name of node hosting the Pod
`status.hostIP`	IP address of node hosting the Pod
`metadata.name`	Pod name
`metadata.namespace`	Namespace in which the Pod is running
`status.podIP`	Pod IP address
`spec.serviceAccountName`	ServiceAccount that is used for the Pod
`metadata.uid`	Unique ID of the Pod
`metadata.labels['`*key*`']`	Value of the Pod's label *key*
`metadata.annotations['`*key*`']`	Value of the Pod's annotation *key*

resourceFieldRef.resource

Name	Description
`requests.cpu`	A container's CPU request
`limits.cpu`	A container's CPU limit
`limits.memory`	A container's memory request
`requests.memory`	A container's memory limit

Exercise


Batch
Job

How to run short-lived Pods reliably until completion

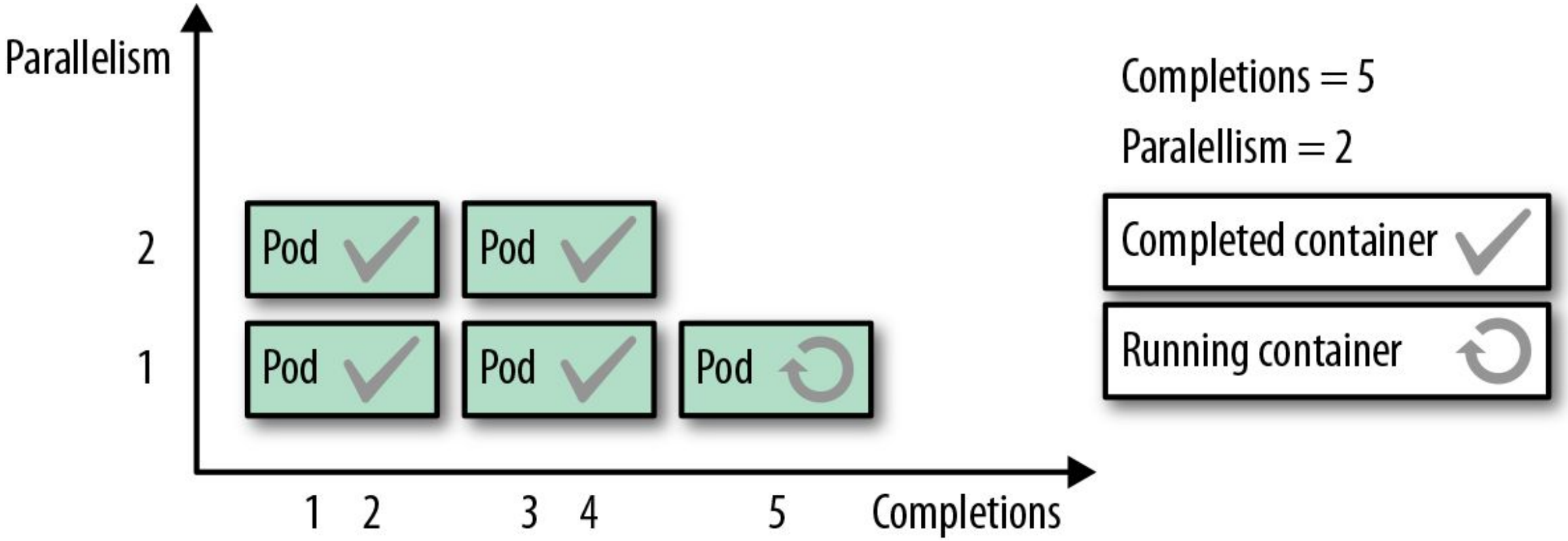
Job

- Resource for a predefined *finite* unit-of-work
- Survives cluster restarts and node failures
- Support for multiple Pod runs
- `.spec.completions`
  - How many Pods to run to complete Job
- `.spec.parallelism`
  - How many Pods to run in parallel

Job

```
apiVersion: batch/v1
kind: Job
metadata:
  name: random-generator
spec:
  completions: 5
  parallelism: 2
  template:
    metadata:
      name: random-generator
    spec:
      restartPolicy: OnFailure
      containers:
      - image: k8spatterns/random-generator:1.0
        name: random-generator
        command: [ "java", "-cp", "/", "RandomRunner", "/numbers.txt", "10000" ]
```

Parallel Batch Job



Job Types

- Single Pod Job
  - completions = 1 and parallelism = 1
- Fixed completion count Jobs
  - completions > 1
  - One Pod per work item
- Work queue Jobs
  - completions = 1 and parallelism > 1
  - All Pods need to finish, with at least one Pod exciting successfully
  - Pods needs to coordinate to shutdown in a coordinate fashion


Periodic
Job

How to run a ***Batch Job*** triggered by a temporal event

CronJob

- Executing a *Batch Job* periodically
- Fields:
  - **schedule** - Cron expression to specify when the Job should run
  - **template** - Job specification
  - **startingDeadlineSeconds** - Deadline for starting jobs that missed the schedule time
  - **concurrencyPolicy** - how to react if previous *Batch Job* is still running
  - **suspend** - set to true to pause a CronJob
  - **successfulJobsHistoryLimit** and **failedJobsHistoryLimit** - how many jobs to keep

CronJob

```
apiVersion: batch/v1beta1
kind: CronJob
metadata:
  name: random-generator
spec:
  # Every three minutes
  schedule: "*/3 * * * *"
  jobTemplate:
    spec:
      template:
        spec:
          containers:
          - image: k8spatterns/random-generator:1.0
            name: random-generator
            command: [ "java", "-cp", "/", "RandomRunner", "/numbers.txt", "10000" ]
          restartPolicy: OnFailure
```


Daemon
Service

How to run a Pod replica on every node

DaemonSet

- Enables to run one Pod on every cluster node
- Good for infrastrucure level processes
  - Log collectors, Metrics exporters, ...
- **nodeSelector** for running only on a subset of nodes
- Not managed by the Kubernetes scheduler
- Treated specially by controllers like the descheduler or cluster-auto scaler
- Alternative: Static Pods started by kubelet from a directory

DaemonSet

```
apiVersion: extensions/v1beta1
kind: DaemonSet
metadata:
  name: random-refresher
spec:
  selector:
    matchLabels:
      app: random-refresher
  template:
    metadata:
      labels:
        app: random-refresher
    spec:
      nodeSelector:
        feature: hw-rng
      containers:
      - image: k8spatterns/random-generator:1.0
        volumeMounts:
        - mountPath: /host_dev
          name: devices
      volumes:
      - name: devices
        hostPath:
          path: /dev
```

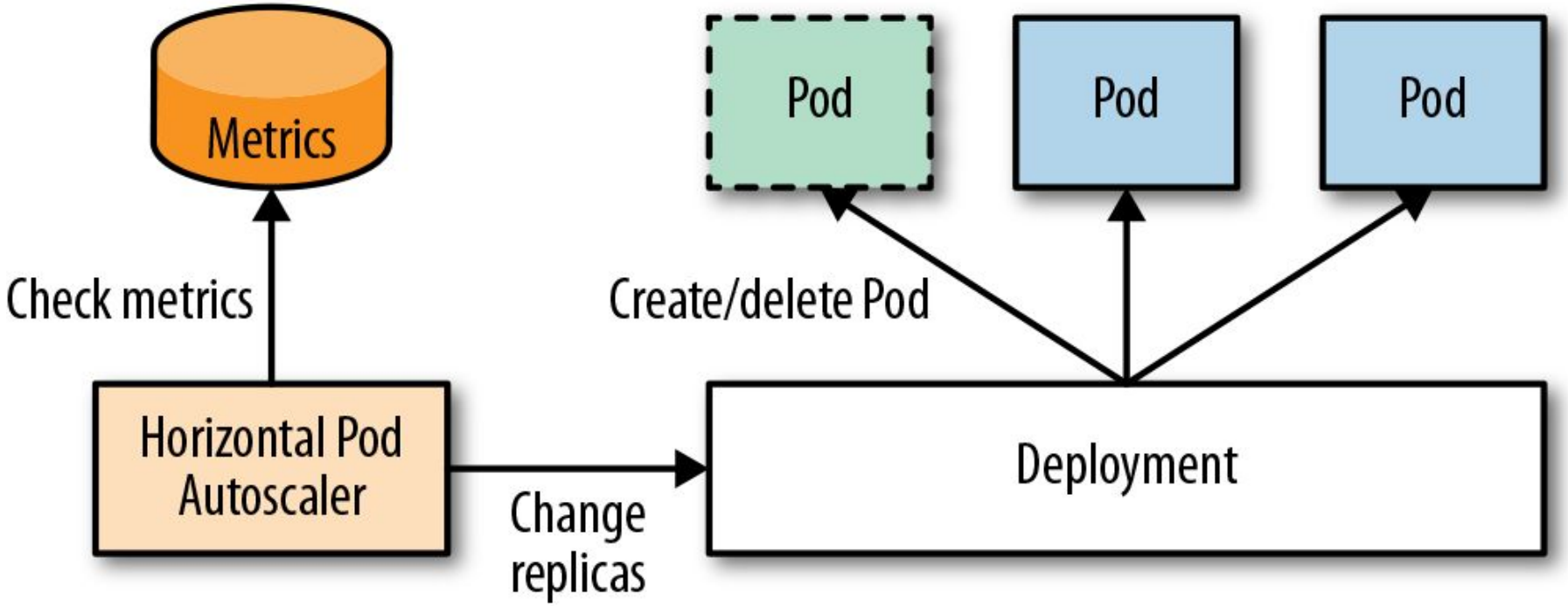

Elastic
Scale

How to automatically react to dynamically changing resource requirements

Scaling

- **Horizontal**: Changing replicas of a Pod
- **Vertical:** Changing resource constraints of containers in a single Pod
- **Cluster**: Adding new nodes to a cluster
- **Manual**: Changing scale parameters manually, imperatively or declaratively
- **Automatic**: Change scaling parameters based on observed metrics

Horizontal Pod Autoscaler (HPA)

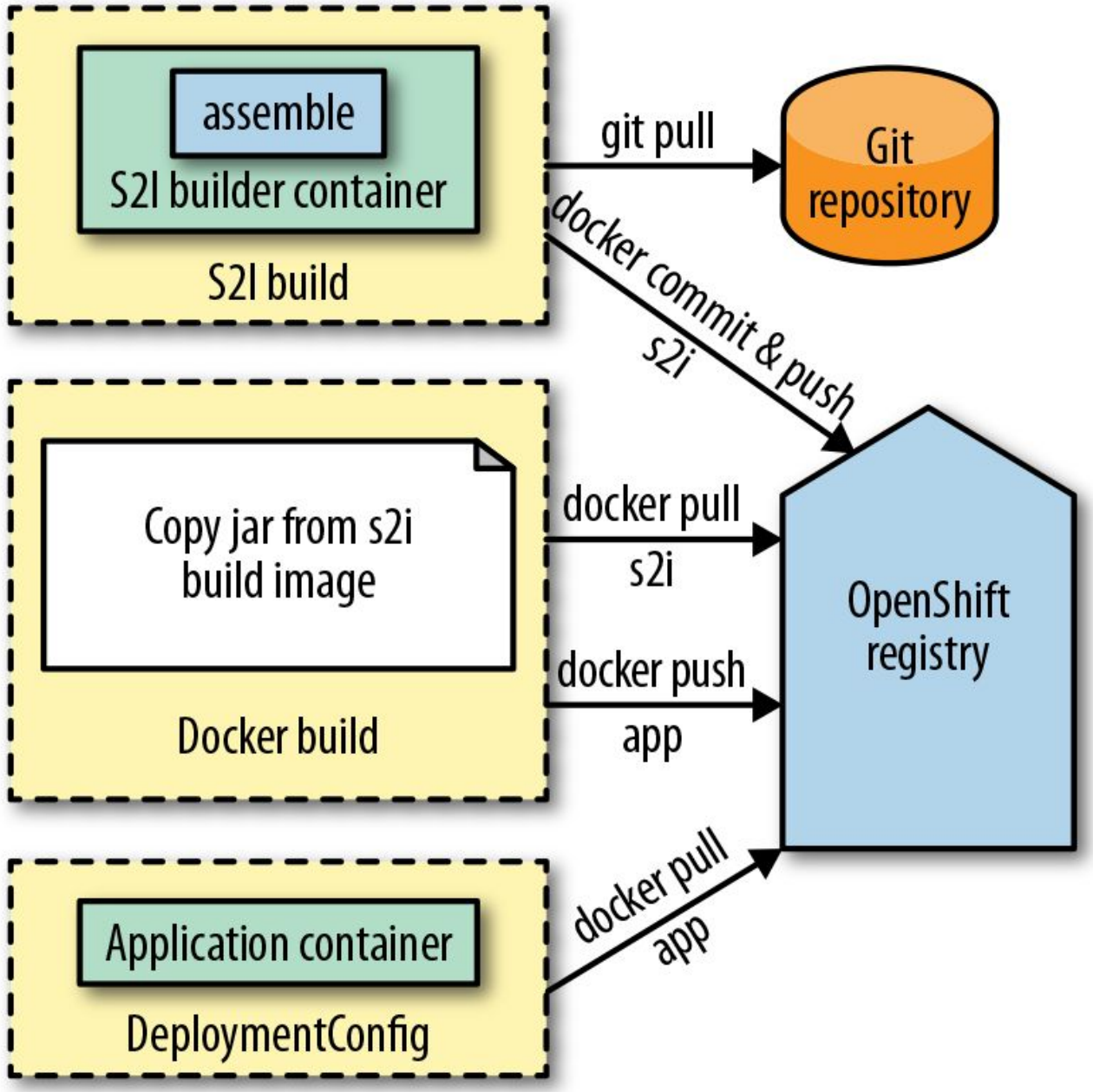



```
kubectl autoscale deployment random-generator --cpu-percent=50 --min=1 --max=5
```

BuildConfig

```
apiVersion: v1
kind: BuildConfig
metadata:
  name: random-generator-build
spec:
  source:
    git:
      uri: https://github.com/k8spatterns/random-generator
  strategy:
    sourceStrategy:
      from:
        kind: DockerImage
        name: fabric8/s2i-java
  output:
    to:
      kind: ImageStreamTag
      name: random-generator-build:latest
  triggers:
  - type: ImageChange
```

S2I Chained Build



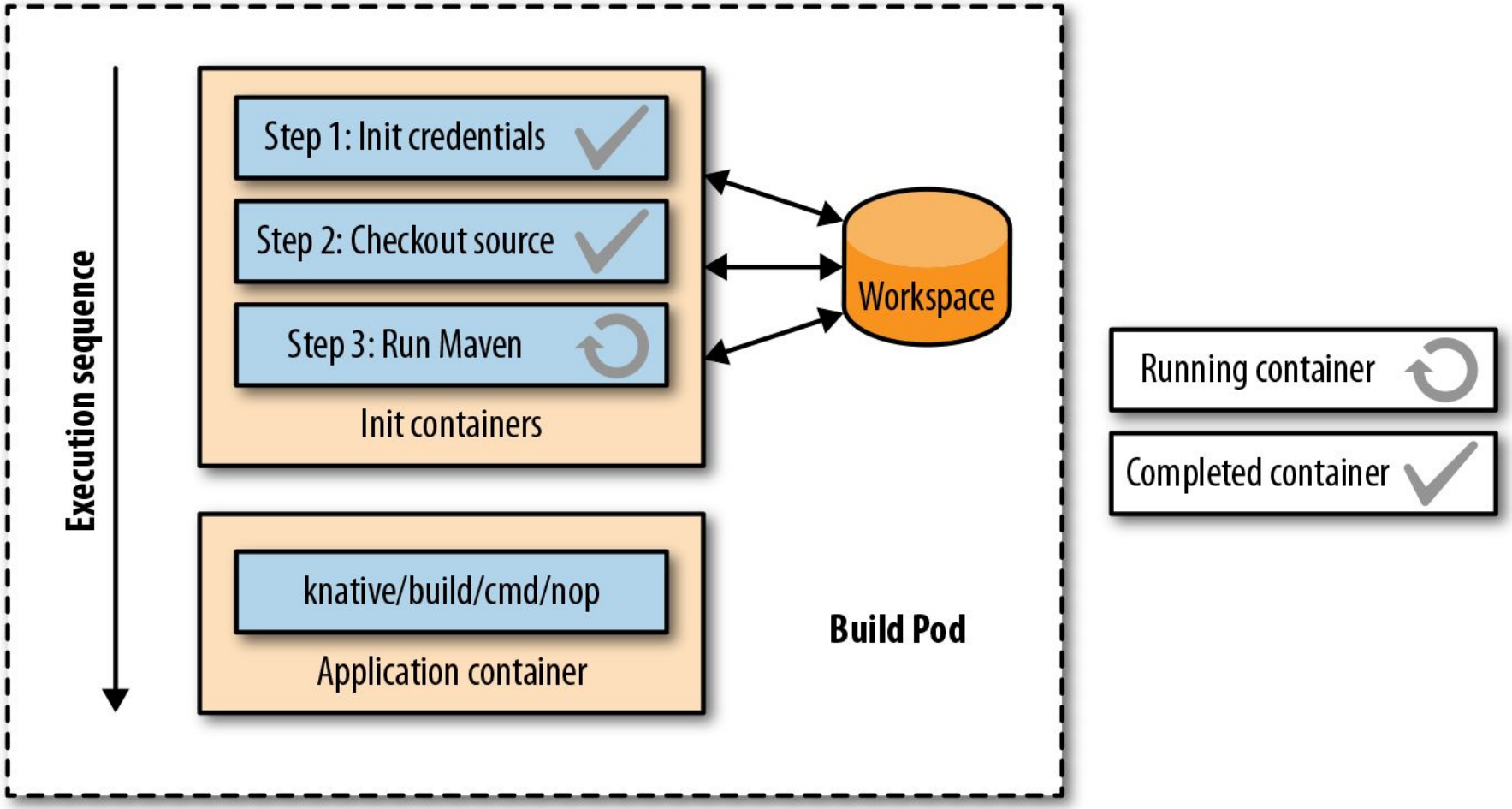
Knative

- **Knative serving**
  - Scale-to-Zero
- **Knative eventing**
  - Event infrastructure for triggering services
- **Knative build**
  - Transforming source to container image
- Build templates allows reusing build strategies

Build

```
apiVersion: build.knative.dev/v1alpha1
kind: Build
metadata:
  name: random-generator-build-jib
spec:
  source:
    git:
      url: https://github.com/k8spatterns/random-generator.git
      revision: master
  steps:
  - name: build-and-push
    image: gcr.io/cloud-builders/mvn
    args:
    - compile
    - com.google.cloud.tools:jib-maven-plugin:build
    - -Djib.to.image=registry/k8spatterns/random-generator
    workingDir: /workspace
```

Knative Build



Exercise