

Kubernetes Patterns

Reusable Elements for Designing
Cloud-Native Applications

Robert Sedor
Senior Cloud Platform Architect

Agenda



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

Dad Jokes

Q: Why was the developer unhappy at their job?

A: They wanted arrays

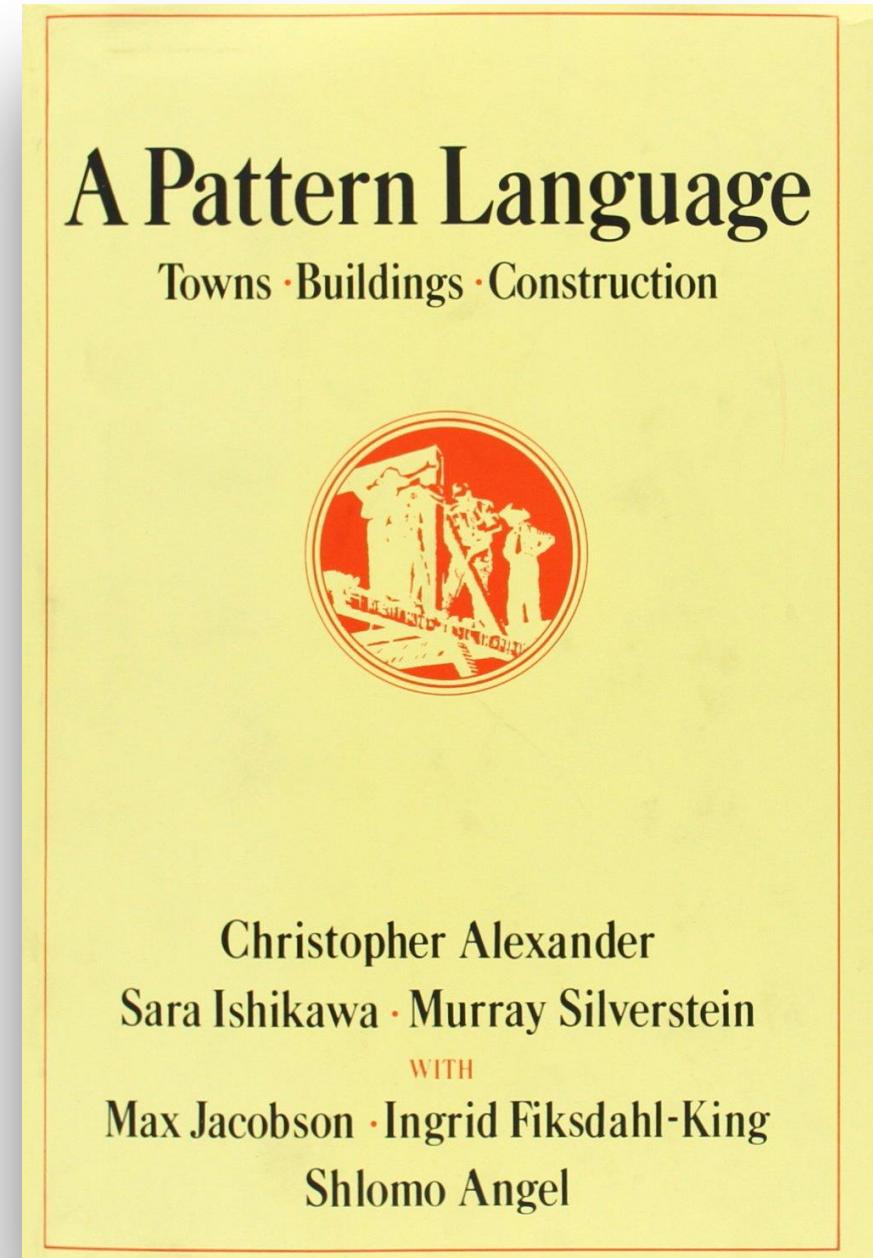
Q: Why was the function sad after a successful first call?

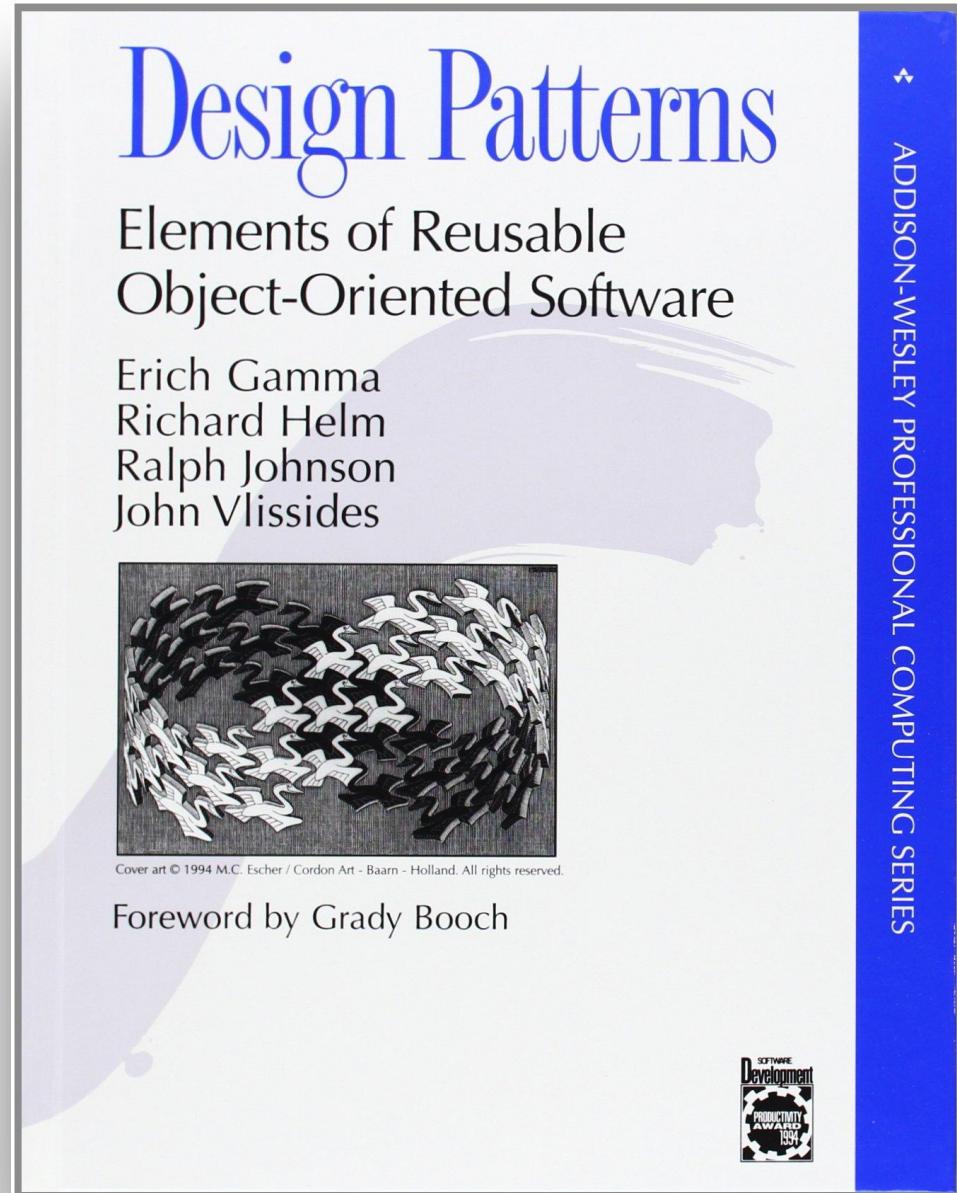
A: It didn't get a callback

PATTERNS

Design Patterns

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





Patterns

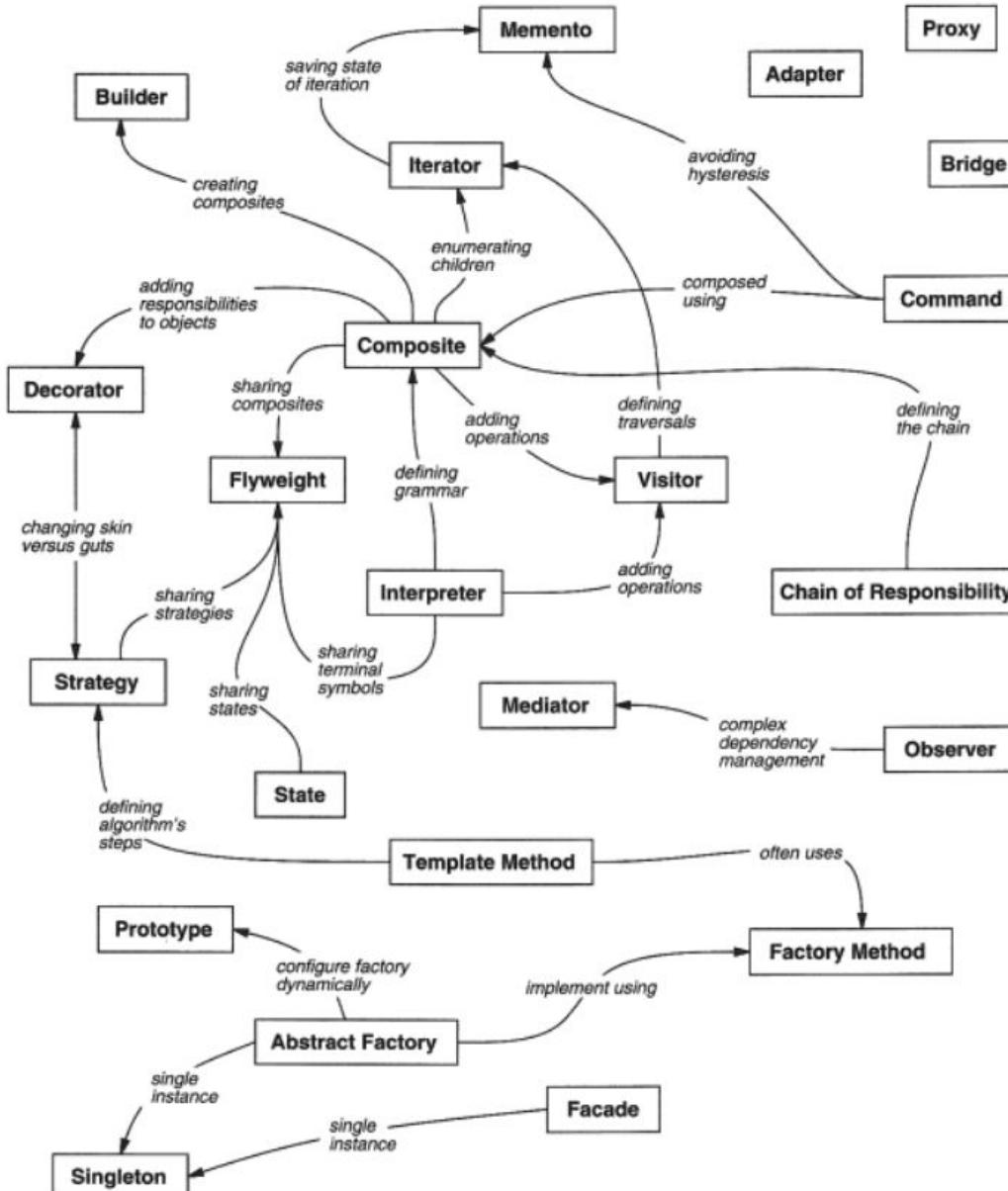
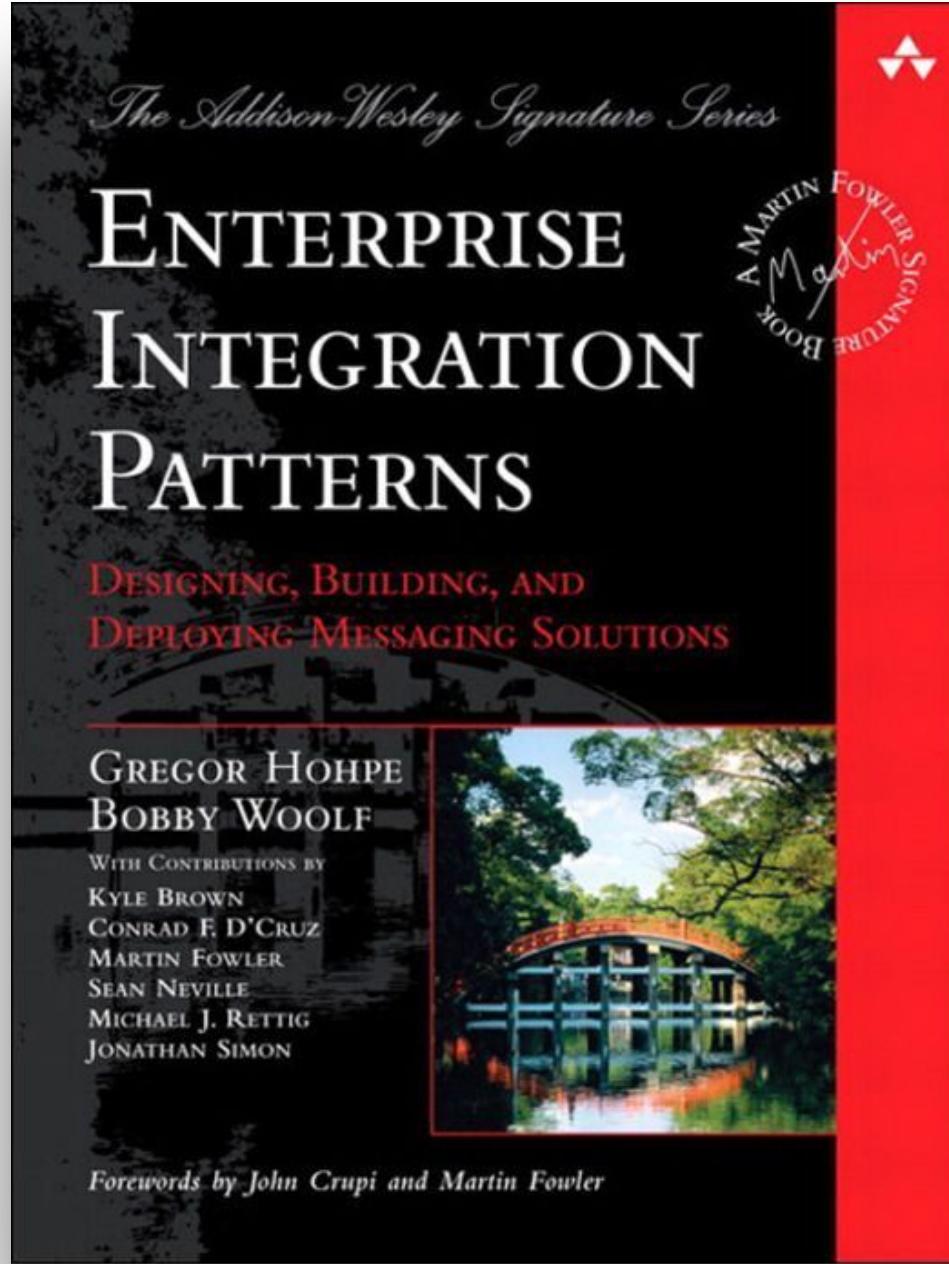
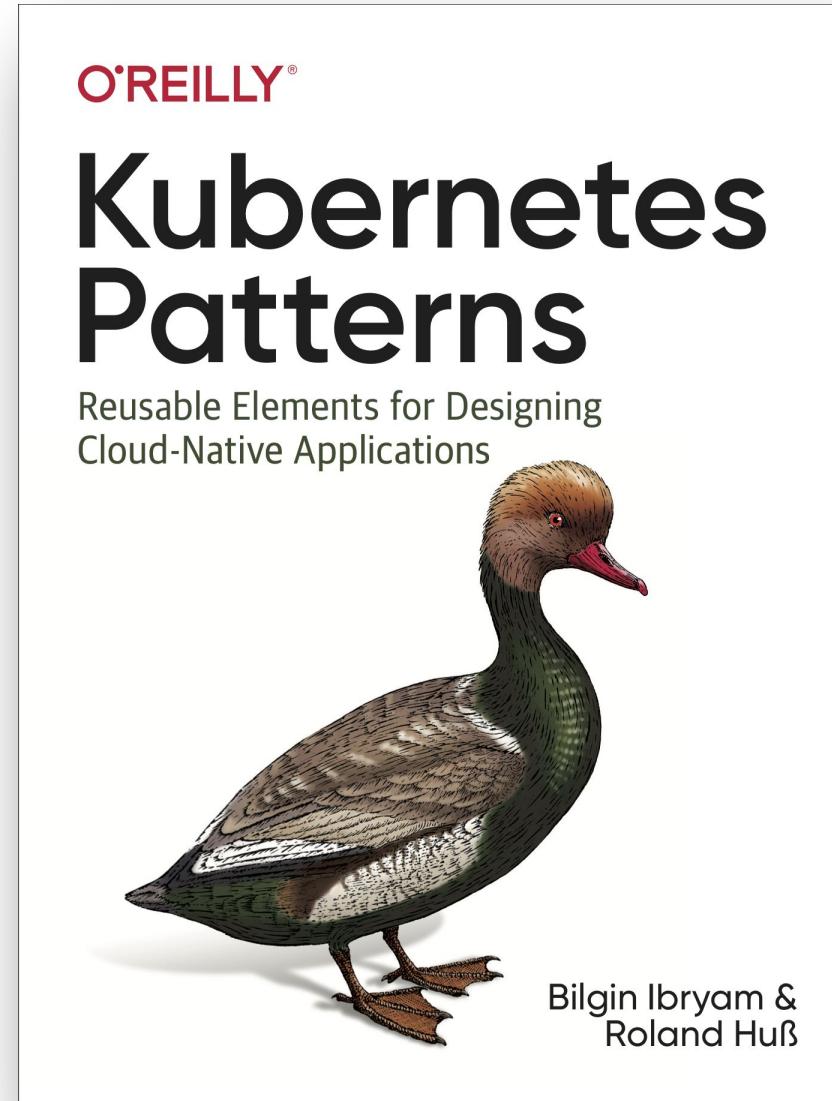


Figure 1.1: Design pattern relationships





Patterns Structure

- Problem
- Patterns:
 - ※ Name
 - ※ Solution

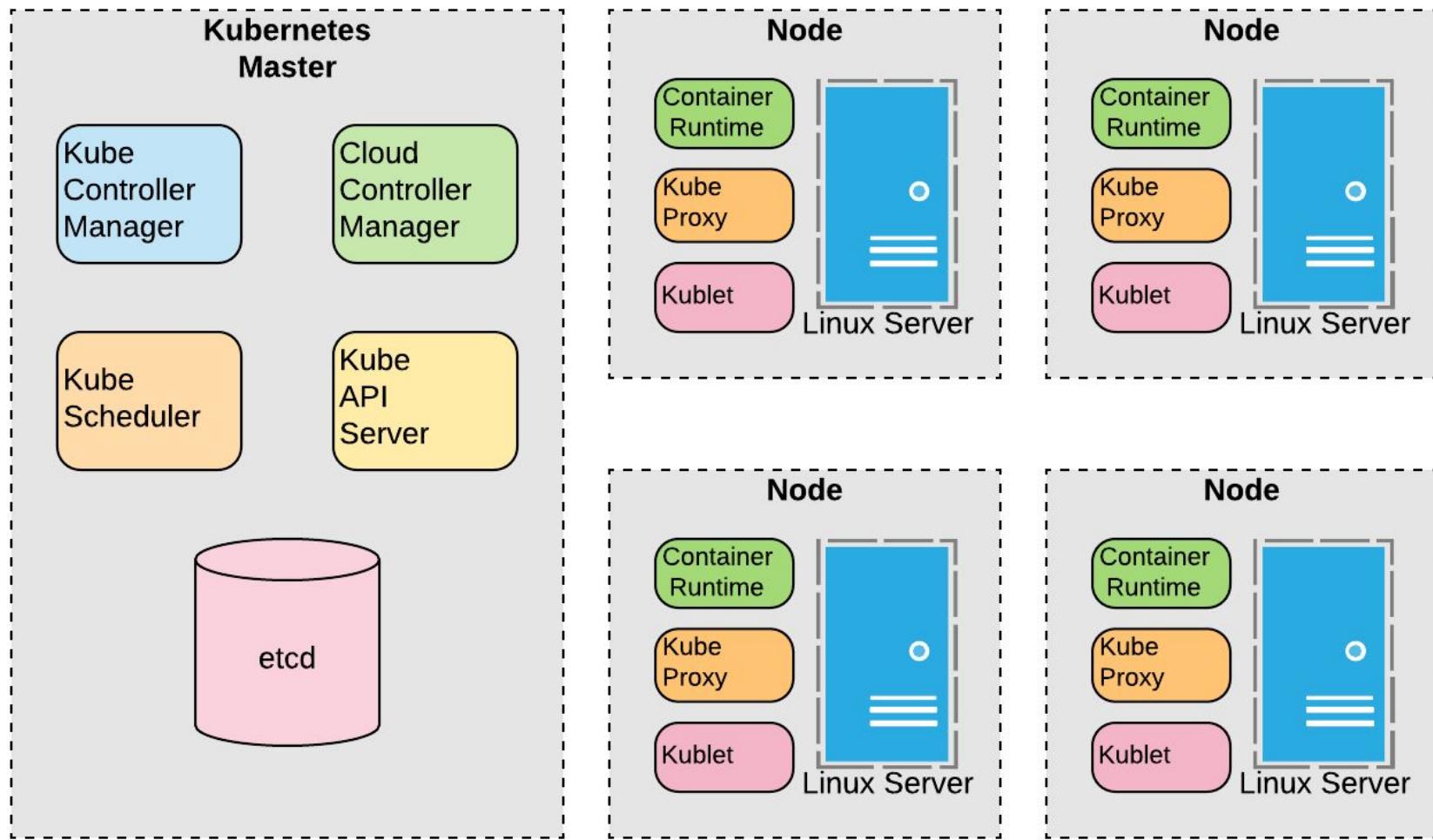
KUBERNETES

Kubernetes

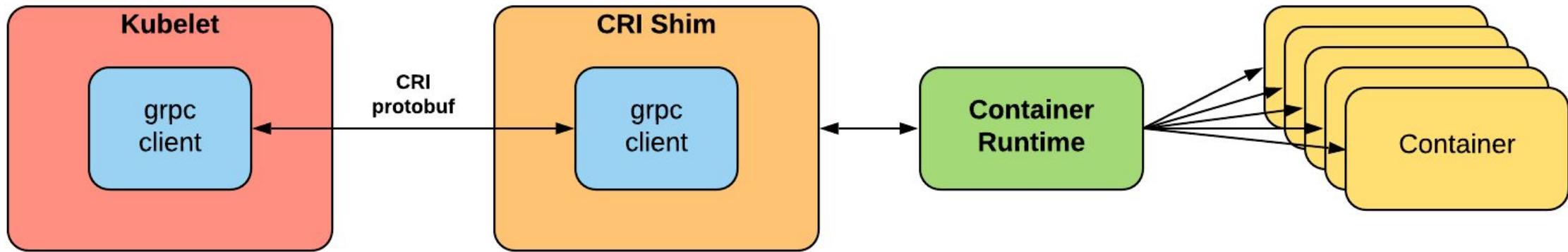


- Open Source container orchestration system started by Google in 2014
 - ※ Scheduling
 - ※ Self-healing
 - ※ Horizontal and vertical scaling
 - ※ Service discovery
 - ※ Automated Rollout and Rollbacks
- Declarative resource-centric REST API

Kubernetes Architecture



Container Runtime



- **Container runtime:** Kubernetes runs containers through an interface called the **CRI** based on **gRPC**.
 - ※ Any container runtime that implements CRI can be used on a node controlled by the kubelet

FOUNDATIONAL PATTERNS

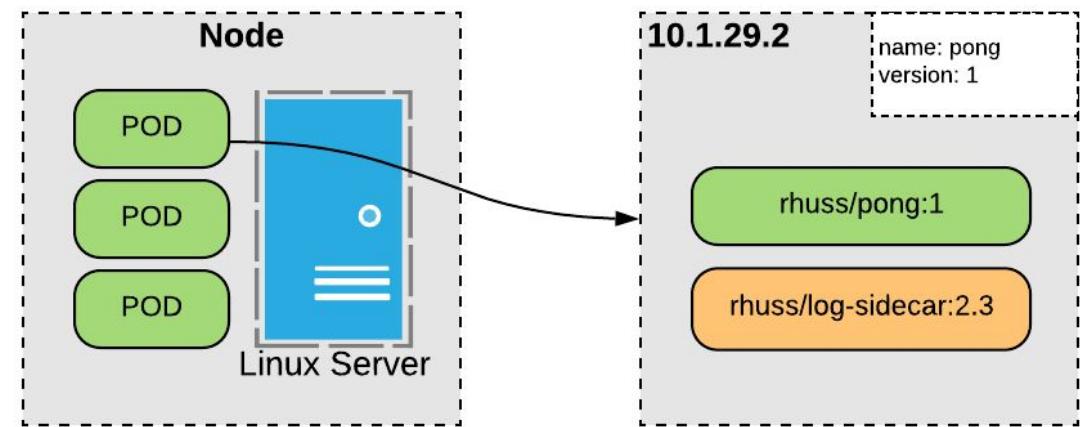
Automatable Unit

How can we create and manage applications with Kubernetes.

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

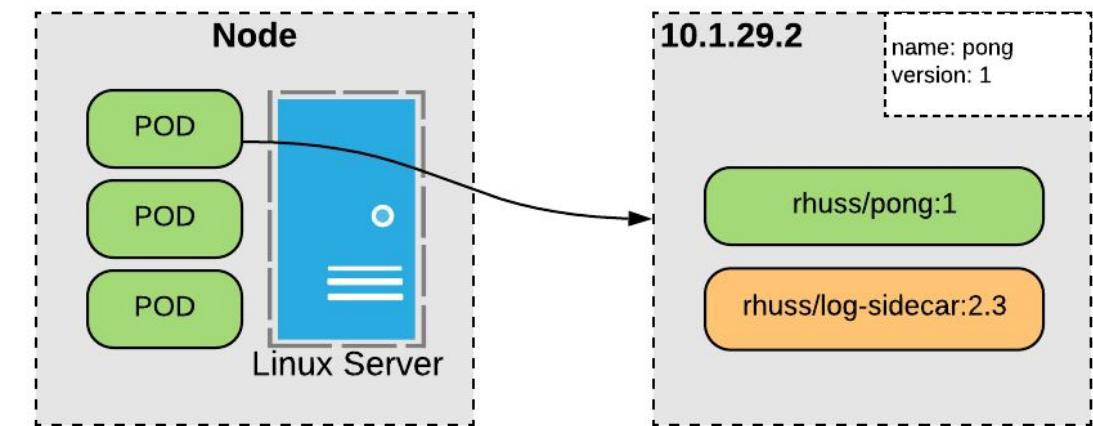
Pod

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

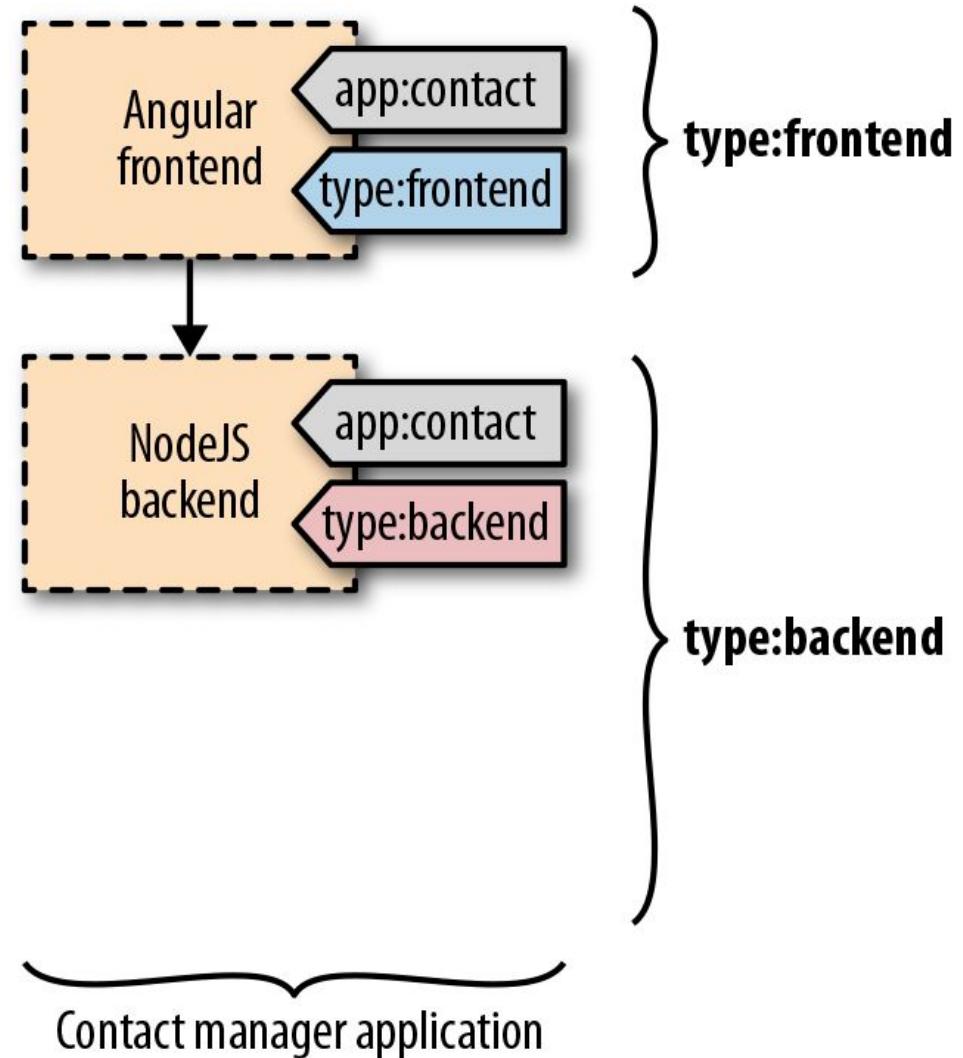
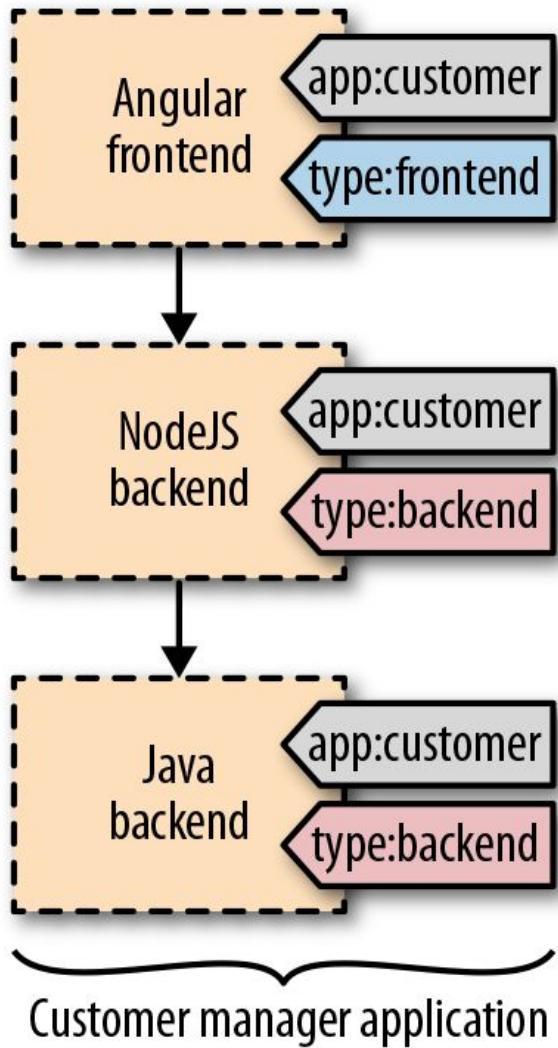


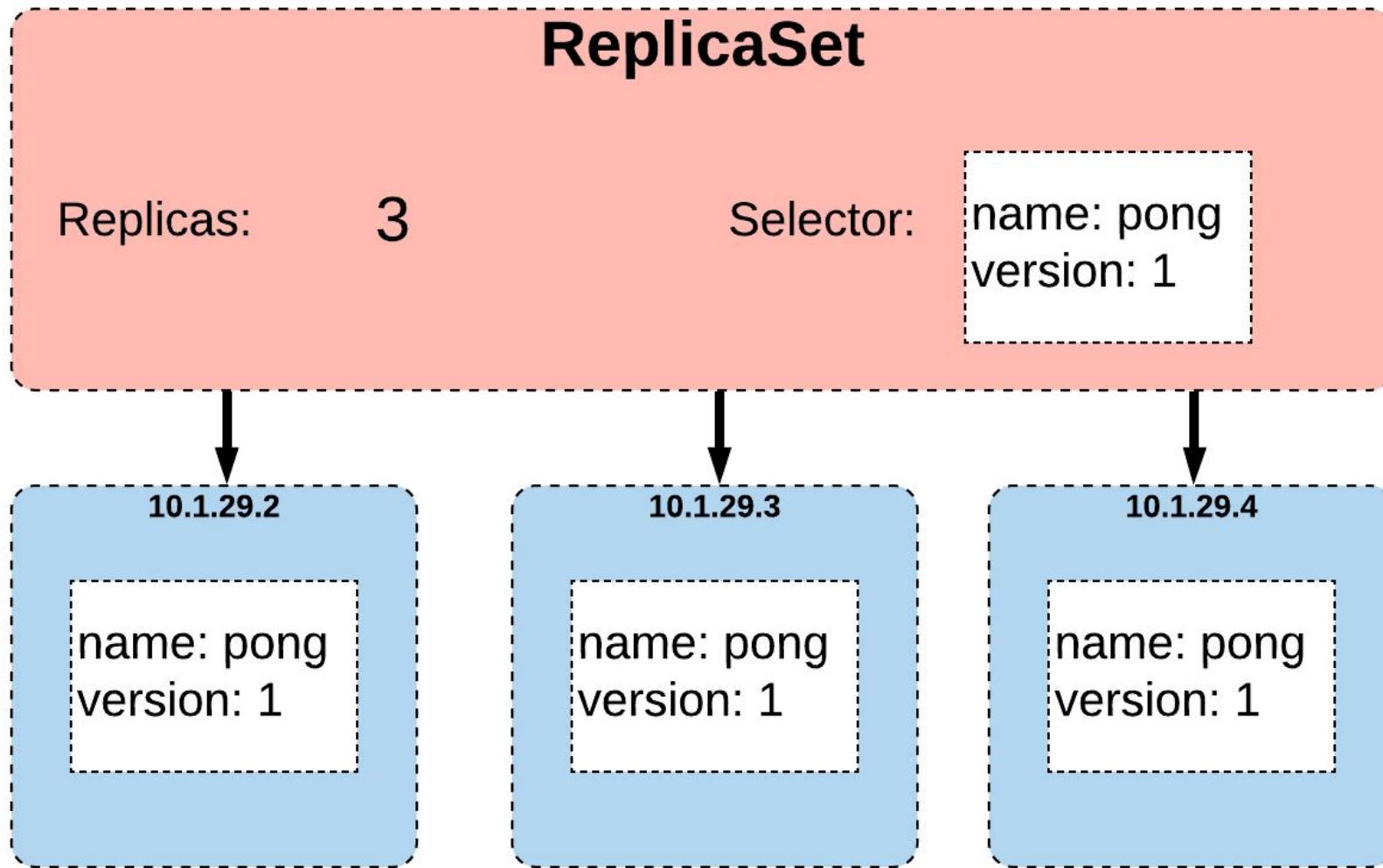
Pod Declaration

```
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
```



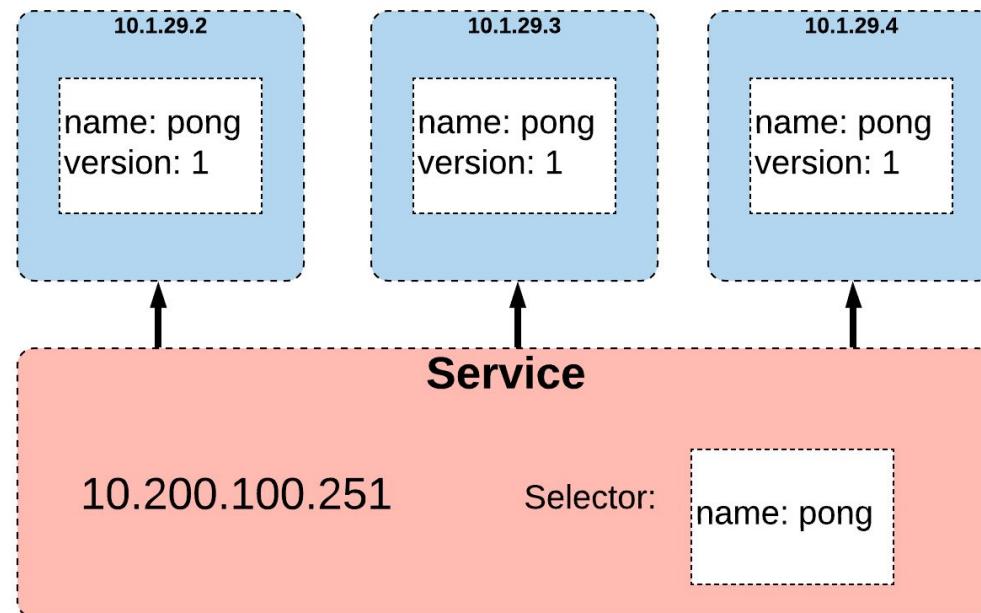
Labels

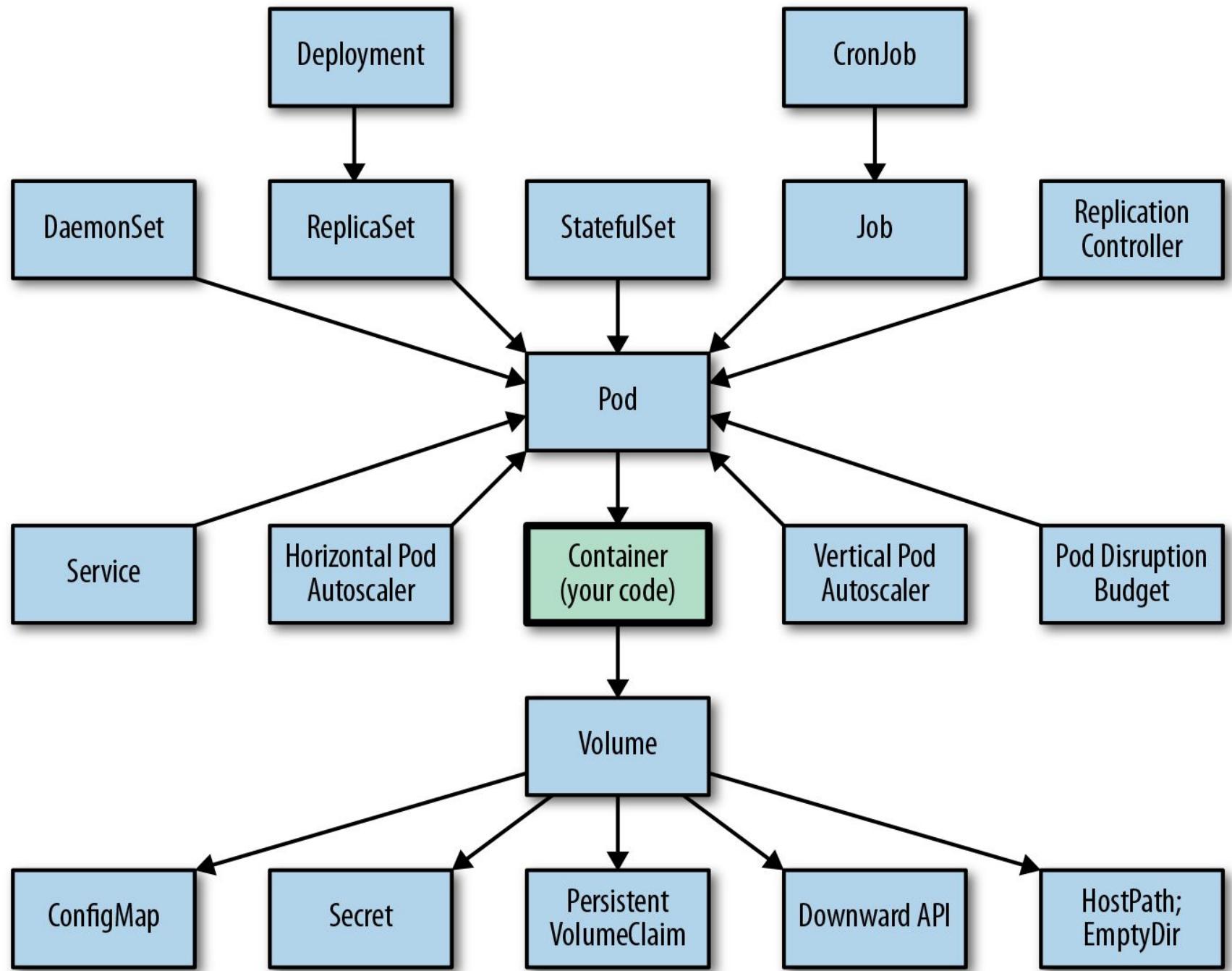




Service

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





Predictable Demands

Application Requirements

How can we handle resource requirements deterministically?

- Declared requirements
 - Scheduling decisions
 - Capacity planning
 - Matching infrastructure services
- Runtime dependencies
 - Persistent Volumes
 - Host ports
 - Dependencies on **ConfigMaps** and **Secrets**

Resource Profiles

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

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

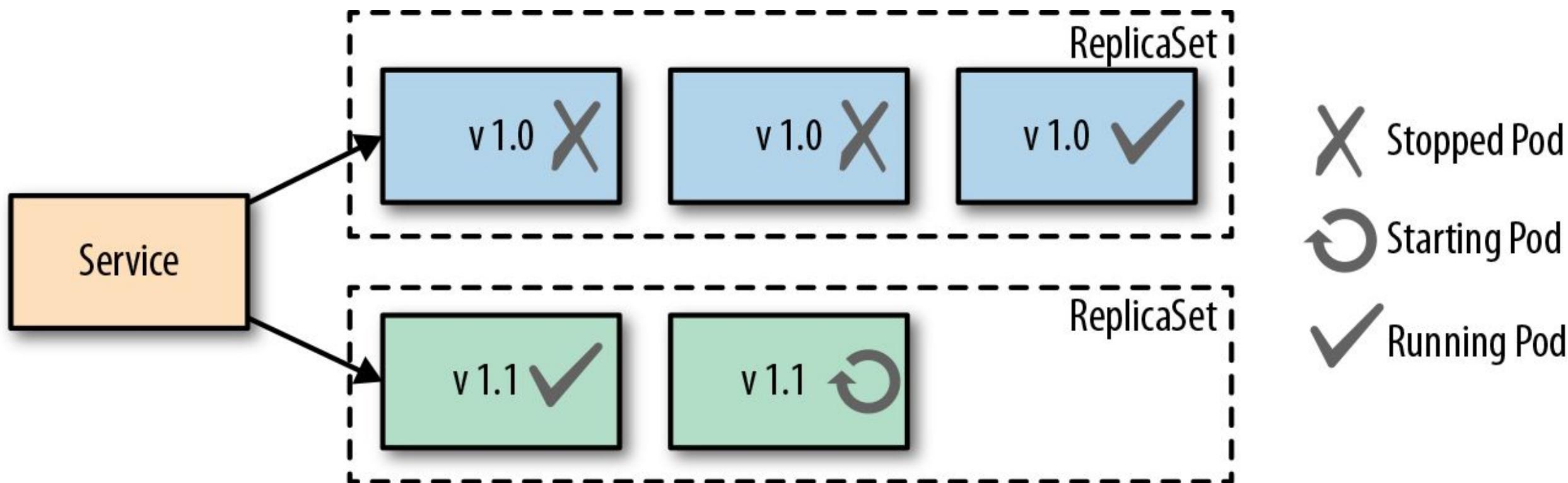
Declarative Deployment

Deployment

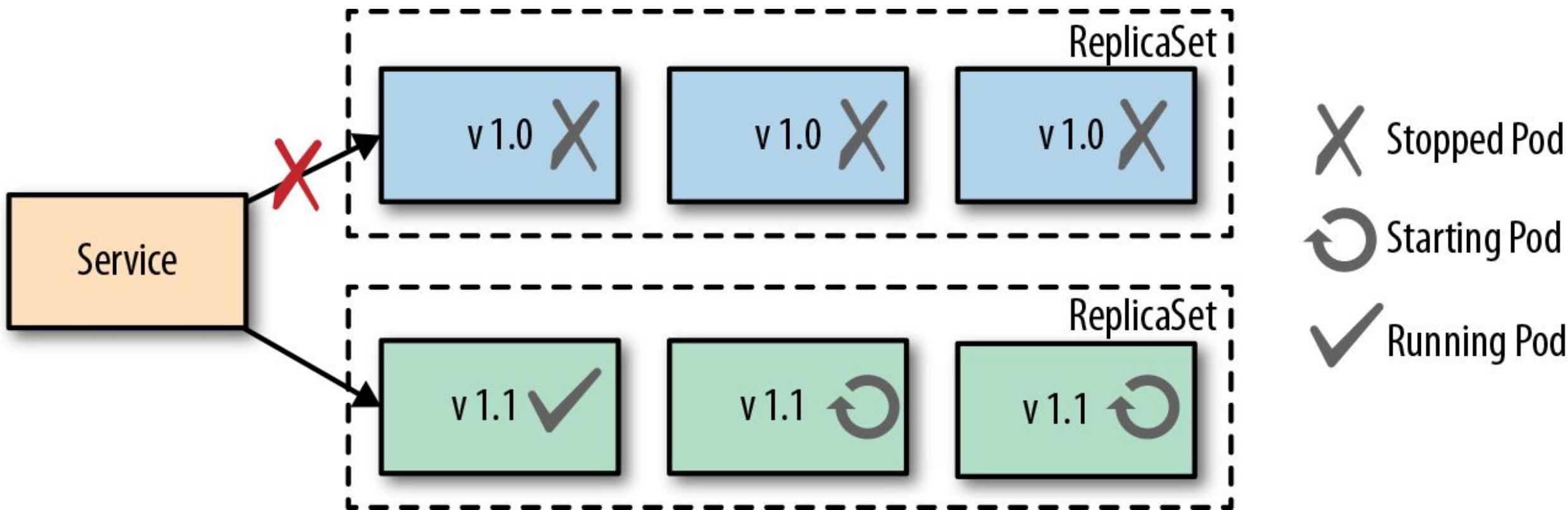
How can applications be deployed and updated?

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

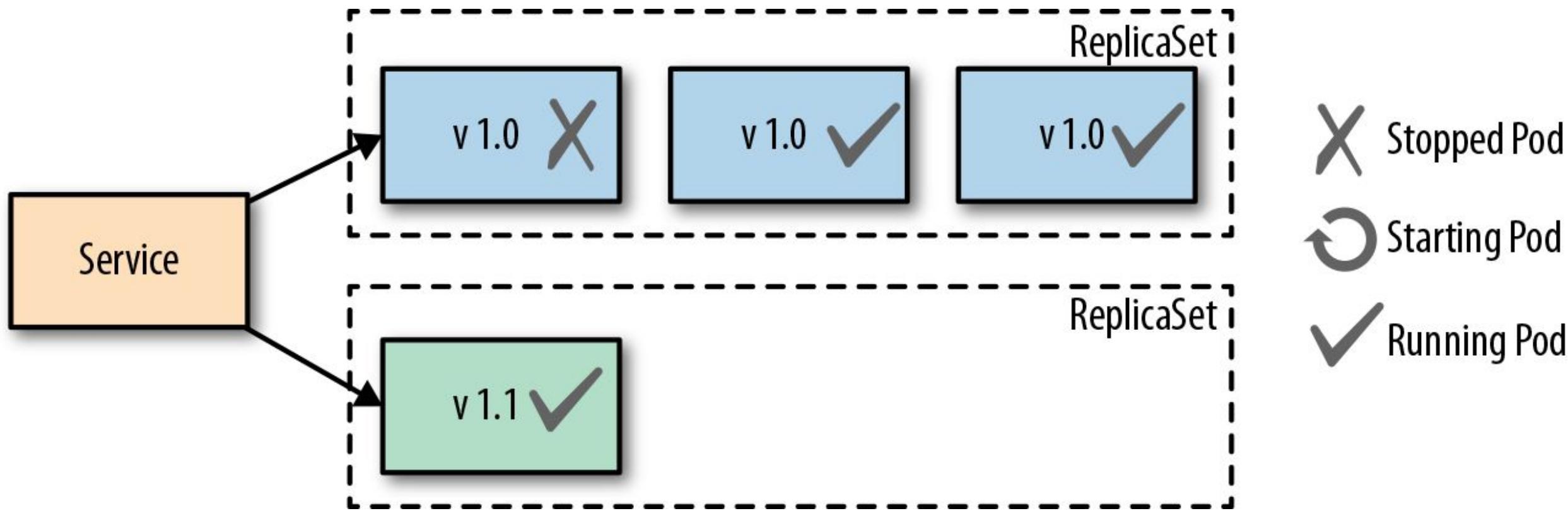
Rolling Deployment



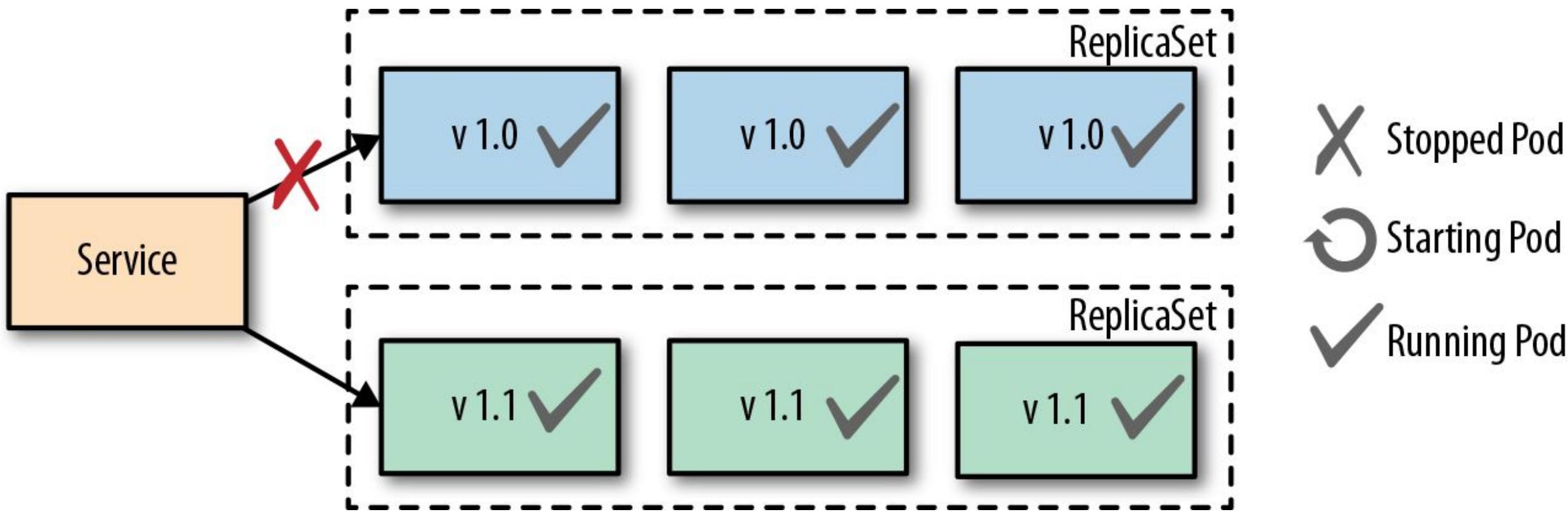
Fixed Deployment



Canary Release

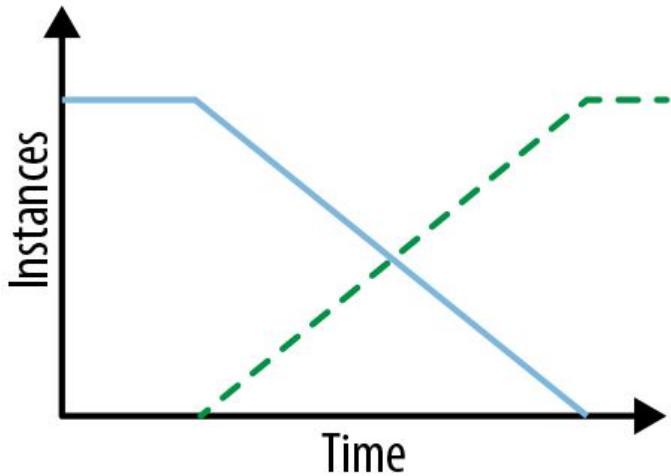


Blue-Green Release

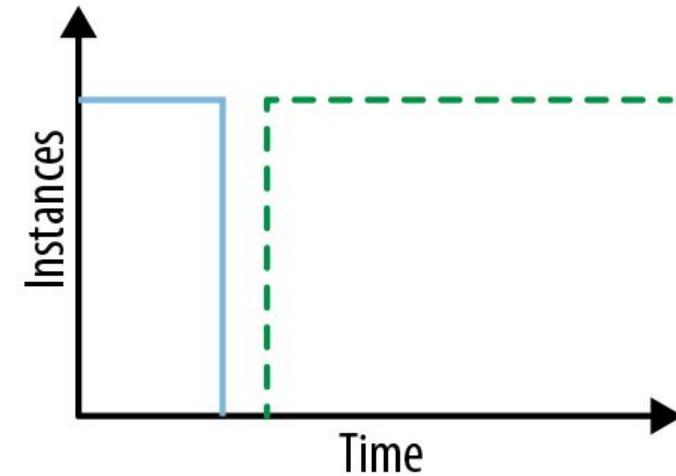


Declarative Deployment

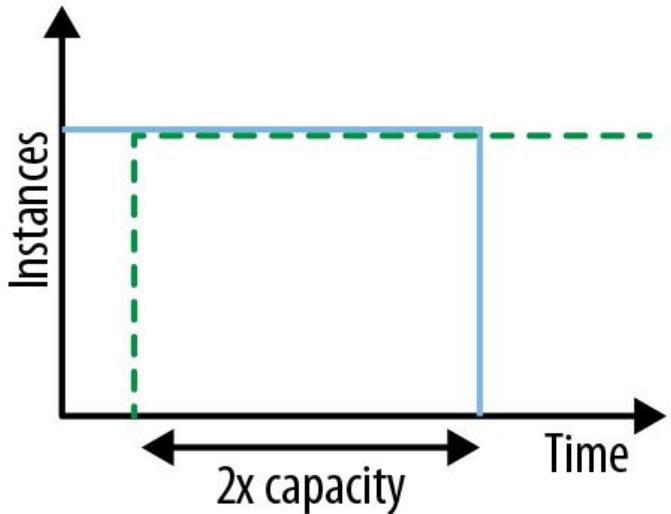
Rolling deployment



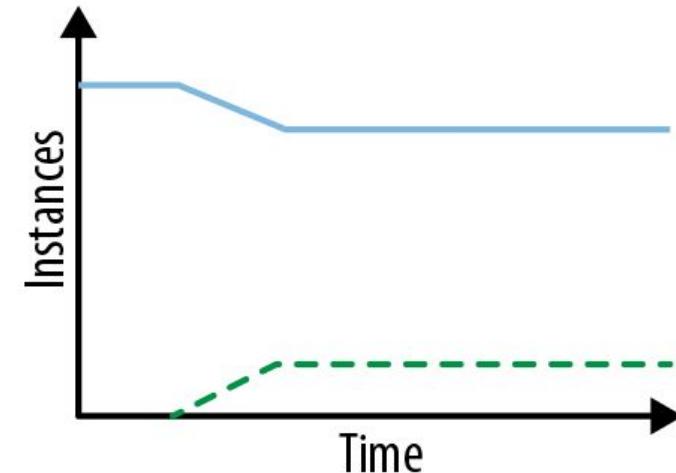
Fixed deployment



Blue-green release



Canary release



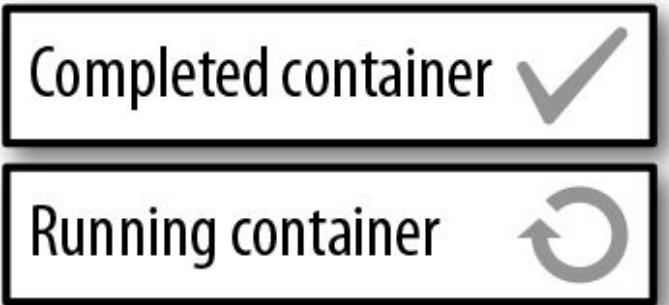
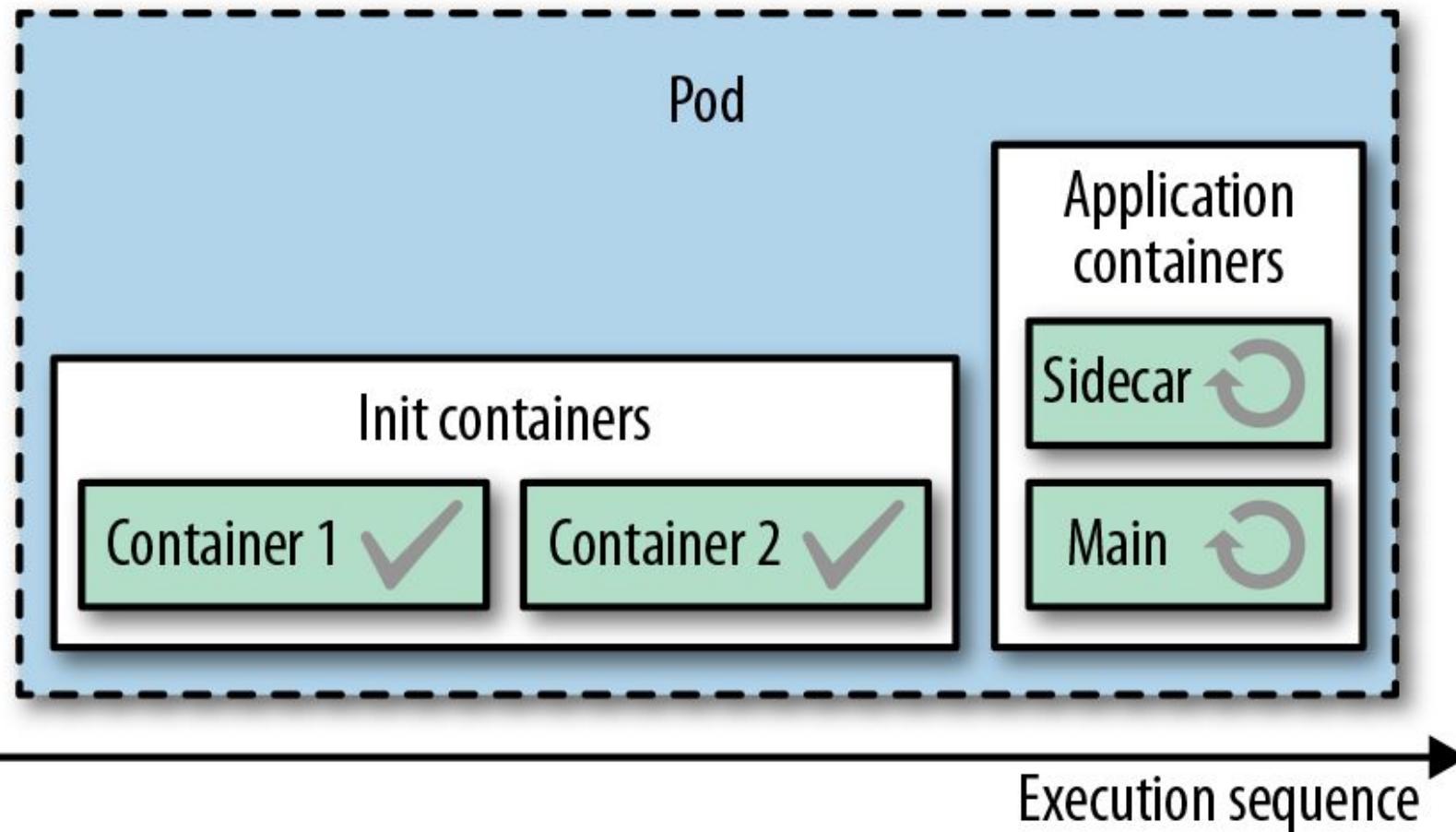
STRUCTURAL PATTERNS

Init Container

How can we initialize our containerized applications?

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

Init Container



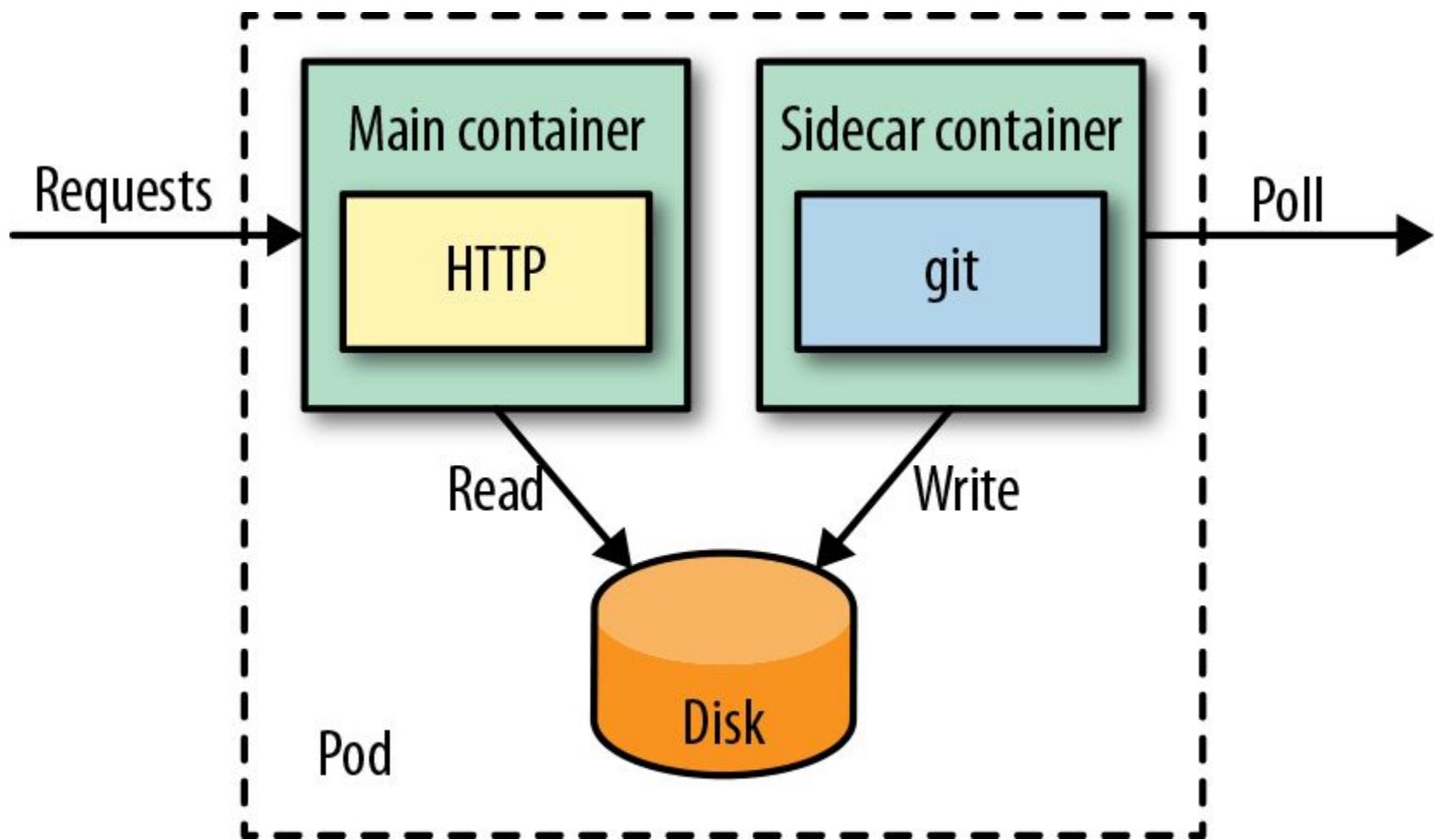
```
apiVersion: v1
kind: Pod
.....
spec:
  initContainers:
  - name: download
    image: axeclbr/git
    command: [ "git", "clone", "https://github.com/myrepo", "/data" ]
    volumeMounts:
      - mountPath: /var/lib/data
        name: source
  containers:
  - name: run
    image: docker.io/centos/httpd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

Sidecar Pattern

How do we enhance the functionality of an application without changing it?

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

Sidecar

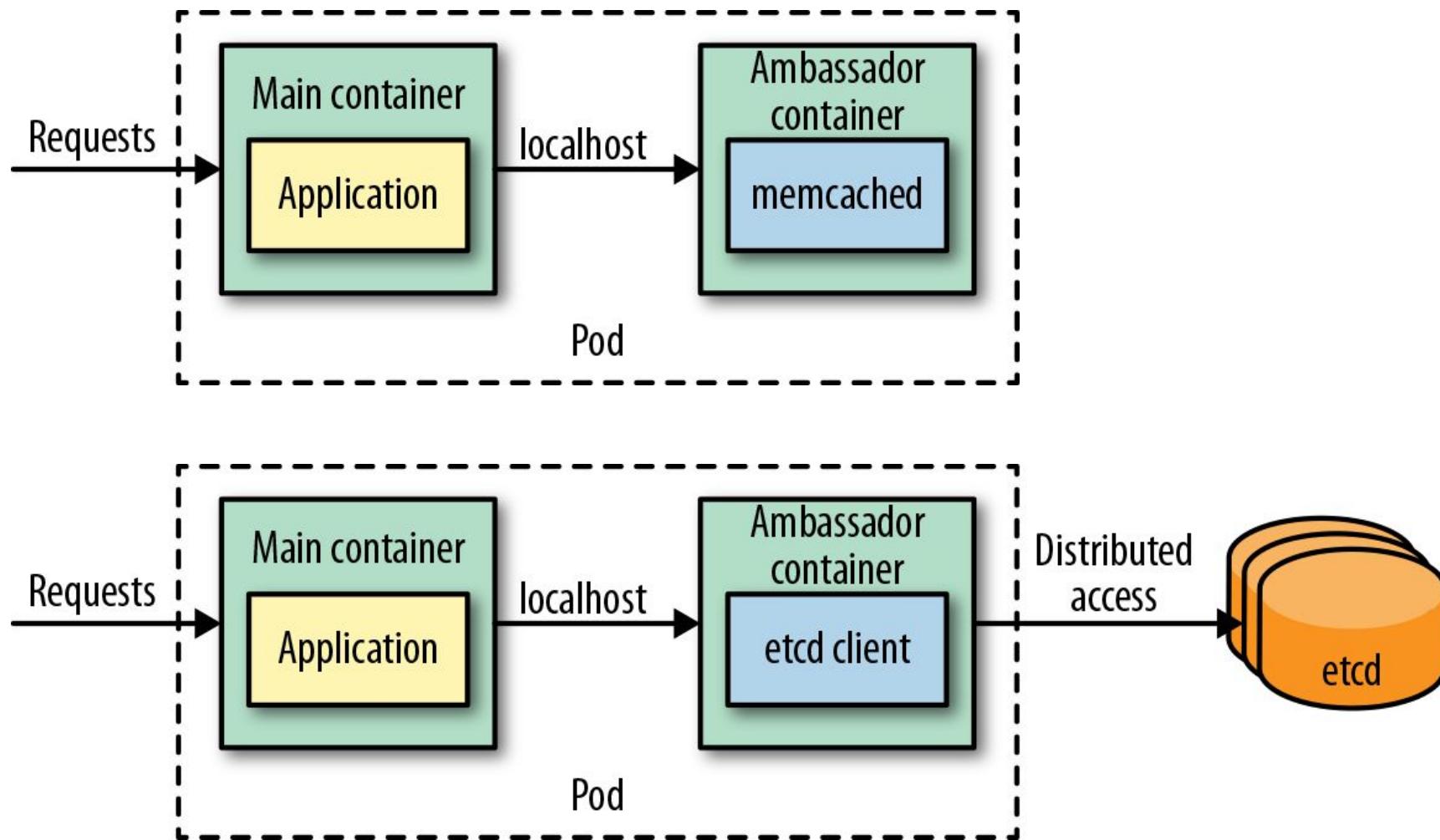


Ambassador Pattern

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

- Also known as **Proxy**
- Specialization of a Sidecar
- Examples for infrastructure services
 - Circuit breaker
 - Tracing

Ambassador

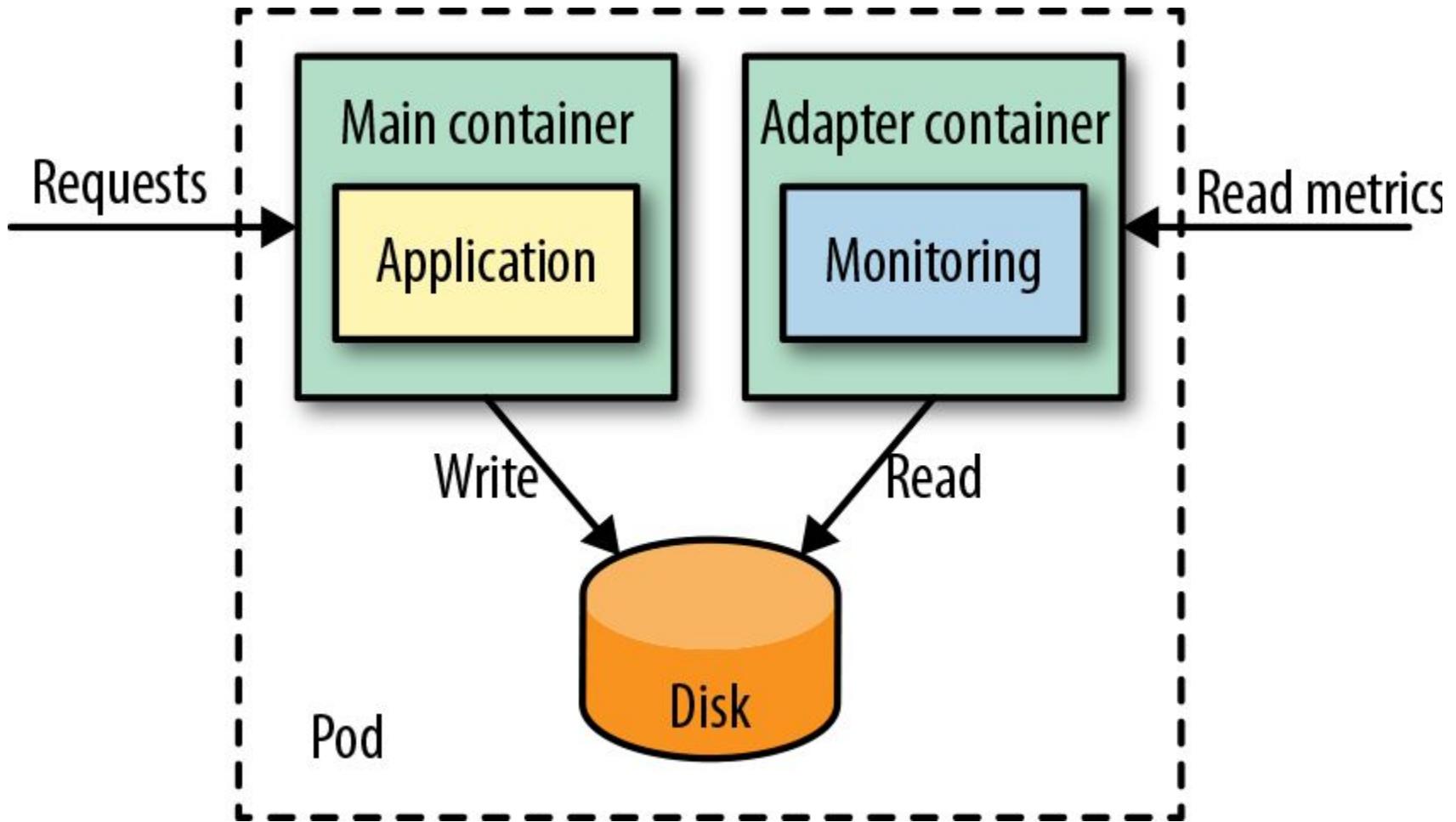


Adapter Pattern

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

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

Adapter



CONFIGURATIONAL PATTERNS

How can applications be configured for different environments?

EnvVar Configuration

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

EnvVar Configuration

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

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
```

ConfigMap

```
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
```

Secret

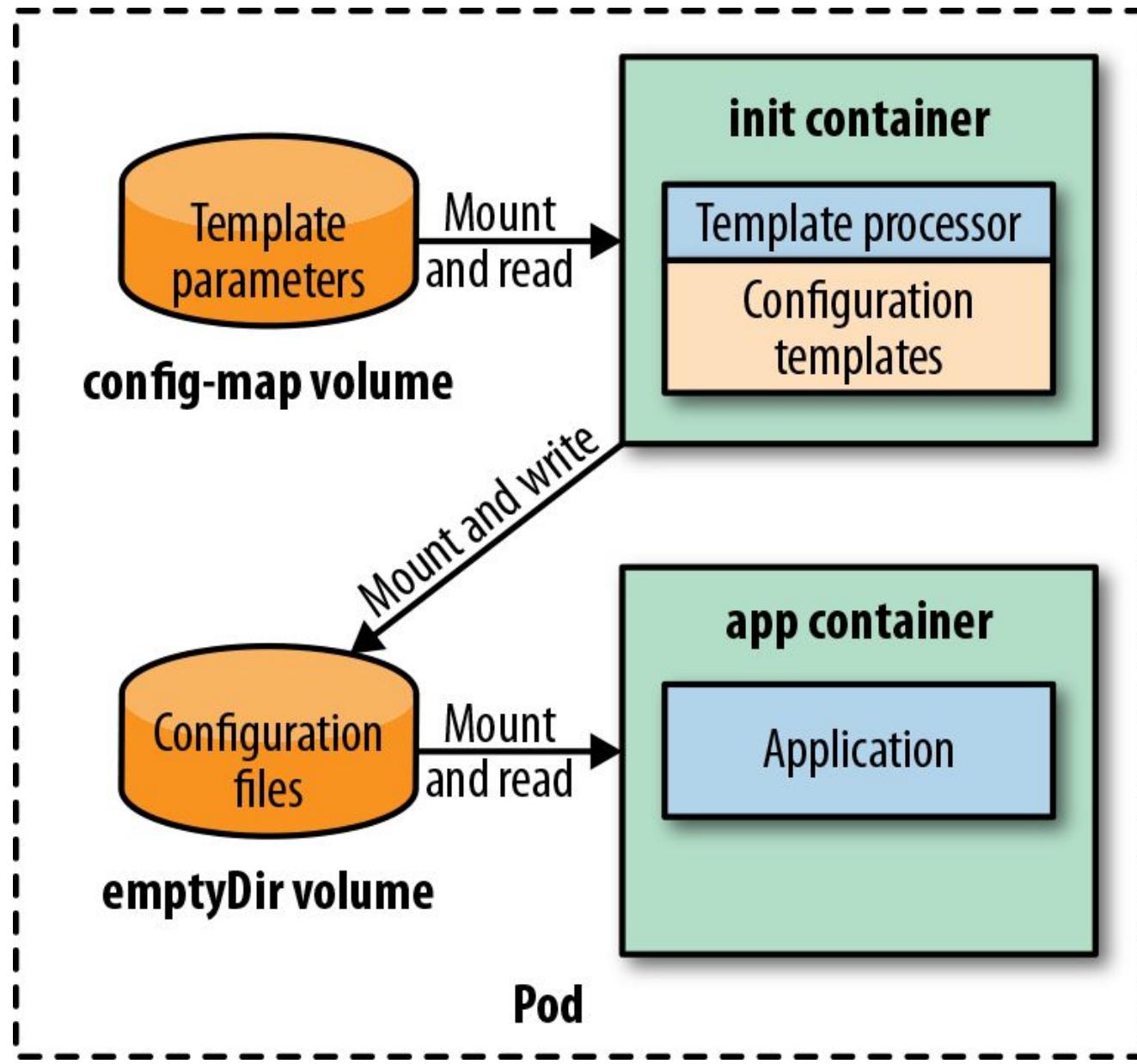
- Like ConfigMap but content Base64 encoded
- Secrets are ...
 - ... only distributed to nodes running Pods that need it
 - ... only stored in memory in a tmpfs and never written to physical storage
 - ... stored encrypted in the backend store (etcd)
- Access can be restricted with RBAC rules
- But: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

Configuration Template

How to manage large and complex similar configuration data?

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

Configuration Template

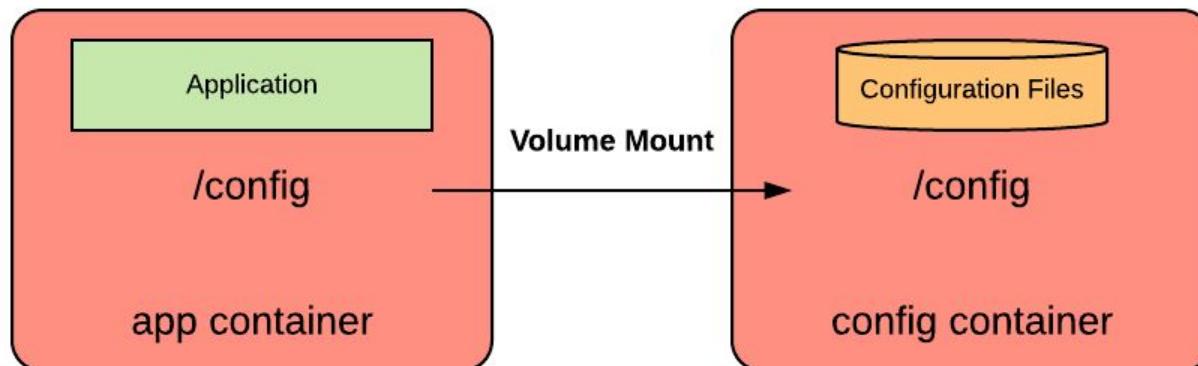


Configuration Template

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

Immutable Configuration

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



- Not directly supported by Kubernetes

ADVANCED PATTERNS

Operator

How to encapsulate operational knowledge into executable software?

- We want to encapsulate operational knowledge so we can
 - Manage installations
 - Manage configuration
 - Manage updates and fail-overs

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

OperatorHub.io

OperatorHub.io

Search OperatorHub... Contribute ▾

Welcome to OperatorHub.io

OperatorHub.io is a new home for the Kubernetes community to share Operators. Find an existing Operator or list your own today.

CATEGORIES 46 ITEMS VIEW ▾ SORT A-Z ▾

Category	Operator Name	Description	Provider
AI/Machine Learning	Aqua Security Operator	The Aqua Security Operator runs within Kubernetes cluster and provides a means to	provided by Aqua Security, Inc.
Big Data	AWS Service Operator	The AWS Service Operator allows you to manage AWS services	provided by Amazon Web Services, Inc.
Cloud Provider	Camel K Operator	Apache Camel K (a.k.a. Kamel) is a lightweight integration	provided by The Apache Software Foundation
Database	CockroachDB	CockroachDB Operator based on the CockroachDB helm chart	provided by Helm Community
Developer Tools	Community Jaeger Operator	Provides tracing, monitoring and troubleshooting microservices-based	provided by CNCF
Integration & Delivery	Couchbase Operator	The Couchbase Autonomous Operator allows users to easily deploy, manage, and main	provided by Couchbase
Logging & Tracing	Crunchy PostgreSQL Enterprise	PostgreSQL is a powerful, open source object-relational	provided by Crunchy Data
Monitoring	Dynatrace OneAgent	Install full-stack monitoring of Kubernetes clusters with the Dynatrace OneAgent.	provided by Dynatrace LLC
Networking			
OpenShift Optional			
Security			
Storage			
Streaming & Messaging			
Other			

PROMISER

- Amazon Web Services (1)
- Aqua Security (1)
- Banzai Cloud (2)

Wrap Up

- Kubernetes offers a rich feature set to manage containerised applications
- Patterns can help in solving recurring Kubernetes, legacy application and Microservices challenges
- Patterns will continue to emerge

Thank you



<https://k8spatterns.io>



@ro14nd



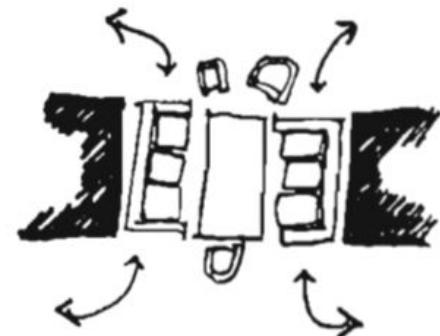
@bibryam



@k8spatterns

Appendix

90 BEER HALL



The open alcove—supports the fluidity of the scene.

criss-cross paths



activities

open alcoves

Kubernetes Architecture and Foundational Elements

KUBERNETES

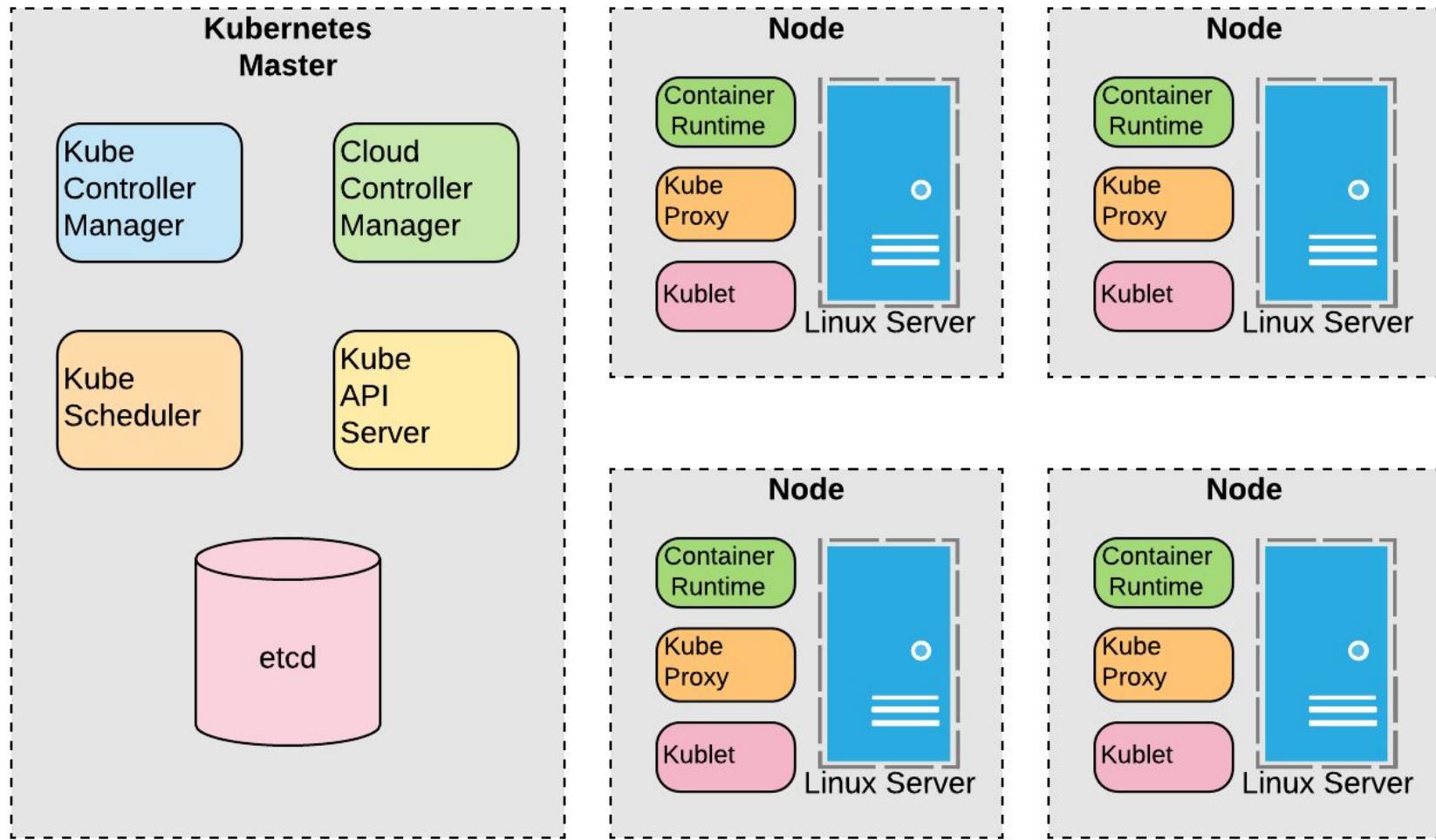
Kubernetes



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

Kubernetes Foundational Elements

Kubernetes Architecture



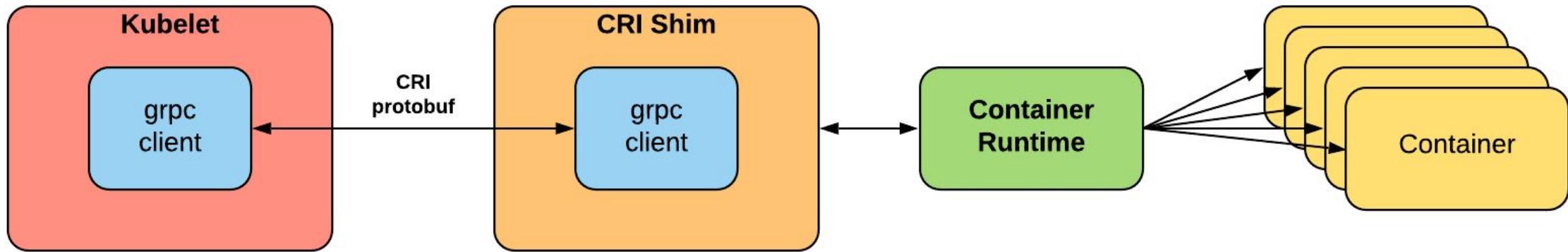
Control Plane Components

- **API Server:** kube-api-server exposes the Kubernetes API to the world. Stores cluster state in etcd
- **etcd metadata store:** a consistent and reliable distributed key-value store (<https://coreos.com/etcd/>)
- **Scheduler:** kube-scheduler schedules pods to worker nodes
- **Controller manager:** kube-controller manager is a single process that contains multiple controller watching for events and changes to the cluster
- **Cloud controller manager:** Embeds cloud-specific control loops.

Data Plane Components

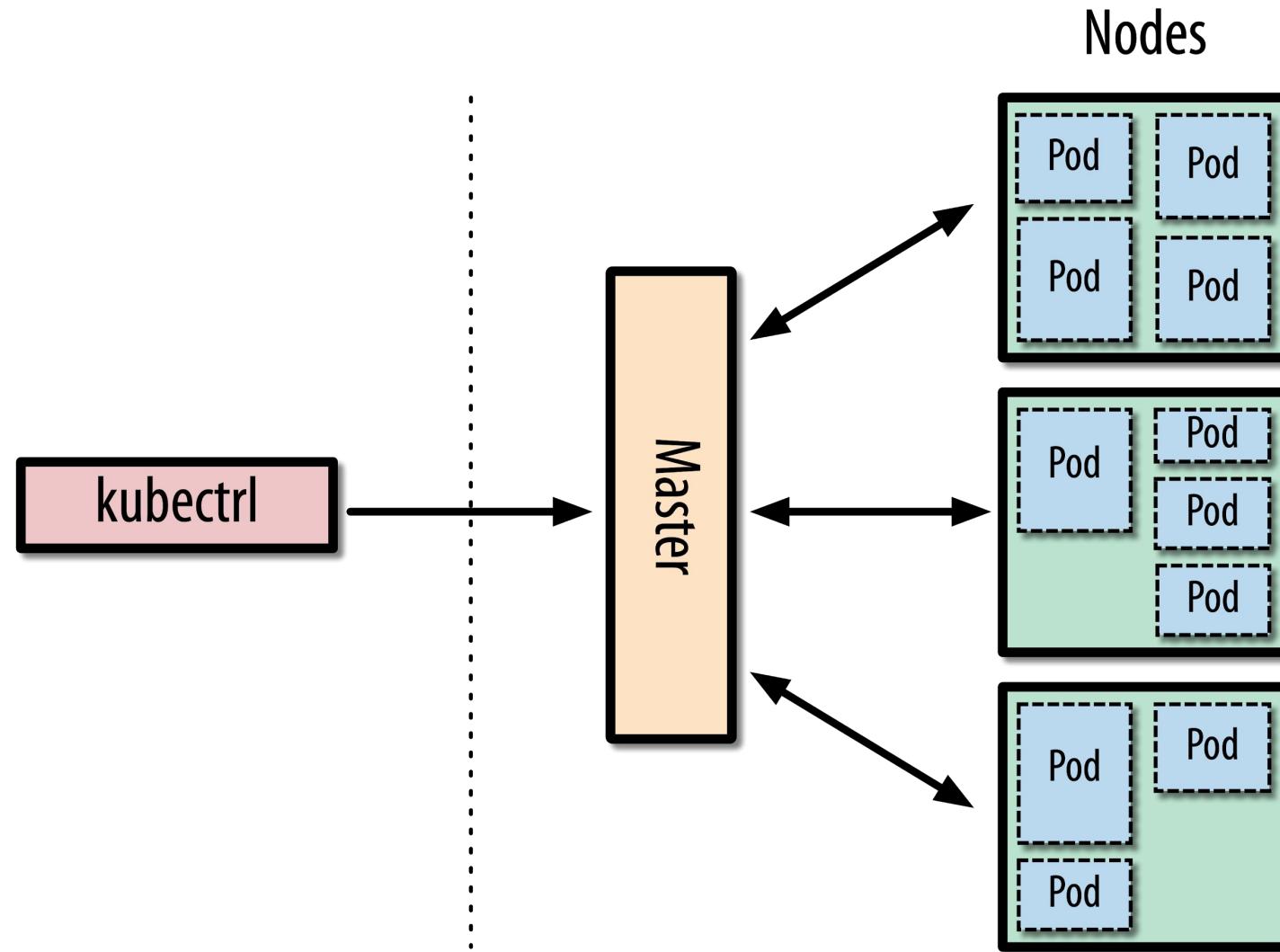
- **A collection of nodes that run containerized workloads as pods**
- **kubelet**: Responsible for communicating with the API server and running and managing pods on the node
- **kube-proxy**: Responsible for the networking of the node. Fronts services and can forward TCP and UDP packets and also discovers addresses of services via DNS or environment variables
- **Kubectl**: The command-line interface (CLI) to the Kubernetes cluster (`kubectl <command> --help`)

Data Plane Components

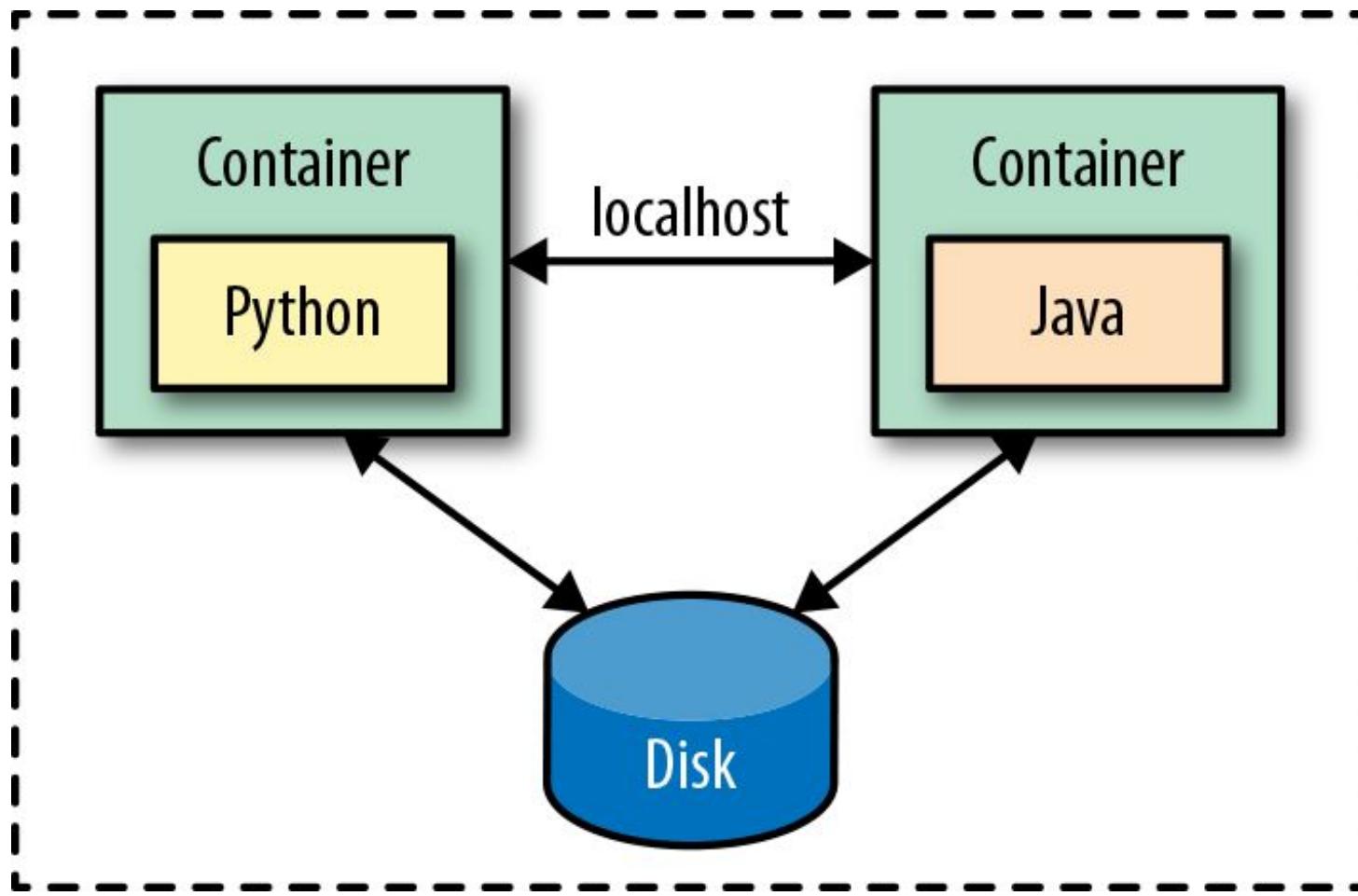


- **Container runtime:** Kubernetes runs containers through an interface called the **CRI** based on **gRPC**.
 - ※ Any container runtime that implements CRI can be used on a node controlled by the kubelet

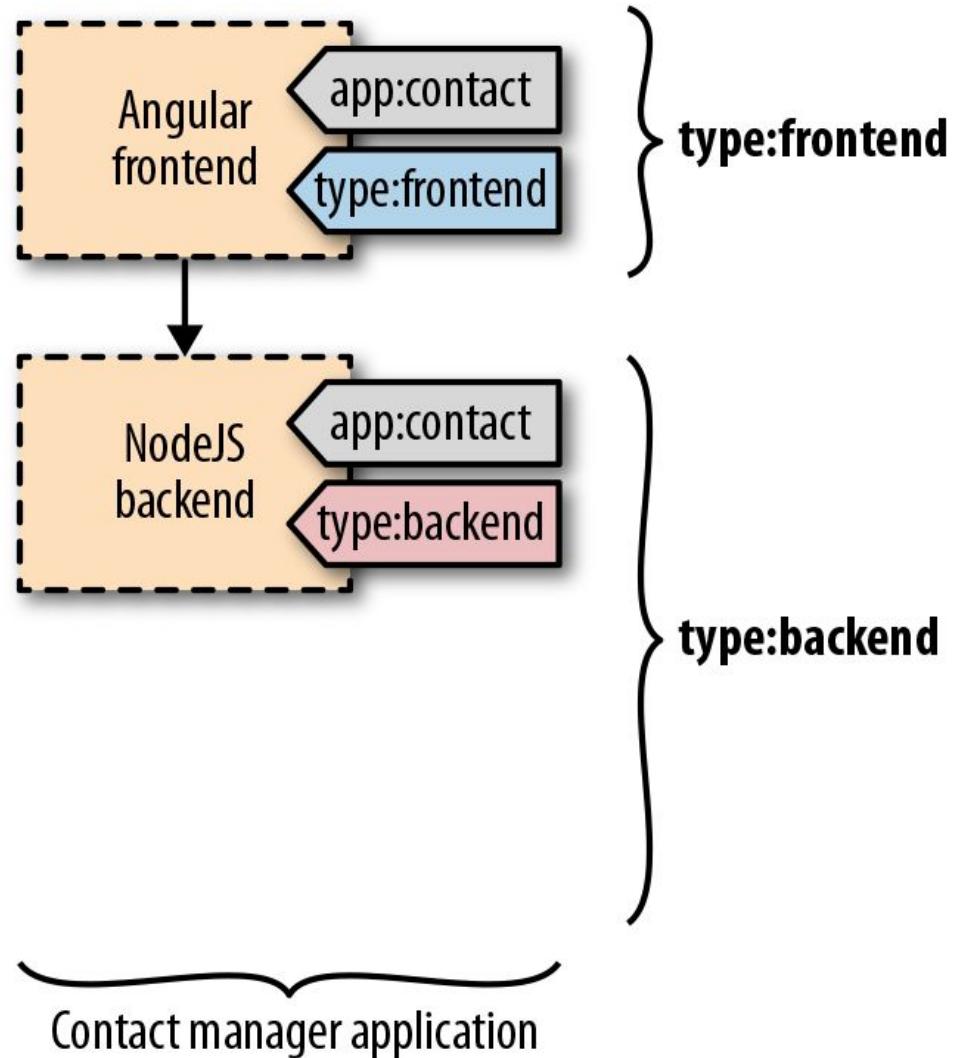
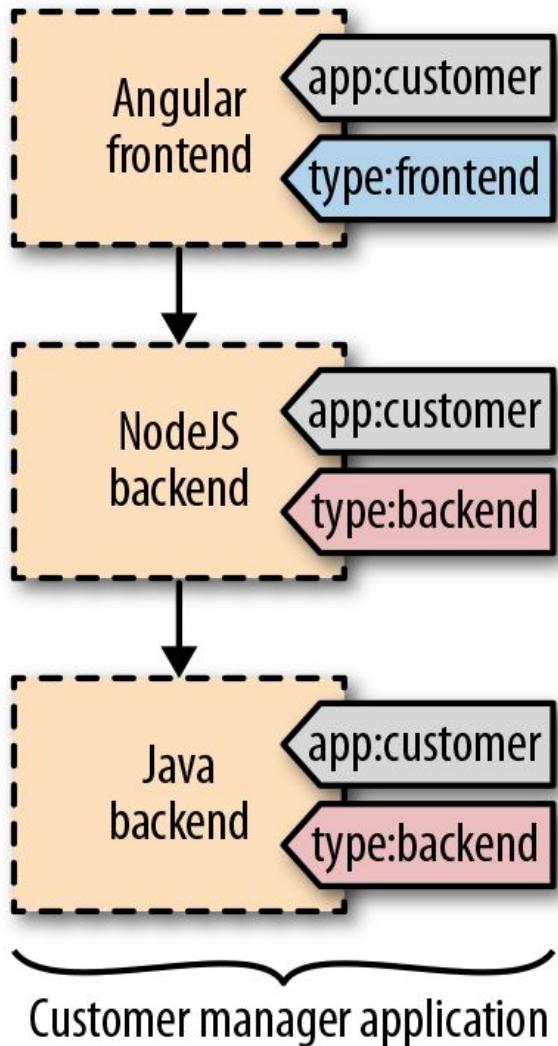
Architecture



Pod



Labels



Health Probe Pattern

Monitoring Health

How to communicate an application's health state to Kubernetes?

- 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

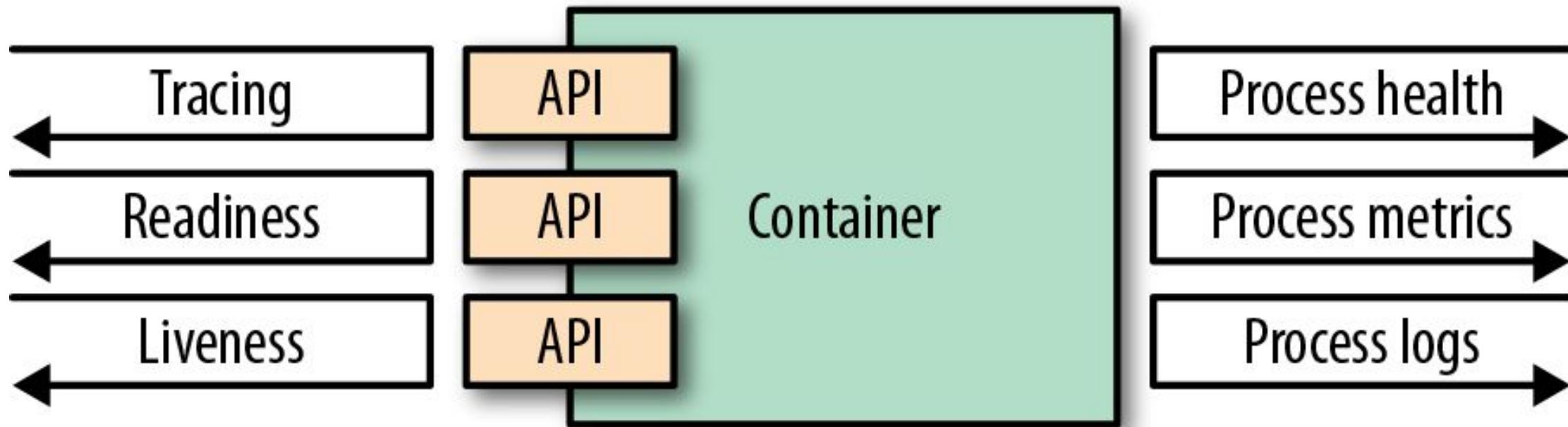
Liveness & Readiness

- 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

Health Probe

```
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

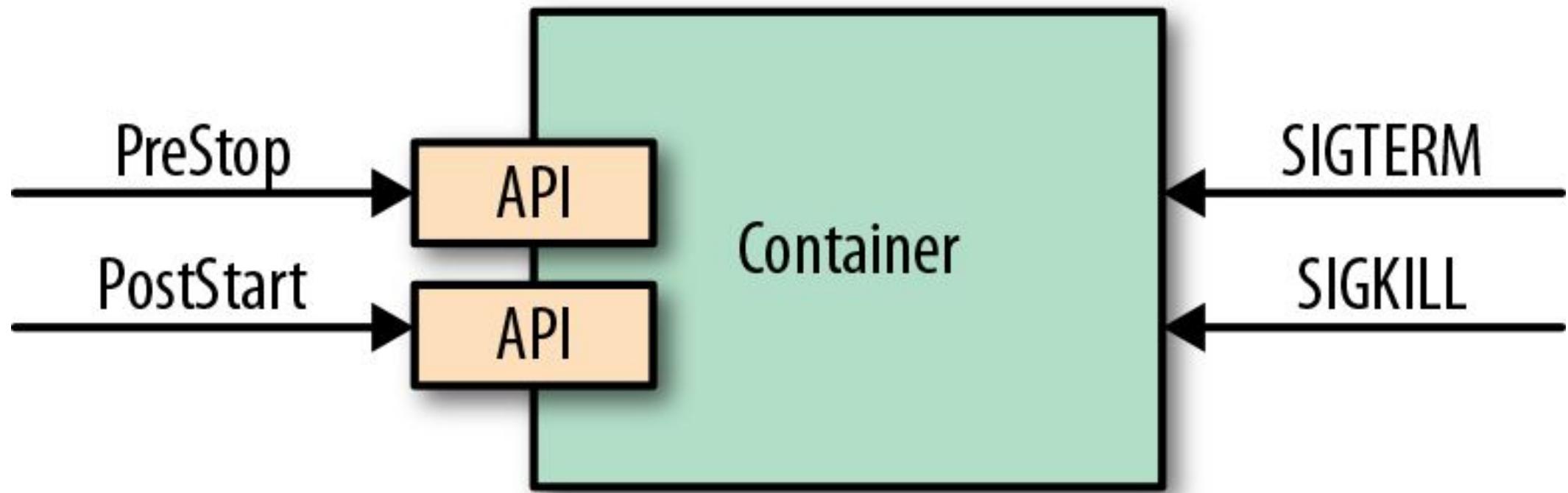


Managed Lifecycle Pattern

Lifecycle Events

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

Managed Lifecycle

```
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
```

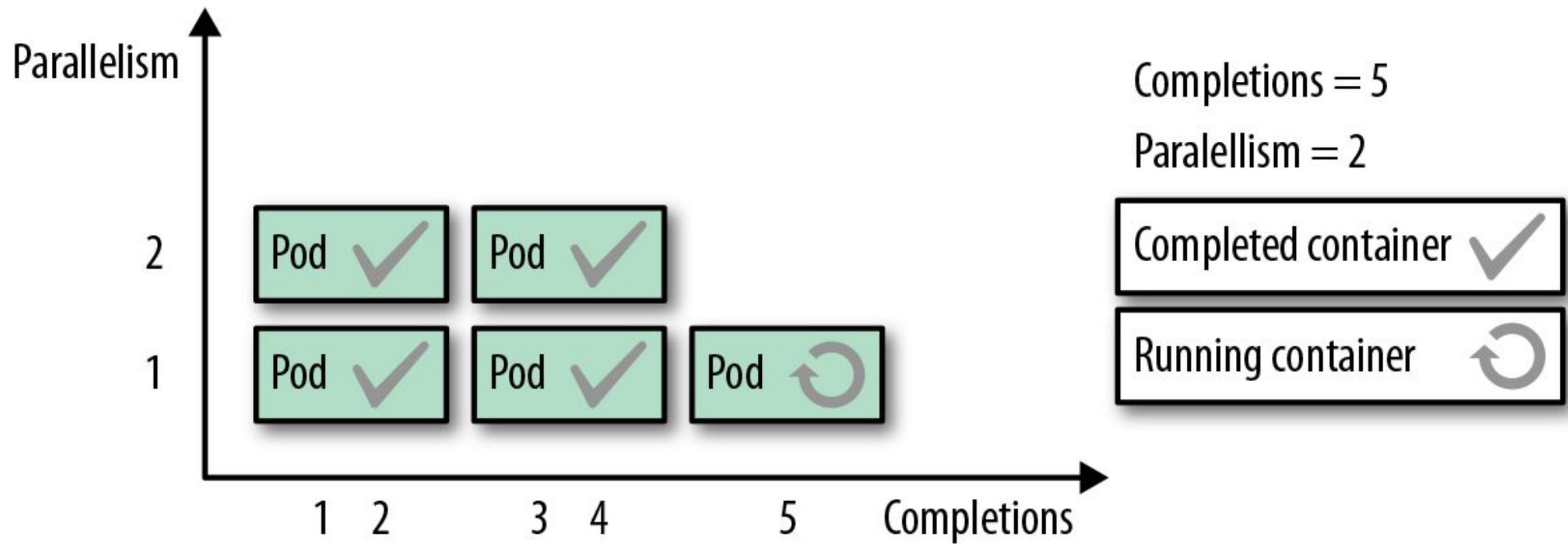
Batch Job Pattern

Job

How to run short-lived Pods reliably until completion?

- 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

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 exiting successfully
 - Pods needs to coordinate to shutdown in a coordinate fashion

Stateful Service Pattern

Distributed Stateful Services

How to manage stateful workloads?

- 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** automatically 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
```

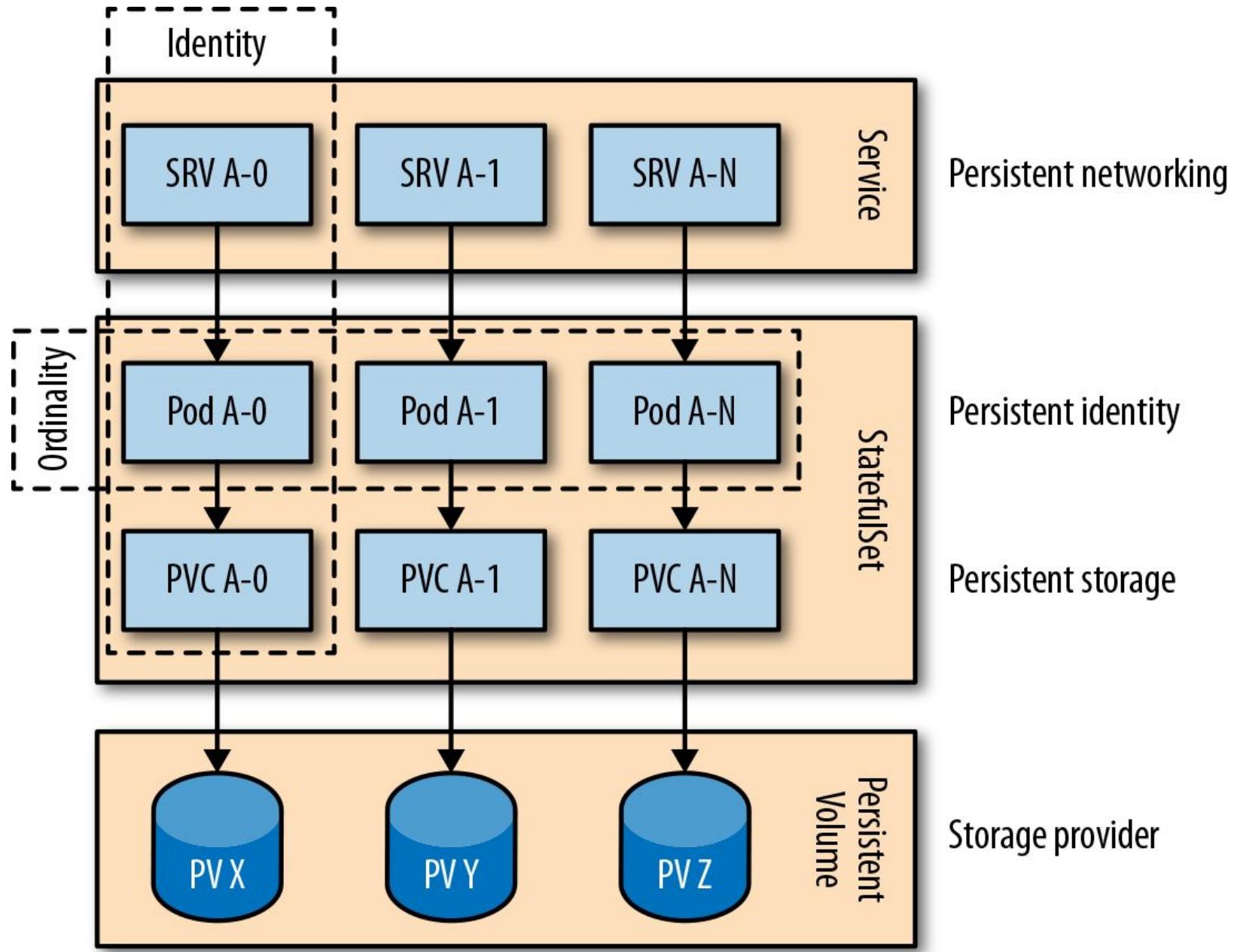
rg-0.random-generator.default.svc.cluster.local
rg-1.random-generator.default.svc.cluster.local

...

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

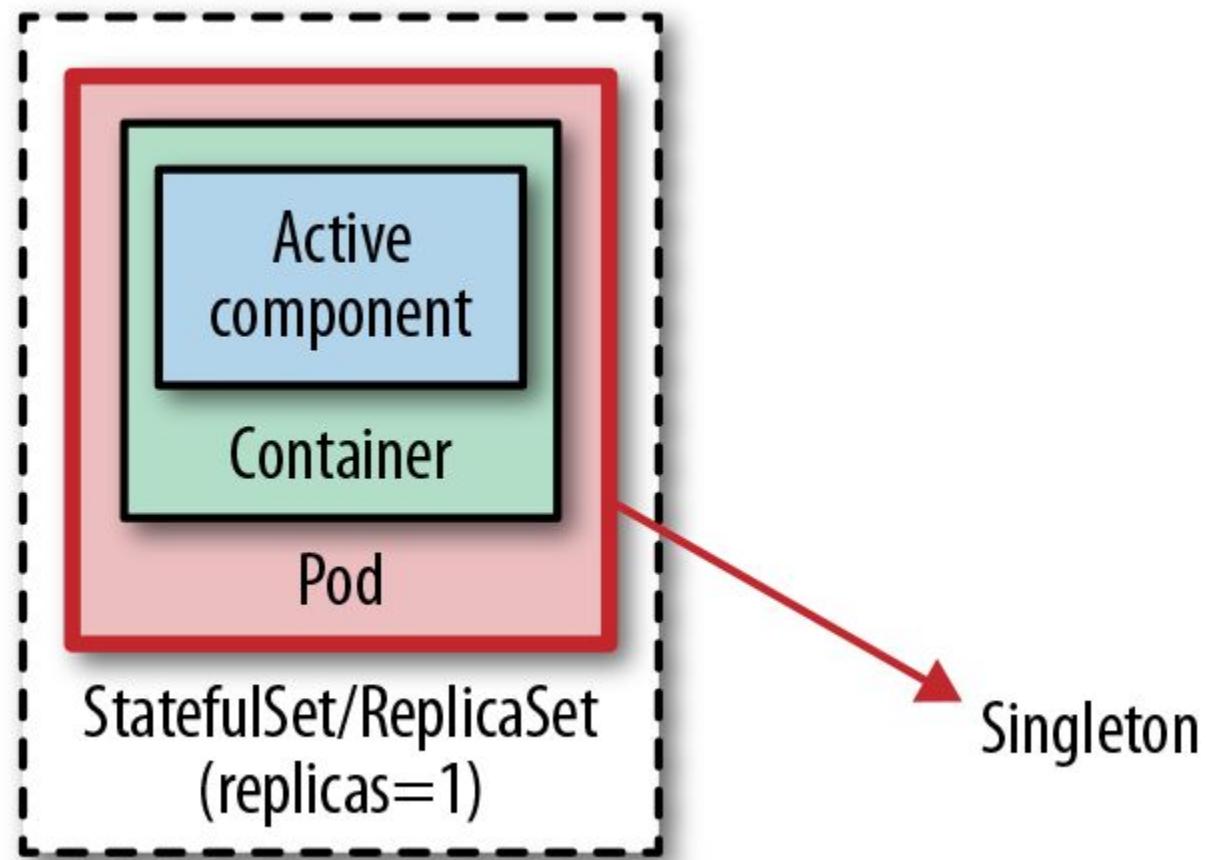


Singleton Service Pattern

Singleton Service

How to ensure that only one application instance is active?

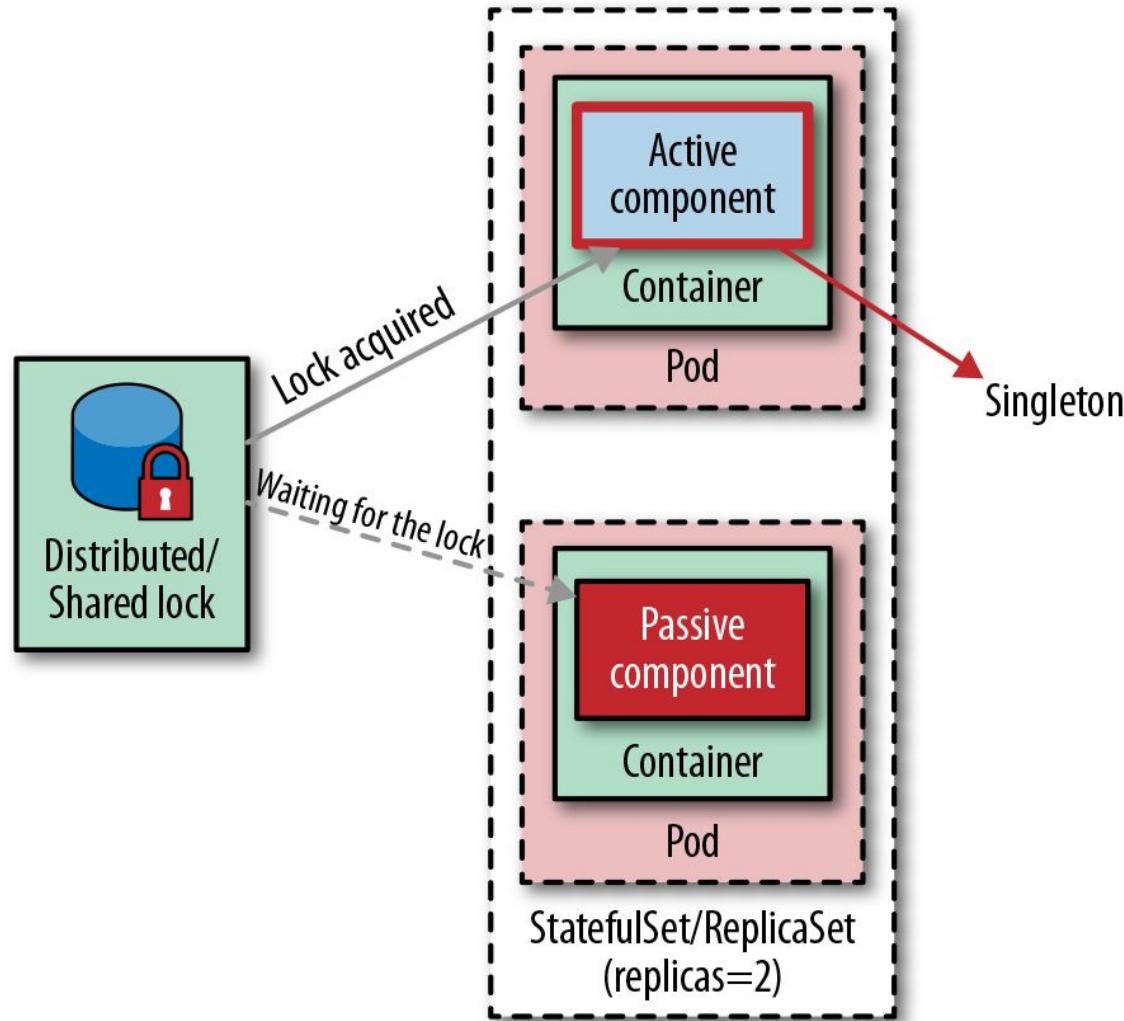
Out-of-Application Locking



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 exist 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

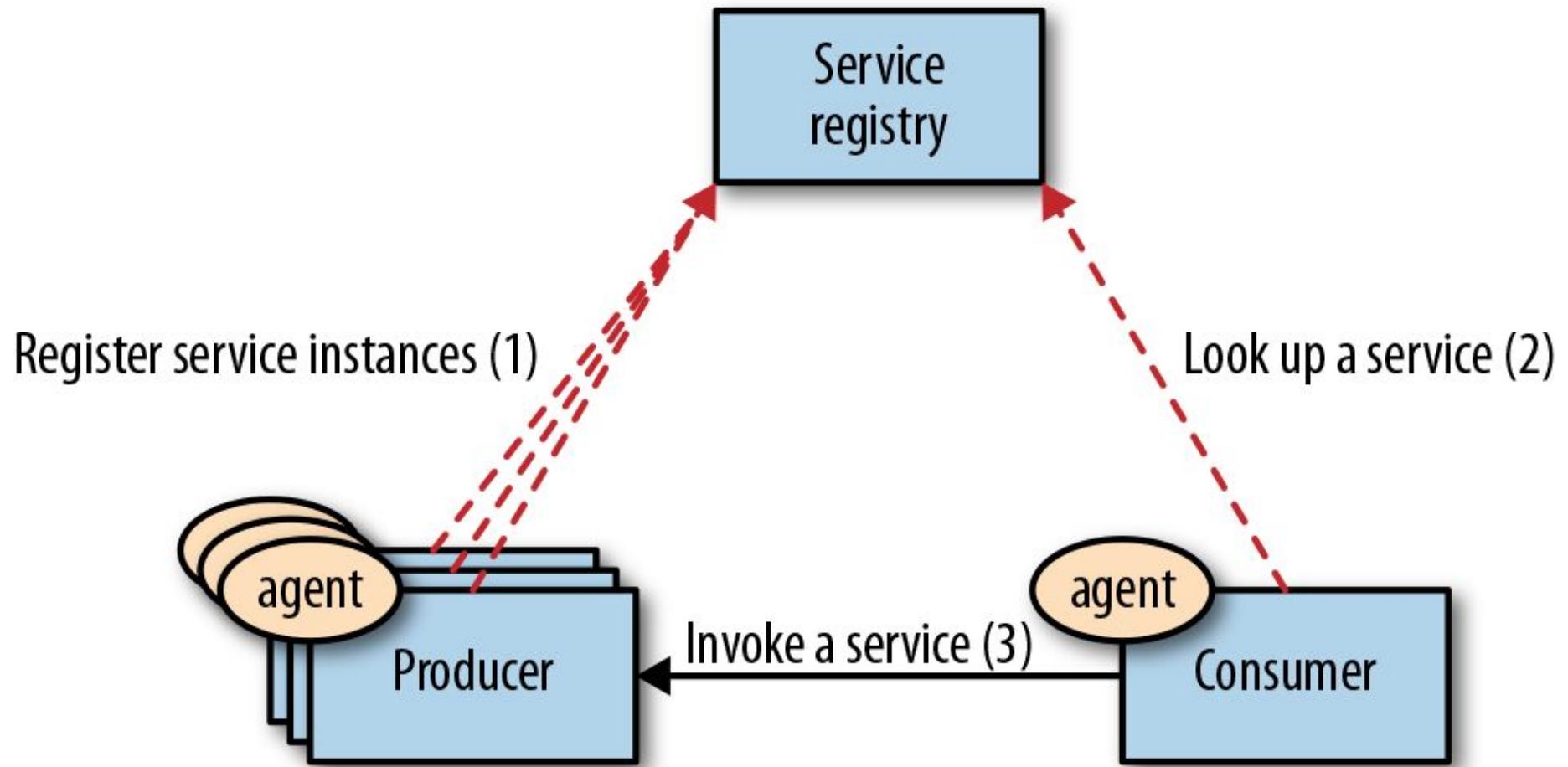
```
apiVersion: policy/v1beta1
kind: PodDisruptionBudget
metadata:
  name: random-generator-pdb
spec:
  selector:
    matchLabels:
      app: random-generator
  minAvailable: 2
```

Service Discovery Pattern

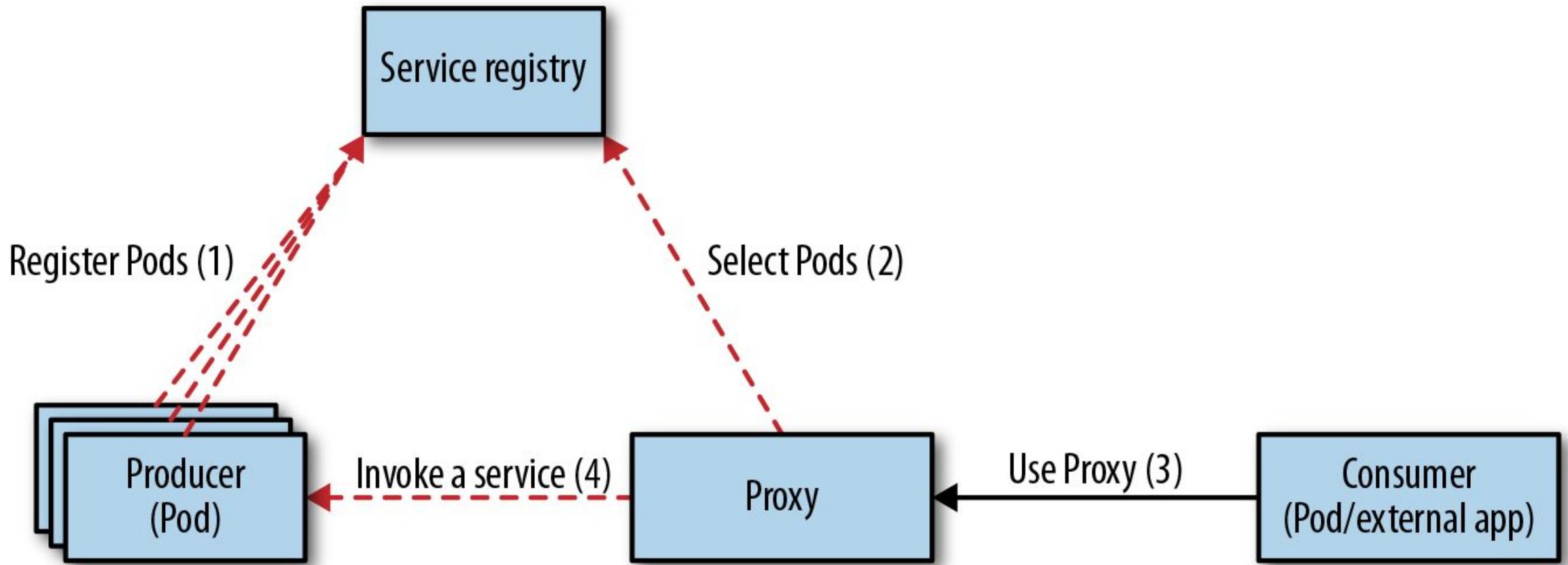
Service Discovery

How to discover and use services?

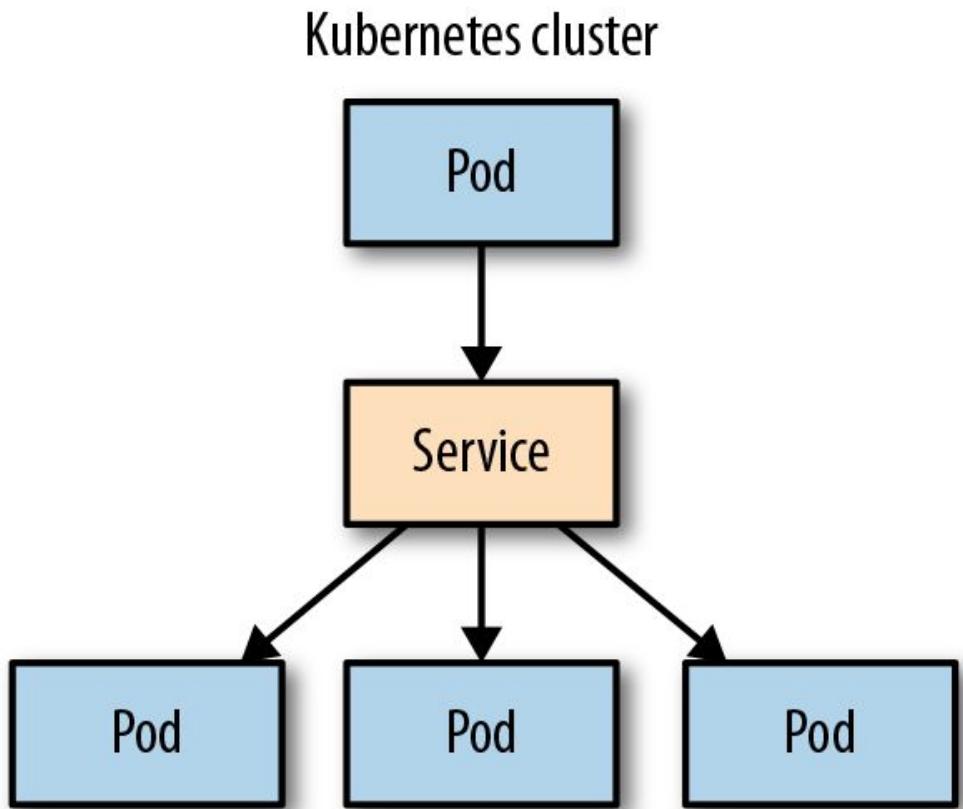
Client-side Service Discovery (non Kubernetes)



Server-side Service Discovery (Kubernetes)

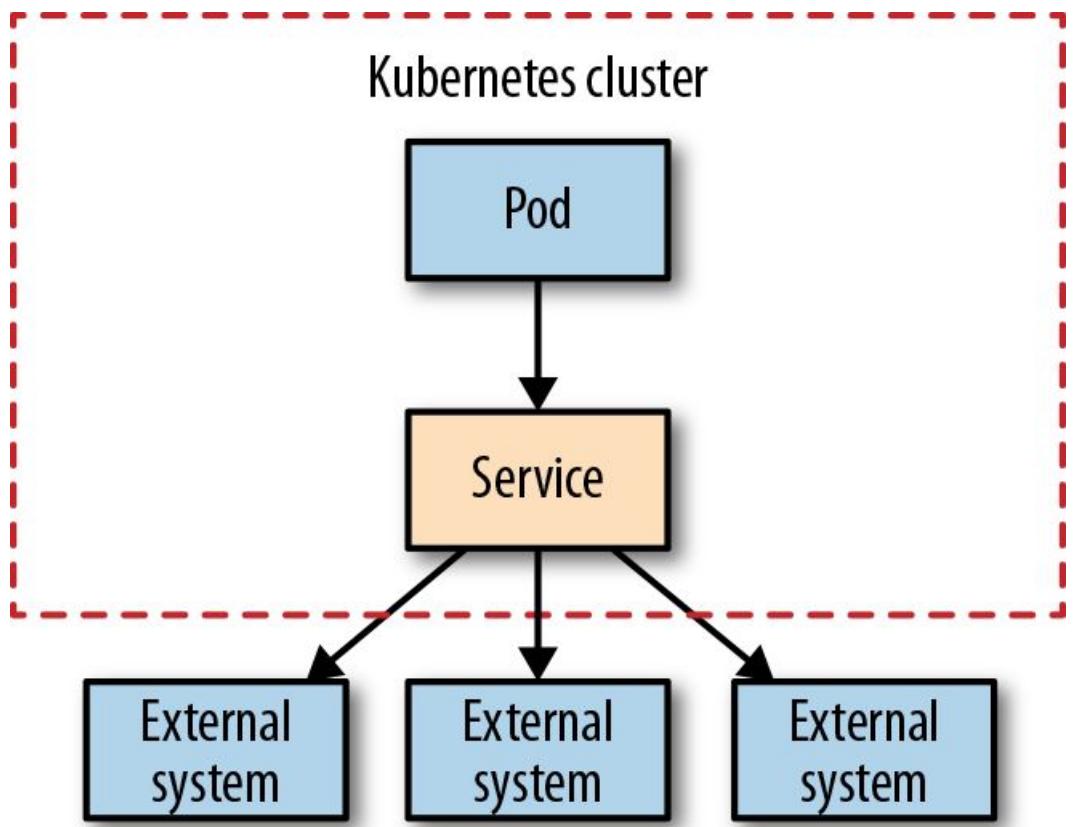


Internal Service Discovery



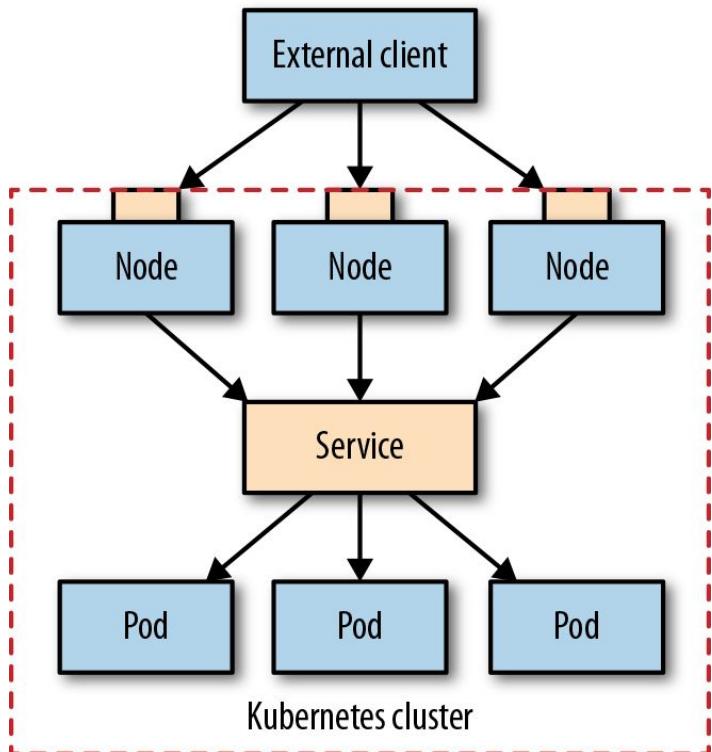
- Discovery through DNS lookups
- Pods picked by label selector
- Multiple ports per Service
- Session affinity on IP address
- Successful readiness probes required for routing
- Virtual IP address for each Service

Manual Service Discovery

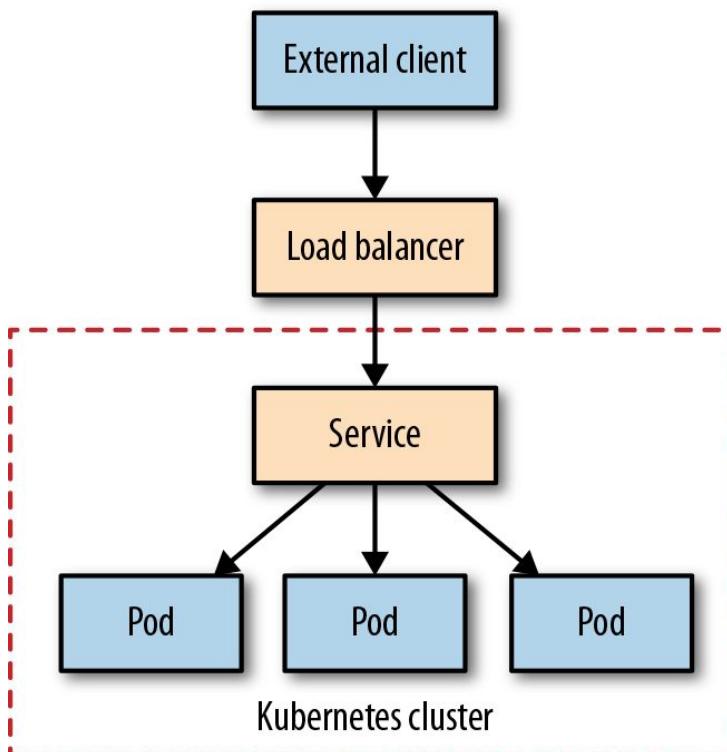


- Service without selector
- Manually creating Endpoint resource with the same name as the Service
- Service of type **ExternalName** map are registered as DNS CNAMEs

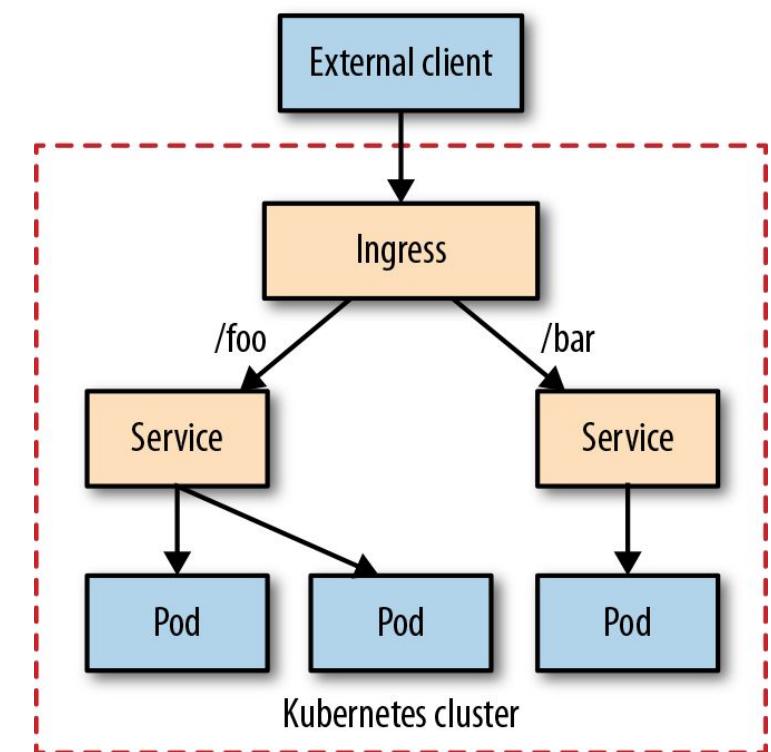
Node Port



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

Controller Pattern

Controller

How to get from the current state to the 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

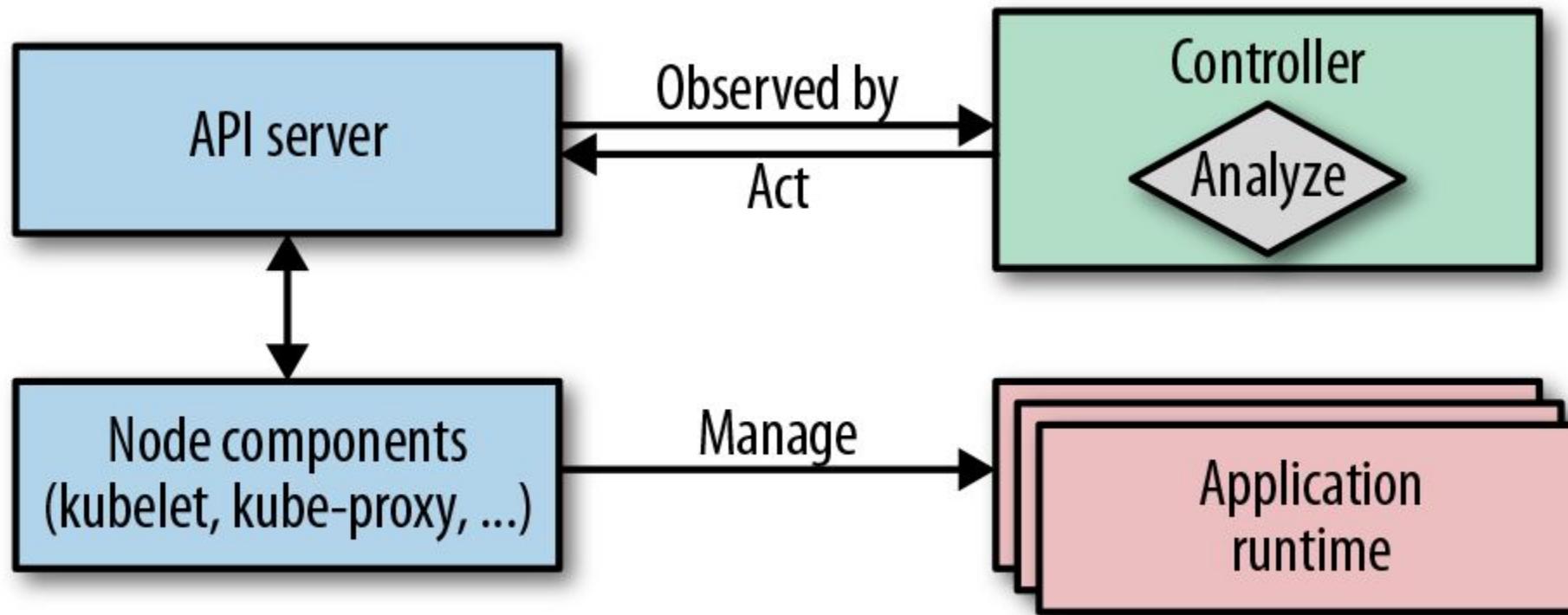


Analyze - Determine difference to target state



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 charset 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
```

Elastic Scale Pattern

Scaling

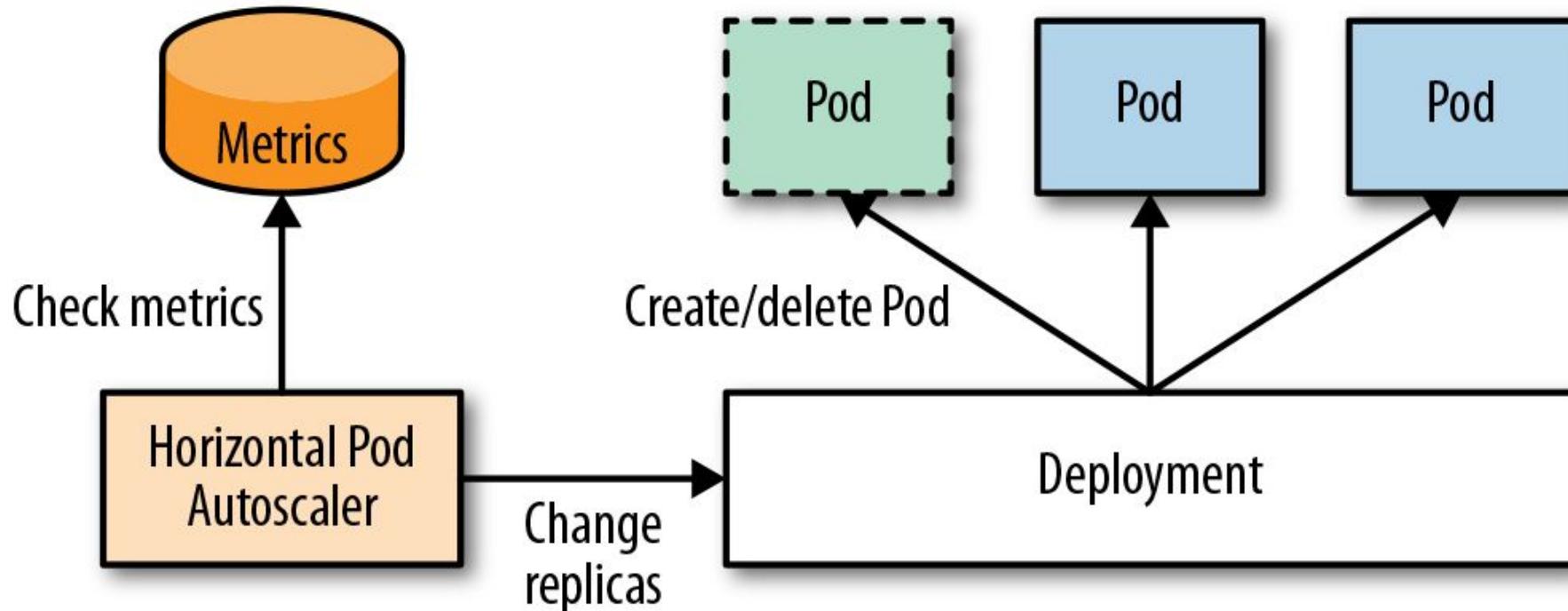
How to automatically react to dynamically changing resource requirements?

- **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

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
```

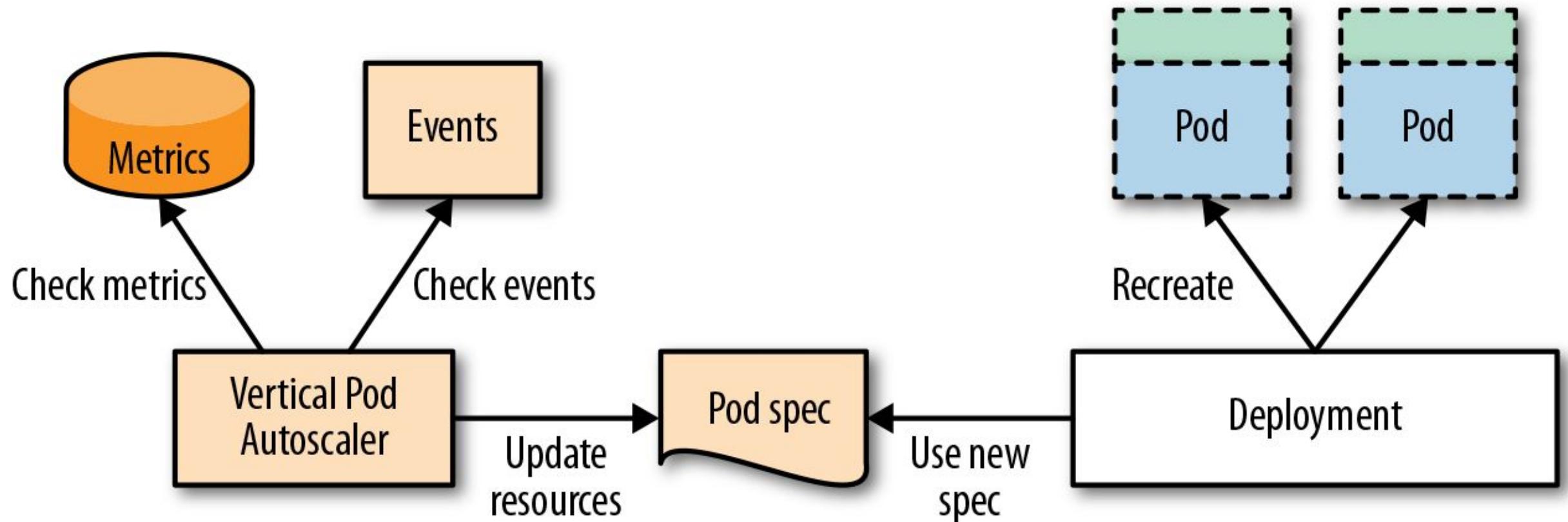
HorizontalPodAutoscaler

```
apiVersion: autoscaling/v2beta2
kind: HorizontalPodAutoscaler
metadata:
  name: random-generator
spec:
  minReplicas: 1
  maxReplicas: 5
  scaleTargetRef:
    apiVersion: extensions/v1beta1
    kind: Deployment
    name: random-generator
  metrics:
  - resource:
      name: cpu
      target:
        averageUtilization: 50
        type: Utilization
    type: Resource
```

HPA: Metrics & Challenges

- Metrics
 - **Standard Metrics** - CPU & Memory Pod data obtained from Kubernetes metrics server
 - **Custom Metrics** - Metrics delivered via an aggregated API server at the custom.metrics.k8s.io API path
 - **External Metrics** - Metrics obtained from outside the cluster
- Challenges
 - **Metric Selection** - Correlation between metric value and replica counts
 - **Preventing Thrashing** - Windowing to avoid scaling on temporary spikes
 - **Delayed Reaction** - Delay between cause and scaling reaction

Vertical Pod Autoscaler (VPA)



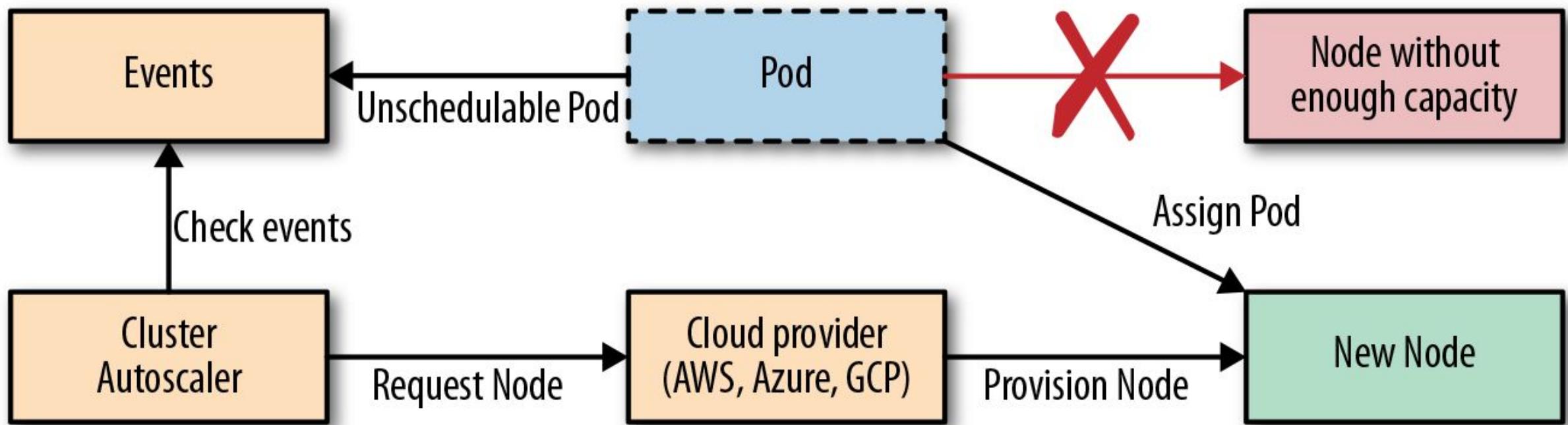
VerticalPodAutoscaler

```
apiVersion: poc.autoscaling.k8s.io/v1alpha1
kind: VerticalPodAutoscaler
metadata:
  name: random-generator-vpa
spec:
  selector:
    matchLabels:
      app: random-generator
  updatePolicy:
    updateMode: "Off"
```

VPA: Update Mode

- updateMode : Off
 - Recommendations are stored in the status : section of the VPA resource
 - No changes to the selected resources are performed
- updateMode : Initial
 - Recommendations are applied during creation of a Pod
 - Influences scheduling decision
- updateMode : Auto
 - Automatically restarts Pods with updated resources based on recommendation

Cluster Autoscaler



Cluster Autoscaler

- Scale-Up
 - Adding a new node if a Pod is marked as *unschedulable* because of scarce resources.
 - Cluster API: Kubernetes API for dynamically managing node groups.
- Scale-Down
 - ... more than half of a nodes capacity is unused
 - ... all movable Pods can be placed on other nodes
 - ... no other reasons to prevent node deletion
 - ... no Pods that can not be moved

Elastic Scale

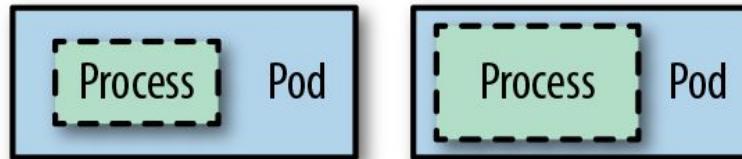
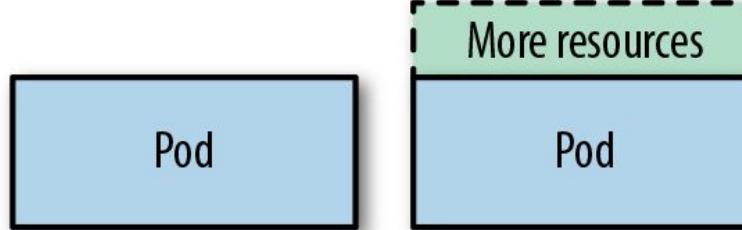
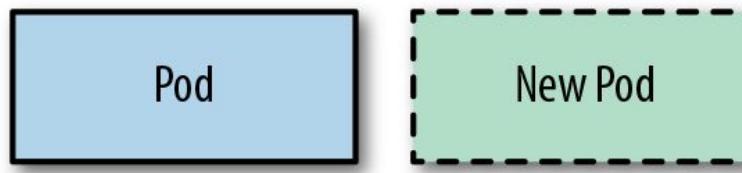
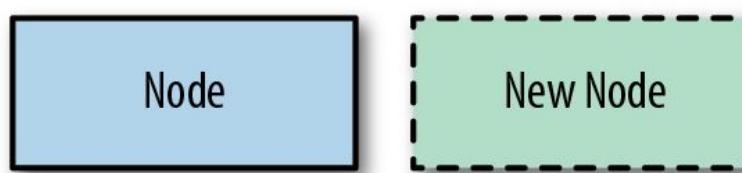
Technique	Action	Scaling Example
Application Tuning	Tune process (threads, heap, etc.)	
Vertical Pod Autoscaler	Increase/reduce container resources	
Horizontal Pod Autoscaler	Add/remove Pods	
Cluster Autoscaler	Add/remove Nodes	

Image Builder Pattern (OpenShift)

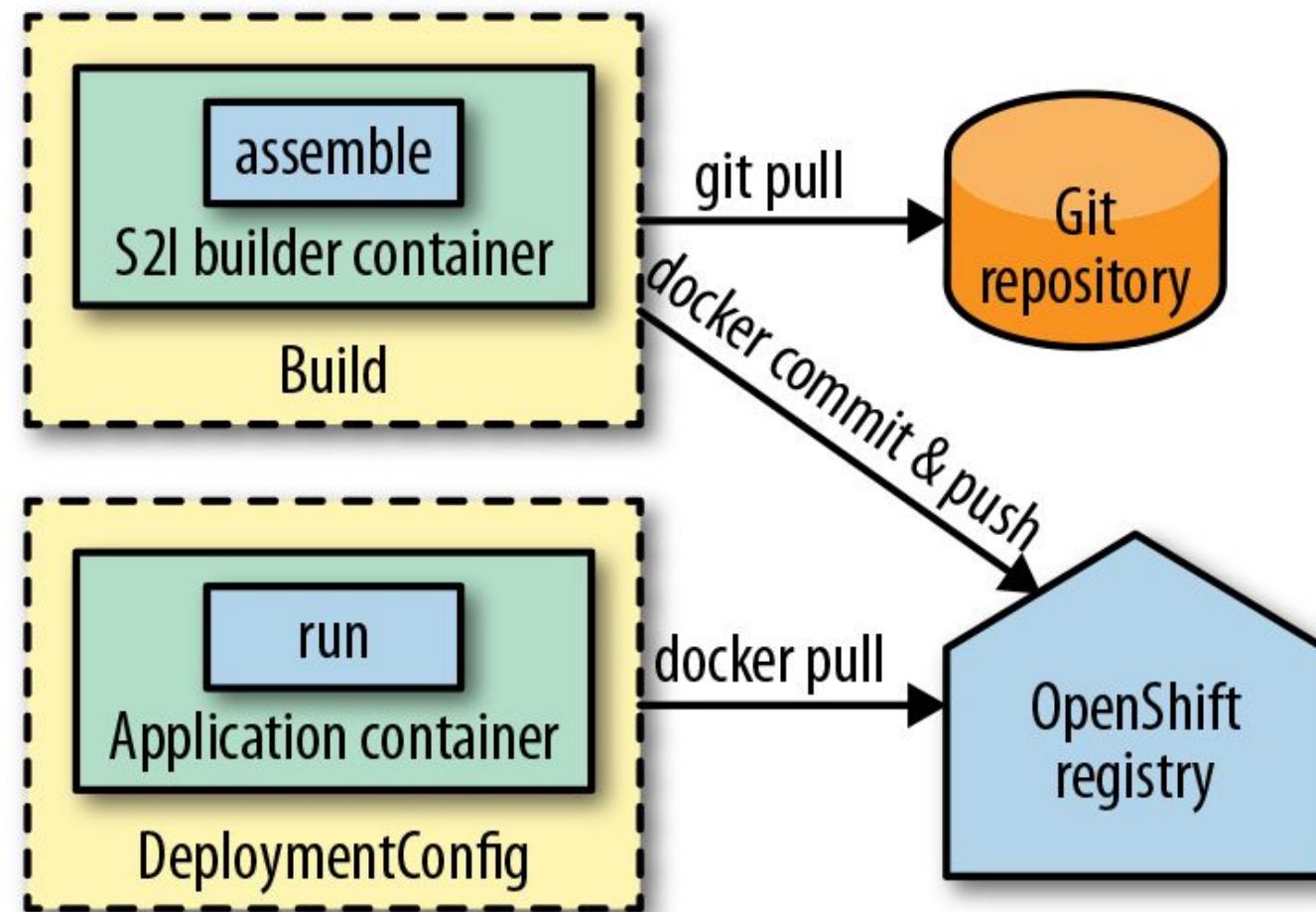
Image Builder

How to build container images within the cluster?

OpenShift Build

- Build types:
 - Source-to-Image (S2I)
 - Docker
 - Pipeline
 - Custom
- Source can be from
 - Git
 - Container Image
 - Secret
 - Binary Input when starting build
- ImageStreams connect build with deployment

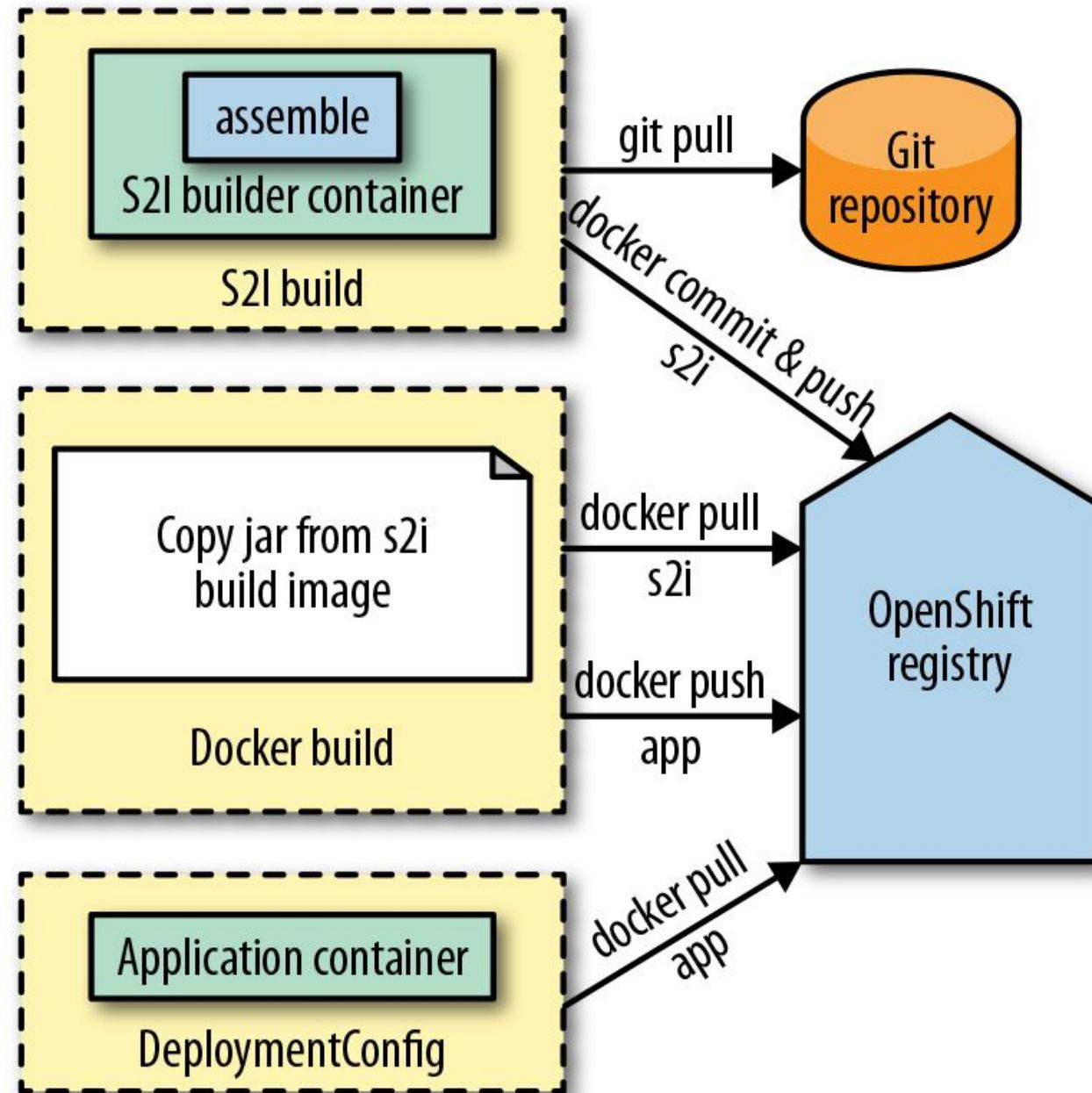
OpenShift S2I Build



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



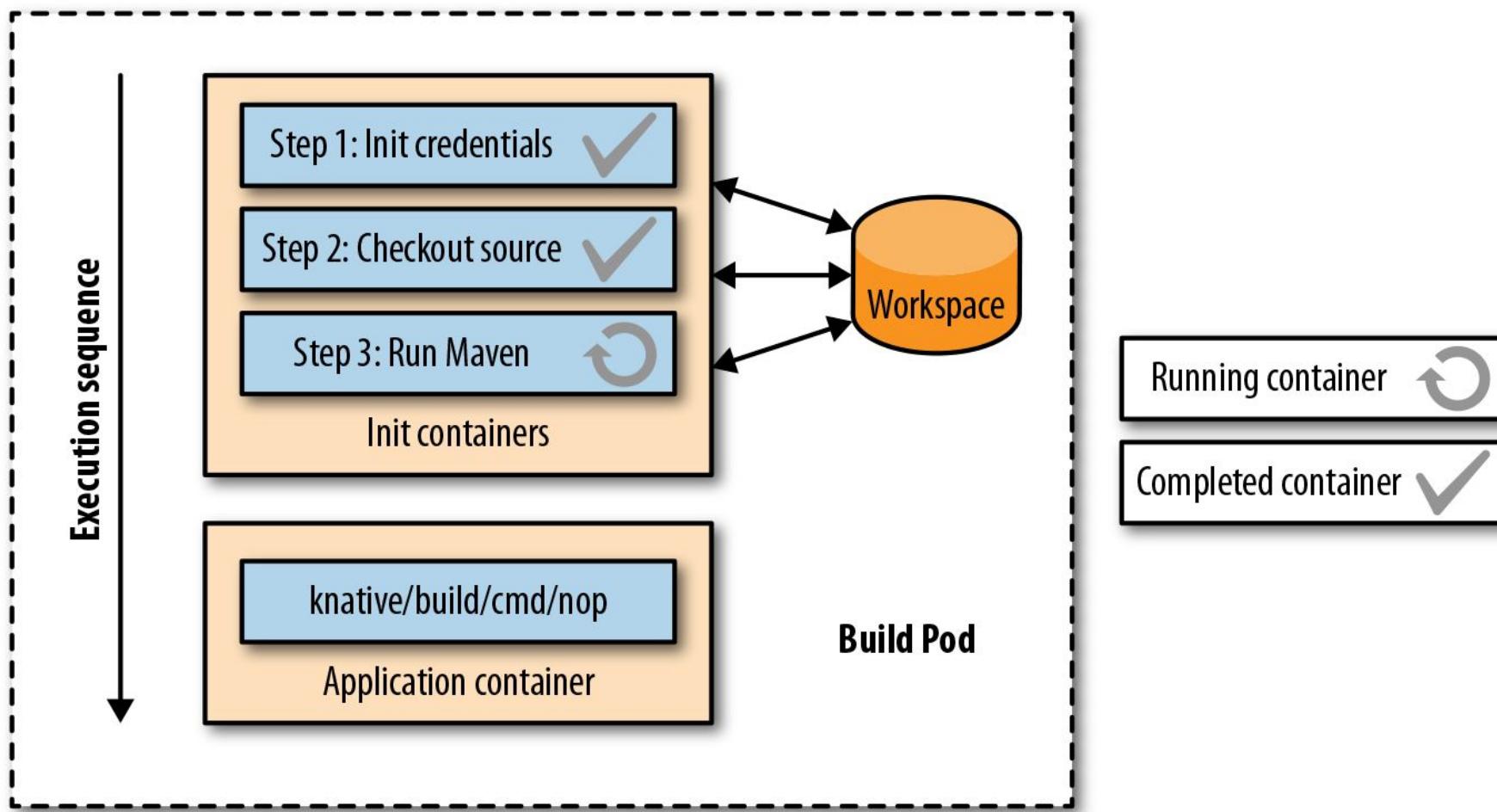
Knative

- **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



Operator Pattern

CustomResourceDefinition

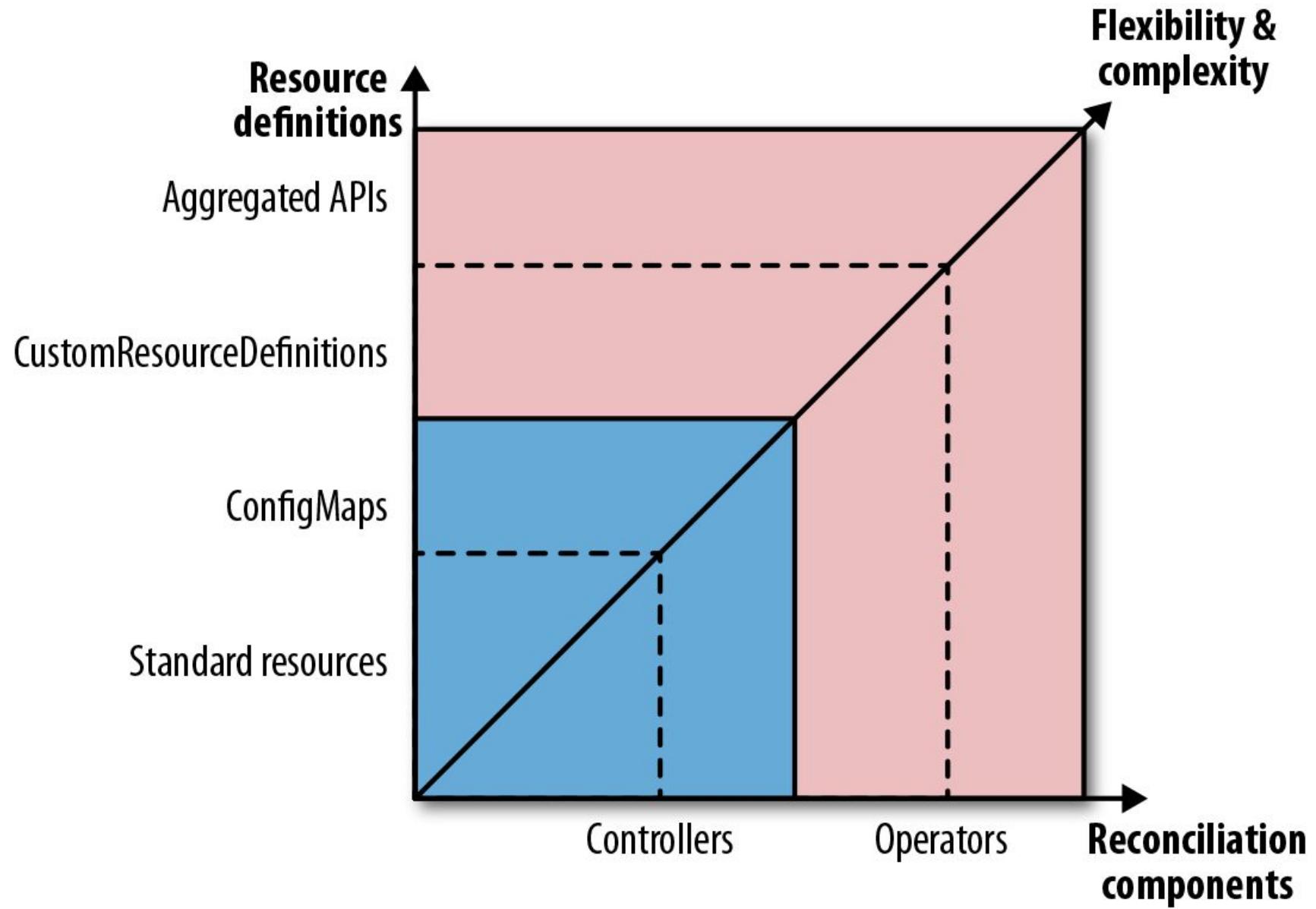
Custom resource is modelling a custom domain and managed through the Kubernetes API

```
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

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



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 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>

Configuration Template

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

```
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: {}
```