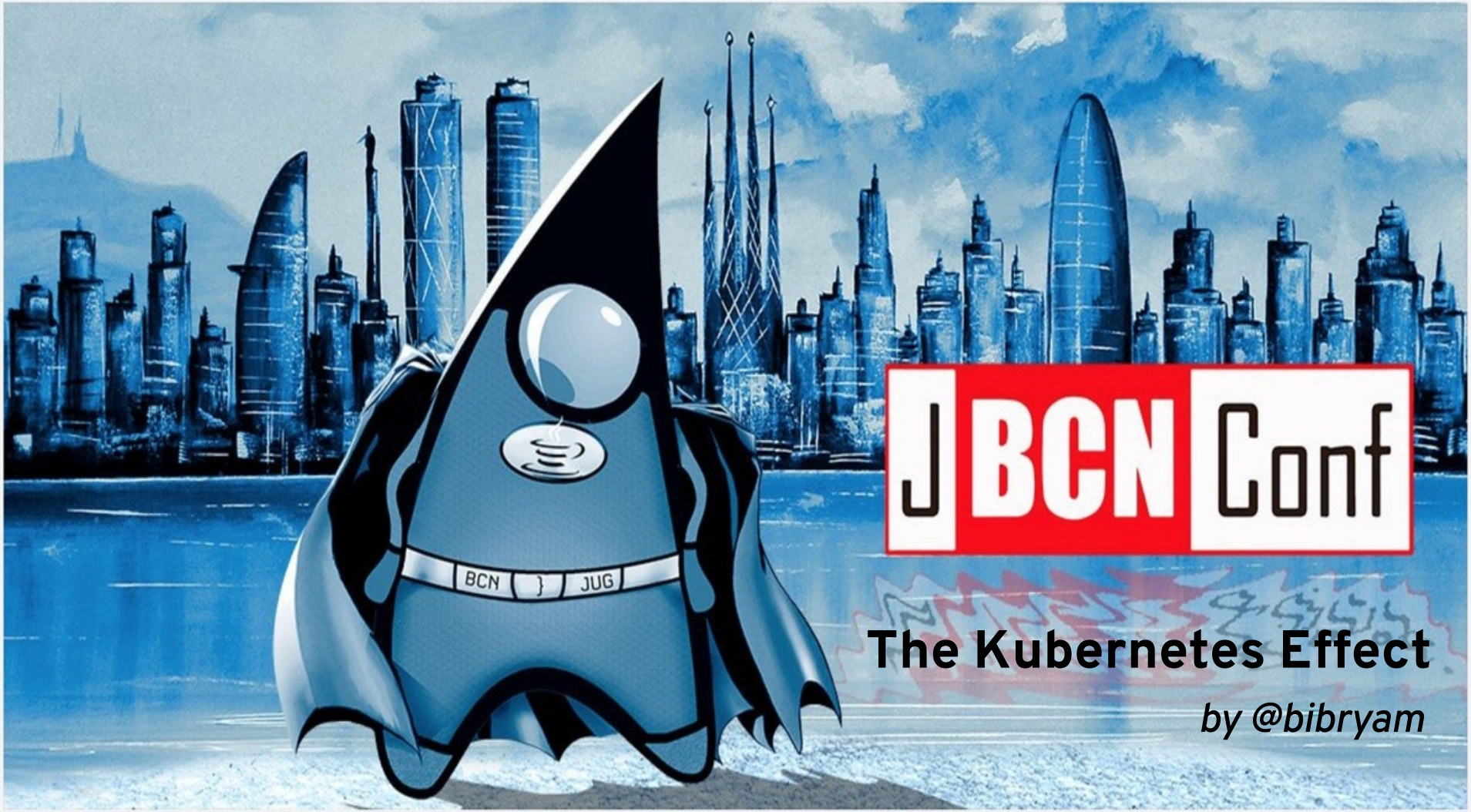
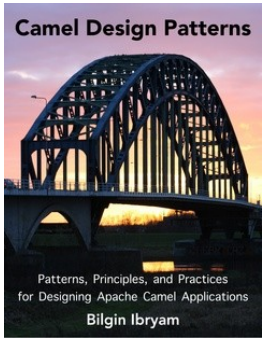
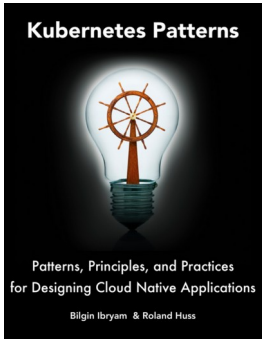

BCN } JUG
JBCN Conf
The Kubernetes Effect
by @bibryam

Who am I?

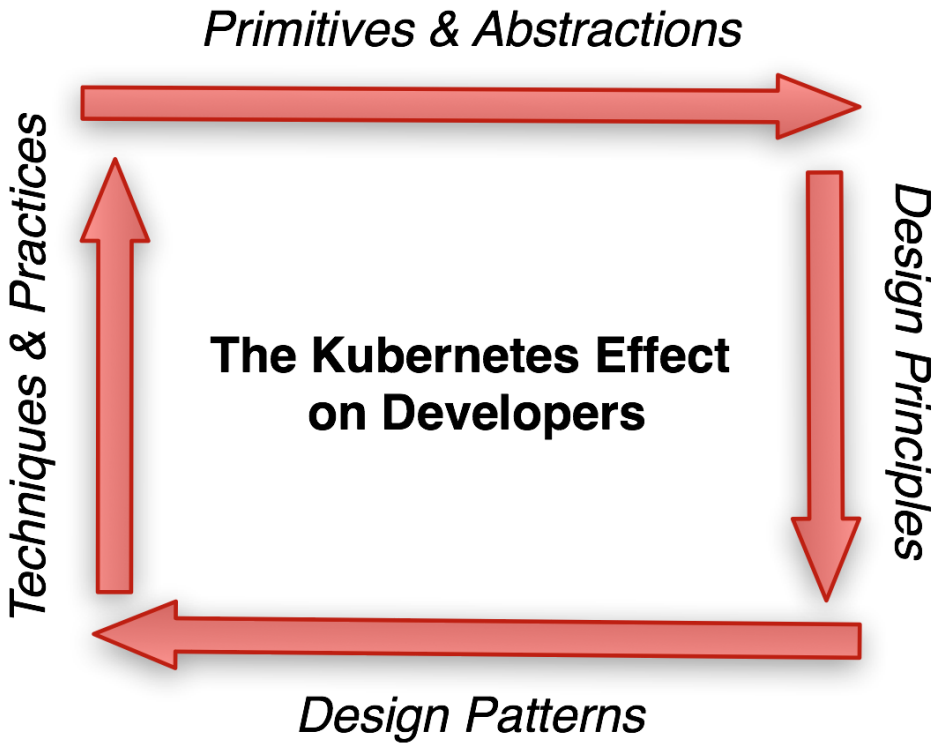
- Bilgin Ibryam
- @bibryam
- http://ofbizian.com/
- http://github.com/bibryam/

- Integration Architect at Red Hat
- Committer for Camel, OFBiz, Isis at ASF
- Author of Camel Design Patterns, Kubernetes Patterns
- Interested in Integration, Cloud Native, Blockchain





Agenda


Primitives & Abstractions
Design Principles
Design Patterns
Techniques & Practices
The Kubernetes Effect
on Developers

Cloud native...

A common definition:

Cloud Native is structuring

teams, **culture** and **technology**

to utilize **automation** and **architectures**

to manage complexity and unlock velocity.

@jbeda

A typical application:

Applications adopting the principles of

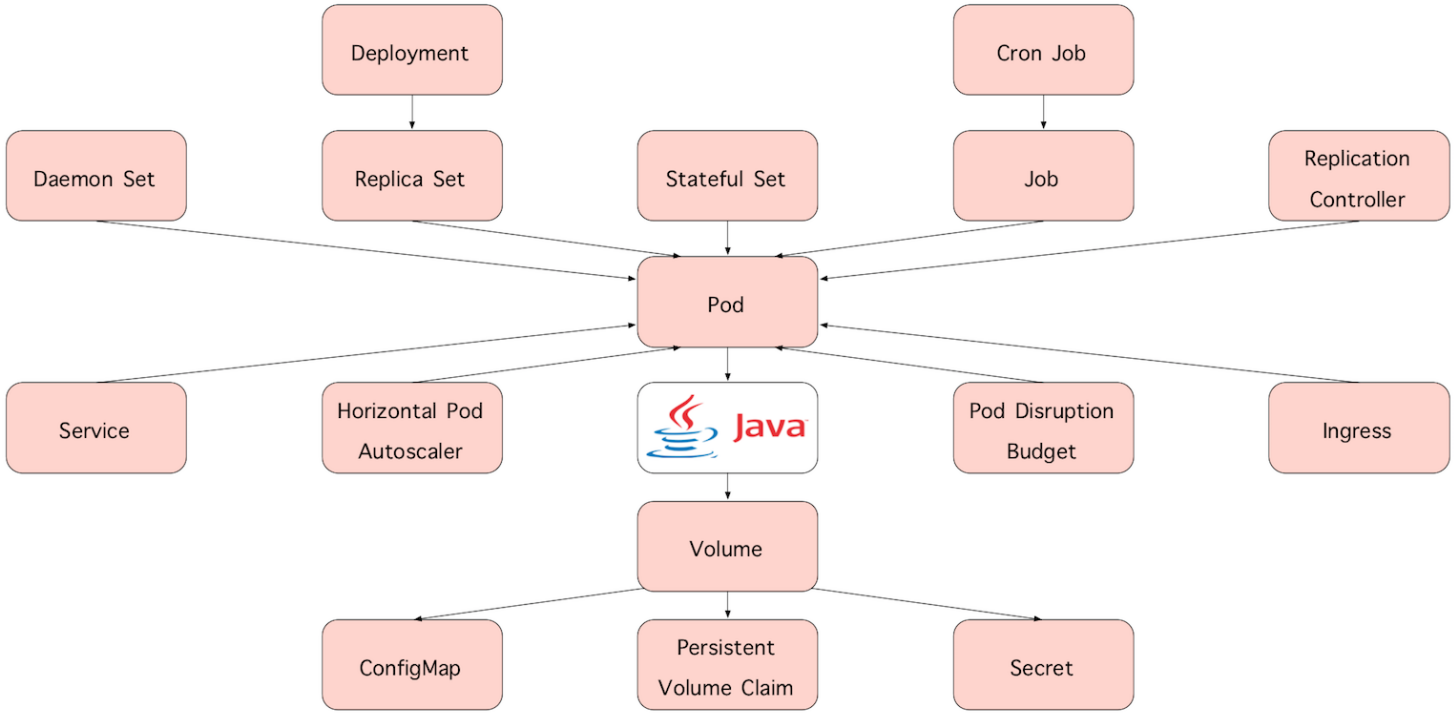
microservices packaged as

containers orchestrated by

platforms running on top of

cloud infrastructure.

A Kubernetes based microservice



Local vs distributed primitives

Concern / Abstraction	Local / Java	Distributed / Kubernetes
Behaviour encapsulation	Class, Object	Container Image, Container
Unit of reuse	.jar	Container Image
Deployment unit	.jar, .war, .ear	Pod
Buildtime/Runtime isolation	Module, Package, Class	Container Image, Namespace
Initialization preconditions	Constructor	Init-container
Post initialization	init-method	PostStart hook
Pre destroy	destroy-method	PreStop hook
Cleanup procedure	finalize(), ShutdownHook	Defer-container***
Asynchronous, Parallel execution	ThreadPoolExecutor, ForkJoinPool	Job
Periodic task	Timer, ScheduledExecutorService	CronJob
Background task	Daemon Thread	DaemonSet
Configuration management	Properties	ConfigMap, Secret
Service discovery	ZooKeeper, Consul...	Service

Distributed abstractions and primitives

- Application packaging (**Container**)
- Deployment unit (**Pod**)
- Auto scaling (**HPA**)
- Atomic work unit (**Job**)
- Recurring execution (**CronJob**)
- Service discovery (**Service**)
- Load balancing (**Service**)
- Application placement (**Scheduler**)
- Lifecycle management (**Deployment**)
- Health checks (**liveness/readiness**)
- Lifecycle hooks (**PostStart/PreStop**)
- Artifact grouping (**Label**)
- Custom Resource Definition (**CRD**)

Software design principles

Principles represent abstract guidelines or believes that help create systems with higher-quality attributes.

- **KISS** - Keep it simple, stupid
- **DRY** - Don't repeat yourself
- **YAGNI** - You aren't gonna need it
- **SoC** - Separation of concerns
- **SOLID** Principles by Robert C. Martin:
  - **S**ingle responsibility
  - **O**pen/Closed
  - **L**iskov substitution
  - **I**nterface segregation
  - **D**ependency inversion

The twelve-factor app i.e. the Heroku way

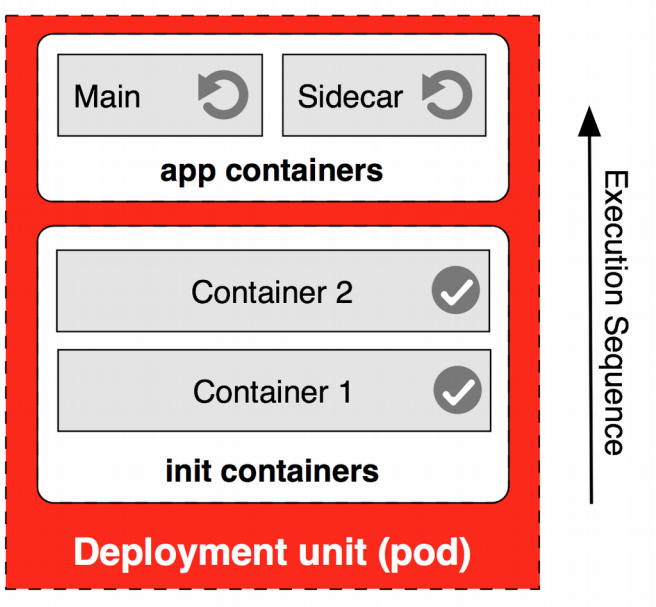
1. One codebase tracked in revision control, many deploys
2. Explicitly declare and isolate dependencies
3. Store config in the environment
4. Treat backing services as attached resources
5. Strictly separate build and run stages
6. Execute the app as one or more stateless processes
7. Export services via port binding
8. Scale out via the process model
9. Maximize robustness with fast startup and graceful shutdown
10. Keep development, staging, and production as similar as possible
11. Treat logs as event streams
12. Run admin/management tasks as one-off processes

Principles of container-based application design

- Build time:
  - Single Concern Principle (SCP)
  - Self-Containment Principle (S-CP)
  - Image Immutability Principle (IIP)
- Runtime:
  - High Observability Principle (HOP)
  - Lifecycle Conformance Principle (LCP)
  - Process Disposability Principle (PDP)
  - Runtime Confinement Principle (RCP)

https://www.redhat.com/en/resources/cloud-native-container-design-whitepaper

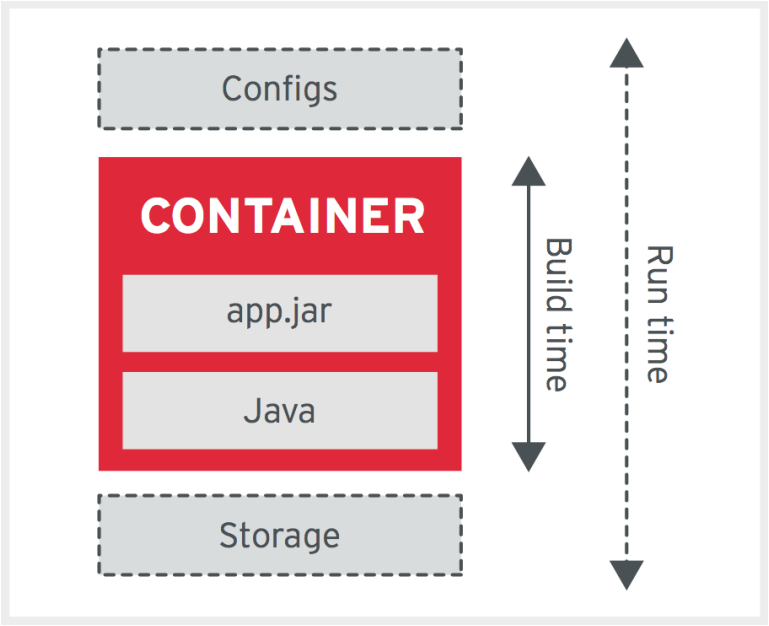
Single concern principle



Design patterns:

- Sidecar
- Ambassador
- Adapter
- Init-container
- Defer-container

Self-containment principle



Anti-patterns:

- Locomotive

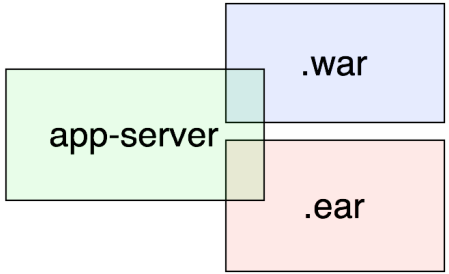
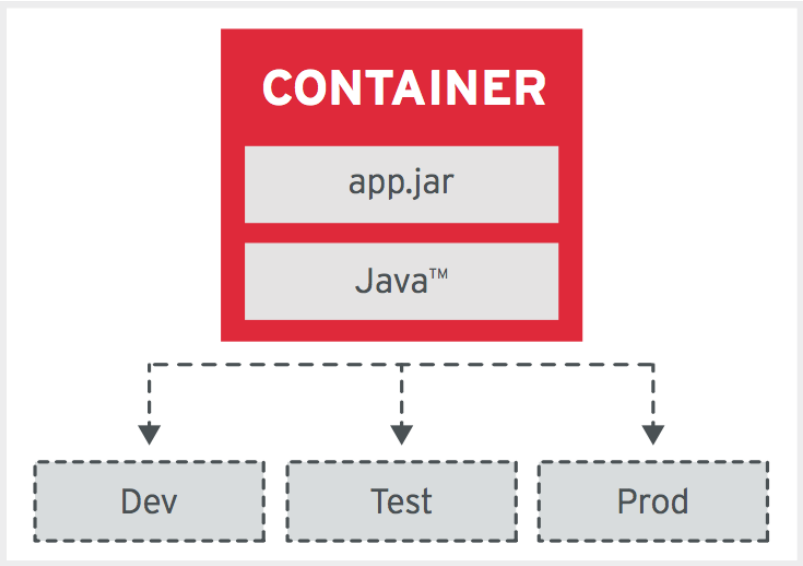


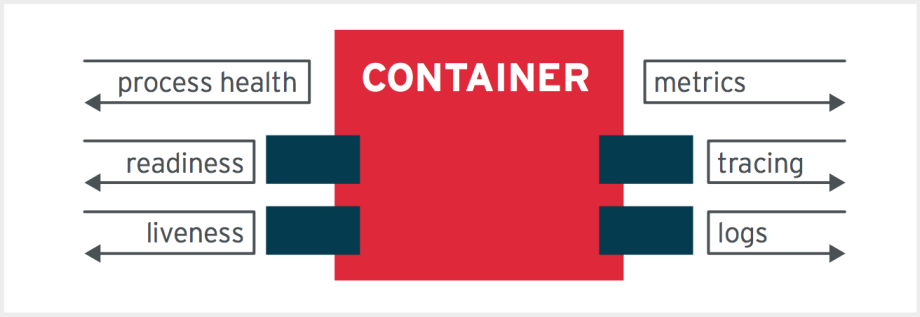
Image immutability principle



Also known as:

- Dev/Prod parity
- Snowflakes vs Phoenix
- Impedance mismatch

High observability principle



Healthcheck implementations:

- Spring Boot Actuator
- Dropwizard Metrics
- WildFly Swarm Monitor
- MicroProfile Healthchecks
- Apache Camel
- And many others!

Lifecycle conformance principle

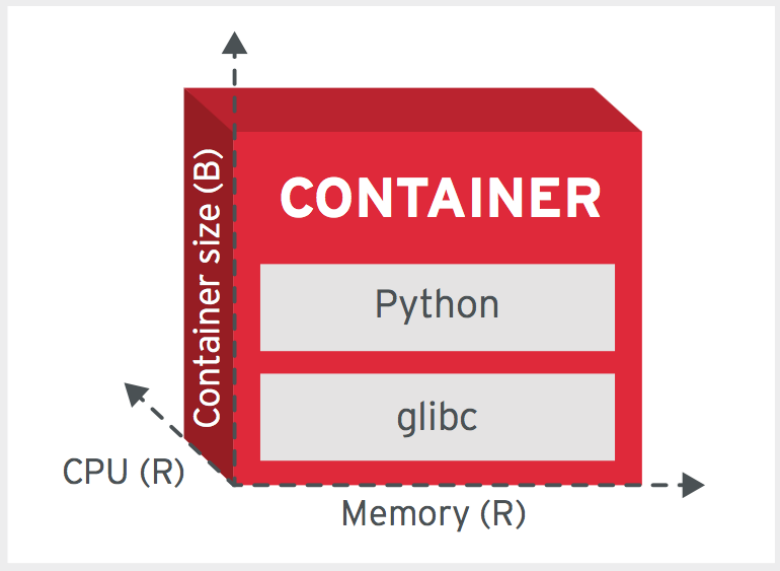
- Graceful shutdown:
  - SIGTERM
  - SIGKILL
- Lifecycle hooks:
  - PreStop
  - PostStart

Process disposability principle

Also known as:

- Cattle rather than pets
  - Don’t rely on a particular instance.
  - Be aware of shots at your cattle.
  - Be robust against sudden death.
- Stateless or with replicated state
- Idempotent startup
- Graceful shutdown

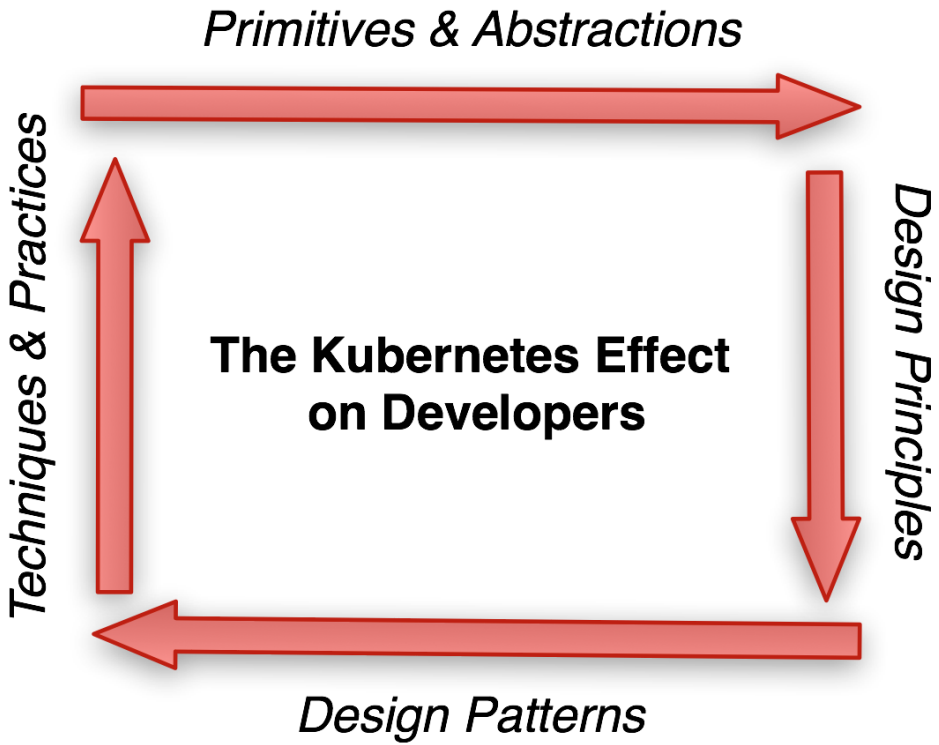
Runtime confinement principle



Implications:

- Pod scheduling
- Pod auto scaling
- Pod eviction
- Pod QoS classes:
  - Best Effort
  - Burstable
  - Guaranteed
- Capacity management

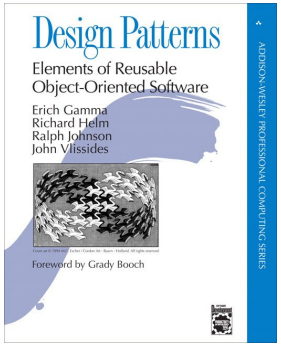
Agenda


Primitives & Abstractions
Design Principles
Design Patterns
Techniques & Practices
The Kubernetes Effect
on Developers

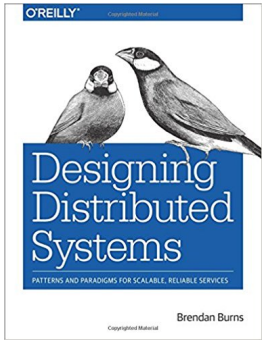
Software design patterns

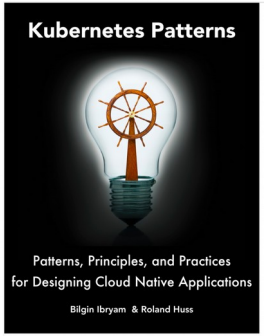
A reusable solution to a **reoccurring** problem within a given **context.**

Object-oriented patterns

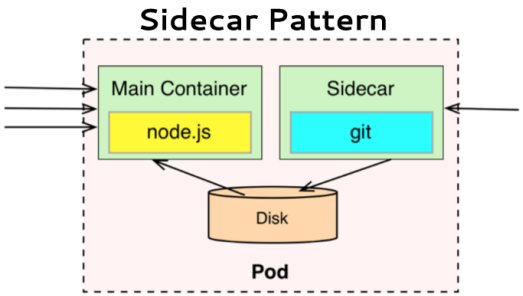


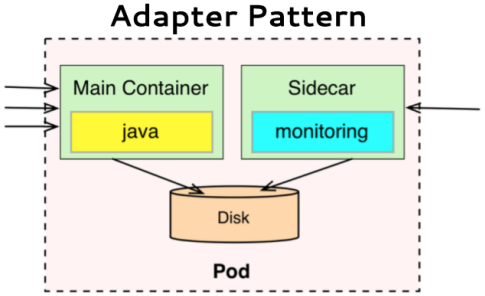
Container orchestration patterns

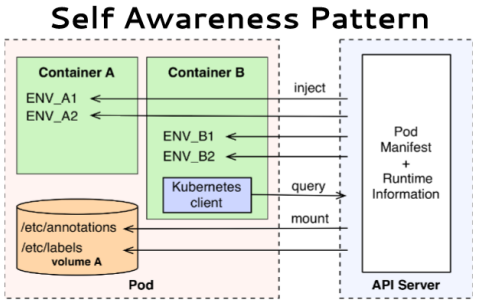


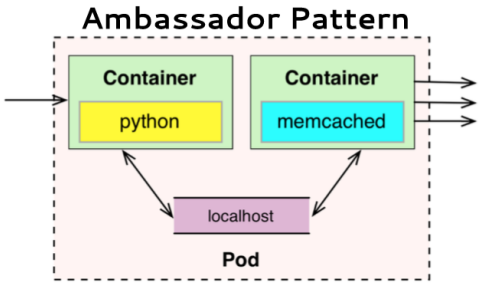


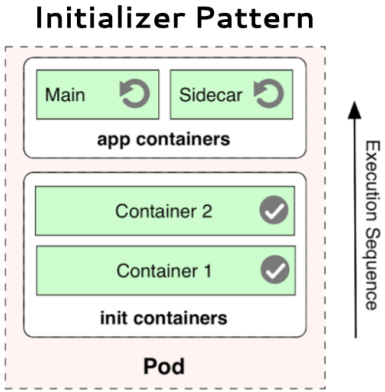
Container design patterns

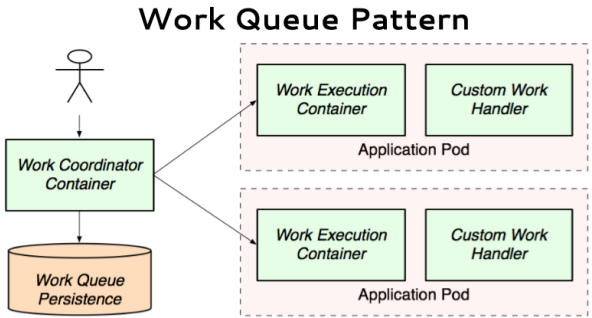




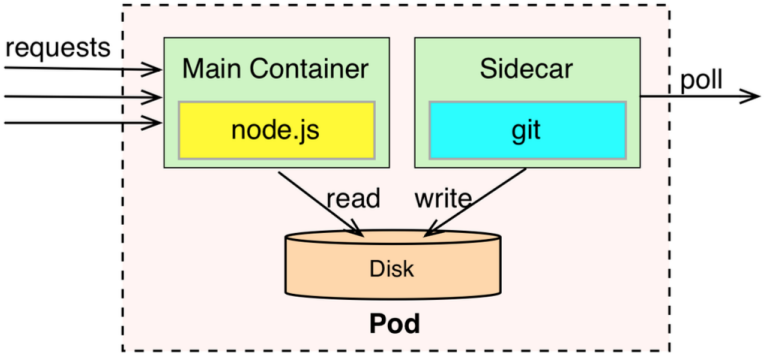








Sidecar pattern

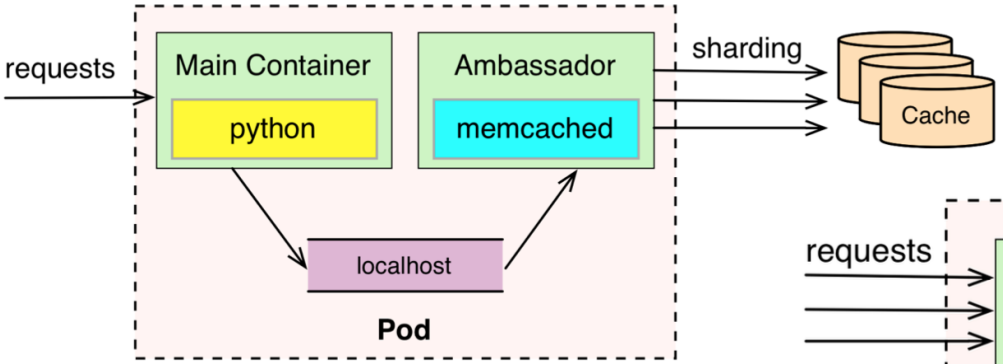


- Use cases
  - Monitoring, health checks, watchdog
  - Logging
  - Configuration
  - Networking
  - Offload proxy

```yaml
- apiVersion: v1
  kind: Pod
  metadata:
    name: web-app
  spec:
    containers:
    # Main container is a stock httpd serving from /var/www/html
    - name: app
      image: centos/httpd
      ports:
        - containerPort: 80
      volumeMounts:
      - mountPath: /var/www/html
        name: git
    # Sidecar cloning a given git repository every 10 minutes
    - name: poll
      image: axeclbr/git
      volumeMounts:
      - mountPath: /var/lib/data
        name: git
      env:
        - name: GIT_REPO
          value: https://github.com/mdn/beginner-html-site-scripted
      command: ['sh', '-c', 'git clone $(GIT_REPO) . && watch -n 60 git pull']
      workingDir: /var/lib/data
    # The shared directory for holding the files
    volumes:
    - emptyDir: {}
      name: git
```

Specialized Sidecars

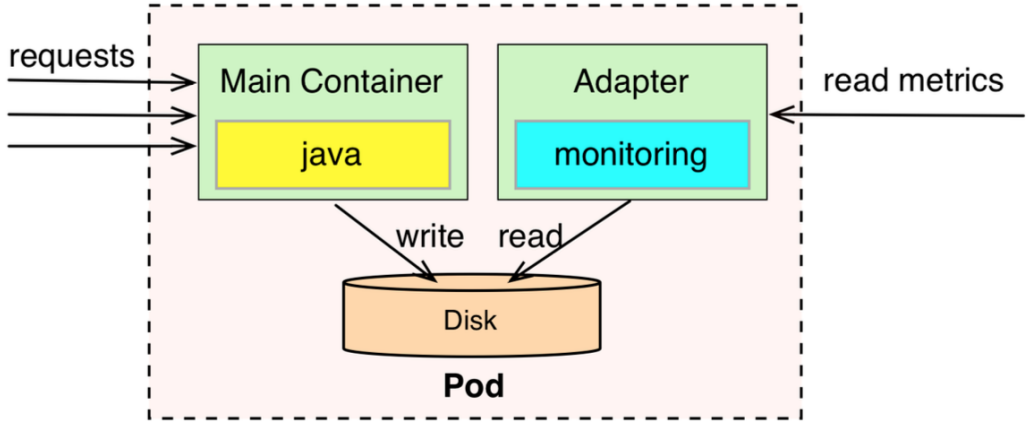
Ambassador Pattern



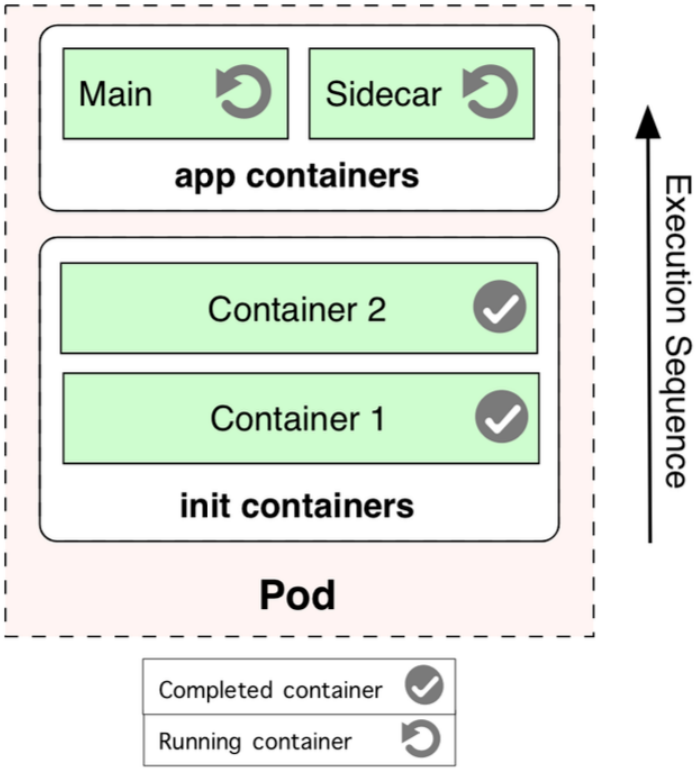
- Use cases
  - Out-of-process proxy
  - Resiliency
  - Monitoring
  - Security

- Use cases
  - Reverse proxy to a heterogeneous system
  - Log normalizer
  - Metrics exporter

Proxy Pattern



Init-container pattern



- Use cases
  - Wait for external dependency
  - Configuration
  - Initialization

```yaml
- apiVersion: v1
  kind: Pod
  metadata:
    name: web-app
  spec:
    containers:
    # Main container is a stock httpd serving from /var/www/html
    - name: app
      image: centos/httpd
      ports:
        - containerPort: 80
      volumeMounts:
      - mountPath: /var/www/html
        name: git
    initContainers:
    # Init container cloning a given git repository at startup
    - name: download
      image: axeclbr/git
      env:
       - name: GIT_REPO
         value: https://github.com/mdn/beginner-html-site-scripted
      command: ['sh', '-c', 'git clone $(GIT_REPO) /var/lib/data']
      volumeMounts:
      - mountPath: /var/lib/data
        name: git
    # The shared directory for holding the files
    volumes:
    - emptyDir: {}
      name: git
```

More Kubernetes Patterns

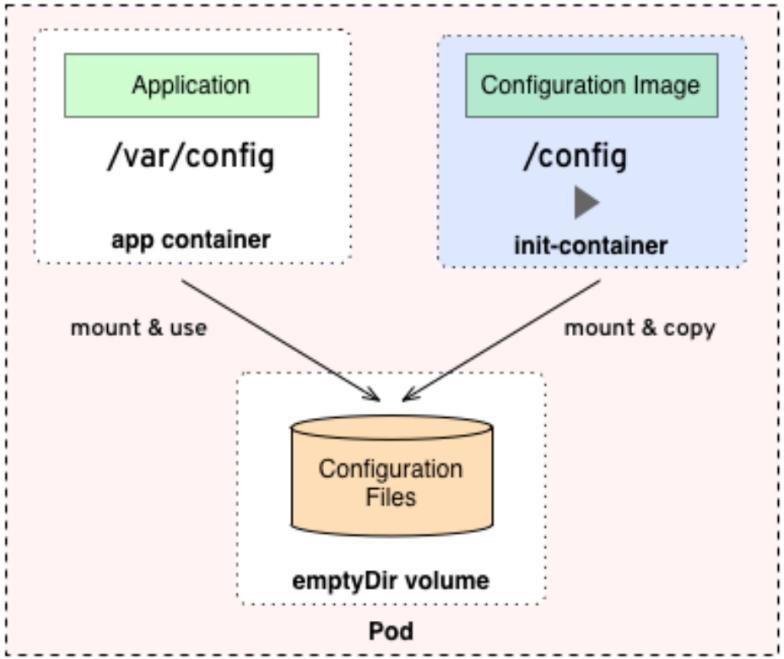
Configuration Patterns

- EnvVar Configuration
- Configuration Resource
- Configuration Template
- Immutable Configuration

Advanced Patterns

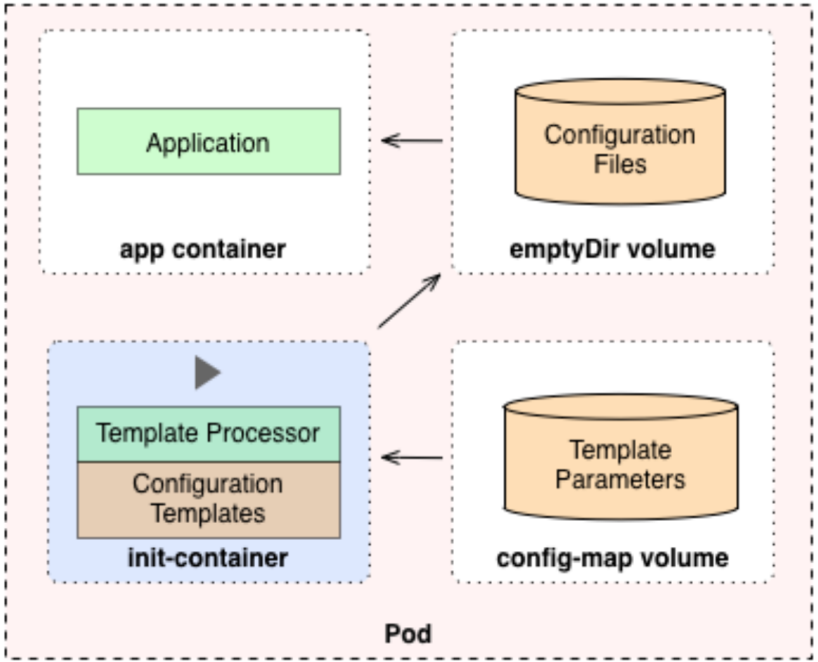
- Stateful Service
- Custom Resource Descriptors
- Custom Controller
- Build Container

Immutable configuration pattern



- Use cases
  - Immutable configuration data
  - Large configuration files

Configuration template pattern



- Use cases
  - Complex templating logic

Custom controller/Operator patterns

```
for {
  desired := getDesiredState()
  current := getCurrentState()
  makeChanges(desired, current)
}
```

Extension controllers

- Expose controller (fabric8)
- Configmap controller (fabric8)
- Linux Update Operator

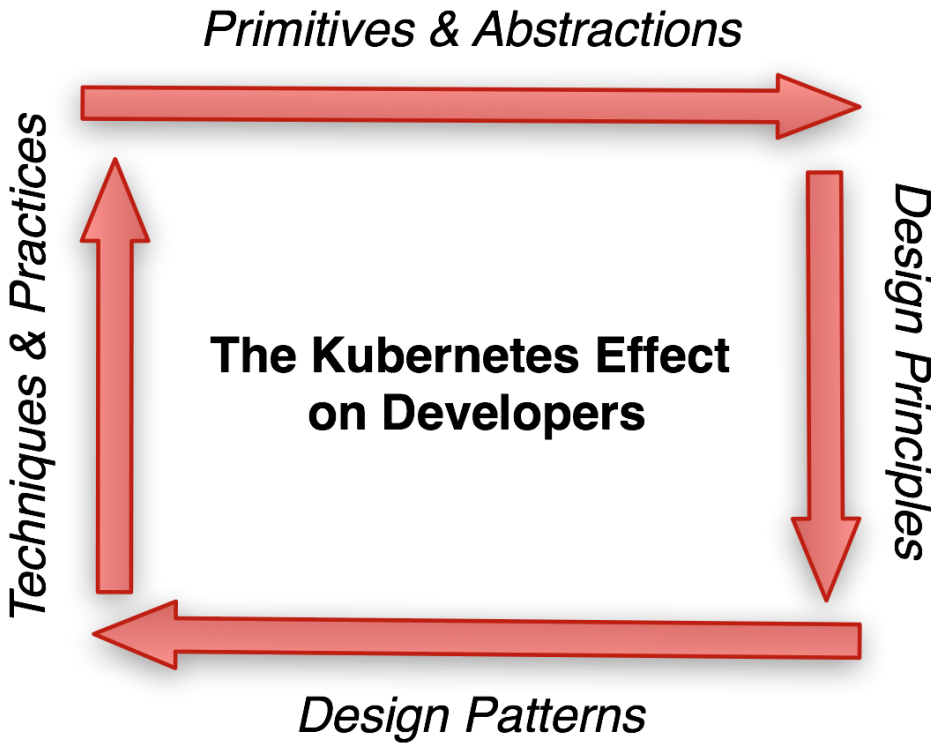
Application Controller

- Prometheus operator
- Zookeeper operator
- Infinspan operator
- Strimzi - Kafka operator

Techniques and practices

- **Aim for small images** - this reduces container size, improves build, and deployment time.
- **Support arbitrary user IDs** - avoid using the sudo command or requiring a specific user ID.
- **Mark important ports** - declare ports using the EXPOSE command.
- **Use volumes for persistent data** - the data that needs to be preserved after a container is destroyed.
- **Set image metadata** - Image metadata in the form of tags, labels, and annotations.
- **Synchronize host and image** - attributes such as time and machine ID.
- **Log to STDOUT and STDERR** - to ensure container logs are picked up and aggregated properly.

Agenda


Primitives & Abstractions
Design Principles
Design Patterns
Techniques & Practices
The Kubernetes Effect
on Developers

Kubernetes resources

- **The Kubernetes Effect (blog post)**
  https://www.infoq.com/articles/kubernetes-effect

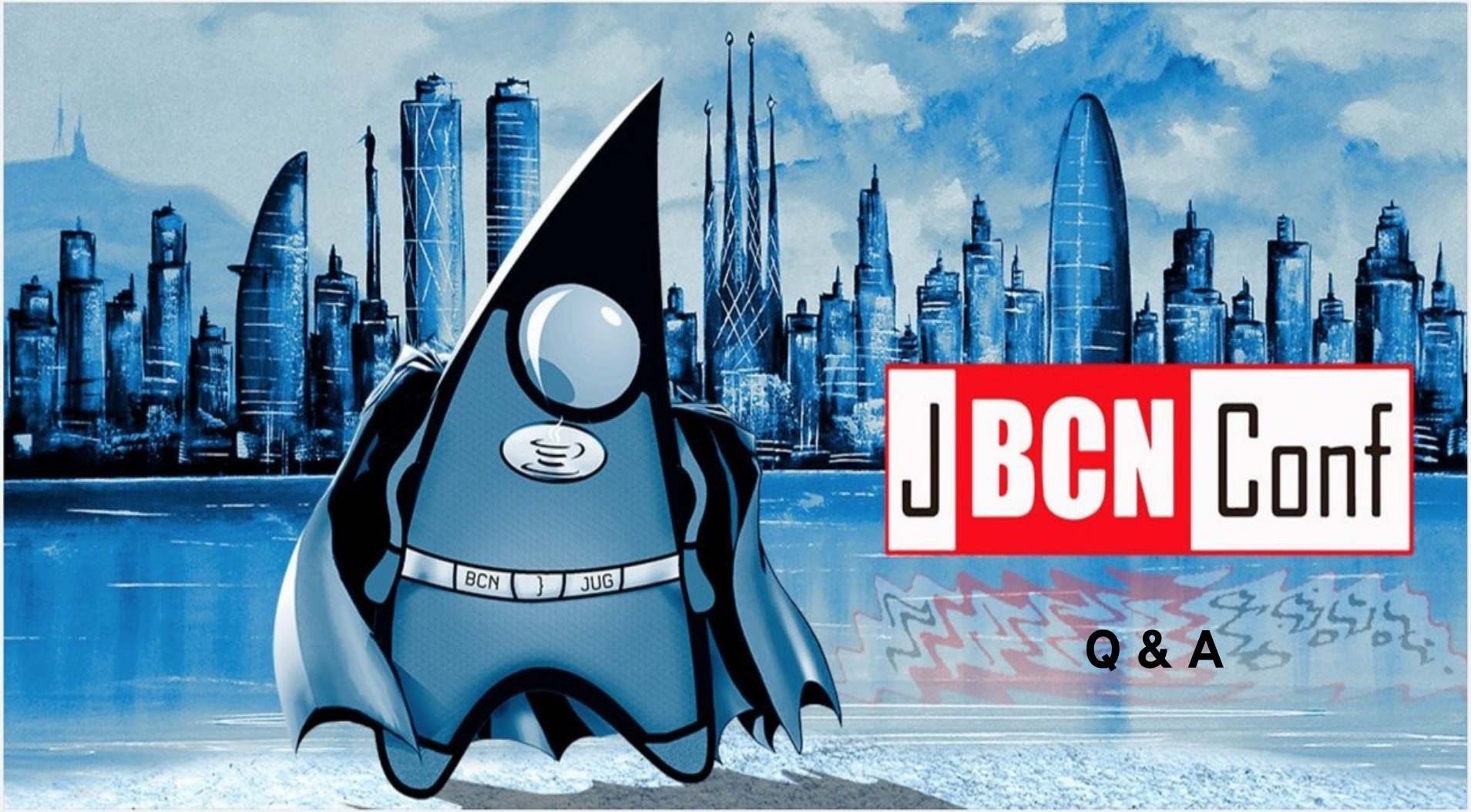
- **Principles of container-based application design (white paper)**
  https://www.redhat.com/en/resources/cloud-native-container-design-whitepaper

- **Design patterns for container-based distributed systems (white paper)**
  https://www.usenix.org/system/files/conference/hotcloud16/hotcloud16_burns.pdf

- **Designing Distributed Systems (free ebook)**
  https://azure.microsoft.com/en-us/resources/designing-distributed-systems

- **Kubernetes Patterns (ebook)**
  https://leanpub.com/k8spatterns

- **Kubernetes in Action (ebook)**
  https://www.manning.com/books/kubernetes-in-action


BCN } JUG
JBCN Conf
Q & A