KubeCon | CloudNativeCon

Europe 2019

Learn how to Leverage Kubernetes to Support 12 Factor for Enterprise Apps

Dr. Brad Topol

IBM Distinguished Engineer

@bradtopol

Michael Elder

IBM Distinguished Engineer

@mdelder

Let’s deploy our apps on the cloud!

1. **Cloud** has evolved as a strategy for disruption driven by continuous delivery.

2. Cloud elasticity enables **microservices** architectures to scale out quickly, but also roll new updates out at immense speeds.

3. **Data** becomes the fuel for business innovation.

4. **AI** becomes the catalyst to turn data into **brilliant** user experiences.

5. **Profit**! Or really, reduce overall cost.

Why?

What is a 12-factor app?

- “12-Factor” is a software methodology for building scalable microservice applications
- Originally created by Heroku
- Best practices designed to enable applications to be built with portability, resilience, and scalability when deployed to the web

Why 12 factor apps?

I. Codebase
One codebase tracked in revision control, many deploys

II. Dependencies
Explicitly declare and isolate dependencies

III. Config
Store config in the environment

IV. Backing services
Treat backing services as attached resources

V. Build, release, run
Strictly separate build and run stages

VI. Processes
Execute the app as one or more stateless processes

VII. Port binding
Export services via port binding

VIII. Concurrency
Scale out via the process model

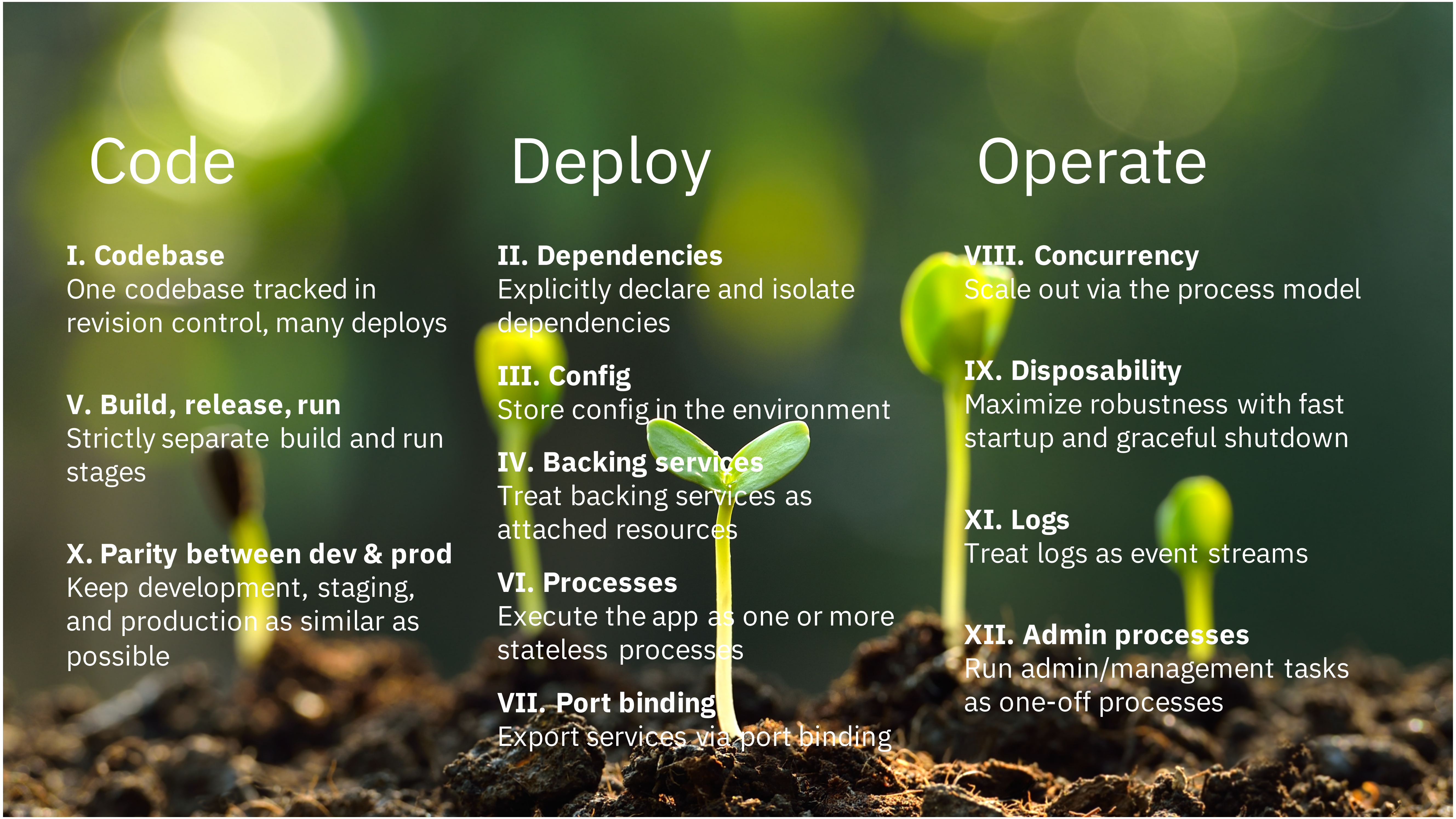
IX. Disposability
Maximize robustness with fast startup and graceful shutdown

X. Dev/prod parity
Keep development, staging, and production as similar as possible

XI. Logs
Treat logs as event streams

XII. Admin processes
Run admin/management tasks as one-off processes

- **Make it easier to run, scale, and deploy applications**
- **Keep parity between development and production**
- **Provide strict separation between build, release, and run stages**



Code

I. Codebase
One codebase tracked in revision control, many deploys

V. Build, release, run
Strictly separate build and run stages

X. Parity between dev & prod
Keep development, staging, and production as similar as possible

Deploy

II. Dependencies
Explicitly declare and isolate dependencies

III. Config
Store config in the environment

IV. Backing services
Treat backing services as attached resources

VI. Processes
Execute the app as one or more stateless processes

VII. Port binding
Export services via port binding

Operate

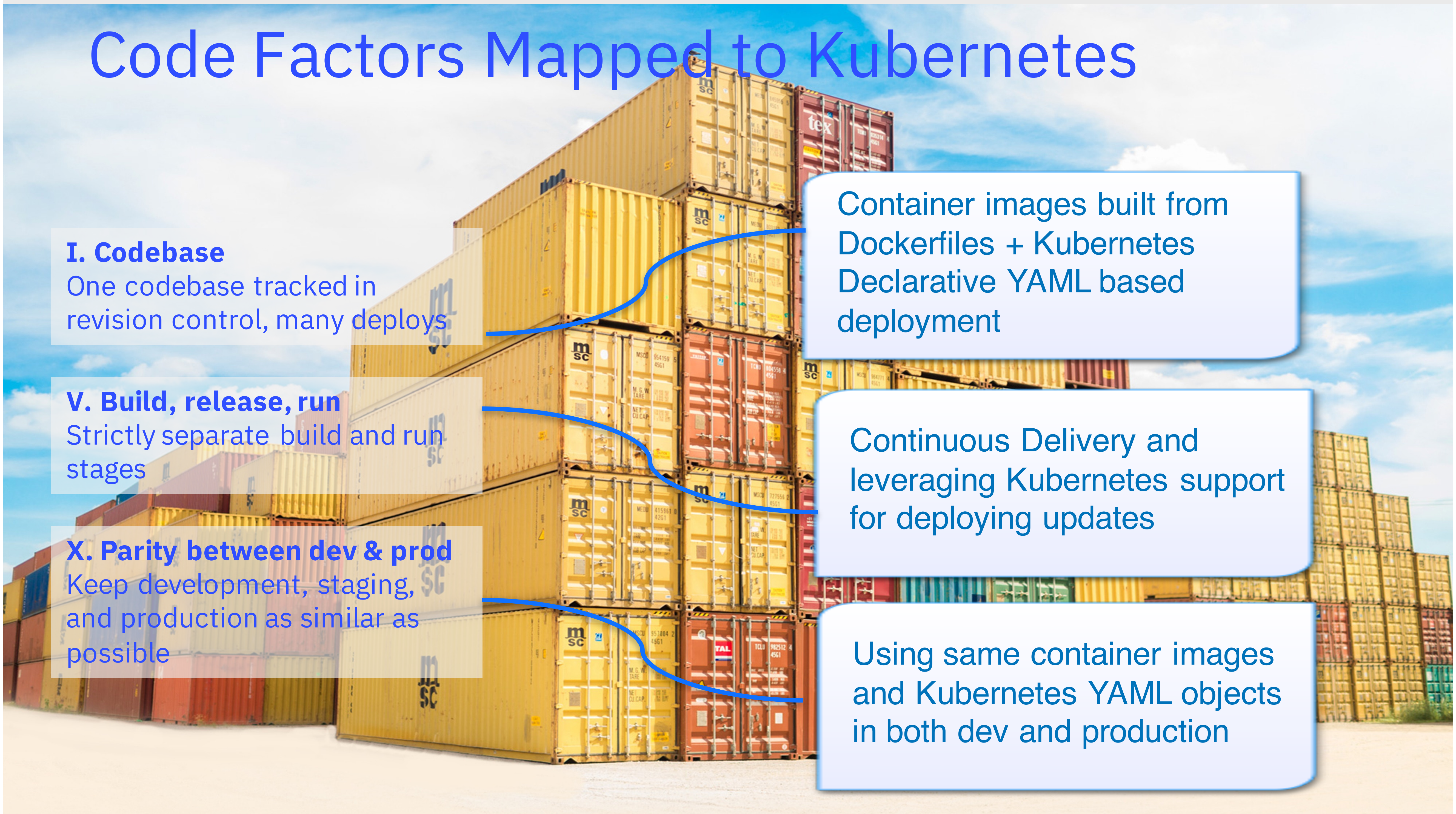
VIII. Concurrency
Scale out via the process model

IX. Disposability
Maximize robustness with fast startup and graceful shutdown

XI. Logs
Treat logs as event streams

XII. Admin processes
Run admin/management tasks as one-off processes

Code Factors Mapped to Kubernetes



I. Codebase
One codebase tracked in revision control, many deploys

Container images built from Dockerfiles + Kubernetes Declarative YAML based deployment

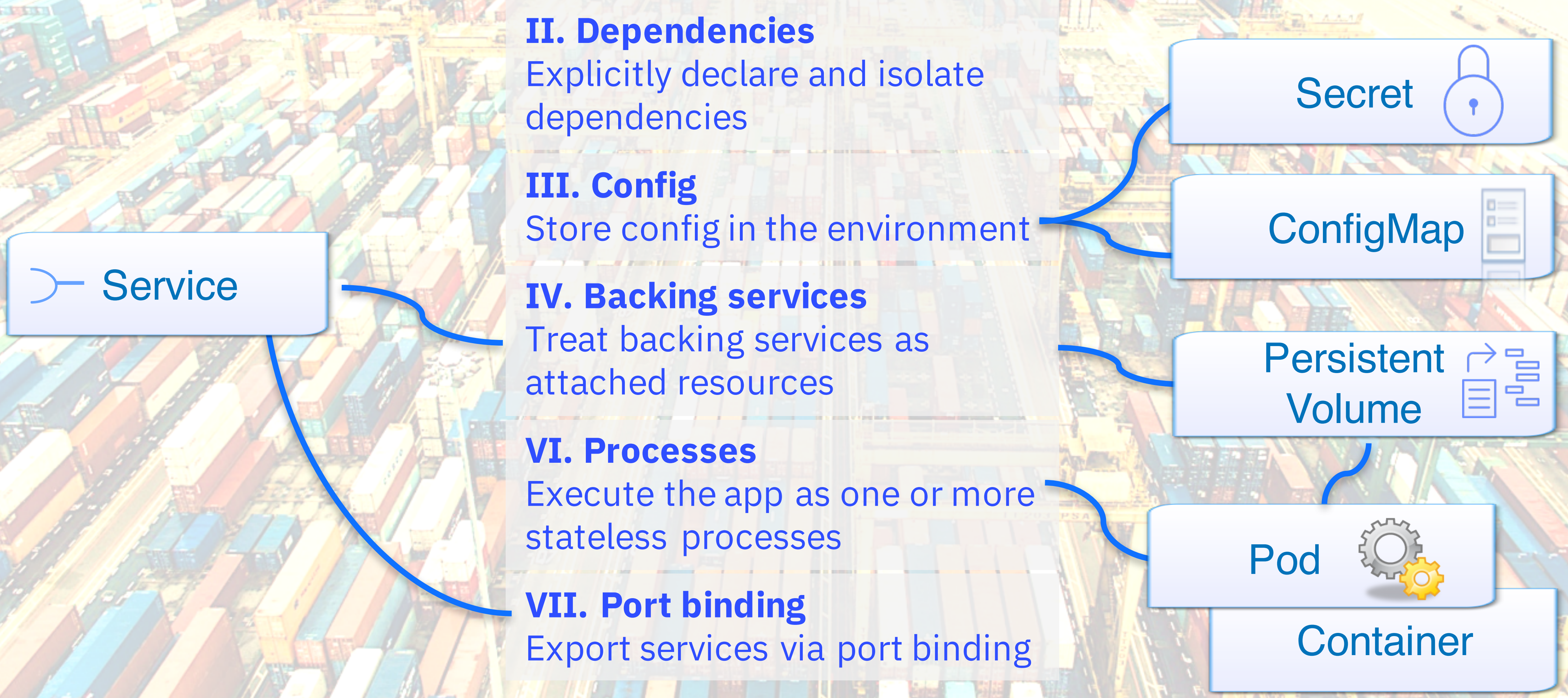
V. Build, release, run
Strictly separate build and run stages

Continuous Delivery and leveraging Kubernetes support for deploying updates

X. Parity between dev & prod
Keep development, staging, and production as similar as possible

Using same container images and Kubernetes YAML objects in both dev and production

Deploy Factors Mapped to Kubernetes



II. Dependencies
Explicitly declare and isolate dependencies

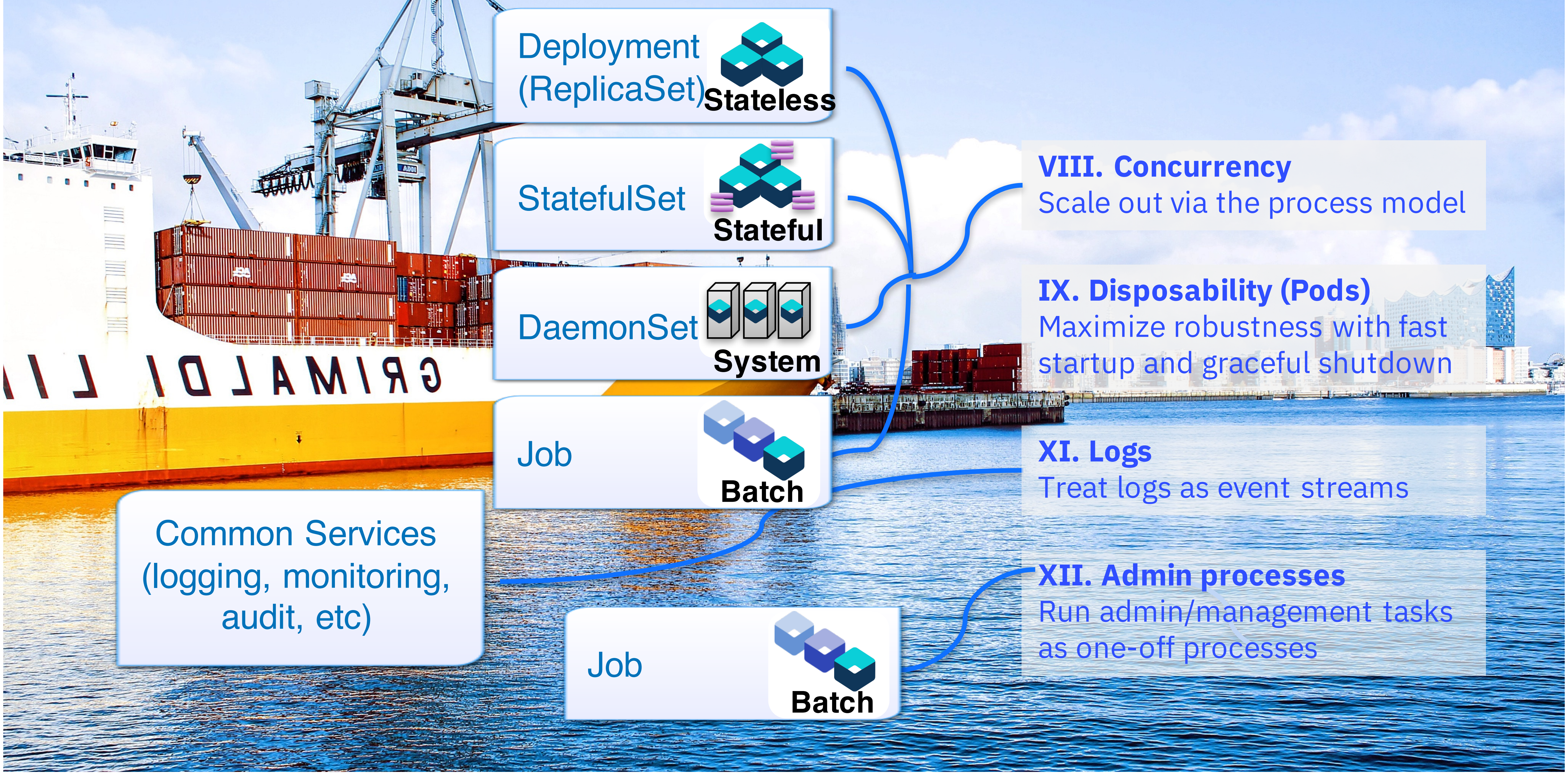
III. Config
Store config in the environment

IV. Backing services
Treat backing services as attached resources

VI. Processes
Execute the app as one or more stateless processes

VII. Port binding
Export services via port binding

Operate Factors Mapped to Kubernetes



Deployment (ReplicaSet) Stateless

StatefulSet Stateful

DaemonSet System

Job Batch

Common Services (logging, monitoring, audit, etc)

Job Batch

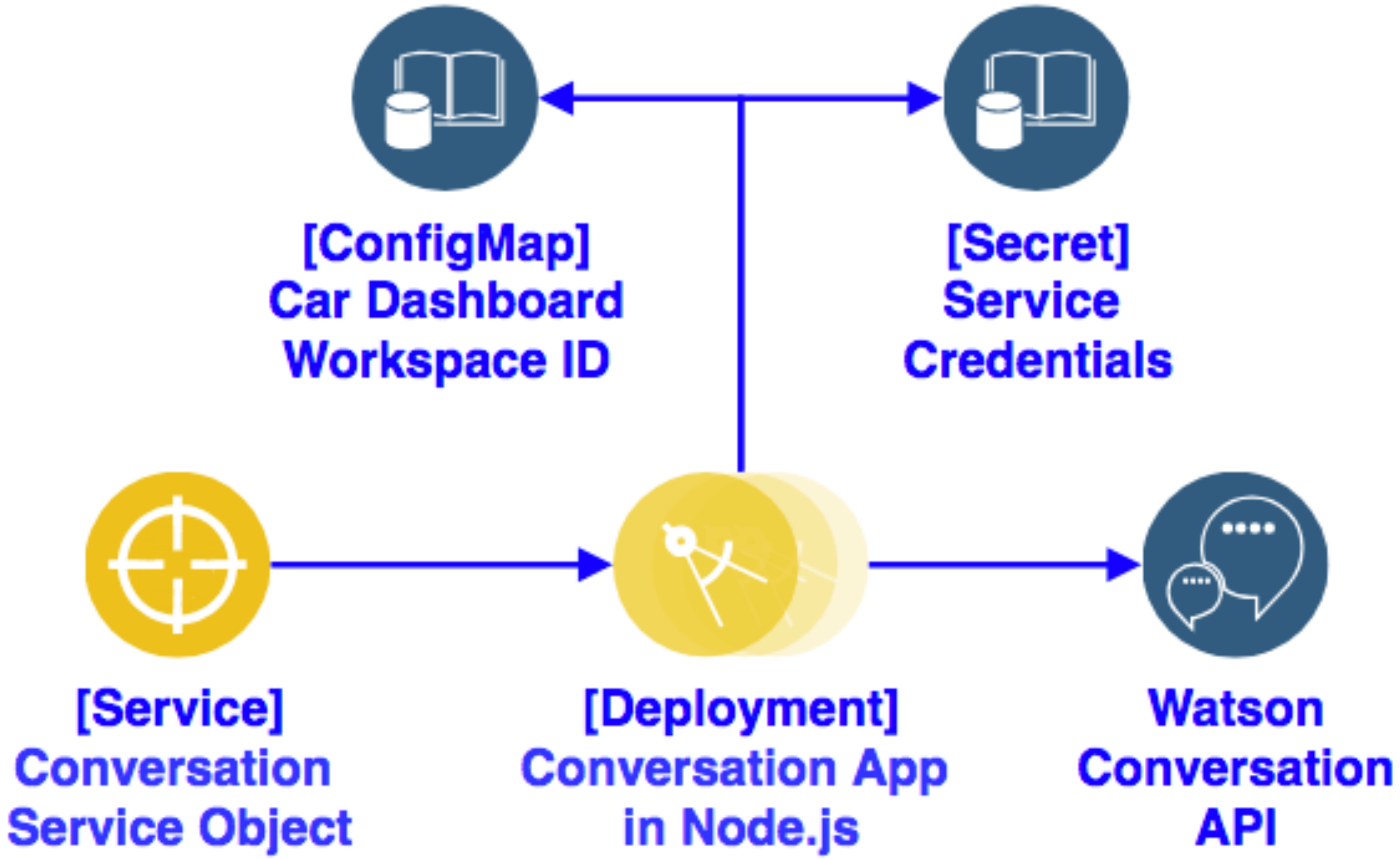
VIII. Concurrency
Scale out via the process model

IX. Disposability (Pods)
Maximize robustness with fast startup and graceful shutdown

XI. Logs
Treat logs as event streams

XII. Admin processes
Run admin/management tasks as one-off processes

An app to talk about



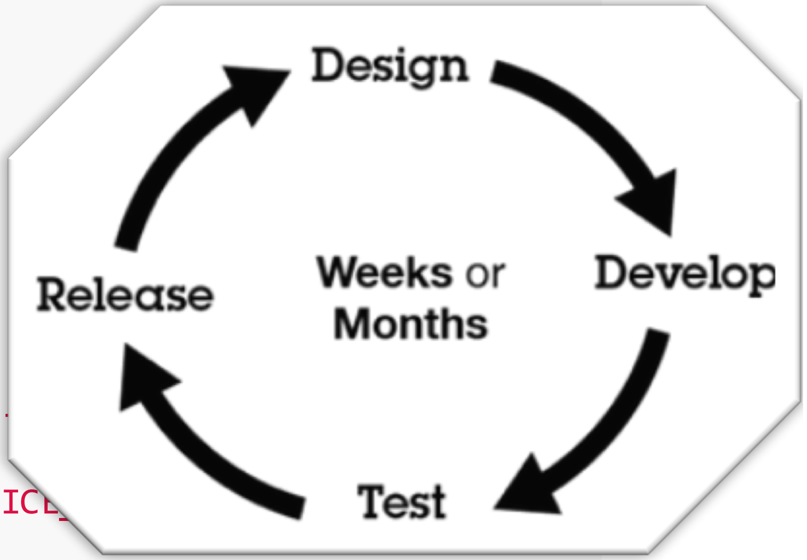
Code factors for our app

1. **Container Images** are built from Dockerfiles. **Kubernetes Deployments, etc** are managed as YAML (Factor #I)

2. Having a strong artifact-driven model makes it easier to follow a **Continuous Delivery** lifecycle (Factor #V)

3. Using the same **images** and YAML objects make it easier for **dev teams** to match what’s running in **production** (Factor #X)

```
# Application to deploy
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
name: watson-conversation-app
spec:
replicas: 2 # tells deployment to run 2 pods matching the
template
template: # create pods using pod definition in this template
  metadata:
    labels:
      app: watson-conversation-app
      tier: frontend
  spec:
    containers:
    - name: watson-conversation-app
      image: mycluster.icp:8500/default/conversation-
simple:alt
      resources:
        requests:
          cpu: 100m
          memory: 100Mi
      env:
        - name: WORKSPACE_ID
          valueFrom:
            configMapKeyRef:
              name: car-dashboard-
              key: workspace_id
        - name: CONVERSATION_SERVICE
          valueFrom:
            secretKeyRef:
              name: binding-conversation-service-car
              key: binding
```

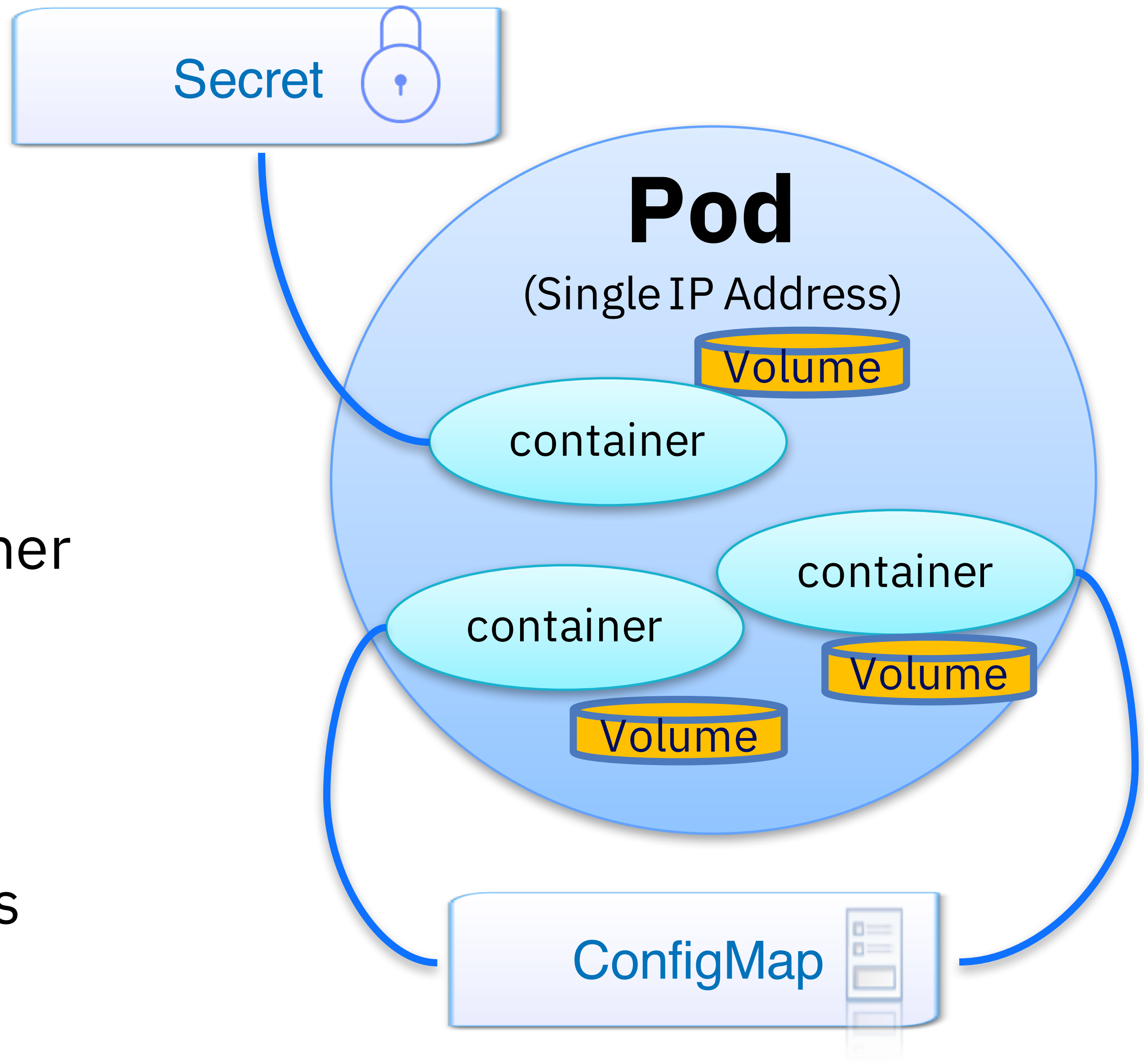


Deploy factors for our app

1. **ConfigMaps** and **Secrets** can be managed in source repositories or built dynamically via commands (Factor #III)

2. Our **container image** runs as a container **process** in a **Pod** with other **containers** (Factor #VI)

3. A collection of **Pods** can expose or consume **Services** via port bindings (Factor #IV & Factor #VII)

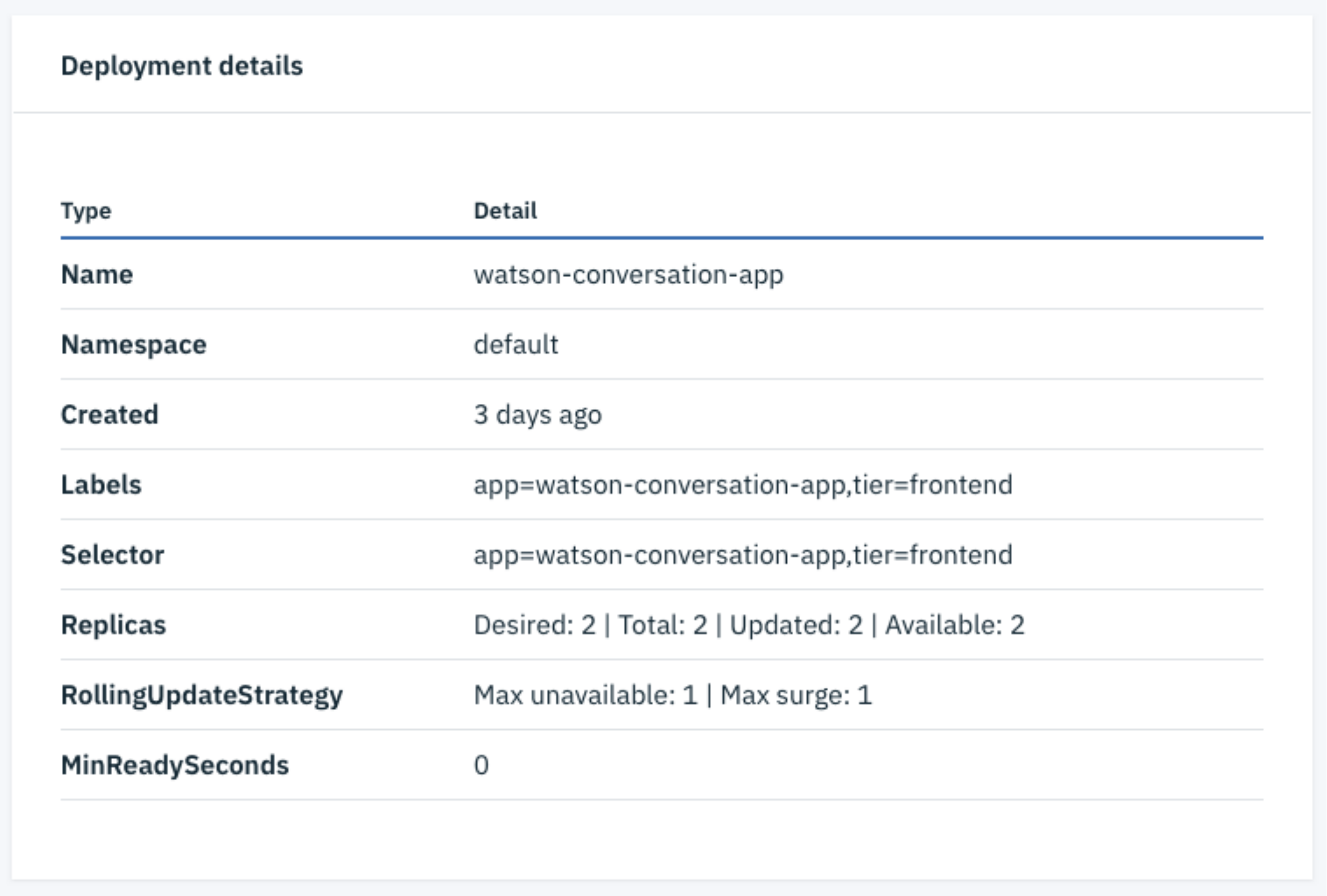


Operate factors for our app

1. A **Deployment** includes a **ReplicaSet** which declares the desired availability policy (Factor #VIII)

2. If a **Pod** fails, Kubernetes will attempt to recover it via restarting the Pod or scheduling it to a new node (Factor #IX)

3. Running our app as a container makes it possible to capture all logs, metrics, and other management functions in a consistent way (Factor #XII)



Deployment details

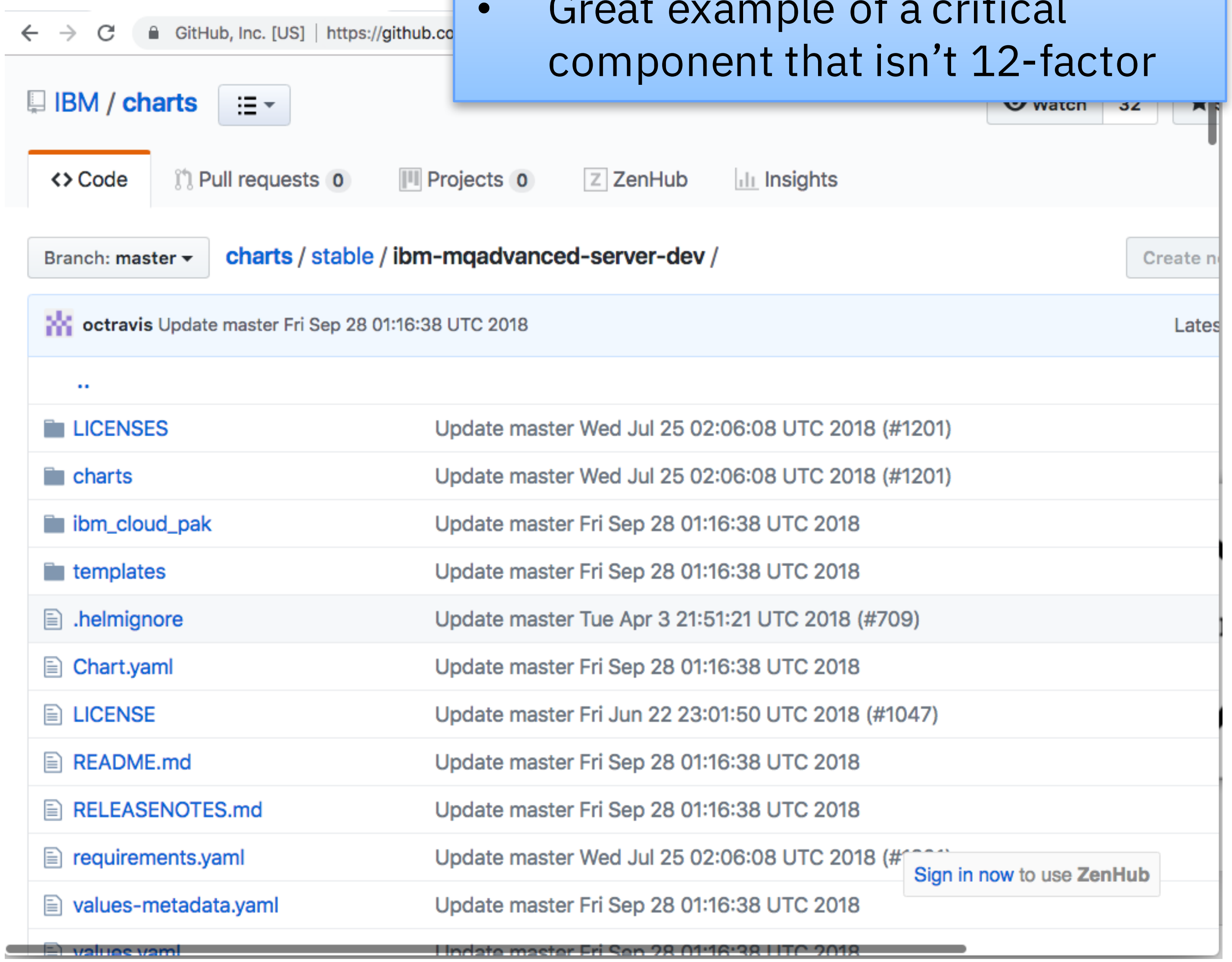
Type	Detail
Name	watson-conversation-app
Namespace	default
Created	3 days ago
Labels	app=watson-conversation-app,tier=frontend
Selector	app=watson-conversation-app,tier=frontend
Replicas	Desired: 2 | Total: 2 | Updated: 2 | Available: 2
RollingUpdateStrategy	Max unavailable: 1 | Max surge: 1
MinReadySeconds	0

Great, but 95% of my workloads do not fit 12-factor!

Code factors for middleware

1. **Helm Charts** are an open way to package 12-factor apps, but also middleware like IBM MQ Series (F#I)
2. Providing a catalog of Helm Charts either from the community or your internal teams makes it easier to **build production-like environments** (F#V)
3. Just like apps, build these into Continuous Delivery pipelines for **canary testing your upgrades** of critical supporting services

- IBM MQ Series is a leading provider of messaging services for enterprise apps;
- Great example of a critical component that isn't 12-factor



charts / stable / ibm-mqadvanced-server-dev /

octravis Update master Fri Sep 28 01:16:38 UTC 2018 Lates

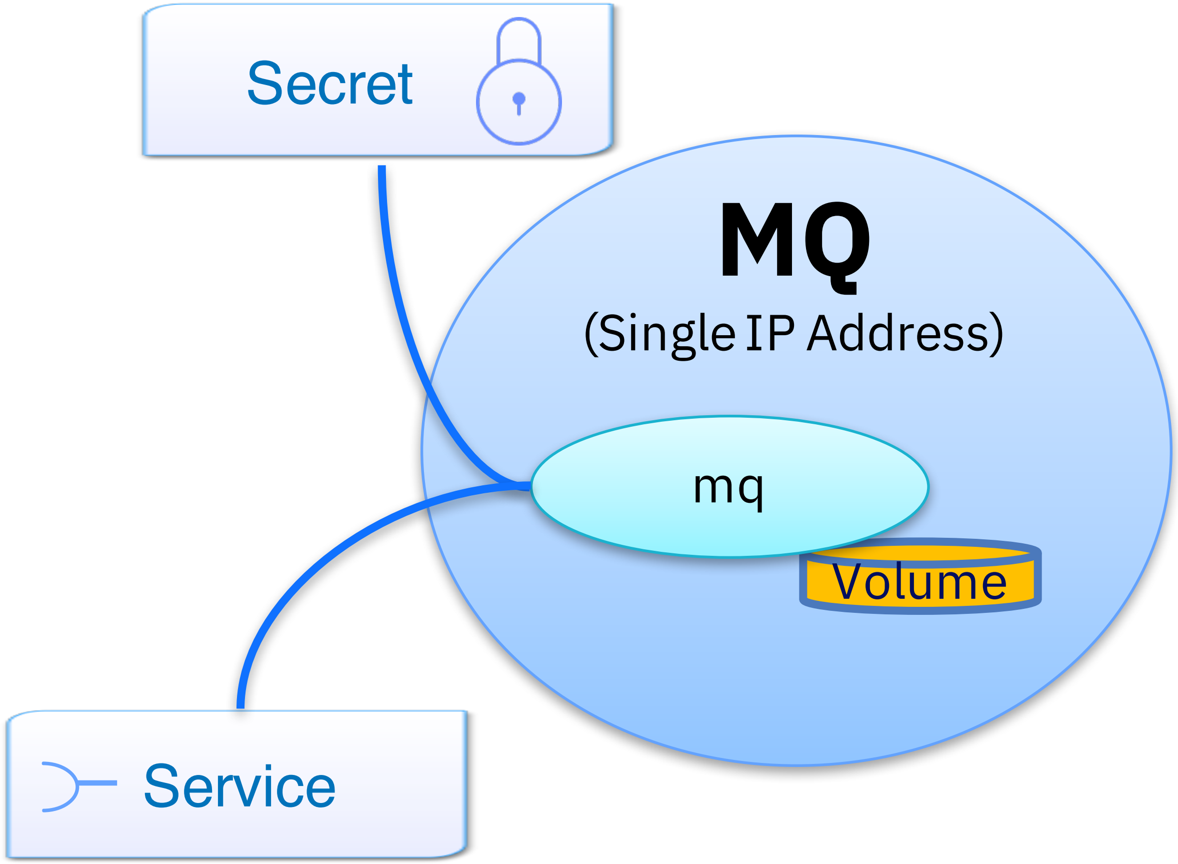
..	
LICENSES	Update master Wed Jul 25 02:06:08 UTC 2018 (#1201)
charts	Update master Wed Jul 25 02:06:08 UTC 2018 (#1201)
ibm_cloud_pak	Update master Fri Sep 28 01:16:38 UTC 2018
templates	Update master Fri Sep 28 01:16:38 UTC 2018
.helmignore	Update master Tue Apr 3 21:51:21 UTC 2018 (#709)
Chart.yaml	Update master Fri Sep 28 01:16:38 UTC 2018
LICENSE	Update master Fri Jun 22 23:01:50 UTC 2018 (#1047)
README.md	Update master Fri Sep 28 01:16:38 UTC 2018
RELEASENOTES.md	Update master Fri Sep 28 01:16:38 UTC 2018
requirements.yaml	Update master Wed Jul 25 02:06:08 UTC 2018 (#1201)
values-metadata.yaml	Update master Fri Sep 28 01:16:38 UTC 2018
values.yaml	Update master Fri Sep 28 01:16:38 UTC 2018

Deploy factors for middleware

1. **Secrets** used to configure credentials and TLS certificates (F#III)

2. MQ built as a **container image** that runs as a container **process** in a **Pod** (F#VI)

3. Admin console, app messaging ports, and metrics ports exposed via **Services** with port bindings (F#IV & F#VII)

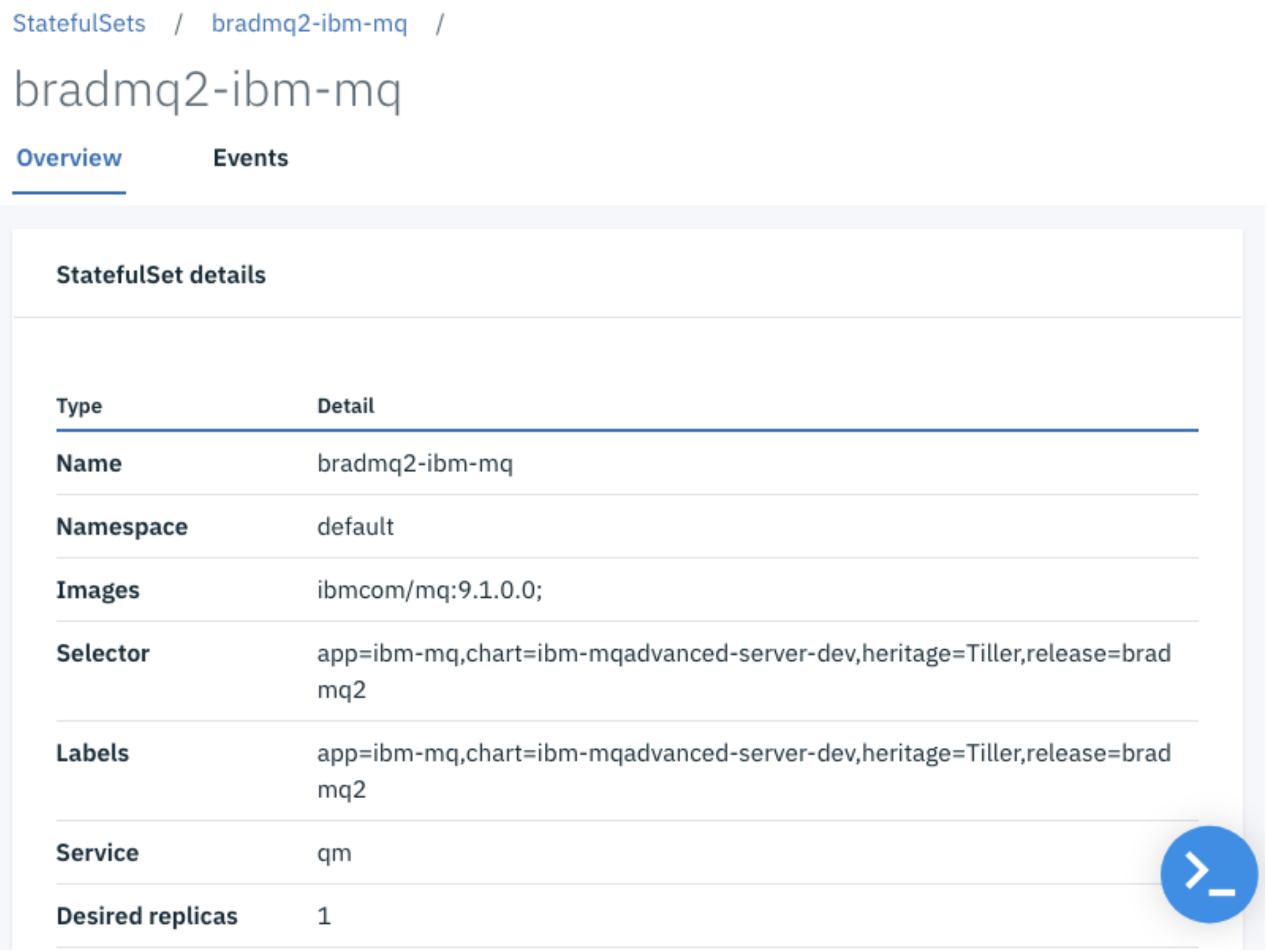


Operate factors for middleware

1. A **StatefulSet** that declares the desired availability policy for MQ; a database might scale out replicas or prepare primary/secondary failover (F#VIII)

2. Recovered **Pods** re-mount the same **PersistentVolumes** (F#IX)

3. All **Pods** can have **logs**, **metrics**, or other **management details** captured automatically by your Kubernetes provider (F#XII)



bradmq2-ibm-mq

StatefulSet details

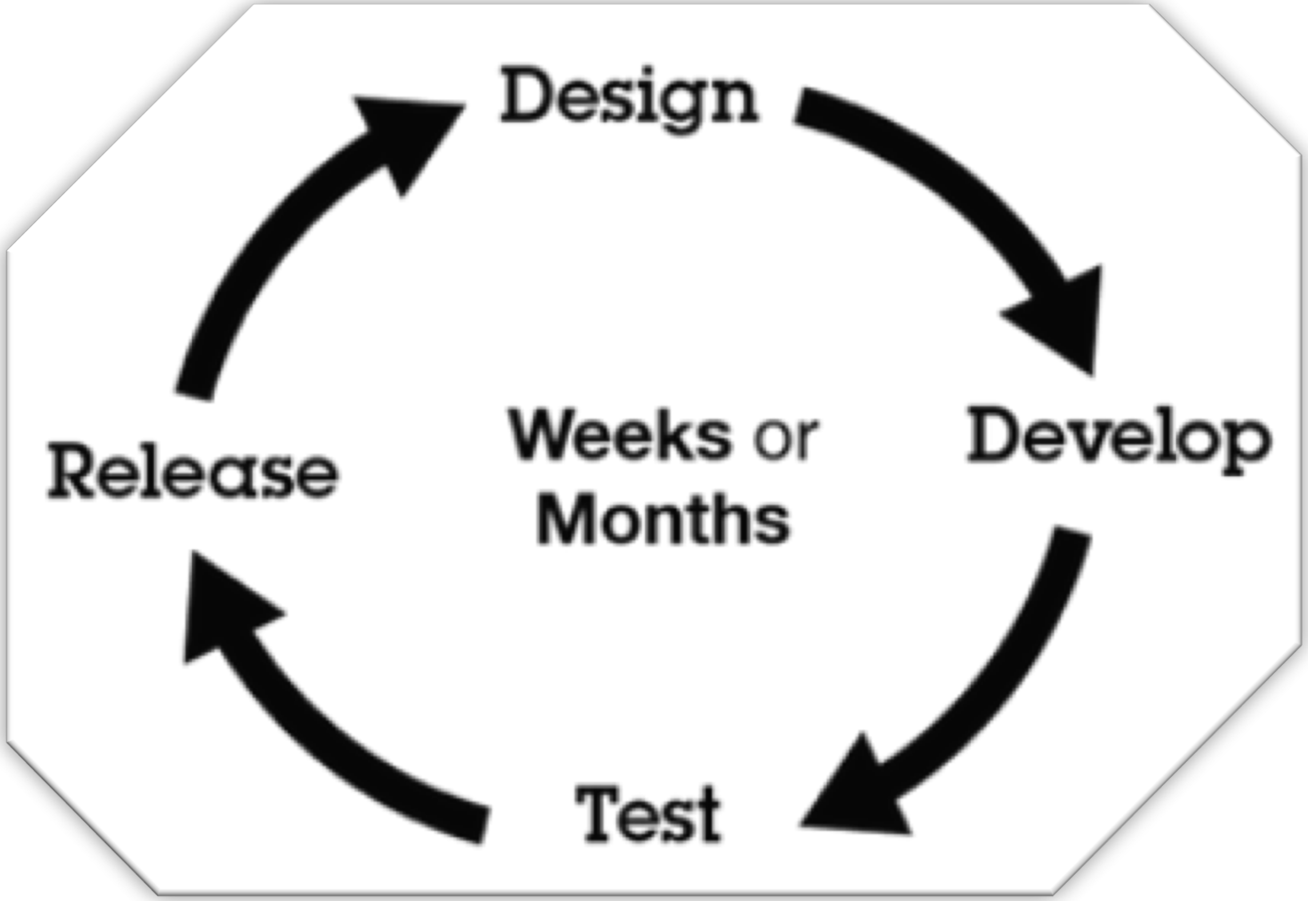
Type	Detail
Name	bradmq2-ibm-mq
Namespace	default
Images	ibmcom/mq:9.1.0.0;
Selector	app=ibm-mq,chart=ibm-mqadvanced-server-dev,heritage=Tiller,release=bradmq2
Labels	app=ibm-mq,chart=ibm-mqadvanced-server-dev,heritage=Tiller,release=bradmq2
Service	qm
Desired replicas	1

Continuous Integration & Delivery

1. Exposing all of your datacenter via container images with a Kubernetes orchestrator will take time for full maturity

2. Potential to dramatically simplify delivery of services and ongoing operations with built-in control planes for running containers

3. Start **NOW** to leverage these same features designed for 12-factor apps to expose more **production-like environments** (F#V) for your devs/LOBs



Enough talking, let’s see it LIVE!

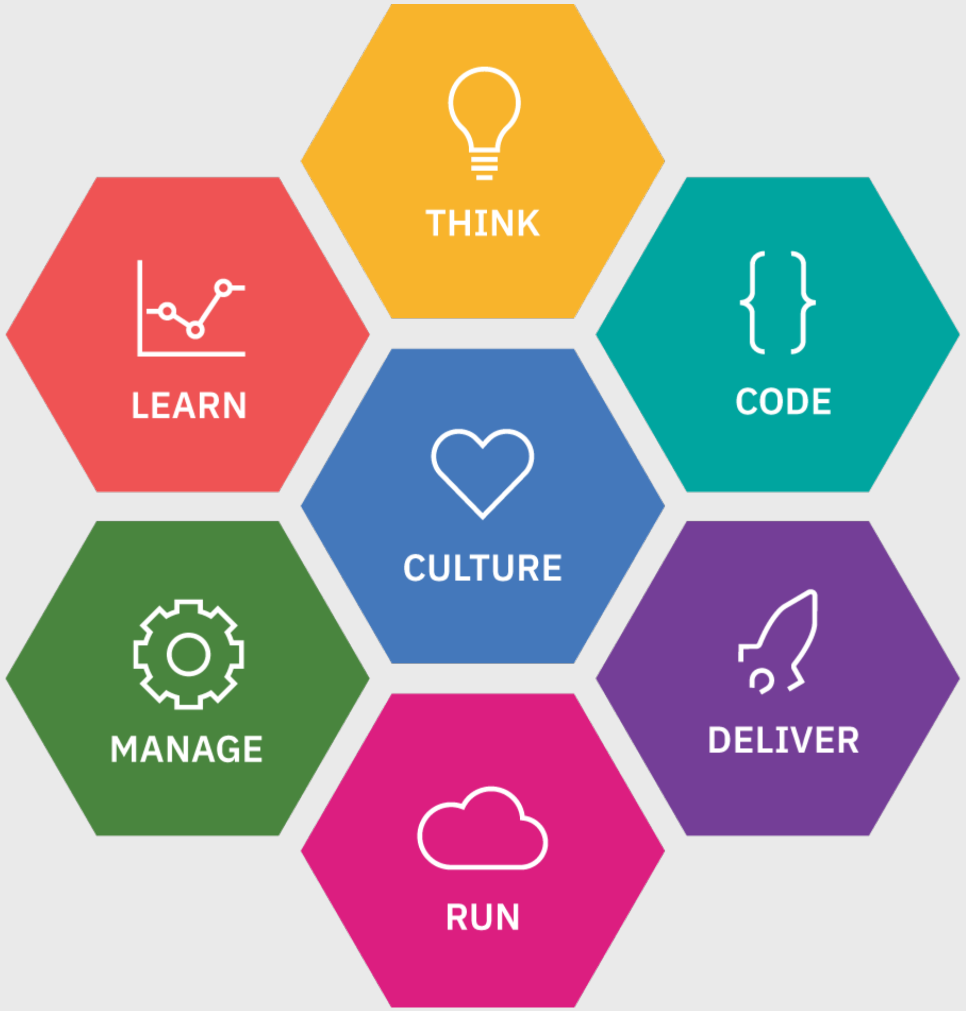
Leverage the IBM Cloud Garage Method to change how you work.

ibm.com/cloud/garage >

Provides an in-depth collection of practices, tutorials, and architectures to help you on this journey.

Completely open forum for learning at your own pace.

We offer hands-on guidance and services, if needed.



Defined Practices ▶ **Business Benefits** ✚ **Technical Benefits**

Find IBM Cloud Garages worldwide

AUSTIN

COPENHAGEN

DUBAI

LONDON

MELBOURNE

MUNICH

NEW YORK

NICE

SAN FRANCISCO

SÃO PAULO

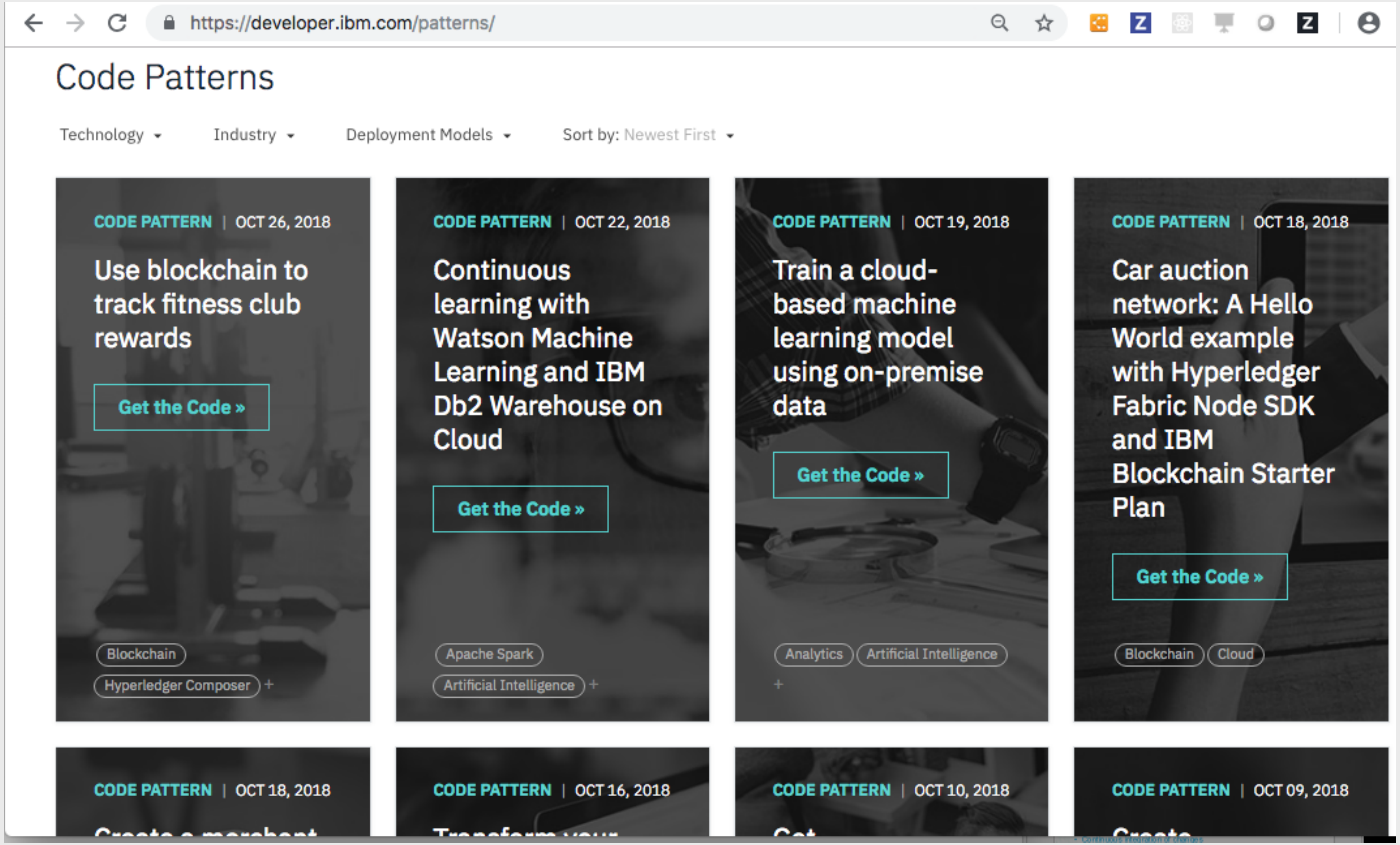
SINGAPORE

TOKYO

TORONTO

Get your hands in the code

developer.ibm.com/patterns >



https://developer.ibm.com/patterns/

Code Patterns

Technology ▾ Industry ▾ Deployment Models ▾ Sort by: Newest First ▾

CODE PATTERN | OCT 26, 2018

Use blockchain to track fitness club rewards

Get the Code »

Blockchain · Hyperledger Composer +

CODE PATTERN | OCT 22, 2018

Continuous learning with Watson Machine Learning and IBM Db2 Warehouse on Cloud

Get the Code »

Apache Spark · Artificial Intelligence +

CODE PATTERN | OCT 19, 2018

Train a cloud-based machine learning model using on-premise data

Get the Code »

Analytics · Artificial Intelligence +

CODE PATTERN | OCT 18, 2018

Car auction network: A Hello World example with Hyperledger Fabric Node SDK and IBM Blockchain Starter Plan

Get the Code »

Blockchain · Cloud

CODE PATTERN | OCT 18, 2018

Create a merchant

CODE PATTERN | OCT 16, 2018

Transform your

CODE PATTERN | OCT 10, 2018

Get

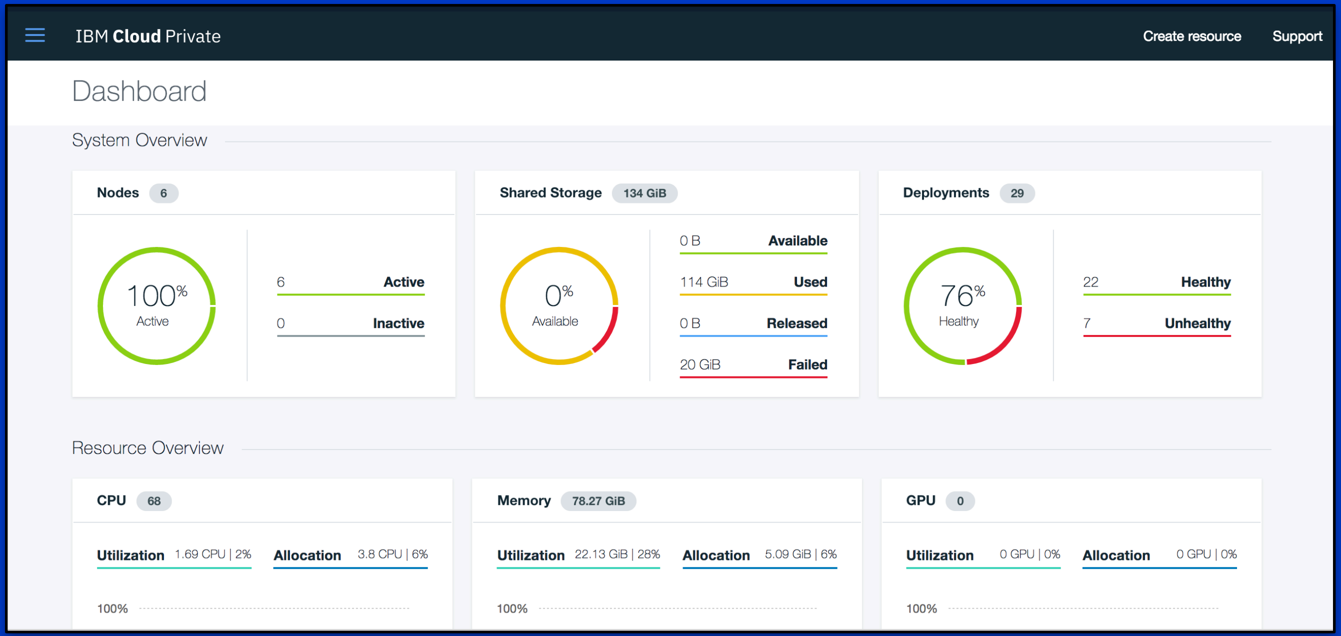
CODE PATTERN | OCT 09, 2018

Create

Try Kubernetes with IBM Cloud Private

ibm.biz/Try-IBMCloudPrivate >

Free Community Edition



IBM Cloud Private

Create resource

Support

Dashboard

System Overview

Nodes 6

100% Active

6 Active

0 Inactive

Shared Storage 134 GiB

0% Available

0 B Available

114 GiB Used

0 B Released

20 GiB Failed

Deployments 29

76% Healthy

22 Healthy

7 Unhealthy

Resource Overview

CPU 68

Utilization 1.69 CPU | 2%

Allocation 3.8 CPU | 6%

100%

Memory 78.27 GiB

Utilization 22.13 GiB | 28%

Allocation 5.09 GiB | 6%

100%

GPU 0

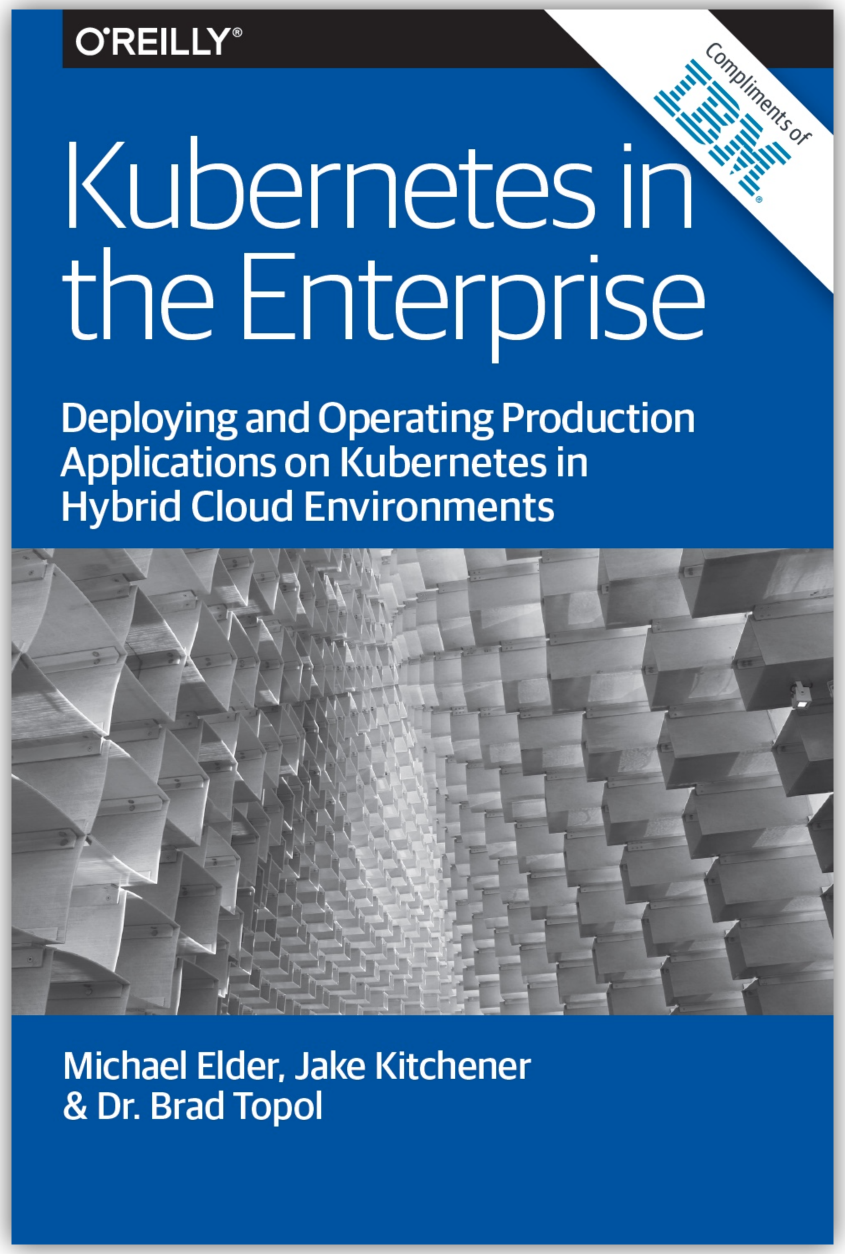
Utilization 0 GPU | 0%

Allocation 0 GPU | 0%

100%

ibm.biz/BdYA4i >

Learn more in our new book!



O'REILLY®

Compliments of IBM

Kubernetes in the Enterprise

Deploying and Operating Production Applications on Kubernetes in Hybrid Cloud Environments

Michael Elder, Jake Kitchener & Dr. Brad Topol

Now available online compliments of IBM (see above)!

Come see me to get it signed!

IBM