KubeCon | CloudNativeCon

OPEN SOURCE SUMMIT

China 2019

KubeCon CloudNativeCon OPEN SOURCE SUMMIT
China 2019

7 Missing Factors for Your Production-Quality 12-Factor Apps

Michael Elder
IBM Distinguished Engineer - IBM Multicloud Platform
@mdelder

Shikha Srivastava
IBM Senior Technical Staff Member
@shikhasthoughts

What is Ready for production application

- Secure
  - Installation, authentication and access
- Resilient, Highly Available and scale
- Repeated deployment
  - with safe upgrades and configuration changes
- Performance
- Observable
- Upgradeable
- more …..
- And AGILE too

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 scalabilit***y when deployed to the web

kubernetes & 12-factor apps

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

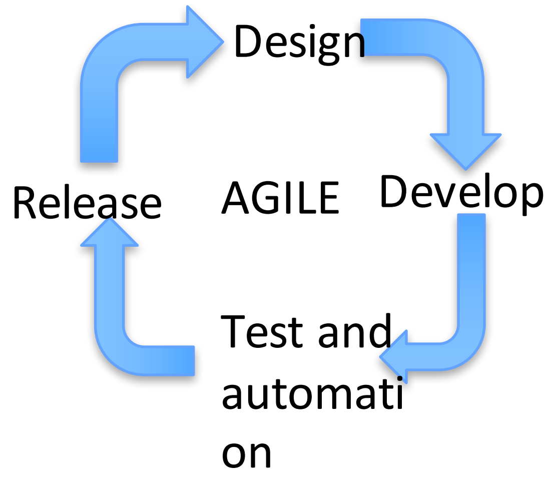
XII. Admin processes
Run admin/management tasks as one-off processes

Developers dream – Code factors

- One codebase for my application tracked in revision that runs anywhere: build, ship and run anywhere

AND

- I can offload deployment, HA, scaling, upgrade strategy and not worry about it



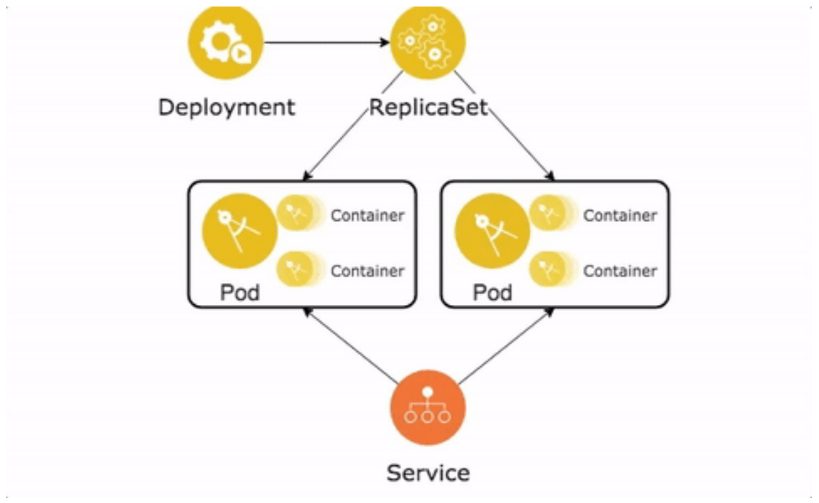
- **Container Images** built from Dockerfiles using trusted small image. **Kubernetes Deployments, etc** managed as YAML (F#I- ***Codebase***)

- Having a strong artifact-driven model makes it easier to follow a **Continuous Delivery** lifecycle (F#V- ***Build, release, run***)

- Using the same **images** and YAML objects make it easier for **dev teams** to match what's running in **production** (F#X- ***Dev/prod parity***)

Operate factors: Concurrency (F#VIII) & Disposability (F#IX)

- Ensure scale for your app
- Replica set ensures specified number of pods are always running



- Is this enough?

  Remember load is never constant in the real world

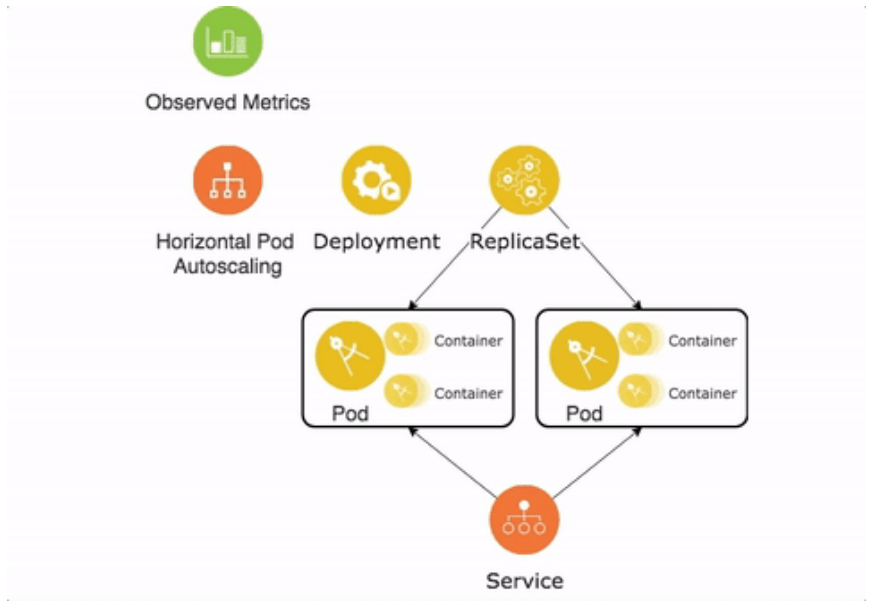
```
kind: Deployment
metadata:
 name: nginx
spec:
 replicas: 2
 template:
   metadata:
     labels:
       service: http-server
   spec:
     containers:
     - name: nginx
       image: nginx:1.10.2
       imagePullPolicy: IfNotPresent
       ports:
       - containerPort: 80
```

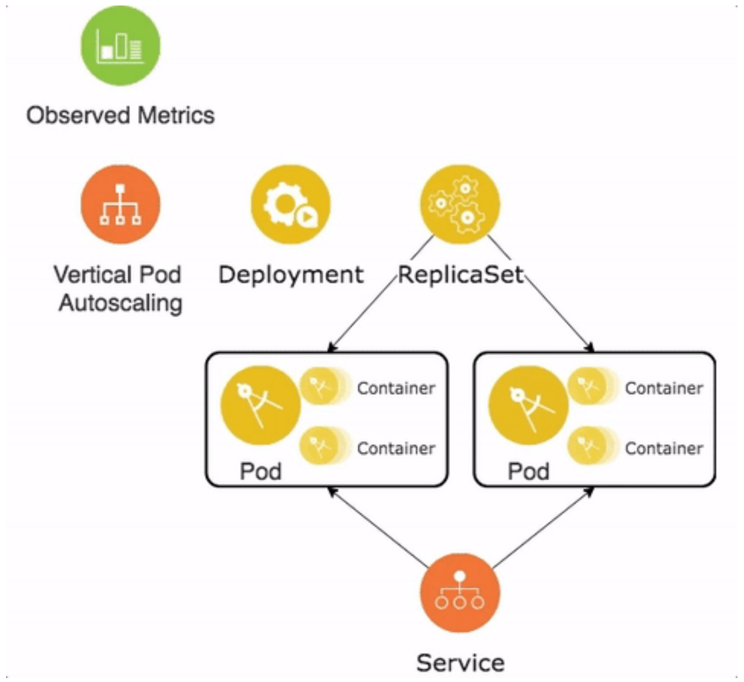
Operate factors: Concurrency (F#VIII)

Leverage autoscaling to automate computation resources based on load

- Horizontal Pod Scaler (HPA)
  - Controls the number of replicas
  - Use cpu or memory as a trigger or use custom metric
  - Applicable for stateless app

- Vertical Pod Scaler (VPA)
  - Controls the memory and cpu for pod
  - Use cpu or memory as a trigger or use custom metric
  - Applicable for statefull apps





XIII. Observable
Apps should provide visibility about current health and metrics

XIV. Schedulable
Apps should provide guidance on expected resource constraints

XV. Upgradable
Apps must upgrade data formats from prior generations

XVI. Least privileged
Apps should provide guidance on expected resource constraints

XVII. Auditable
Apps should provide appropriate audit logs for compliance needs

XVIII. Access Control (Identity, Network, Scope, Certificates)
Protect app and resources from the world

XIX. Measurable
Apps usage should be measurable for quota or chargebacks

7 missing factors

SCAN ME

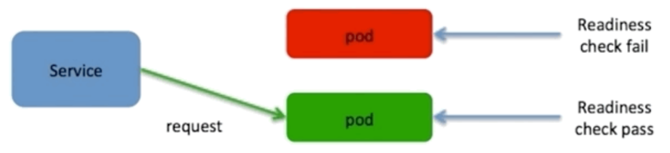
7 missing factors from 12 factor application

Observable: Application health (F#XIII)

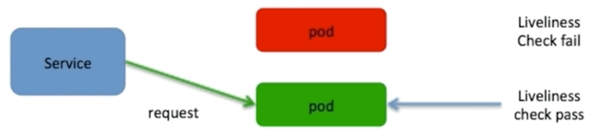
Know your application health

- Kubernetes probes
  - Is the app ready to accept traffic?: Readiness
  - Is the app responsive? : Liveliness

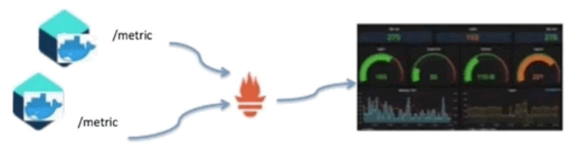
```
readinessProbe:
  # an http probe
  httpGet:
    path: /readiness
    port: 8080
  initialDelaySeconds:
20
  periodSeconds: 5
```



```
livenessProbe:
  # an http probe
  httpGet:
    path: /healthcheck
    port: 8080
  initialDelaySeconds: 15
  timeoutSeconds: 1
```

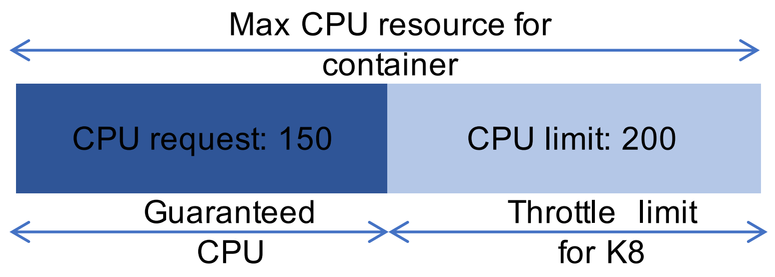


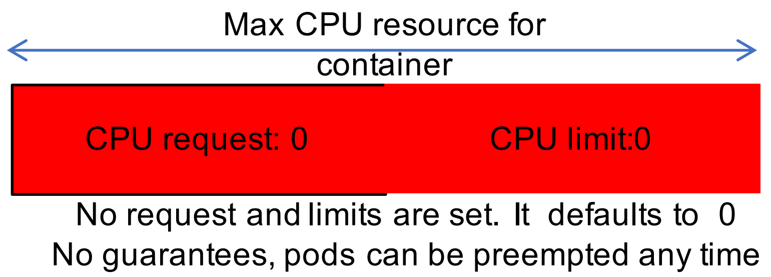
- Is this enough?
  - What about transactions, traffic, memory usage ?



Schedulable: Resource requests, limits, & quotas (F#XIV)

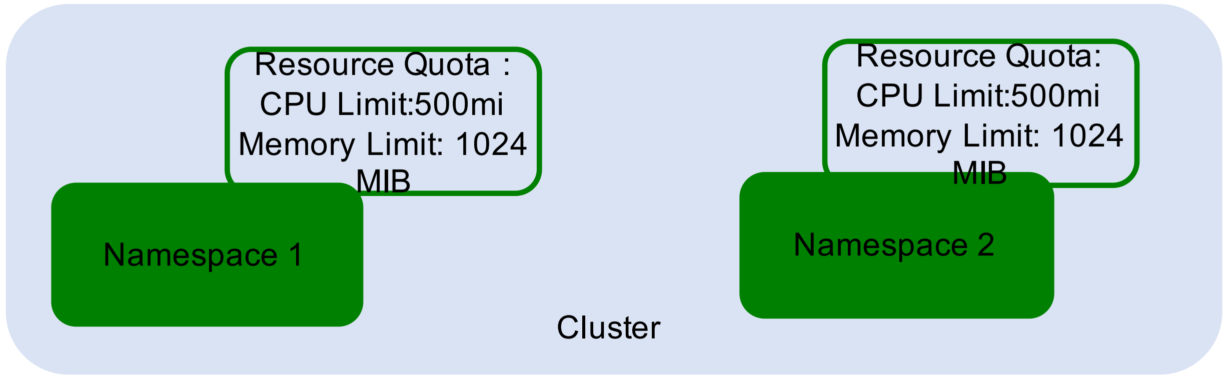
Guarantee resources for your containers: Specify request and limits for the compute resources




Max CPU resource for container
CPU request: 0
CPU limit:0
No request and limits are set. It defaults to 0
No guarantees, pods can be preempted any time

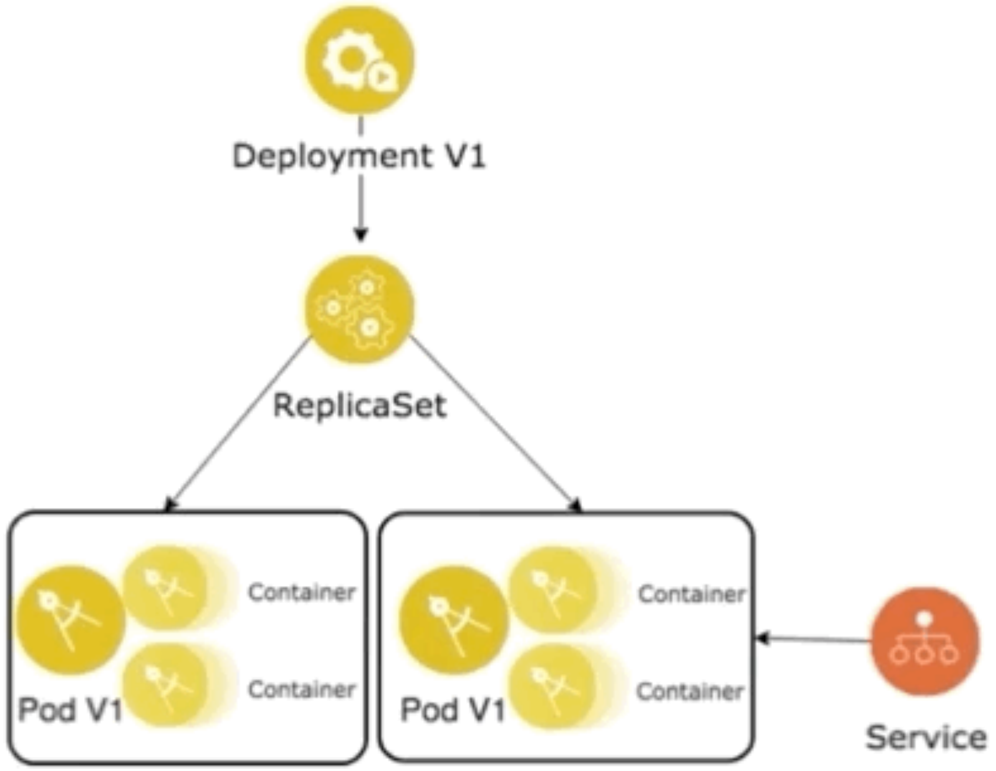
Set resource quota

Once quota in a namespace for compute resources set, the users are forced to set requests or limits for those values



Upgradable (F#XV)

Applications should be able to roll out updates for cases where backward compatible updates (security or feature updates)needs to be made



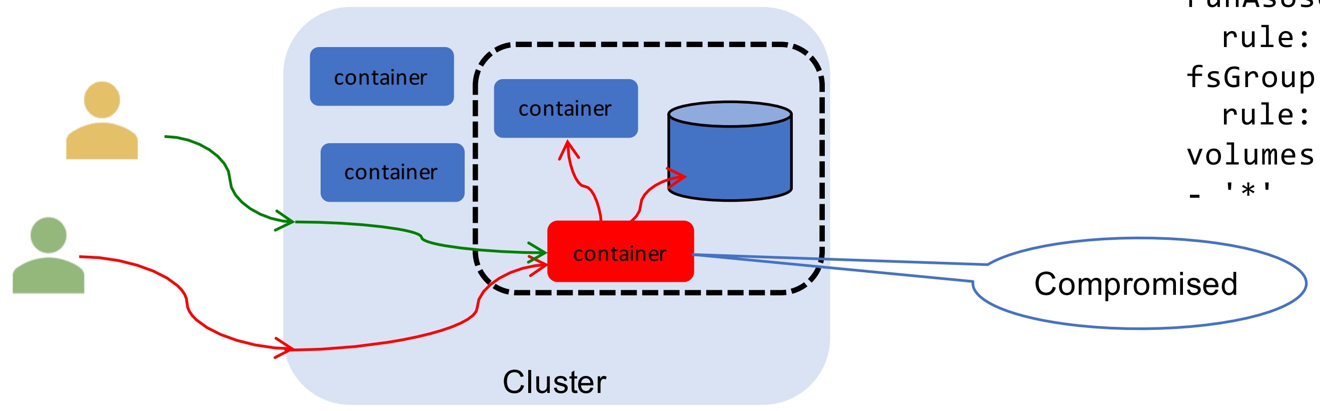
```
minReadySeconds: 5
strategy:
  # indicate which
strategy
  # we want for rolling
update
   type: RollingUpdate
  rollingUpdate:
  maxSurge: 1
  maxUnavailable: 1
```

Least Privilege (F#XVI)

Limit container access to hosts. Every permission is an attack vector

- Use Pod Security Policy and Network Policy to
  - Limit access to filesystem
  - Limit access to Kernel capabilities
  - Use a non-privileged user
  - Limit access to volume types
  - Limit access to ports

```
#sample-psp.yaml
apiVersion: policy/v1beta1
kind: PodSecurityPolicy
metadata:
  name: example
spec:
  privileged: false
  # Don't allow
  #   privileged pods!
  # The rest fills in some
  #   required fields.
  seLinux:
    rule: RunAsAny
  supplementalGroups:
    rule: RunAsAny
  runAsUser:
    rule: RunAsAny
  fsGroup:
    rule: RunAsAny
  volumes:
  - '*'
```



Auditable (F#XVII)

- Know WHAT/WHEN/WHO/WHERE for all CRUD operations
  - Chronological set of records documenting sequence of events affecting system and application by users or components
- Use cloud agnostic industry standard format – CADF (Cloud Auditing Data Federation)
- Control the quantity of logs

CADF event:

<initiator_id>: ID of the user that performed the operation
<target_uri>: CADF specific target URI, (for example: data/security/project)
<action>: The action being performed, typically: <operation>. <resource_type>

Access Control -Identity, Network, Scope (F#XVIII)

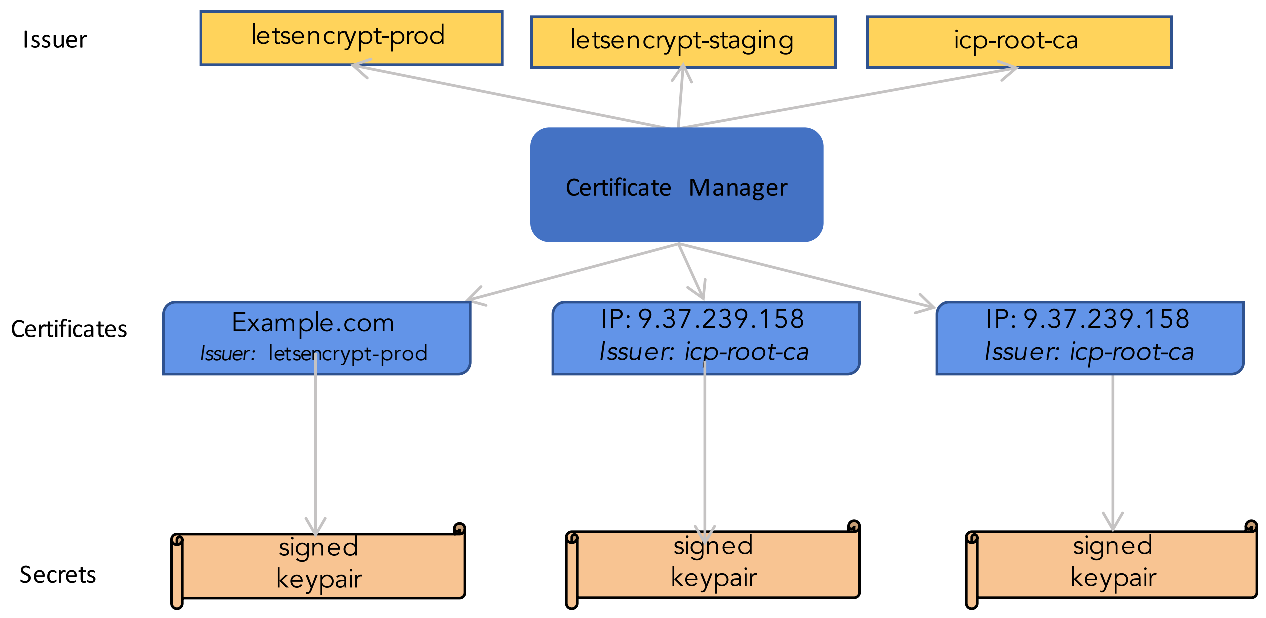
Protect app and resources from the world

- Authentication and Authorization
- Certificate Management
- Data Protection
- Network security
  - Network policy
  - Network Isolation
- Admission Controller
  - Example: Image admission controller

Access Control: Identity, Network, Scope (F#XVIII)

Ensure secure communication

- Generate Certificates
- Enable TLS / mTLS
- Manage Certificates

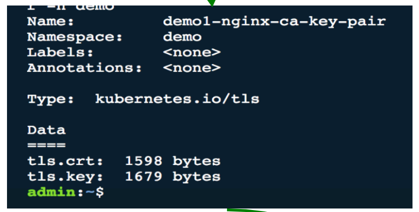


```
# sample issuer.yaml
apiVersion: certmanager.k8s.io/v1alpha1
kind: Issuer
metadata:
    name: demo1-nginx-ca
    namespace: demo
spec:
  ca:
     secretName: demo1-nginx-ca-key-pair
```

1. Issuer creates Certificate

```
# sample certificate.yaml
apiVersion: certmanager.k8s.io/v1alpha1
kind: Certificate
Metadata:
 name: demo1-nginx-cert
spec:
   secretName: demo1-nginx-cert
 issuerRef:
     name: demo1-nginx-ca
  kind: Issuer
    commonName: "foo1.bar
    dnsNames:
      foo1.bar1
```

2. Certificate creates secret



```
r -n demo
Name:         demo1-nginx-ca-key-pair
Namespace:    demo
Labels:       <none>
Annotations:  <none>

Type:  kubernetes.io/tls

Data
====
tls.crt:  1598 bytes
tls.key:  1679 bytes
admin:~$
```

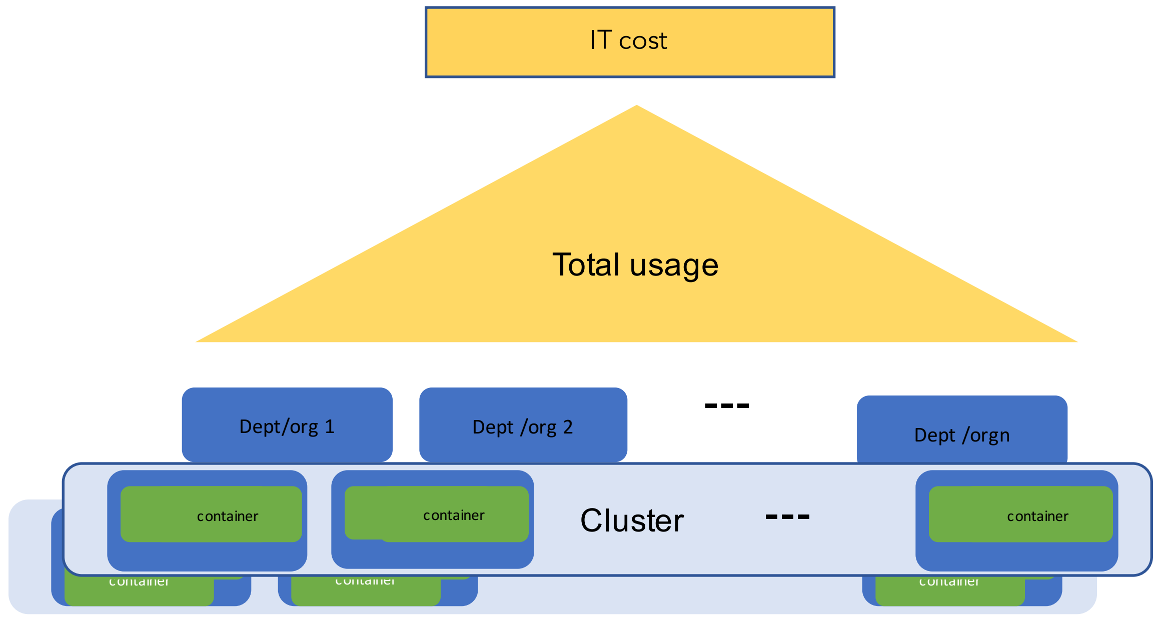
3. Secret mounts to Pod

Pod

Measurable (F#XIX)

Know the cost of the application

- Compute resources allocated to run the containers should be measurable
- Org / department using the cluster should be accountable

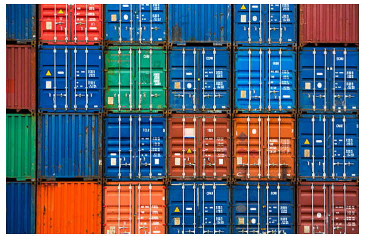


So, What really makes a production-ready app?

A production grade application

Attention to
Cloud provider configurations
Example XII: Observable, Example: XVIII: Access Control. Factor XIX:: Measurable

Attention to
Kubernetes configuration
Example: Factor III: Config, Factor II Config, Factor XIV: Schedulable

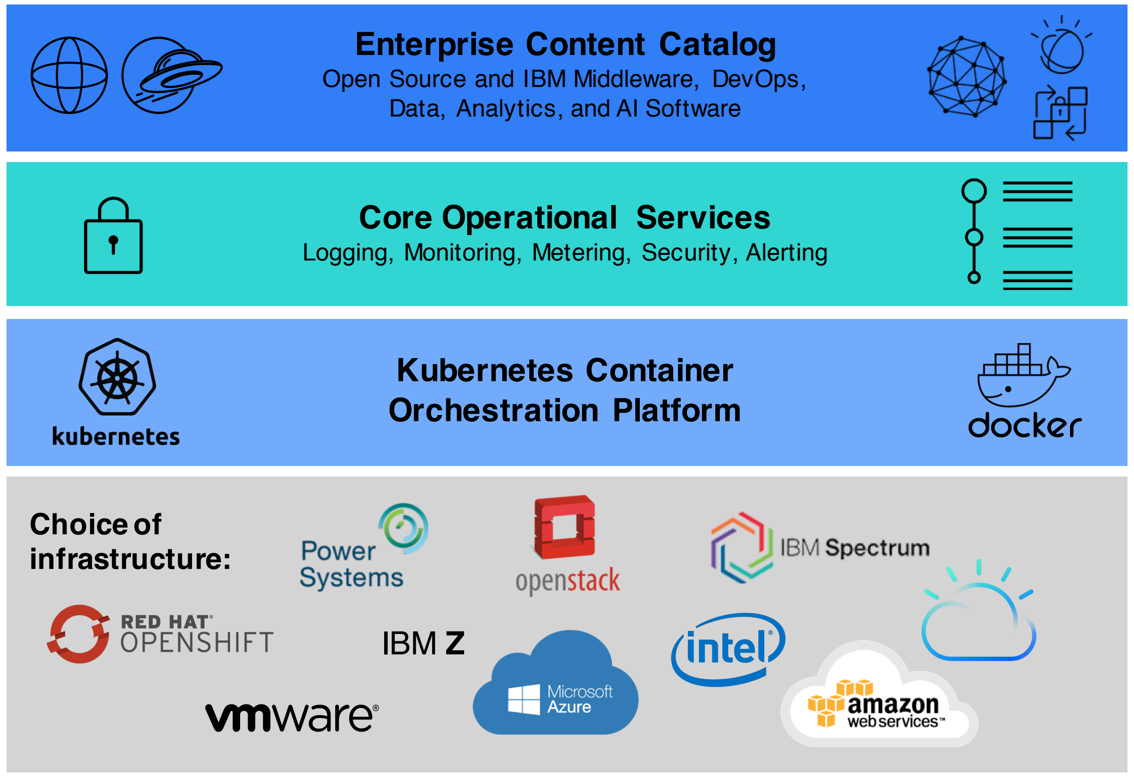


Attention to
Building **containers** and what's inside the containers
Example: Factor I : codebase , Factor X: dev/prod parity, Factor XV

Production thinking needs to be through the entire process

Enough talking, let’s see it LIVE!

IBM Cloud Private (ICP)

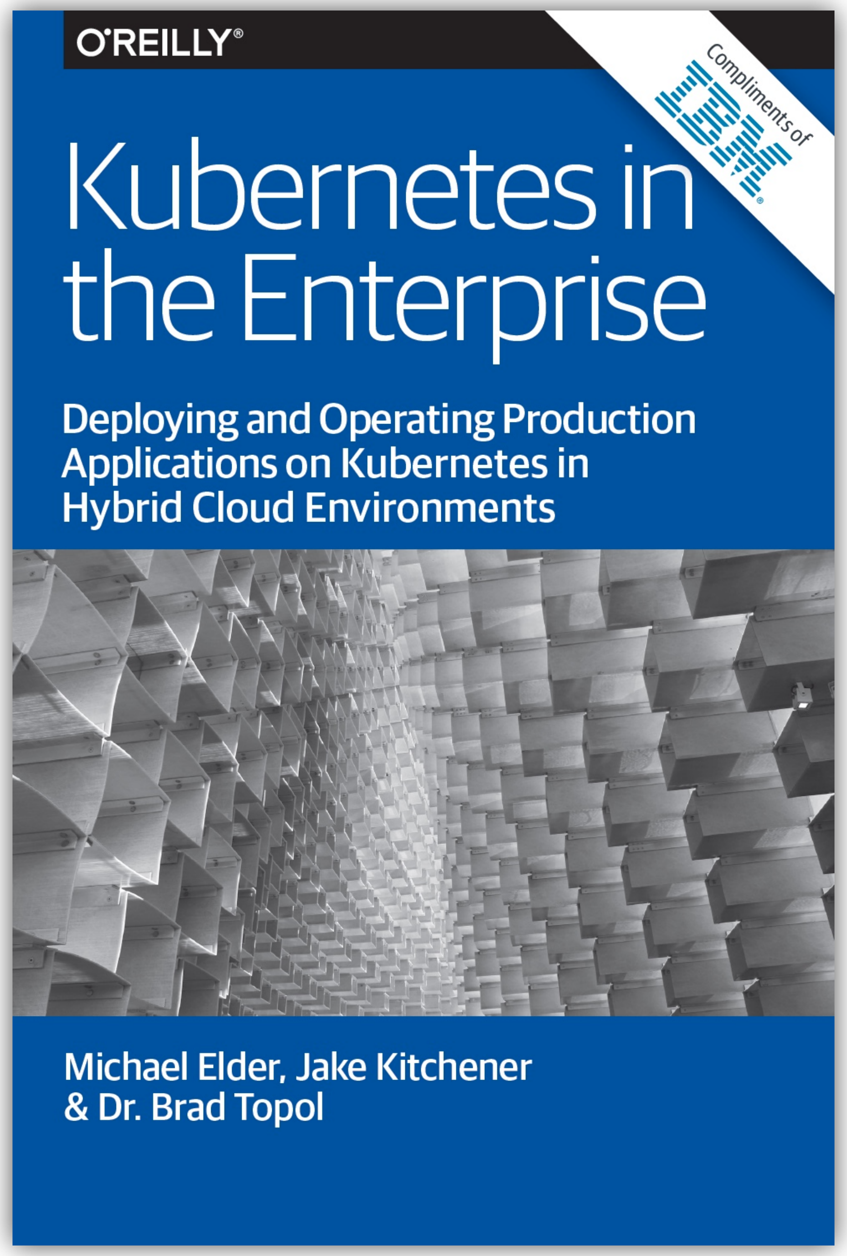

Enterprise Content Catalog
Open Source and IBM Middleware, DevOps, Data, Analytics, and AI Software
Core Operational Services
Logging, Monitoring, Metering, Security, Alerting
Kubernetes Container Orchestration Platform
kubernetes
docker
Choice of infrastructure:
Power Systems
openstack
IBM Spectrum
RED HAT OPENSHIFT
IBM Z
intel
Microsoft Azure
amazon web services
vmware

- Self-service rich catalog of IBM MW
- Helm based parameterized install to simplify complex K8 apps
- Logging : ELK + filebeat
- Monitoring : Prometheus + Grafana
- Usage : IBM Metering Service
- IBM Vulnerability Advisor
- IBM Mutation Advisor
- Authentication/ Authorization
- Certificate Management
- Network security
- Audit trail for any CRUD operations
- Team based organization of resources

Provides the capabilities to run containerized application in secure, scalable and resilient environment

All communication enabled over TLS. Data secured in transit and at rest

Now available online compliments of IBM: ibm.biz/BdYA4i >


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

Need Details on signing

#7678A: Tech Talk: Deploying Kubernetes in the Enterprise (with the authors)

When: Wednesday, 11:30 AM - 12:10 PM

Where: Table Top Tap Room at the Metreon | Code Cafe Tech Talks Area

Get a **signed** copy with all of the authors at the Code Café Mezzaine on Wednesday (7 – 7:30PM)!

Learn more in our new book!