From Containerized Applications to secure and scaling with Kubernetes

Shikha Srivastava
@shikhasloughs
IBM

Erica Brown
@ericabrEngineer
IBM
Ready for production application

- Secure
  - Installation and access
- Resilient, Highly Available and scale
- Repeated deployment
  - with safe upgrades and configuration changes
- Performance
- Obervability
- more ..... 
- And AGILE too
Production Setup with failover

Ahh… think of:

• Code maintenance
  • multiple language across multiple different machine
• Reliable code updates and rollback
• Scaling
• Transaction and Data replication
• And more
Developer’s wish

• I can have 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

( developers dream)

Containers + Kubernetes + Cloud Scale
What does this mean for the application

• Application: collection of independent microservices

• Microservice: packaged as containers, developed and delivered following continuous delivery and continuous integration model

• Containers: managed and orchestrated by some one else (like Kubernetes)

AND

• Applying security best practices through out

Microservice and Containers are foundation for Cloud Native architecture
Building Containers

- Small Trusted Images
  - private trusted repo for organization ensuring authenticity of the base image
- Remove any vulnerabilities
  - scanning tools integrated into ongoing process.
- Sensitive Data (e.g., password) not on docker file.
  - It should be passed at runtime
- Self Contained
- Immutable
- Single process recommended per container
Inside Container

- Use a non root user inside the container
- Explicitly call out and isolate dependencies
  - No ownership of supporting subsystems like storage
- Configurations in config maps
- Use secrets for sensitive info
- Long lasting data should be stored in a database or shared PV
- Run with minimal privileges
- Disposable: Let container orchestrator handle the failure
- Log to stdout and stderr
What's next – The hosting part

Production application is composed of multiple containers and needs to be deployed across multiple host.

Need orchestration in place to manage containers for

• Resiliency, HA, Scale out and in
• Security- install and access
• Upgrade and rollback strategy
• Monitoring, Logging and metering
• Storage management
• And more
Kubernetes provides deployment automation, scaling and management of containerized application

- Controls and automates application deployments and updates.
- Provides HA and Scales containerized applications and their resources on the fly.
- Provides health-check and self-heal for the applications
- Mounts and adds storage to run stateful apps.
- And more
Pod: group of related containers with storage option and a unique IP
Service: group of related Pods and policy defining how to access them
Volume: data in container dies with container,
Nodes: where pods and services run. Nodes can be across VMs
Master coordinates all the Nodes in the cluster and manages replicas
Labels: simple key value pairs that can be applied to pods, services etc.
Scaling

For: Reliability, High availability, Scaling and rolling updates

Deployments: Instructs cluster how to create and scale apps

Replica set: ensures specified number of pods are always running specified in deployment

Auto-scaling: HPA
Dynamically scaling the pods based on resource usage
Also see CA and VPA
Deployments allows for rolling out updates for cases where backward compatible updates (security or feature updates) need to be made.

- **minReadySeconds**: time it takes for an app to boot.
- **maxSurge**: number of pods in addition to desired ones.
- **maxUnavailable**: number of pods that can be unavailable.

```
minReadySeconds: 5
strategy:
  # indicate which strategy we want for rolling update
type: RollingUpdate
rollingUpdate:
  maxSurge: 1
  maxUnavailable: 0
```
Application stability

Default behavior: Kubernetes sends traffic to pods
The default behavior can be changed with

- **Liveliness Probe**: indicator that application is running
- **Readiness Probe**: indicator that application is ready to serve traffic

Types of probes and configuration
- HTTP
- Command
- TCP

livenessProbe:
  httpGet:
    path: /healthcheck
    port: 8080
  httpHeaders:
    - name: X-Custom-Header
      value: Awesome
  initialDelaySeconds: 5
  periodSeconds: 3

Any code greater than or 200 and less than 400 = success
Specify Compute resources for containers
- specify how much CPU and memory (RAM) each
  Container needs.
  - Set **requests and limits**

Set **resource quota**
A resource quota provides constraints towards
aggregate resource consumption per namespace

Once quota in a namespace for compute resources like
cpu and memory is set, the users are forced to set
requests or limits for those values
If not set, the quota system may reject pod creation

```yaml
kind: ResourceQuota
metadata:
  name: demo
spec:
  hard:
    requests.cpu: 500m
    requests.memory: 10Mib
    limit.cpu: 700m
    limit.memory: 500Mib
```

Resources:
requests:
  memory: “64Mi”
  cpu: “200m”
limits:
  memory: “64Mi”
  cpu: “200m”
Security

Provided by K8s provider
• All components should use certificates
• TLS enablement
• Disable any and all anonymous logins
• Audit trail for any CRUD operations
• Disable any and all insecure ports
• Authentication, Authorization and Admission Controller
  • Image admission controller
  • Pod Security policy: Cluster level resource controlling actions that pods can perform
• Vulnerability and mutation monitoring
Organizing in K8s

• Use Namespaces to segregate the cluster for use by departments or dev teams
• Provides isolated env to set policies and naming resources
  • Resources created in one namespace is hidden from other namespaces
  • Admins can control resources allocated to namespaces
  • RBAC policy scoped for appropriate authority
Summary – CI + CD + CS

- Small base images
- Secure
- Immutable
- Signed images
- Deny unknown images
- Separate automation from user actions
- No elevation of privileges
- Permissions on volume
- Audit trail
- PSP
- Resource management
- Fine grained RBAC
- Handle sensitive data
- Scaling
- HA
- Upgrade strategy
# IBM Cloud Private (ICP) Overview

## Choose your infrastructure:

- **IBM Z**
- **openstack**
- **vmware**
- **IBM Spectrum**
- **Intel**

## Content Catalog

- Open Source and IBM Middleware, Data, Analytics, and AI Software

## Core Operational Services

- Log Management, Monitoring, Metering, Security, Alerting, Deployment

## Kubernetes Container Orchestration Platform

- **docker**

## Strategic Value:

- Self-service catalog
- Agility, scalability, and elasticity
- Self-healing
- Enterprise security
- No vendor lock-in
ICP-Built with open standards, preventing vendor lock-in

Containers

- **Docker**: Executable package of software that includes everything needed to run it

Orchestration

- **Kubernetes**: Automate deployment, scaling, and management of containerized applications

Management

- **Helm**: Define, install, and upgrade Kubernetes applications

Provisioning

- **Terraform**: Infrastructure as code to provision public cloud and on-premises environments
Home Page

- Private cloud overview
- Intro videos
- IBM Cloud Private for Dummies book

http://ibm.biz/ICP-Home

Garage Method

- Reference architectures
- Best practices

http://ibm.biz/ICP-Garage

Digital Technical Engagement

- Guided demos
- Proof of Technology

http://ibm.biz/ICP-DTE
ICP - Security

**Identity**
- OpenID Connect provider in Liberty profile is used to authenticate users
- Integrates with client’s enterprise LDAP
- Users and groups are imported into the ICP platform for authorization purposes

**Access Control**
- Role-based access control based on teams
- A ‘team’ is a logical grouping of resources, users, and user groups
- Users and user groups are assigned roles within a team that gives them permissions associated with each assigned role on resources within this team
- Access control gateway enforces role based access control for all registered services

**Network Security**
- Calico based
- Isolate traffic based on Namespaces (logical scope) & or Nodes (physical scope) or both
- Fine grained control over the sharing of objects within a single namespace.

**Data at Rest Protection**
- Any ICP state is encrypted
- All ICP secrets are encrypted at rest and isolated through Role Based Access Control

**Image Security**
- Vulnerability/Mutation Advisor
- Private Image Repository
- Image Admission Controller Policies for Namespace gating

Audit Logging, Compliance, Inter-Node Communication, Cert Management …