Openshift on IBM Cloud Workshop

Jaffa Sztejnbok
Developer Advocate
IBM

jaffas@il.ibm.com

IBM **Developer**

IBM

20 CALL FOR 20 CODE®
Global Challenge
The clock is ticking. Take on climate change now.
Join the movement where over 210,000 participants have stepped up in the past 2 years to build over 8,000 applications for urgent societal issues. Your ideas to help halt and reverse the impact of climate change could earn you $200K and actually be deployed.
Accept the challenge
UN
SHAPING OUR FUTURE

Winners of the Grand Prize receive **$200,000 USD cash** and open source deployment support through Code and Response™ with The Linux Foundation.

Answer the call. Take on climate change.

In a recent global IBM survey of more than 3,000 developers, first responders, and social activists, 77% agree that climate change is the single most pressing issue of our time. In the same survey, 79% of respondents agree that climate change is something that can be reduced or combatted with technology.

Innovate with the latest technology to address energy sustainability, water sustainability, or disaster resiliency. Winning solutions are deployed in communities that need the most help.

Water sustainability

Energy sustainability

Disaster resiliency

Presentation and Lab

https://github.com/jaffaszt/jaffa-openshift101

Agenda

Containers

Docker

Orchestration

Kubernetes

Red Hat OpenShift

Lab Materials

redhat®

+

IBM

Do you use orchestration services, and if so which? (2016)

Do you use orchestration services, and if so which? (2017)

Do you use orchestration services, and if so which? (2018)

What are Containers?

OS level virtualization

- Isolate user space instances on a single host machine
- Gives isolated view of processes, user space and file system for user owner
- Share host Linux kernel
- A self-contained sealed unit of software
- Contains everything required to run the code

A Container Includes :
Code, configs, Processes, Networking, Dependencies, OS (just enough)

VM vs Container

Traditional Deployment **Virtualized Deployment** **Container Deployment**

Containers: Dev vs Ops

DEV	**OPS**
Code	Logging
Libraries	Remote Access
Config	Network Config
Runtime	Monitoring
OS	

Why Containers?

- Agile
- Continuous Deployment
- Dev and Ops
- Observability
- Consistency
- Management
- Microservices
- Resource Isolation
- Resource Utilization
- Portable
- Easy to manage
- Containers provide “just enough” isolation
- Immutable

What is Docker?

Docker is the world's leading containerization standard platform

Docker is a platform for developing, shipping and running applications using **container** virtualization technology.

What is Docker?

Image
Template to use for creating containers. Snapshot of Read-Only stored in Docker Hub。

Container
Basic unit on which application service runs。

Docker Hub
Store, distribute and share images of containers.
Establish and use in SaaS or enterprise.

Docker Repository
Pulling an image from Docker Trusted Registry is the same as pulling an image from Docker Hub or any other registry. Since DTR is secure by default, you always need to authenticate before pulling images.。

Docker Engine
A program for creating containers and running them.
Run on a physical server or virtual machine.
User can operate with command。

Docker :

Docker makes it really easy to install and run software without worrying about setup or dependencies

Common images available on the cloud

hub.docker.com

Dockerfile

```
FROM ubuntu:18.04
COPY . /app
RUN make /app
CMD python /app/app.py
```

Each instruction creates one layer:

- `FROM` creates a layer from the `ubuntu:18.04` Docker image.
- `COPY` adds files from your Docker client's current directory.
- `RUN` builds your application with `make`.
- `CMD` specifies what command to run within the container.

Layers of Docker container

Any RUN commands you specify in the Dockerfile creates a new layer for the container. In the end when you run your container, Docker combines these layers and runs your containers. Layering helps Docker to reduce duplication and increases the re-use.

Docker Commands

Example：

command	description
run	Run a new container from the image
start	Start a container that has stopped
stop	Stop a running container
exec	Execute a **second** command in the running container
ps	Browse the system's container
images	Browse the system image
build	Build the container from the build script Dockerfile
inspect	Check the container's system information
logs	Check the container's log

https://docs.docker.com/engine/reference/commandline/docker/

$ docker images
$ docker container ls
Images
Containers
local docker instance
My Computer

Docker registry

$ docker pull hello-world
Images
Containers
local docker instance
PULL
Docker registry
My Computer

$ docker load -i ~/backup/ubuntu_image.docker
Images
Containers
local docker instance
Load
backup.tar
My Computer

Docker registry

artifactory
npm
$ docker build
Dockerfile
WorkDIR
Images
Containers
Run
local docker instance
Load
backup.tar
Docker registry
My Computer

Images
Run
Containers
local docker instance
$ docker run hello-world
My Computer

Docker registry

How to manage containers?

How to manage containers?

amazon.com

NETFLIX

Container Orchestration !

- Scheduling
- Cluster management
- Service discovery
- Provisioning
- Monitoring
- Configuration management

What is Kubernetes?

- **Container Orchestrator**
  - Provision, manage, scale applications
- **Manage infrastructure resources needed by applications**
  - Volumes
  - Networks
  - Secrets
  - And many many many more...
- **Declarative model**
  - Provide the "desired state" and Kubernetes will make it happen
- **What's in a name?**
  - Kubernetes (K8s/Kube): "Helmsman" in ancient Greek

Some comparisons

Container Ecosystem Layers

What is Kubernetes?

At its core, Kubernetes is a database (etcd).
With "watchers" & "controllers" that react to changes in the DB.
The controllers are what make it Kubernetes.
This pluggability and extensibility is part of its "secret sauce".

DB represents the user's desired state
Watchers attempt to make reality match the desired state

"API Server" is the HTTP/REST front-end to the DB

More on controllers later...

Kubernetes resource model

- A resource for every purpose

Config Maps
Daemon Sets
Deployments
Events
Endpoints
Ingress
Jobs
Nodes
Namespaces
Pods
Persistent Volumes
Replica Sets
Secrets
Service Accounts
Services
Stateful Sets, and more...

Kubernetes aims to have the building blocks on which you build a cloud native platform.

Therefore, the internal resource model **is** the same as the end user resource model.

<u>Key Resources</u>

Pod: set of co-located containers

- Smallest unit of deployment
- Several types of resources to help manage them
- Replica Sets, Deployments, Stateful Sets, ...

Services

- Define how to expose your app as a DNS entry
- Query based selector to choose which pods apply

Kubernetes Architecture: How apps are accessed

Pod

- Smallest deployment unit – runs containers
- Each pod has its own IP
- Shares a PID namespace, network, and hostname

Service

- Collection of pods exposed as an endpoint
  - state and networking info propagated to all worker nodes
- Types of service exposure
  - ClusterIP – Exposes cluster-internal IP
  - NodePort – Exposes the service on each Node's IP at a static port
  - LoadBalancer – Exposes externally using a cloud provider's load balancer
  - ExternalName – Maps to an external name (such as foo.bar.example.com)

Kubernetes in Action!

1. User via "kubectl" deploys a new application
2. API server receives the request and stores it in the DB (etcd)
3. Watchers/controllers detect the resource changes and act upon it
4. ReplicaSet watcher/controller detects the new app and creates new pods to match the desired # of instances
5. Scheduler assigns new pods to a kubelet
6. Kubelet detects pods and deploys them via the container runtime (e.g. Docker)
7. Kubeproxy manages network traffic for the pods – including service discovery and load-balancing

Kubernetes in Action!

1. A request is sent to your app by using the public IP address of your worker node and the NodePort on the worker node.

2. The request is automatically forwarded to the NodePort service's internal cluster IP address and port. The internal cluster IP address is accessible inside the cluster only.

3. kube-proxy routes the request to the Kubernetes NodePort service for the app.

4. The request is forwarded to the private IP address of the pod where the app is deployed. If multiple app instances are deployed in the cluster, the NodePort service routes the requests between the app pods.

Kubernetes in Action – Deployment

controllers/nginx-deployment.yaml

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  labels:
    app: nginx
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.7.9
        ports:
        - containerPort: 80
```

Why Openshift ?

Red Hat OpenShift

OpenShift Architectural Overview

OpenShift Architectural Overview

Openshift resource model

Openshift resource model

OpenShift Overview

- Container Host & Runtime
- Enterprise Kubernetes
- Validated Integrations
- Integrated Container Registry
- Developer Workflows
- Access to Services

OpenShift Overview

- OpenShift Service Mesh
- OpenShift Serverless
- OpenShift Pipelines

Why Openshift ?

- OKD (Origin Community Distribution)
- github.com/openshift/origin
- 30,000+ commits, 350+ contributors

Workshop website

https://github.com/jaffaszt/jaffa-openshift101

IBM