TECH DAYS

Containers

An Introduction

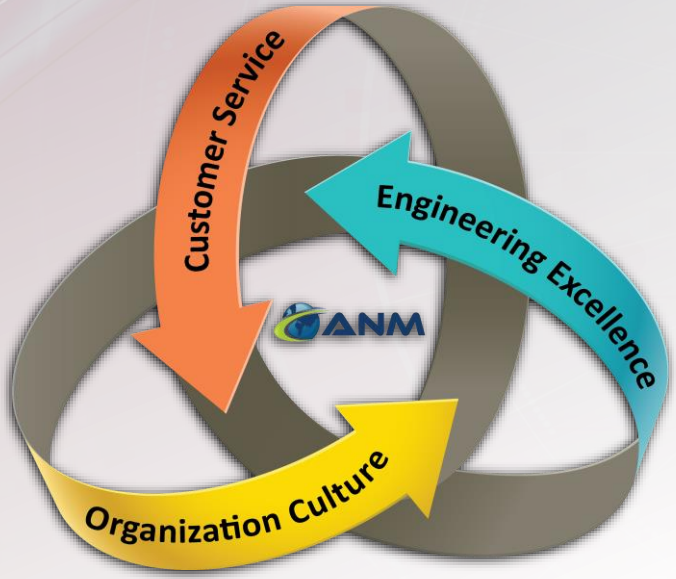
June 6, 2019

Ian Logan, Ian.Logan@anm.com
Sr. Systems Engineer, ANM
Klaus Mueller, Klaus.Mueller@anm.com
Solutions Architect, ANM

Presented by ANM

Sponsored by INGRAM MICRO

ANM: Committed to Innovation



- ☑ Excellence in IT Consulting, Since 1994
- ☑ Culture of Trust & Excellence
- ☑ 98.6% Customer Sat Rating (2018)
- ☑ Commitment to Innovation

TOP WORK PLACES 2018 — The Denver Post

TOP WORK PLACES 2018 — AlbuquerqueJournal

BPtW — 2016 BEST PLACES TO WORK

TOP WORK PLACES 2016 — AlbuquerqueJournal

Staying True to our Roots

- Local Leadership, Offices and Staff
- Close to Your Business
- Understanding Local Challenges
- Local Community Involvement

Committed to Innovation: Transformational Services

Delivering IT Expertise in Core Infrastructure:

- Campus and Data Center Networking
- Wired and Wireless Networking
- Data Center Compute and Storage (Converged & Hyperconverged solutions)
- Collaboration – voice, video, teams, conferencing, audio/visual design
- Virtualization and Cloud – VMware, AWS

CISCO

cisco Gold Partner

cisco Master Collaboration Partner

cisco Master Security Partner

aws partner network

vmware

- Information Security - 2016
- App Development (SalesForce) - 2016
- Cloud & Automation (AWS & RedHat) - 2017
- Technology Adoption - 2018

What *is* a container?

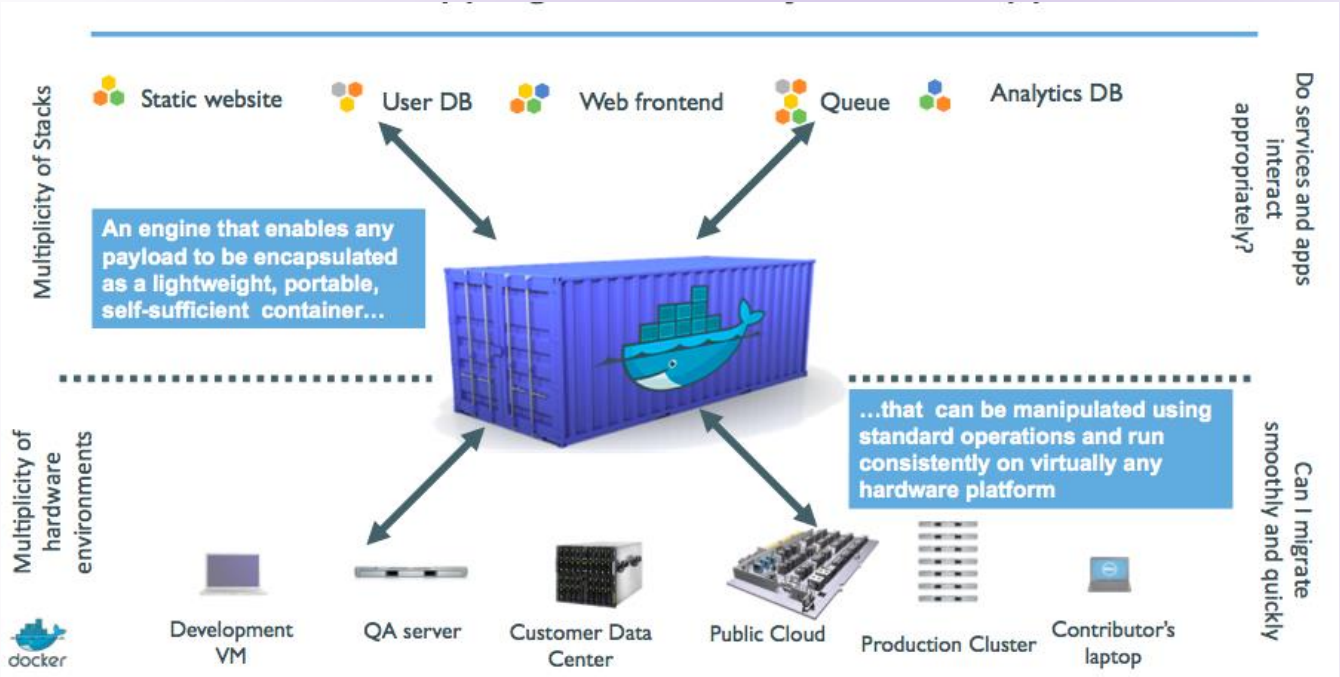
What *is* a container?

Shipping containers

- One size fits most loads
- Massive efficiency improvements
- Standard size and configuration

Source - https://container.training/intro-selfpaced.yml.html#25

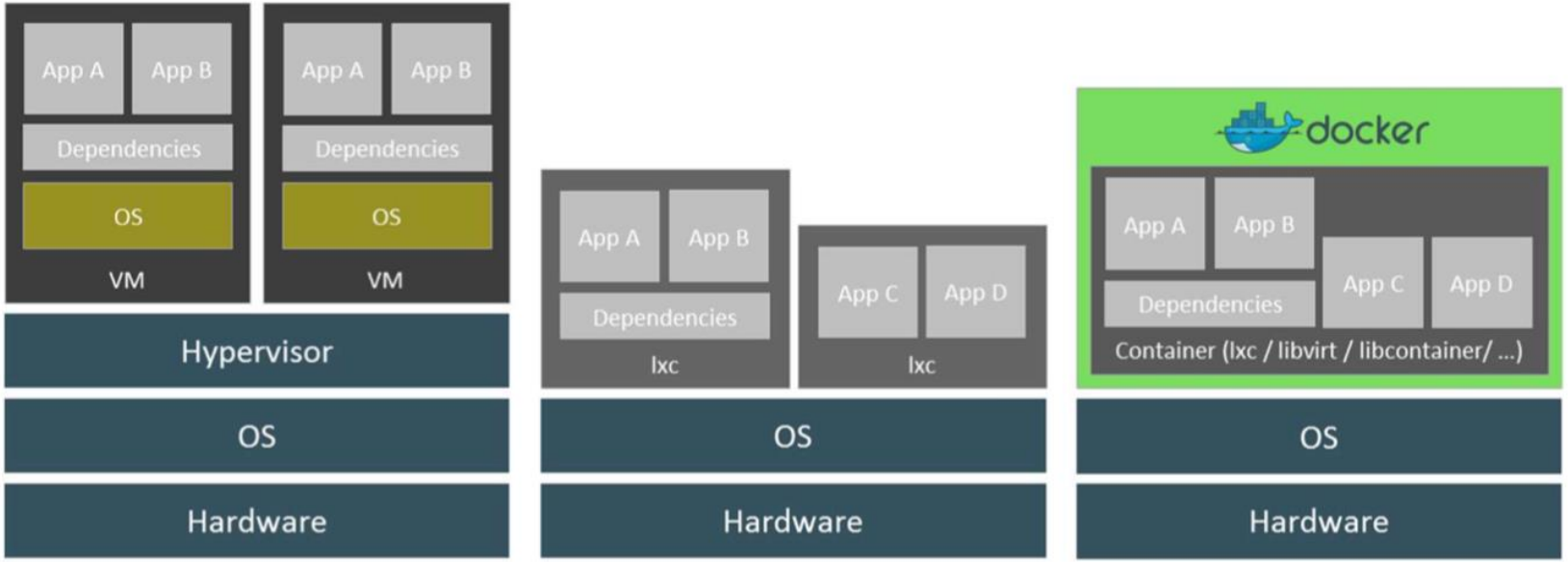
What *is* a container?



Containers:

- Portability for our applications
- Standard format
- Includes all the dependencies for the application
- Workload isolation at runtime

Source - https://container.training/intro-selfpaced.yml.html#27


App A
App B
Dependencies
OS
VM
App A
App B
Dependencies
OS
VM
Hypervisor
OS
Hardware
Type 2 Hypervisor
App A
App B
Dependencies
lxc
App C
App D
lxc
OS
Hardware
LXC
docker
App A
App B
Dependencies
App C
App D
Container (lxc / libvirt / libcontainer/ ...)
OS
Hardware
Docker

What *is* a container?

Compared to a virtual machine containers are

- Extremely lightweight
- Often stateless
- Somewhat OS agnostic
  - Run an Ubuntu flavored container on CentOS
  - Include just the binaries and libraries your app needs, run on any new enough Linux kernel

What *is* a container?

Escape dependency hell

1. Write installation instructions into an `INSTALL.txt` file
2. Using this file, write an `install.sh` script that works *for you*
3. Turn this file into a `Dockerfile`, test it on your machine
4. If the Dockerfile builds on your machine, it will build *anywhere*
5. Rejoice as you escape dependency hell and "works on my machine"

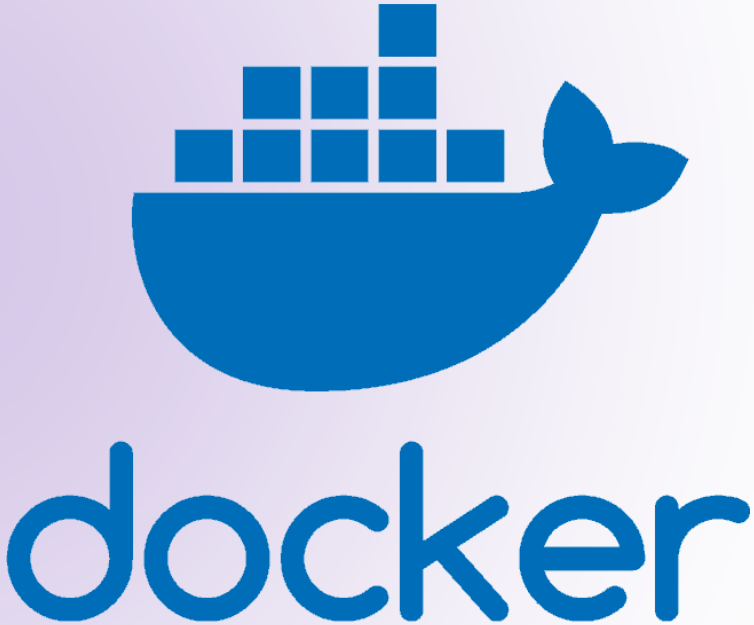
Never again "worked in dev - ops problem now!"

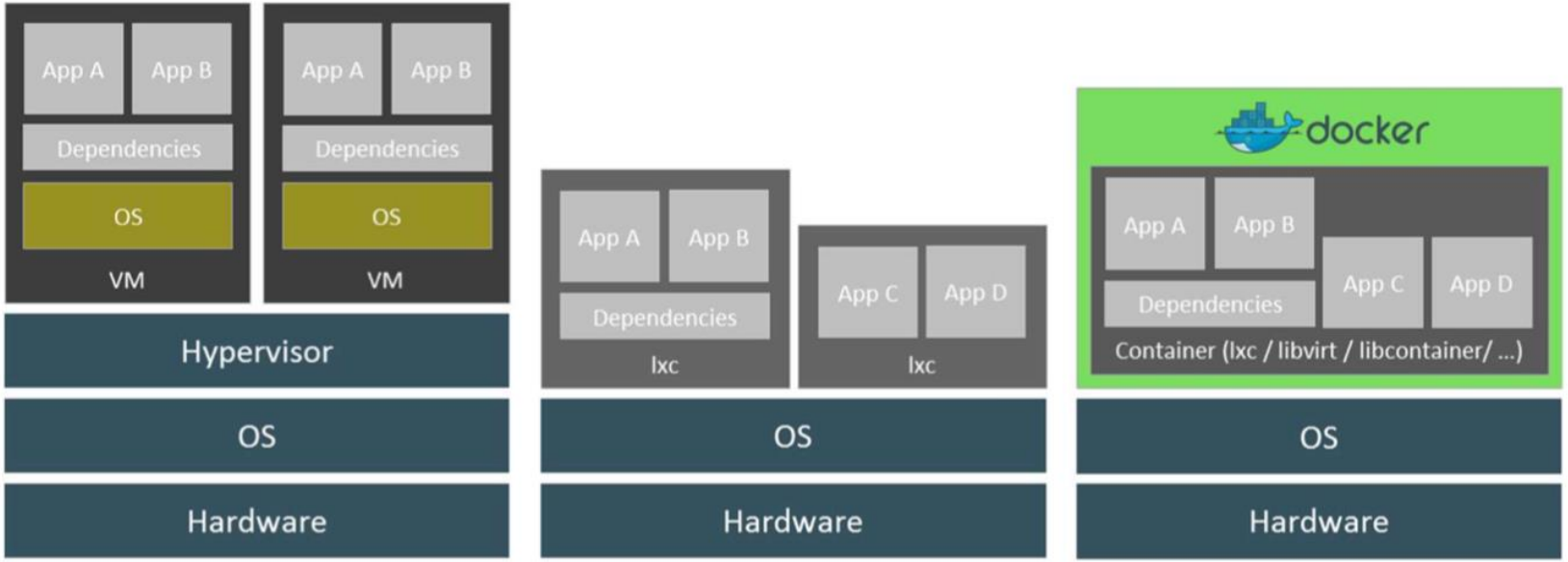
Source: https://container.training/intro-selfpaced.yml.html#31

What is Docker?

Docker

- Made using containers easy.
- Originally Linux centric but runs on many platforms today.
- Includes tools for defining a container, and distributing them (the registry).
- Has become widely used standard for building and packaging applications as containers.

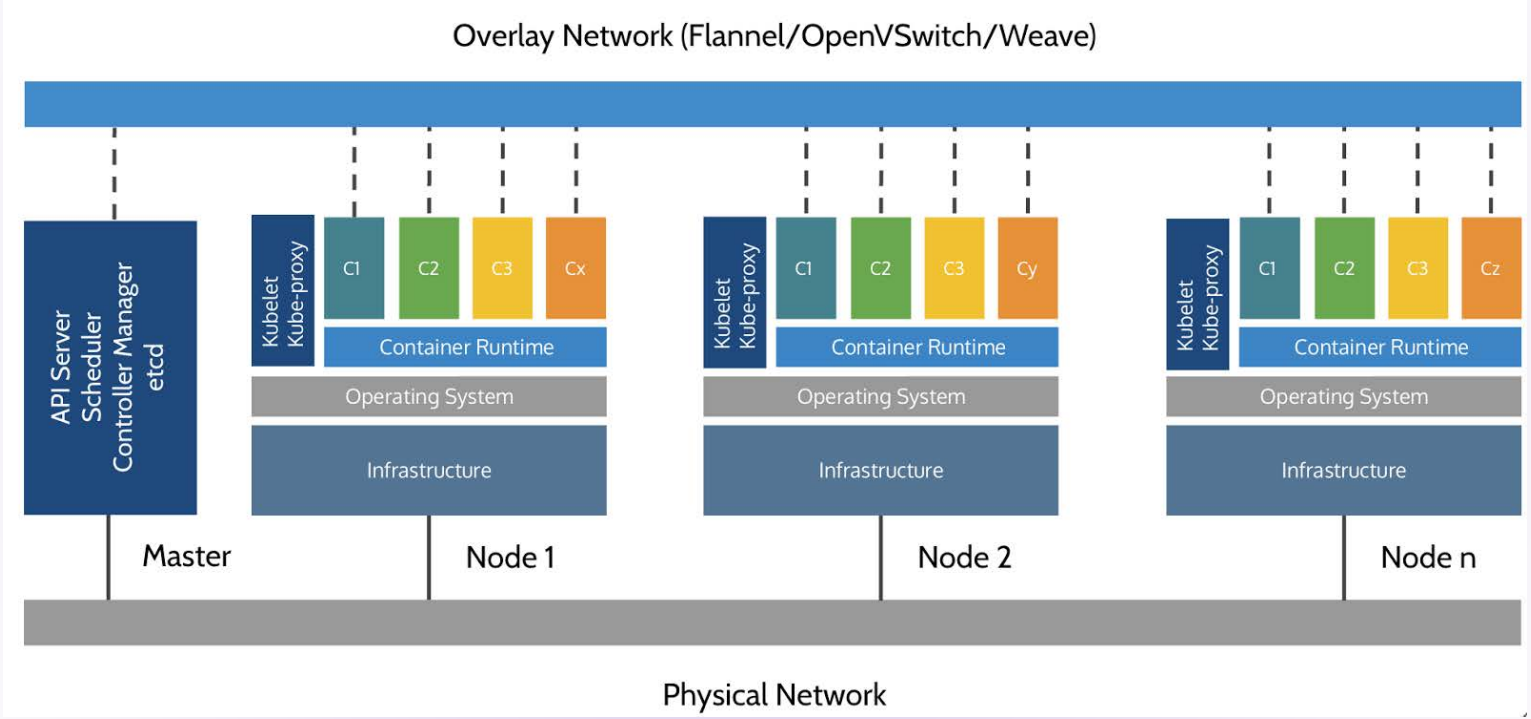



App A
App B
Dependencies
OS
VM
App A
App B
Dependencies
OS
VM
Hypervisor
OS
Hardware
Type 2 Hypervisor
App A
App B
Dependencies
lxc
App C
App D
lxc
OS
Hardware
LXC
docker
App A
App B
Dependencies
App C
App D
Container (lxc / libvirt / libcontainer/ ...)
OS
Hardware
Docker

What is Kubernetes?

- Open source container cluster manager
- Used as a backend in Google's App Engine
- Runs on Private and Public Clouds, and even on Bare metal
- Becoming the de-facto standard for managing and orchestrating container workload clusters

Kubernetes Cluster



Source: https://container.training/kube-halfday.yml.html#74

Key definitions

- **Container** – lightweight standardized unit of software
- **Repository** – An online resource for hosting pre-made containers (images)
- **Cluster** – one or more nodes hosting containers
- **Orchestration** – software responsible for assigning containers to nodes in a cluster, and maintaining the health of those containers (i.e. restart dead containers)

How did we get to containers?

The explosion of computing in the campus

Sun microsystems

digital

SiliconGraphics

IBM

1950s 1960s 1970s 1980s 1990s

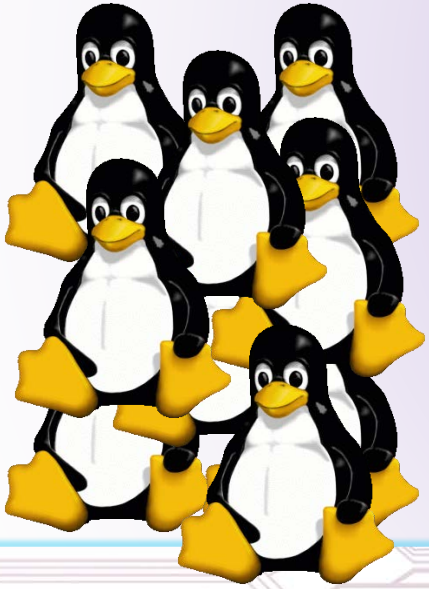
The explosion of computing on the campus

As computers have become more affordable and applications have grown in complexity we've seen:

- Application stability suffers as complexity grows.
- Application/workload isolation becomes critical.
- Applications often can't co-exist on the same server (Hi Java!).
- Servers became affordable enough to dedicate them to a single application.

How did we get to containers?

1990s – 2000s



Application isolation leads to server sprawl

- Most servers are 75-90% idle
- System administrators are stretched thinner and thinner
- Developers have to wait on system admins to order servers, and install software, sometimes for months

How did we get to containers?

2000s

Container technology is developed

- FreeBSD jails are introduced March 2000 in FreeBSD 4.0
- Solaris zones are introduced in 2005 with Solaris 10
- Google supports the development of name spaces in the Linux kernel (a key component of containers)
- Docker is founded in 2010 making Linux containers easy for developers to consume

How did we get to containers?

2010-today

What made Docker special?

- Docker makes it easy for a developer to spin up a container and run it almost anywhere
- Docker with its registry becomes the application delivery system

Downsides to plain Docker

- Docker doesn't address the IT operations pain points around availability and reliability

How did we get to containers?

2010-today

Google releases Kubernetes in 2013

- Kubernetes is an open source system based on Google's internal systems called Borg and Omega
- Kubernetes is a container management platform that addresses most of IT operations pain points while still providing the agility developers crave

Container use cases

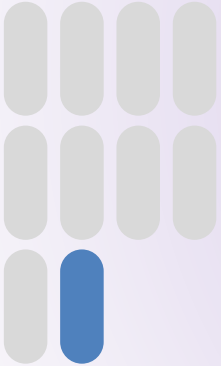
Containers can be used in many different ways, they broadly divide into two categories:

- Empowering developers with APIs for consuming infrastructure.
- Software packaging and distribution.

This has become the standard application design in the devops/cloud native world. Microservice architectures being the prime example of this design.

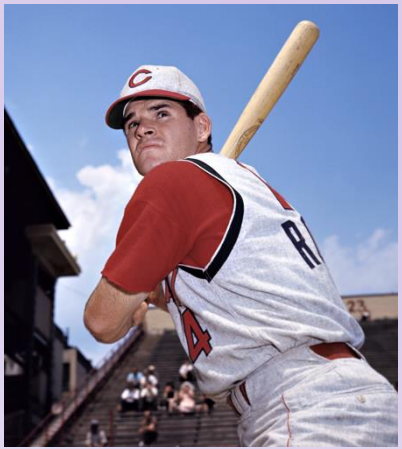
Empowering Developers

At Bats Matter: The VC 10% Rule



Quarterly Releases

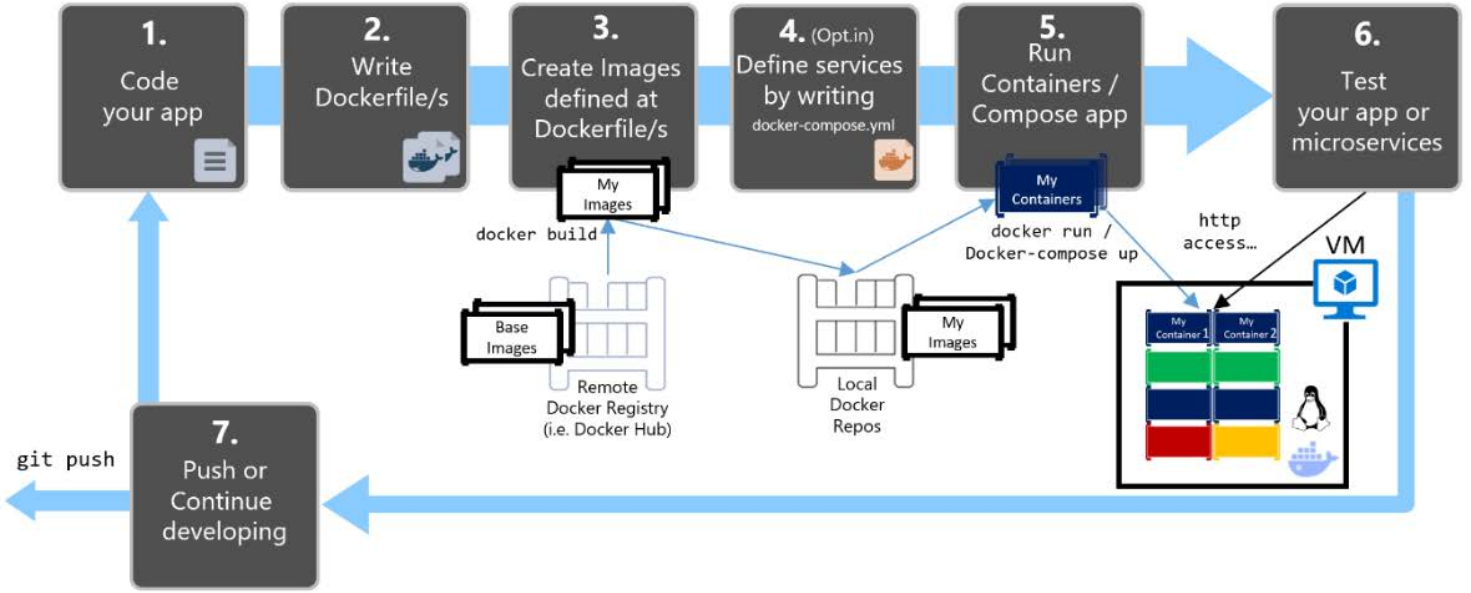
1 Innovation Every 2.5 Years



Monthly Releases

3 Innovations Every 2.5 Years

Inner-Loop development workflow for Docker apps



Source: https://docs.microsoft.com/en-us/dotnet/standard/microservices-architecture/docker-application-development-process/docker-app-development-workflow

Containers and software distribution

- Pre-built containers (images) are distributed via open and private registries.
- Registries are collections of downloadable pre-built images.
  - Hub.docker.com (public)
  - Amazon Elastic Container Registry (private)
  - Google cloud container registry (private)

Container architecture

- Containers were originally intended to be stateless.
- Containers can be ephemeral, being created for a quick task and then going away when its done.
  - Example: resizing an image for a webserver dynamically.
- Containers typically only have one (or a few) processes in them, the ideal microservices design.

Container architecture – stateless design

- Kubernetes' orchestration engine provides the ability to dynamically scale your environment up and down.
- Scaling dynamically is *much harder* when there is local state in the container.

Containers and Networking

Container Networking can refer to some very different use cases.

- How do we network our containers together?
- Running containers on our network infrastructure.

Networking for containers

- Do we need to share a layer 2 segment between containers on different nodes?
  - VXLAN overlays
- Do we need IP routing?
  - BGP, OSPF, static
- Do we need to provide IP services for containers?
  - Load balancing, NAT, DNS, DHCP
- Both open source and vendor solutions are available to enable advanced networking capabilities.
  - E.g. Contiv, Weave, Flannel, Vmware NSX-T, Cisco ACI, etc.

Containers on switches

Linux based switch operating systems with the ability to run containers as a service:

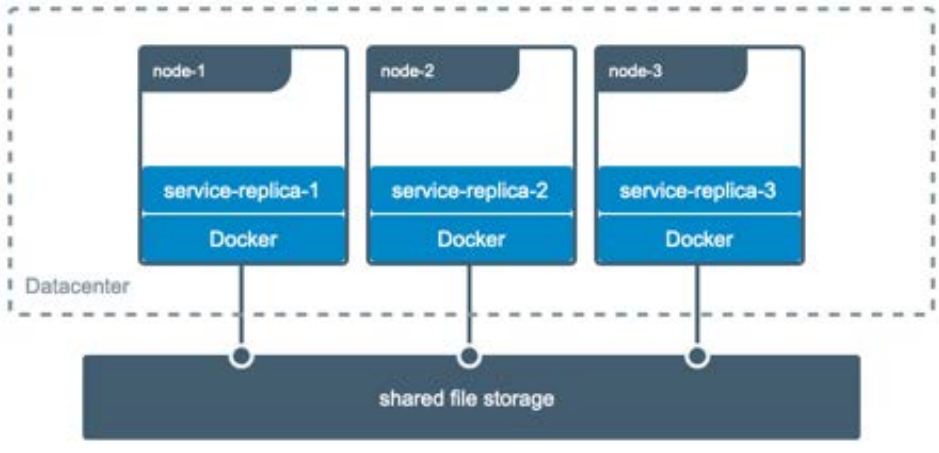
- Cisco IOS XE
- Arista EOS
- Cumulus Linux

Example use cases

- Performance monitoring (PerfSONAR)
- Configuration management (Puppet agent)
- Troubleshooting tools (tcpdump, Wireshark)

Containers and Storage

- Containers were originally stateless.
- Volumes were added afterwards to add stateful storage.
- Docker and Kubernetes both support volumes.
- There are many different kinds of volumes to chose from.
- Not all volumes can be shared with multiple containers/pods.
- There are multiple opensource and commercial solutions available.
    - E.g. Cisco Hyperflex HCI integration, NetApp, Amazon S3 integration, native open source.



Kubernetes commercial products

You can build your own Kubernetes environment using open source components or you can buy a ready-made distribution.

- Cisco Container Platform
- RedHat OpenShift
- Pivotal Container Service (PKS)

An example of commercial Kubernetes

Kubernetes-as-a-Service

Setup

- Deploy Kubernetes clusters on HyperFlex IaaS (VMware)
- Container Networking (Contiv / ACI)
- Persistent storage (Flex Driver)
- Layer-4 and Layer-7 load balancing
- High availability

Consume

- Authentication with Active Directory
- Role based access control
- Communication between containers and external VMs / BMs
- UI – Kubernetes, API
- Security (policies, encryption)

Manage

- Add / remove Kubernetes nodes
- Lifecycle management (OS updates, Kubernetes upgrades)
- Monitoring (Prometheus)
- Logging (EFK)

Buy it or build it?

Buy a commercial distribution of Kubernetes if:

- You're in a hurry to get to production.
- You want technical support for production environments.
- You don't want to become an expert on the intricacies of running a Kubernetes environment.

Kubernetes key concepts

Kubernetes key concepts - Nodes

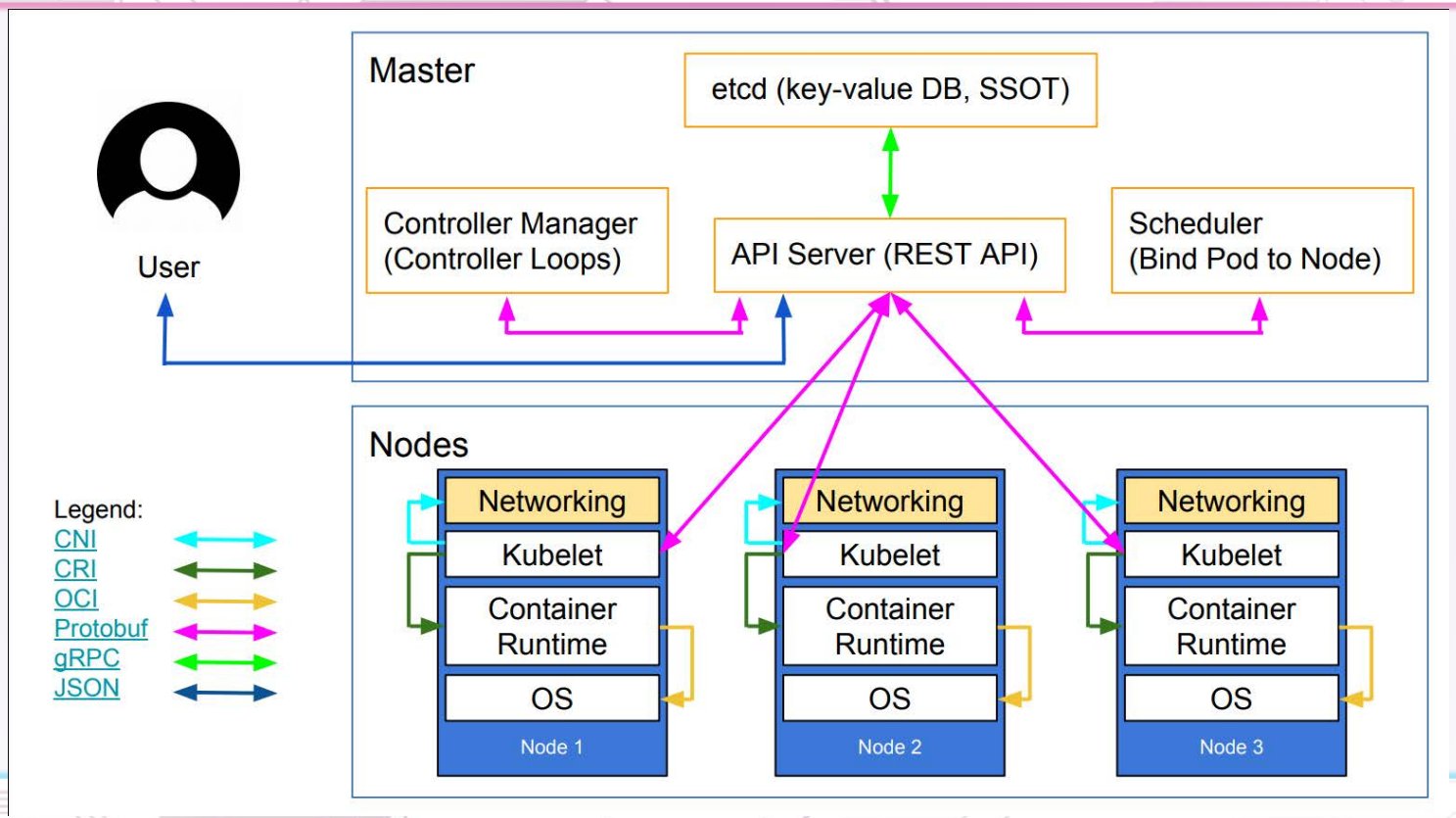
Nodes

- Worker machine in Kubernetes
- Can be physical or virtual
- Used to be called a minion

Master Node

- Runs the k8s control plane
- Can be replicated for high availability

Kubernetes Control Plane



https://container.training/kube-halfday.yml.html#78

Kubernetes Control Plane

The control plane is all about maintaining your desired state. For example:

- If a pod should have 3 copies of a container, it will make sure it always has *exactly* 3.

Kubernetes key concepts – Objects and Names

Objects

- Persistent entities in k8s system
- Containers, resources, policies

Names

- Every object has a unique name which is client provided
- Every object also receives a unique UID automatically

Kubernetes key concepts – Namespaces and Labels

Namespaces

- Multi-tenancy construct that allows for the reuse of names by different tenants
- Also a technology inside the Linux kernel for isolating processes

Labels

- Non-unique strings assigned to objects to group them by role or function
- Examples
  - Tier=production
  - App=web_server
- Policies can reference labels

Kubernetes key concepts – Containers and images

Images

- The collection of binaries, libraries, and other files needed to run an application

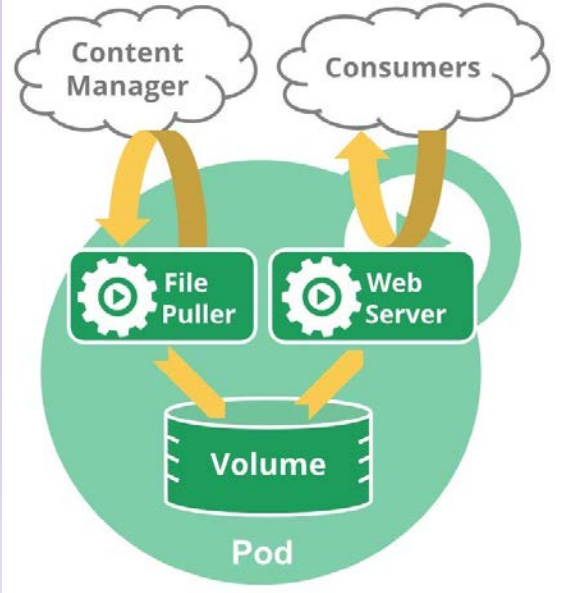
Container

- A running application with its associated resources (i.e. the image)
- Short lived and usually stateless

Kubernetes key concepts – Pods

Pods

- The smallest schedulable resource in k8s
- One or more containers live in a pod.
- All containers within a pod always run on the same Node
- All containers within a pod see the same files and can communicate via IPC, shared files, network stack on localhost
- Each Pod has its own unique intra-cluster IP
- Note: each container in the Pod shares the pods IP.



Kubernetes key concepts – Containers and images

Replicas

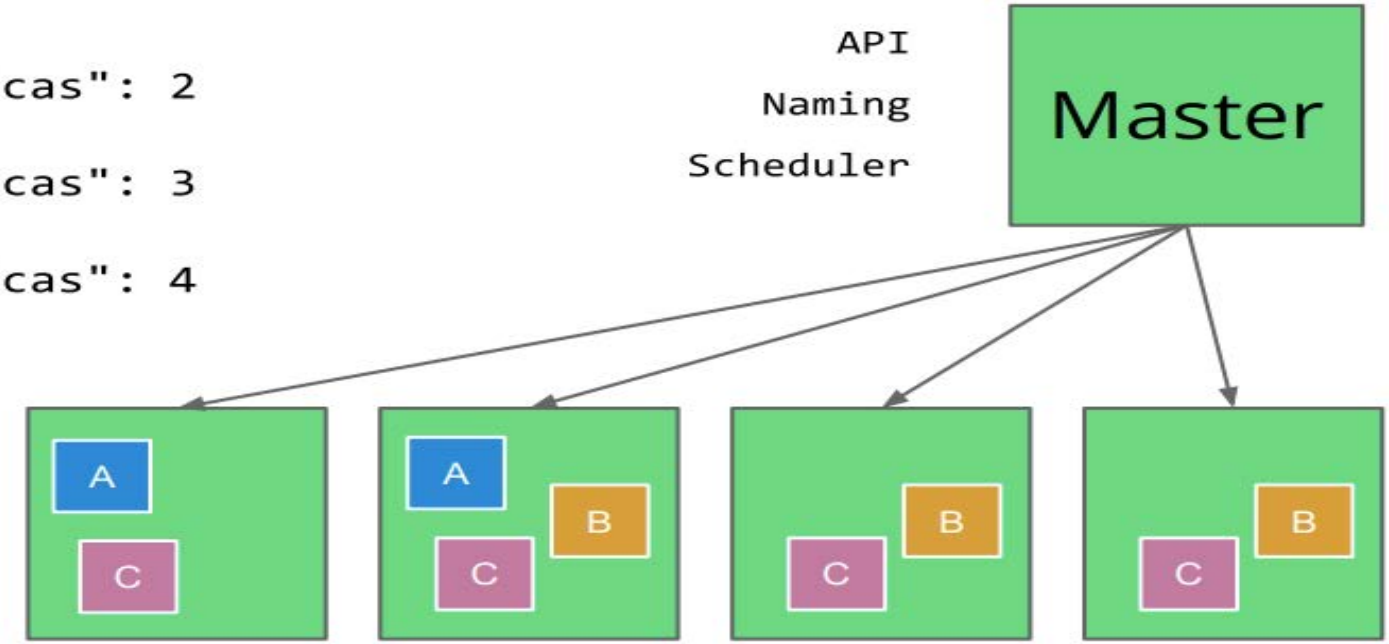
- Multiple copies of a pod.
- Normally scheduled on different nodes.
- Remember containers are normally stateless.

KUBERNETES ARCHITECTURE

A => "replicas": 2

B => "replicas": 3

C => "replicas": 4



Source: https://www.slideshare.net/ZoharStolar/introduction-to-containers-running-dockers-using-kubernetes

Kubernetes key concepts – Services and Volumes

Services

- Any time we expose a Pod to the network outside of the Pod we call that a service
- Services can have 3 types of connectivity
    - Intra-cluster only
    - Node TCP/UDP port
    - Externally Load Balanced

Volumes

- Storage that is associated with a pod
- Over 15 different types of volumes tailored to different use cases

Kubernetes key concepts – CNI

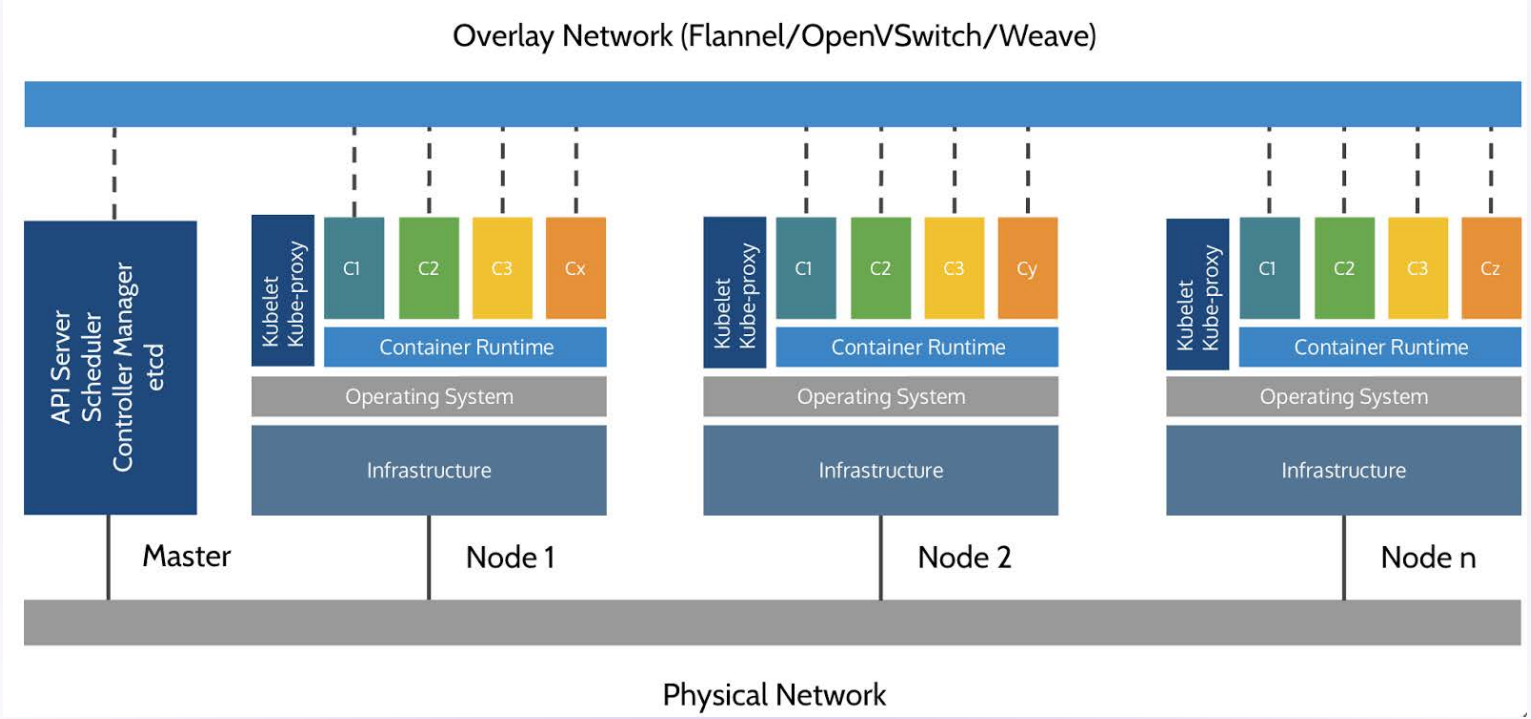
Container Network Interface (CNI)

- A generic plugin interface that allows for a wide variety of networking configurations.
- Anything from simple NAT to full routing with BGP is possible.
- Use VLANs, VXLAN, Geneve for Layer 2 between pods.

Network plugin

- A program that is run by the container management system.
- Creates a virtual ethernet interface for each container namespace and configures the cluster network.

Kubernetes Cluster



Source: https://container.training/kube-halfday.yml.html#74

Contiv – an example CNI plugin

Rich Policy Framework

Set bandwidth and isolation policies in a multi-tenant environment.

Multi-Platforms

Docker, Kubernetes, OpenShift and more.

Multi-Infrastructure

VMs, containers, and bare metal.

Enterprise Grade

Rigorously tested for the cloud.

Networking Support

Layer 2, Layer 3, BGP, ACI

Open Source

Contiv is available through the Apache 2 License and our code is available on GitHub.

Kubernetes – Lets build a pod

This is the specification for a pod

- Every spec must call out the apiVersion, kind, a name, and spec.
- The specification is similar to what we see in docker.
- This will download a MongoDB binary, and all of the Linux support binaries and librarys needed for MongoDB to run in the container

```
1   apiVersion: v1
2   kind: Pod
3   metadata:
4     name: db
5     labels:
6       type: db
7       vendor: MongoLabs
8   spec:
9     containers:
10    - name: db
11      image: mongo:3.3
12      command: ["mongod"]
13      args: ["--rest", "--httpinterface"]
14
```

Kubernetes – Lets start a pod

Now let's create the pod

- The pod specification is saved in db.yml
- We'll use the kubectl command to create the pod
- "kubectl create –f db.yml"

```
→  k8s-specs git:(master) cat pod/db.yml
apiVersion: v1
kind: Pod
metadata:
  name: db
  labels:
    type: db
    vendor: MongoLabs
spec:
  containers:
  - name: db
    image: mongo:3.3
    command: ["mongod"]
    args: ["--rest", "--httpinterface"]
→  k8s-specs git:(master) kubectl create -f pod/db.yml
pod/db created
→  k8s-specs git:(master)
```

Kubernetes – the kubectl command

The **kubectl** command is

- Our primary tool for interacting with k8s
- Developers can also write code to interact with the API if desired, but ops will mostly use kubectl or the dashboard

Resources to learn more about Kubernetes

Resource list

- Minikube – Kubernetes on your laptop
- Docker also runs great on a laptop
- Victor Farcic's Kubernetes live class at safarionline (runs every 2-3 months)
    - The free trial is long enough to attend the class

Resource list

- Cisco DevNet
  - https://developer.cisco.com
  - Cisco Container Platform sandbox
  - Catalyst 9000 sandbox
- Vmware Hands on Lab for NSX-T
  - https://labs.hol.vmware.com
  - HOL-1926-01-NET, HOL-1926-02-NET

Resource list

- Building Catalyst 9000 containers
  - https://github.com/maccioni/centos-vm-on-ios-xe
- Kubernetes docs
  - https://kubernetes.io/docs/
  - https://kubernetes.io/docs/concepts/
- O'Reilly Press
  - Kubernetes: Up and Running
  - Managing Kubernetes

Key take-aways

- Container technology like Docker and Kubernetes empower developers to be self service, improving turn around time between releases.
- Kubernetes provides IT operations tooling for building highly available and reliable container environments.
- Containers is a new software packaging and distribution format we will all need to know at some level.

ANM can help

www.anm.com

AM – Adam.Chavez@anm.com

We have partnerships with Cisco, Vmware, NetApp, RedHat…

- Cisco HyperFlex HCI platform
- Cisco Container Platform
- Vmware PKS
- Cisco ACI and Vmware NSX-T
- And more

Thank You!

Kubernetes – a few more kubectl commands

kubectl delete

- Delete a pod
- Kubectl delete –f <pod_description.yml>
- Kubectl delete <pod name>

Kubectl describe –f <pod_description.yml>

- Report all the configuration and present state information known the k8s for that pod

Kubernetes – a few more kubectl commands

Kubectl exec –it <pod> <command>

- Execute an interactive command (i.e. a shell) in the container
- “Kubectl exec –it –c <container> <pod> <command> “ if your pod has multiple containers in it

Kubernetes – checking pod status

```
→  k8s-specs git:(master) kubectl get pods
NAME   READY   STATUS    RESTARTS   AGE
db     1/1     Running   0          111s
→  k8s-specs git:(master) kubectl get pods -o wide
NAME   READY   STATUS    RESTARTS   AGE     IP           NODE       NOMINATED NODE   READINESS GATES
db     1/1     Running   0          2m34s   172.17.0.5   minikube   <none>           <none>
→  k8s-specs git:(master)
```

Kubectl get pods

- Lists all currently defined pods
- Use –o wide to get more details on each pod

Kubernetes – executing commands in a pod

```
→  k8s-specs git:(master) kubectl exec db ps augxww
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root         1  0.5  1.9 286088 58920 ?        Ssl  03:35   0:03 mongod --rest --httpinterface
root        24  0.0  0.0  17500  2092 ?        Rs   03:45   0:00 ps augxww
→  k8s-specs git:(master)
```

Kubectl exec db ps augxww

- Kubectl exec <pod> <command> <args>
- Here we see that the only two processes running in that container are MongoDB and our ps command. Process isolation!

Kubernetes – Exposing network ports

```
apiVersion: apps/v1beta2
kind: ReplicaSet
metadata:
  name: go-demo-2-db
spec:
  selector:
    matchLabels:
      type: db
      service: go-demo-2
  template:
    metadata:
      labels:
        type: db
        service: go-demo-2
        vendor: MongoLabs
    spec:
      containers:
      - name: db
        image: mongo:3.3
        ports:
        - containerPort: 28017

---

apiVersion: v1
kind: Service
metadata:
  name: go-demo-2-db
spec:
  ports:
  - port: 27017
  selector:
    type: db
    service: go-demo-2
```

In this spec file:

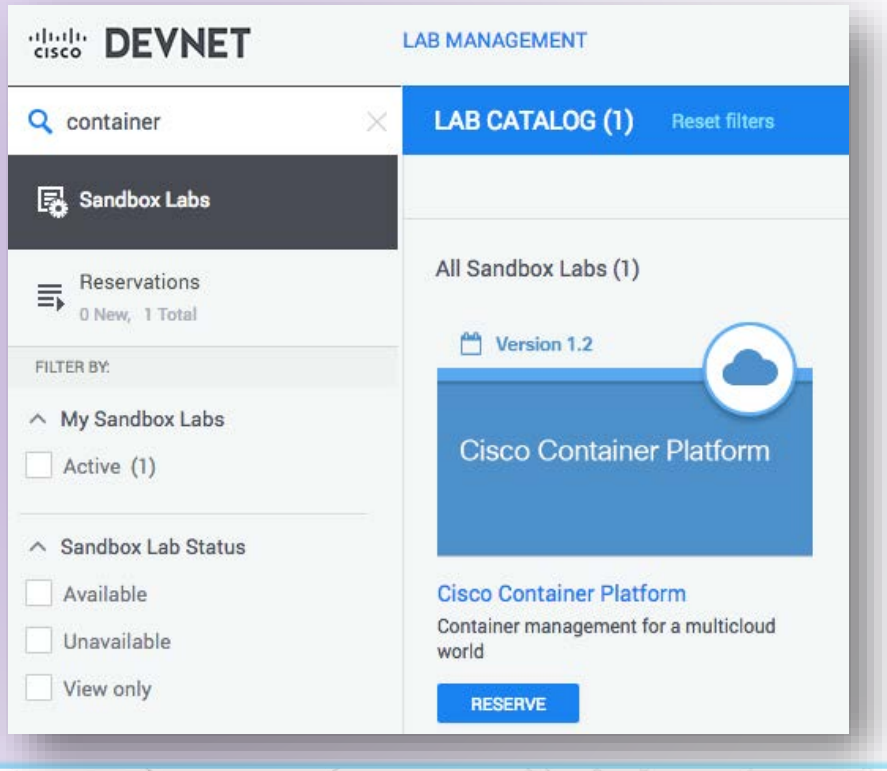
- We create the db container again, with a replicaset for high availability
- We define a k8s service named “go-demo-2-db” that will run on port 27017
- We can also use the ”kubectl expose” command to define a service
- Services can be made available to
  - Other pods in the cluster
  - The external network
  - An external load balancer (ie AWS ELB)

CCP Demo

Explore yourself:

devnetsandbox.cisco.com

Search for “container”

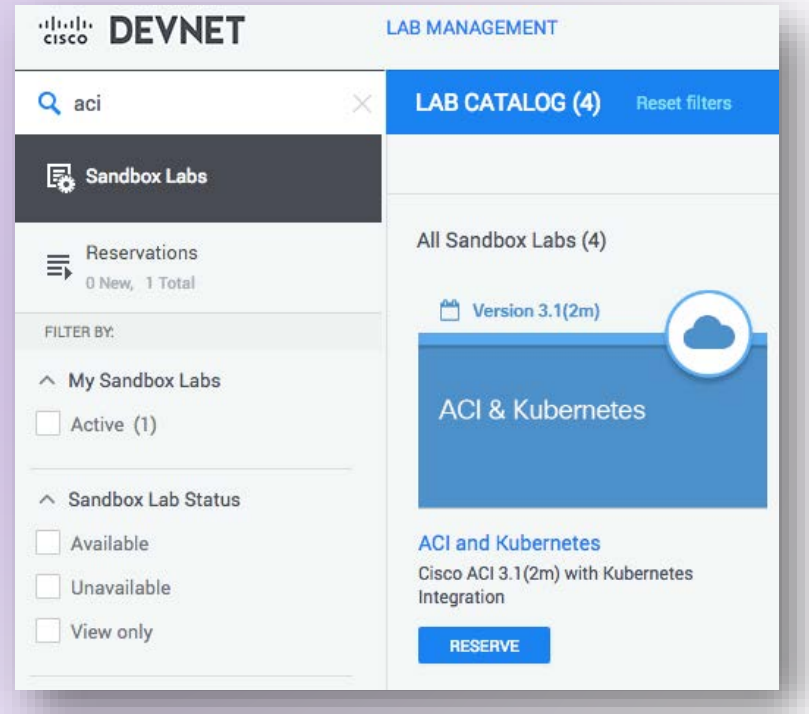


ACI Integration Demo

Explore yourself:

devnetsandbox.cisco.com

Search for “aci”



Running containers on IOS XE

You can try this out even if you don't have a Catalyst 9000 available!

DevNet Catalyst 9000 Sandbox.

- To reserve the sandbox: go to https://devnetsandbox.cisco.com/
- click on Networking on the right pane
- select IOS XE on Catalyst 9000
- click Reserve and follow the instructions.

Running containers on IOS XE

Create a DHCP scope to provide the container with an IP address

```
configure terminal
 ip dhcp excluded-address vrf Mgmt-vrf 192.168.1.1 192.168.1.10

 ip dhcp pool iox-apps2
  vrf Mgmt-vrf
  network 192.168.1.0 255.255.255.0
  default-router 192.168.1.1
  domain-name arm.com
  dns-server 8.8.8.8
```