Docker, Microservices And Kubernetes

training.mirantis.com

Copyright © 2016 Mirantis, Inc. All rights reserved

Presenter Introduction

Name: Reza Roodsari

Email: rroodsari@mirantis.com

Containers, Docker, And Kubernetes

MIRANTIS
Pure Play OpenStack

1. What Containers are
   - How containers make your infrastructure more efficient
   - What is Docker and how it is different from containers
2. What Microservices are
   - How microservices change the application landscape
3. What Kubernetes is
   - How container orchestration makes your applications more resilient
4. How OpenStack and Kubernetes relate to each other

Containers

Operating System Level Virtualization

Application Deployment Models

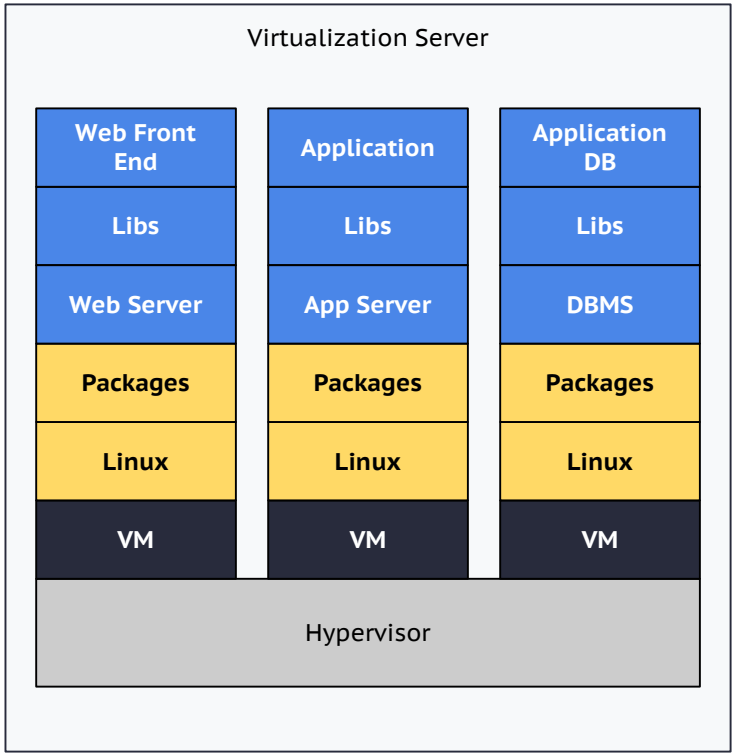
- Different deployment models exist:
  - Model 1: Physical Server
    - Application(s) runs natively on a physical host
  - Model 2: Virtualized Server
    - Application(s) runs within a virtual machine
  - Model 3: Containerized Server
    - Application(s) run within operating system containers

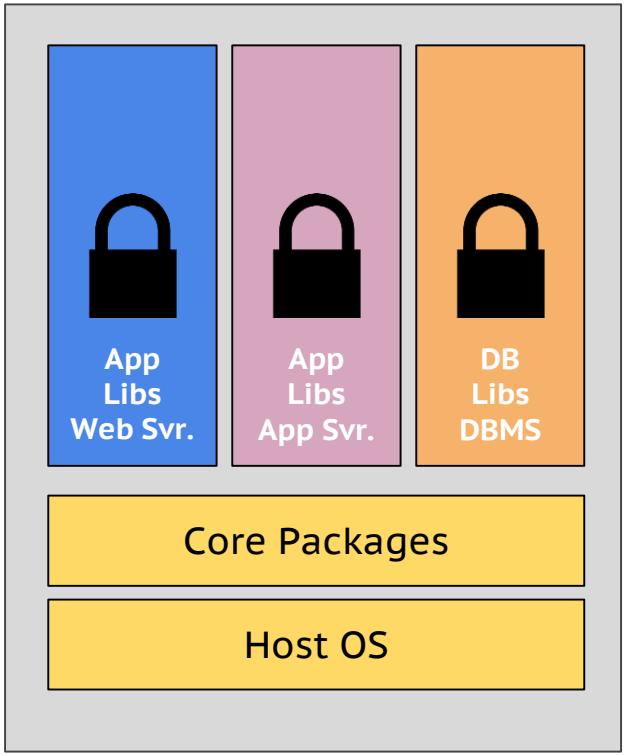
Operating System Container Overview

MIRANTIS
Pure Play OpenStack

- Typical Operating System (OS) has a single user-space
- OS-level virtualization extends the OS kernel to support multiple isolated user-space instances, called containers
- An OS within an OS
- Containers have low overhead compared to Server Virtualization because there is no h/w or s/w “emulation”
- Containers have less flexibility, as they cannot host a guest OS different from the host OS

Virtualized vs. Containerized Architecture





Operating System Container History

There are many container implementations, with varying degrees of capabilities and isolation

- The earliest implementation, chroot, dates back to 1982. It is not full featured, and lacks security features
- Virtuozzo/Parallels proprietary implementation is a full feature implementation dating back to 2000
- OpenVZ (Open Virtuozzo) GNU GPLv2 is a full feature implementation dating back to 2005
- FreeBSD jail BSD license is a full feature implementation dating back to 2000

Linux cgroups and namespaces based Containers

MIRANTIS
Pure Play OpenStack

- Latest generation of Linux containers are based on Linux cgroups and Linux namespaces
- One popular cgroups/namespaces based Linux container is LXC, introduced in 2008 by Canonical. A more recent addition is LXD based on LXC, with a REST API
- Docker also uses cgroups and namespaces, originally based on LXC, but now based on runc/libcontainer
- Rkt from coreos is also based on cgroups and namespaces

Linux Control Groups (cgroups)

MIRANTIS
Pure Play OpenStack

- Control groups or cgroups was contributed to Linux by Google engineering in 2007
- Is a feature of Linux kernel to limit, prioritize, account for, and isolate OS resources:
  - Resource limitations such as memory utilization
  - Resource prioritization such as CPU, GPU, disk and network I/O
  - Utilization accounting which can be used for billing purposes
  - State control allowing process groups to be frozen and restarted

Linux namespaces

- Namespaces of the Linux kernel isolate and virtualize system resources for a collection of processes.
  - **`mnt`**: Mount points
  - **`pid`**: Processes and their properties
  - **`net`**: A complete network stack
  - **`ipc`**: Inter-process communication
  - **`uts`**: Hostname and domain name
  - **`user`**: User IDs and group IDs

Linux Namespace Evolution

MIRANTIS
Pure Play OpenStack

No exact date for the namespaces feature, as there are multiple namespaces and they were added over time

Namespace	Availability	Kernel	Constant	Isolates
Mount	2002	Linux 2.4.19	CLONE_NEWNS	Mount points
IPC	2006	Linux 2.6.19	CLONE_NEWIPC	System V IPC, POSIX message queues
UTS	2006	Linux 2.6.19	CLONE_NEWUTS	Hostname and NIS domain name
Network	2007	Linux 2.6.29	CLONE_NEWNET	Network devices, stacks, ports, etc.
PID	2008	Linux 2.6.24	CLONE_NEWPID	Process IDs
User	2012	Linux 3.8	CLONE_NEWUSER	User and group IDs
Syslog	2013	Linux 3.8	SYSLOG_ACTION_NEW_NS	Kernel message, console behavior, requires user namespace
Cgroup	2016	Linux 4.6	CLONE_NEWCGROUP	Cgroup root directory

Why OS Containers Are Important

MIRANTIS
Pure Play OpenStack

- Containers are important because they fundamentally changes the way we virtualize workloads and application.
- Containers are faster, more portable and can scale more efficiently than hardware virtualization
- It took decades to add isolation and security features to Linux for Containers to be a viable alternative to VMs
- Now that we are almost there, expect this trend to continue, not slow down, for Containers to eventually replace most use-cases involving VMs

Container Adoption in Numbers

MIRANTIS
Pure Play OpenStack

Container adoption

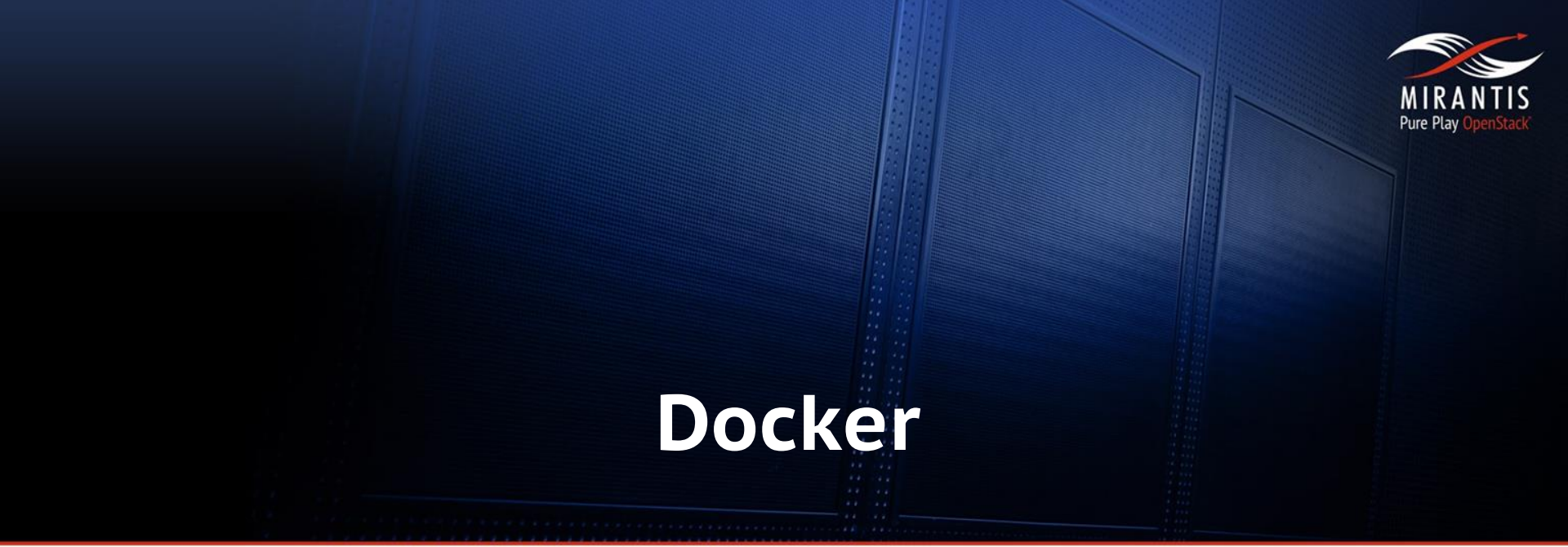
- 79% of organizations use container technologies
- 76% of them in production environments

The biggest drivers of container adoption

- 39% to increase developer efficiency
- 36% to support microservices

Operating system

- GNU/Linux is the dominant platform
- <2% use Windows



Docker

Containerized Application Deployment

OS Image vs. Application Image Containers

MIRANTIS
Pure Play OpenStack

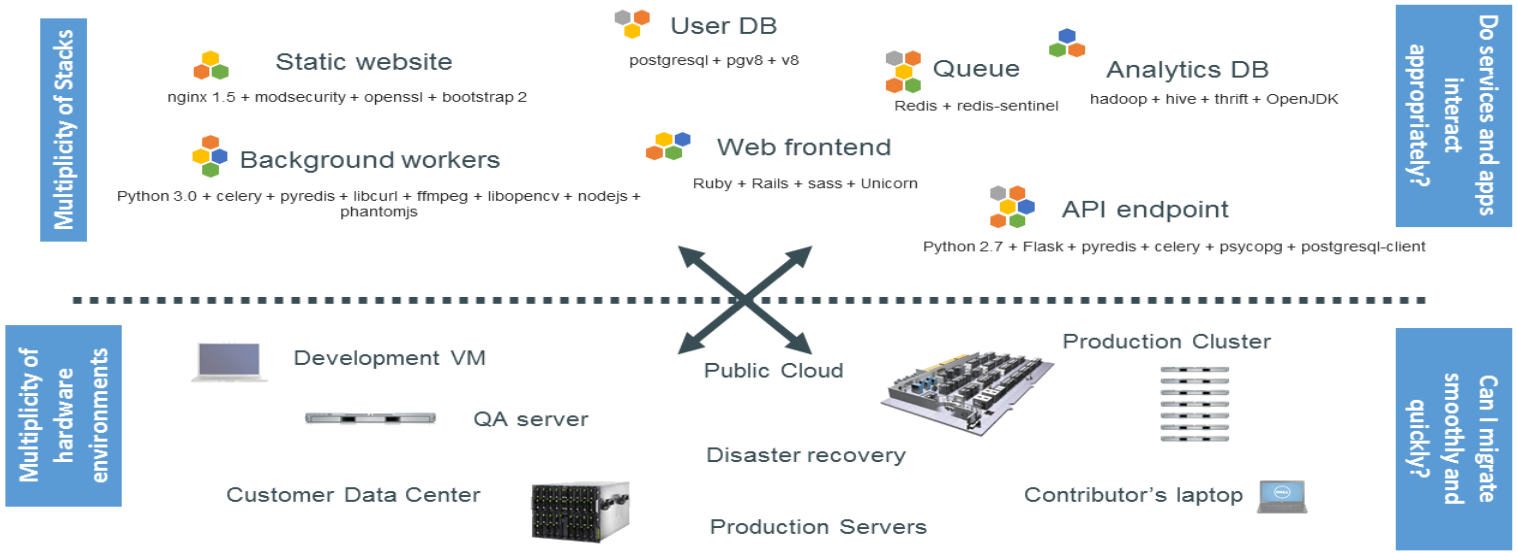
- LXC/LXD and OpenVZ are full OS Containers:
- Container is like a VM with a fully functional OS
- Container is filesystem neutral – persistent data can be saved inside or outside the container
- Docker and RKT are Application Containers:
- Container relies on a union filesystem, made of read-only layers via AUFS/Devicemapper, to avoid modifying shared layers
- Container is designed to support a single application
- Container instance is ephemeral, persistent data is stored outside the container, on the host or volume-containers

Docker != Container

- Docker == Container Image + Image Repository + Container
- Docker's main benefit is in Image creation and distribution
- Using Dockerfile Docker wraps containers and an overlay file system into a developer friendly model:
- Developer starts with a base image
- Encode deployment procedures as part of new image creation
- Reproducibly deploys the image as a container
- This makes developers an integral part of operations team
- Developer creates isolated portable deployment units without the overhead of CM tools

The Docker Revolution

- The rise of Docker is due to its ability to solve the CI/CD "integration matrix" problem



Application Container – Hermetically Sealed

MIRANTIS
Pure Play OpenStack

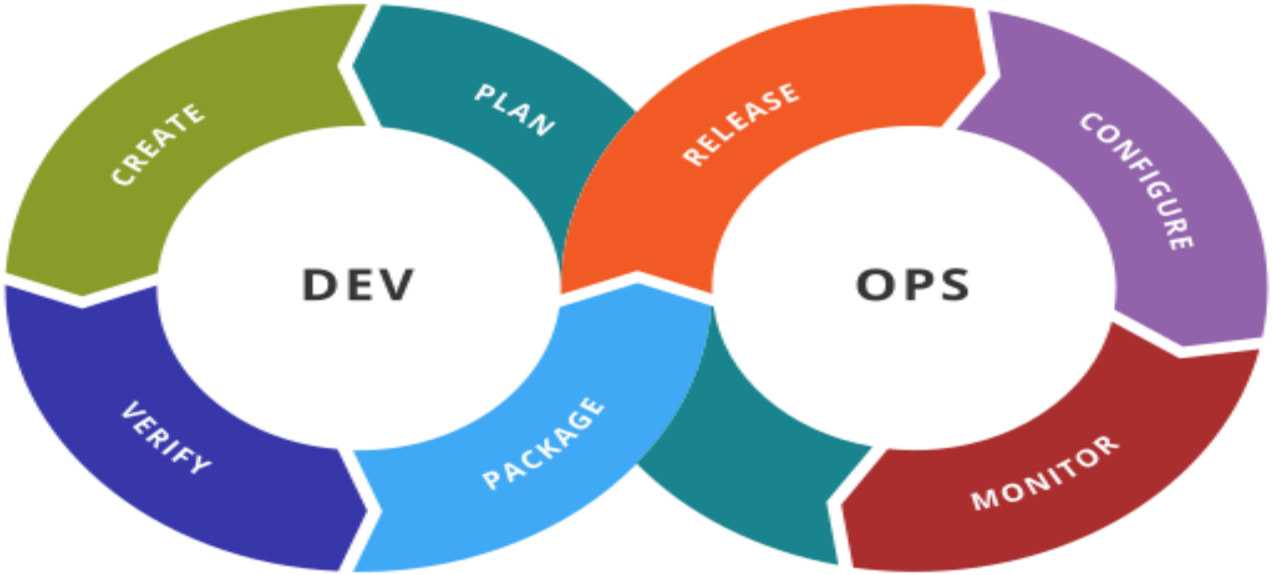
- Docker restricts the container to a single process at startup
- The container image has been carefully constructed by a developer to include everything needed, executable and configuration, and nothing else
- So Docker container emphasizes minimal images size/construction
- From the operations perspective, the container image is a black box
- A small, tactical unit of deployment
- The developer has already done the integration work

Microservices

DevOps ∩ Containers == Microservice

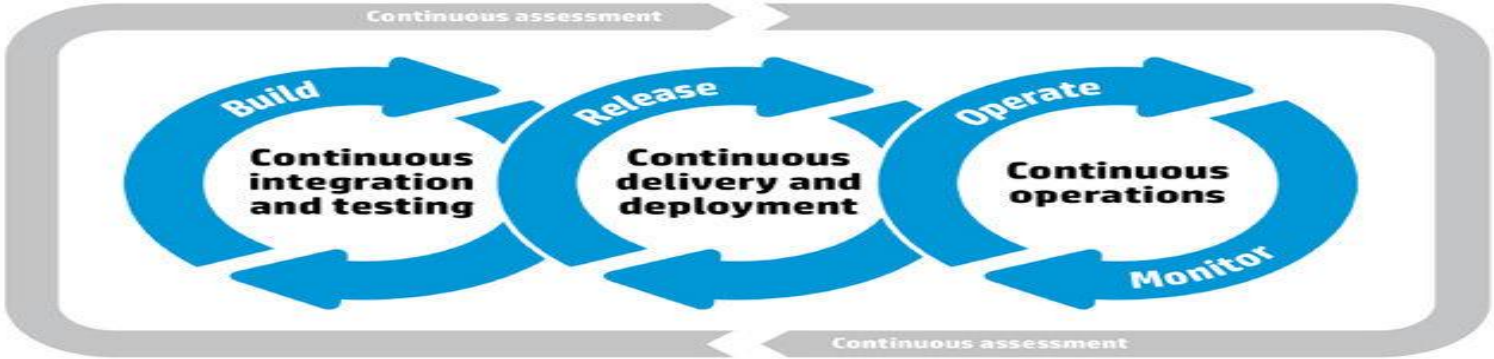
DevOps

A set of practices, and tools that increases an organization's ability to deliver applications and services at high velocity



Traditional DevOps Model

- Virtualization plus Configuration Management tools allows for automated deployment of applications:
- Development team commits the code
- Continuous integration deploys and tests the new code using CM
- Continuous Deployment delivers the new functionality using CM
- Continuous Operation monitors it



Automated Deployment of Applications

MIRANTIS
Pure Play OpenStack

- While application can be monolithic, it's uncommon
- Normally, multiple applications are combined together to create an application stack
- Orchestration and Configuration Management are used to coordinate functional deployment of multiple services to form a multitier solution
- But if services are small and tactical, then virtualization overhead per service is difficult to justify
- So services are combined on a single VM, and deployed together
- Combining service on Linux can lead to multitude of integration issues

12factor And The Adoption Of Docker Containers

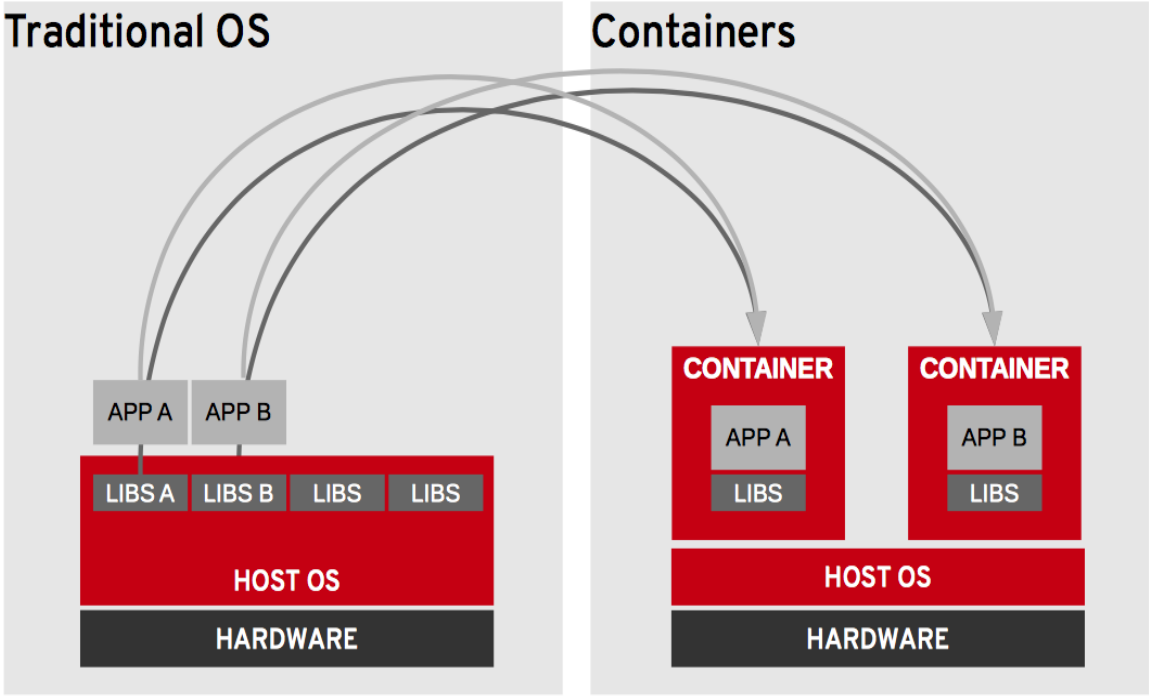
MIRANTIS
Pure Play OpenStack

- Docker containers are small, stateless, independent deployment units
- Hermetically sealed containers allow for isolation and portability
- Adoption of Docker Containers by CI/CD teams lead directly to a microservices-based architecture
- 12factor.net lays down 12 principals for architecting an application by composing smaller, independent, stateless services
- Docker is a perfect fit in this model

Containerized Deployment

Life cycle management at container level allows libraries to be updated without impact to the whole application.

DevOps is the main beneficiary.



Docker Compose – DevOps Packaging Tool

MIRANTIS
Pure Play OpenStack

- Compose is the Docker tool for defining and running multi-container Docker applications on one host
- Containers can share data volumes, Docker networks, environment variables, DNS entries, Linux capabilities, ….
- Compose override file to reuse a single Compose yaml file in different environments, such as development, staging, production
- Compose is Docker's way of defining a service or microservice by composing multiple containers together
- Development team can create the Compose yaml file
- Operations team can use Compose override to deploy to production

Containerized Deployment Summary

MIRANTIS
Pure Play OpenStack

- Microservices are essential to an event-driven IT model
- Containers are essential for microservices adoption
- However, this deployment model still uses a VM as the basis of the deployment
- Containers are tightly coupled to a single host/VM
- Microservices networking and persistence doesn't span beyond a host
- Deployment unit is the service logical host, not microservice
- All that is changed is the "how" of integration and deployment, not the "what"

What's there not to like?

MIRANTIS
Pure Play OpenStack

Tobby Banerjee of flockport.com wrote:

"Take a simple application like WordPress. You would need to build 3 containers that consume services from each other. A PHP container, an Nginx container and a MySQL container plus 2 separate containers for persistent data for the Mysql DB and WordPress files. Then configure the WordPress files to be available to both the PHP-FPM and Nginx containers with the right permissions, and to make things more exciting figure out a way to make these talk to each other over the local network, without proper control of networking with randomly assigned IPs by the Docker daemon! And we have not yet figured cron and email that WordPress needs for account management."

Kubernetes

Declarative Container Orchestration

Containers As First Class Citizens

MIRANTIS
Pure Play OpenStack

- Kubernetes tackles the problem of running microservices at scale as first class citizens, with all the rights bestowed upon VMs
- In other words, a cloud of microservices
- To do so it must solve the networking and persistent issues across multiple hosts that Docker never solved

k8s fundamentals: Pod

MIRANTIS
Pure Play OpenStack

- Kubernetes redefines a microservice (smallest unit of deployment) as one or more application containers (Docker images) in a Pod
- This preserves the DevOps benefit of containers, while tacitly acknowledging that container != microservice
- Pods share some namespaces, such as PID, network, IPC, UTS
- Pods may also have shared persistent volumes
- Kubernetes Pod is roughly an improved Docker Compose
- Kubernetes decouples the Pod from Service ports

k8s fundamentals: Volumes

MIRANTIS
Pure Play OpenStack

- Directory on disk shared between all containers in a Pod
- Different Volume types have different lifecycle:
- An emptyDir type volume and pod have the same lifecycle (is similar to Nova ephemeral disks) – but emptyDir services containers restarts
- rbd, glusterfs, nfs, iscsi, AWS EBS, GCE PD, … volumes have lifecycle beyond that of the Pod, and must have been created before the Pod

k8s fundamentals: Labels and Selectors

MIRANTIS
Pure Play OpenStack

- If in the cloud we have “cattle” then selectors pick a herd of cattle by the labels that identifies them
- Labels are arbitrary key/value pairs assigned to any object in kubernetes. Multiple labels can be assigned.
- Selectors return matching objects for operations on them. Self healing or self managing operations rely on the continuous update of selector matched object to achieve Kubernetes declarative goals.

k8s fundamentals: Service

MIRANTIS
Pure Play OpenStack

- A Kubernetes service is a networking endpoint which proxies requests to pods chosen by a Kubernetes selector
- A service is discoverable by other pods through Kubernetes (optional) internal DNS service. It is also discoverable through environment variables
- Services can be exposed outside of the Kubernetes cluster by assigning an external ip address to them
- A service without a selector can be bound to external endpoints manually, to connect to backing services

k8s fundamentals: Controllers

MIRANTIS
Pure Play OpenStack

- Controllers are responsible for maintaining/enforcing the declarative deployment models supported by Kubernetes
- The most basic model is a load-balanced set of stateless mircoservices, supported by:
- Replica Set / Replication controller
- Job
- Other controllers include:
- Stateful Set / Pet Set
- Daemon Set

Kubernetes as an Operational Support System (OSS)

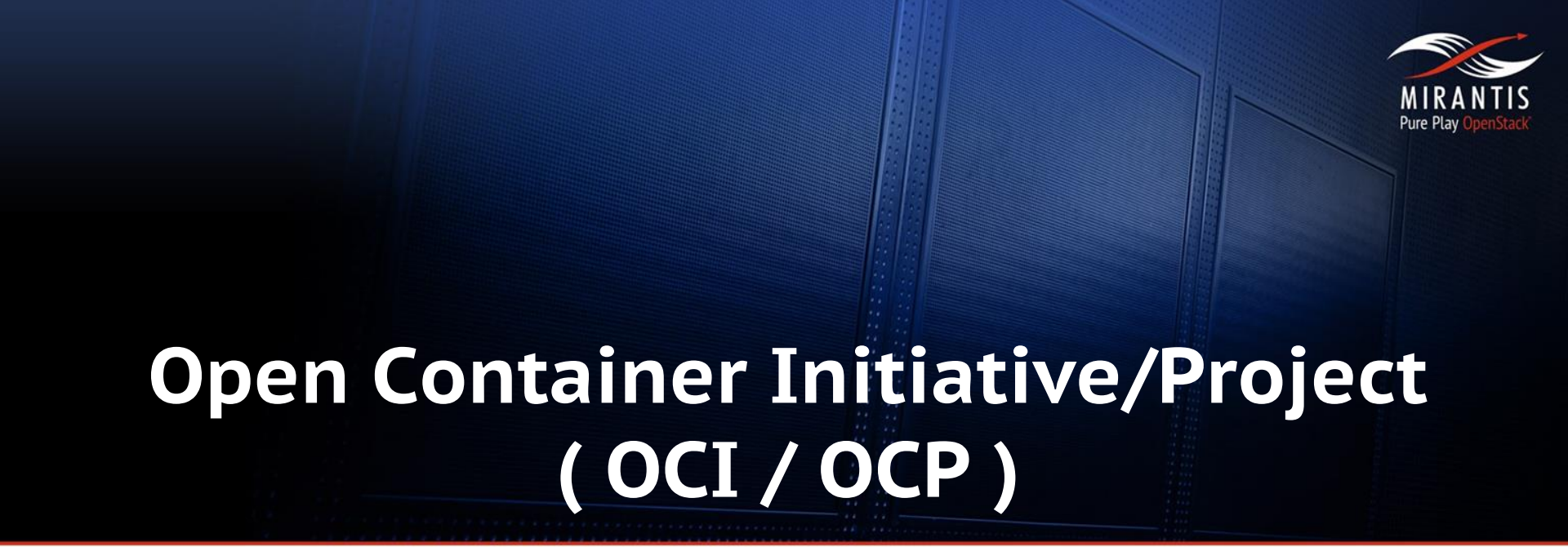
MIRANTIS
Pure Play OpenStack

- Kubernetes views itself as a “building block” of an OSS, not the OSS itself
- If the Kubernetes CLI seems cumbersome, it is most likely because it is meant to be used by an OSS implementing a higher level of abstraction/semantics
- Every component of Kubernetes is API accessible independently to give maximum flexibility and control to the system above that drives it

Putting It All Together

MIRANTIS
Pure Play OpenStack

- If Kubernetes is the new operating system, then:
- Pod is the new process – how we define a deployment unit
- Docker is the new apt – how we package a deployment unit
- Container is the new VM – how we isolate portable deployment units



Open Container Initiative/Project (OCI / OCP)

libcontainer and runc

Open Container Initiative (OCI)

MIRANTIS
Pure Play OpenStack

- Unlike Solaris zones, the Linux kernel lacks a “container” abstraction, so the notion of a OS container is created on top of Linux cgroup and namespace kernel primitives
- But more is needed to fully isolate a container
- OCI was launched in 2015 under Linux Foundation as the governance body for open standards for containers
- OCI defines the API for Application Containers
- Application Containers can then be implemented on any OS, using the underlying OS primitives

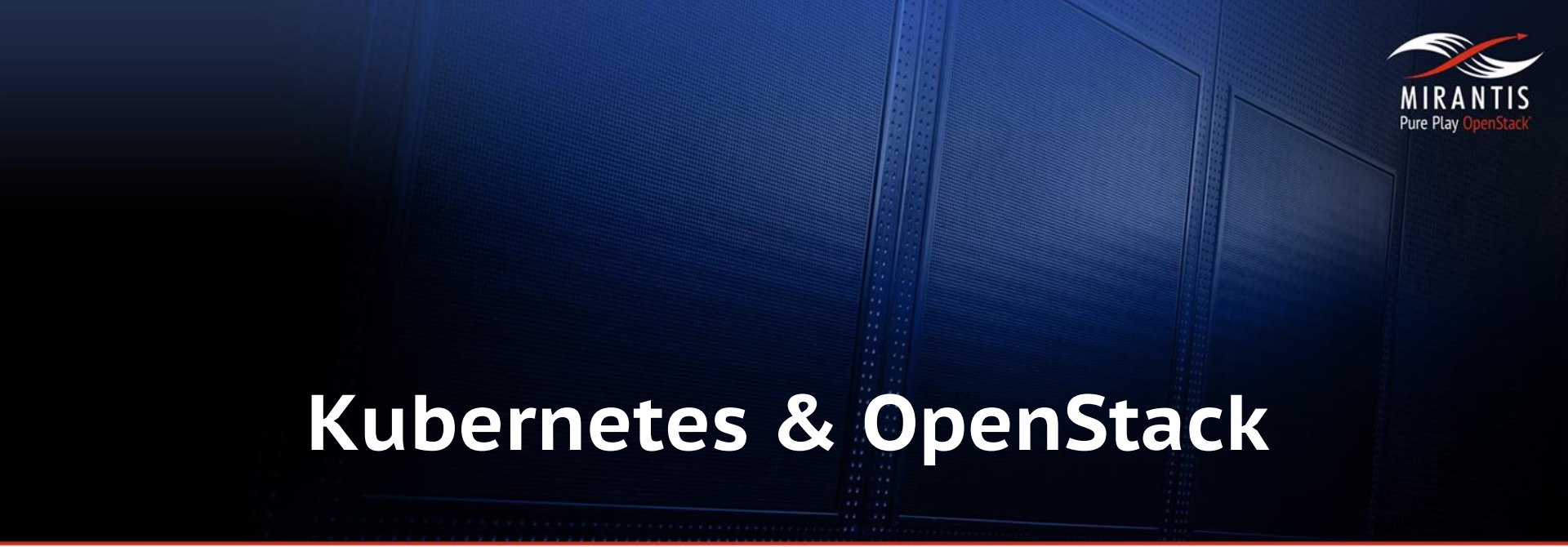
OCI runtime-spec

- Container runtime-spec outlines how to run an unpacked on disk “filesystem bundle” as a container
- Docker contributed “runc” and libcontainer are an OCI runtime-spec implementations. They control:
- mounts and environment variables
- user/group id, hostname, os type, architecture
- selinux and/or apparmor profile
- netlink and netfilter
- capbilities, rlimits
- pre/post start and post stop hooks

OCI image-spec

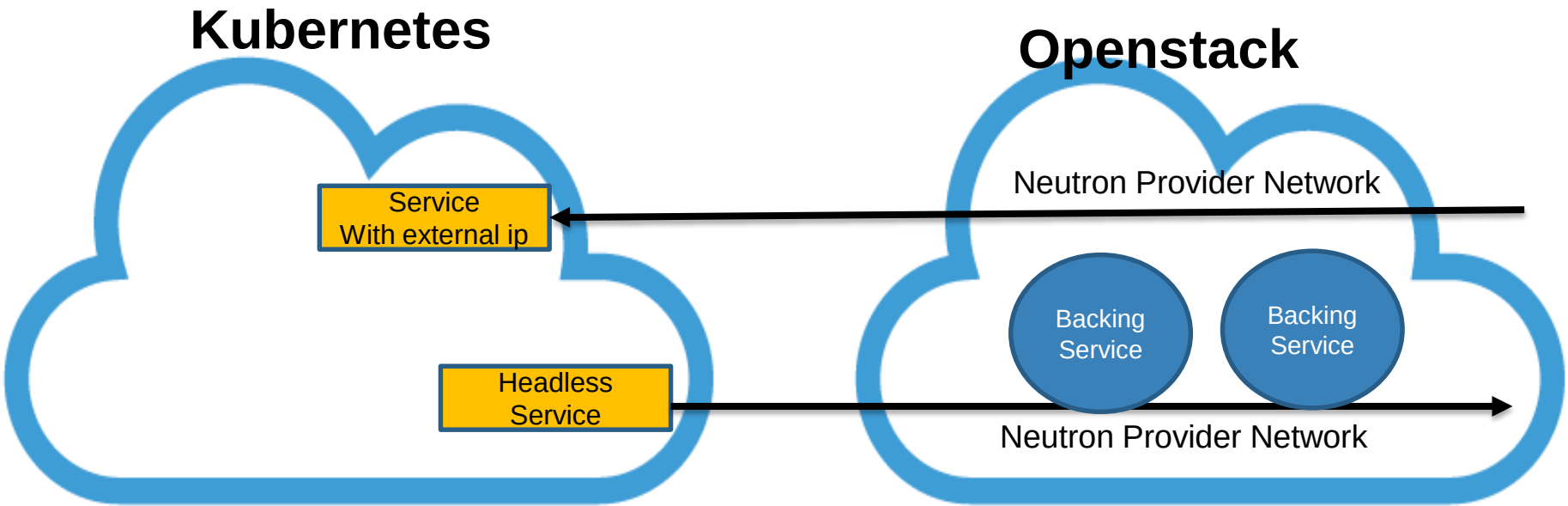
MIRANTIS
Pure Play OpenStack

- Container image-spec outlines how to build, transport, and unpack container images into a “filesystem bundle”
- “filesystem bundle” is the root filesystem of the container plus configuration necessary for launching the container
- **Image Manifest** is a configuration and set of layers for a single container image for a specific architecture and operating system
- **Image Layer Changeset** can be used to present a series of image layers as if they were one cohesive filesystem, typically by using a union filesystem such as AUFS or device-mapper
- **Image Configuration** is an immutable JSON description of image attributes required for launching a container based on the image

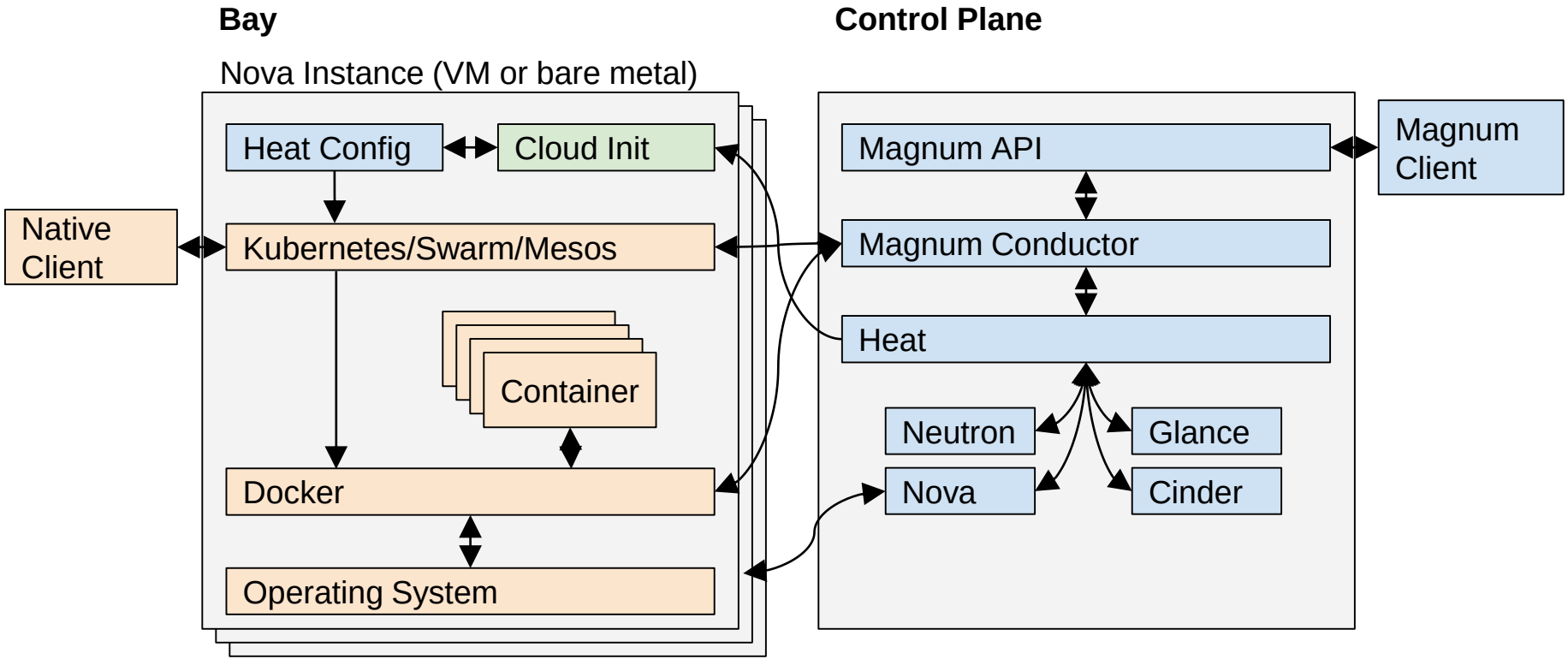


Kubernetes & OpenStack

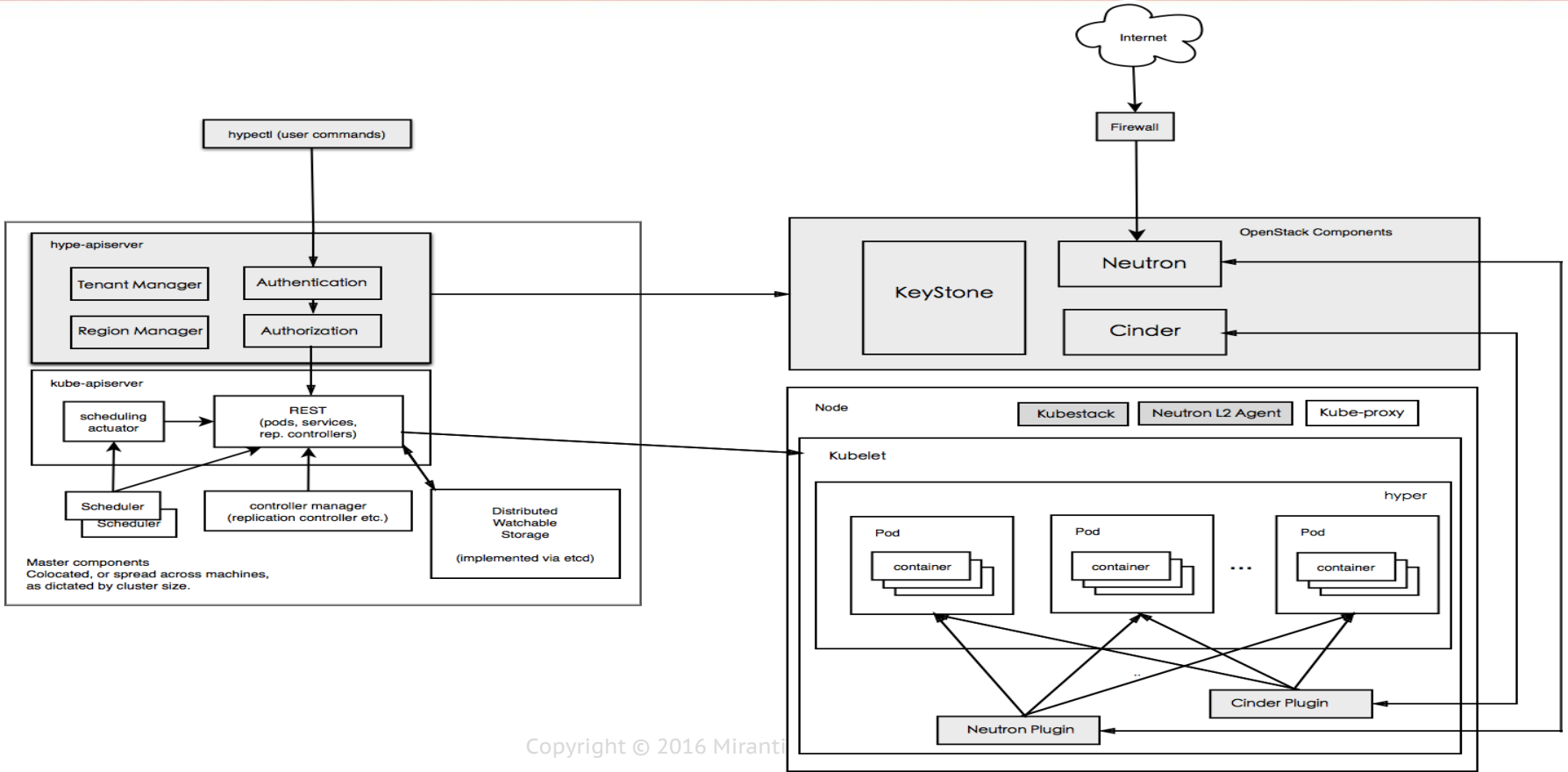
External Connectivity – Best Of Bread Approach



Kubernetes In OpenStack - Magnum



The Sky Is The Limit Hybrid - Hypernetes



Kubernetes & Docker: Courses & Certification

MIRANTIS
Pure Play OpenStack

Kubernetes & Docker Certification (KDC100)

First vendor-agnostic Kubernetes Certification

Hands-on, 30-task exam - $600 USD

Bundle training + exam (KD110) - $2,395

Online Kubernetes Bootcamp (KD100)

Self paced

1 Year access to content + 72 Hours of hands-on labs

Pre-registration at a discounted price of $195 (regularly $395)

Special Offer for Webinar Attendees

Register for Kubernetes & Docker Bootcamp or Certification exam before January 31, 2017 to receive a 20% discount

During registration use discount code: **MIRA-K8S**

Complete schedule:

training.mirantis.com

Schedule

KD110 Training + Exam Bundle

KDC100 Exam Only

Next sessions:

Dec 27 KD110

Dec 29 KDC100 Certification Exam

MIRANTIS
Pure Play OpenStack

Thank you!

Please submit questions in the question pane

Download presentation:
http://bit.ly/kubernetes-docker-slides