OpenShift 3 Technical Architecture

Clayton Coleman, Dan McPherson
Lead Engineers

Principles

The future of *aaS

Redefine the “Application”

- Networked components wired together
  - Not just a “web frontend” anymore
  - Service-oriented-Architectures / microservices are real
  - Make it easy to build, manage, deploy components
  - HTTP frontends are just one type of component
- Critical: relationships between components
  - If you can’t abstract the connection between components you can’t evolve them independently

Immutable images as building blocks

- Image based deployment (Docker)
  - Create once, test everywhere
  - Build a single artifact containing the dependency chain
  - Need tools to manage the build process and security updates
- Declarative application descriptions
  - Record “intent” - this links to this, this should be deployed like this, and let system converge to that intent

Decouple Dev and Ops

- Reduce complexity of application topology
  - Less need for complicated post-deploy setup
- Allow teams to share common stacks
  - Better migration from dev stacks to production stacks
  - Reduce vendor lock-in
- Ops and Devs can use same tools
  - OpenShift is an app, so are most organizational tools
  - Ensure patterns work cleanly for both

Decouple Dev and Ops (cont.)

- Templatize / blueprint everything
  - Ensure that organizations have clear configuration chains
  - Define common patterns for rolling out changes
- Easily provision new resources
  - Allow infrastructure teams to provision at scale
  - Subdivide resources for organizational teams with hard and soft limits

Abstract Operational Complexity

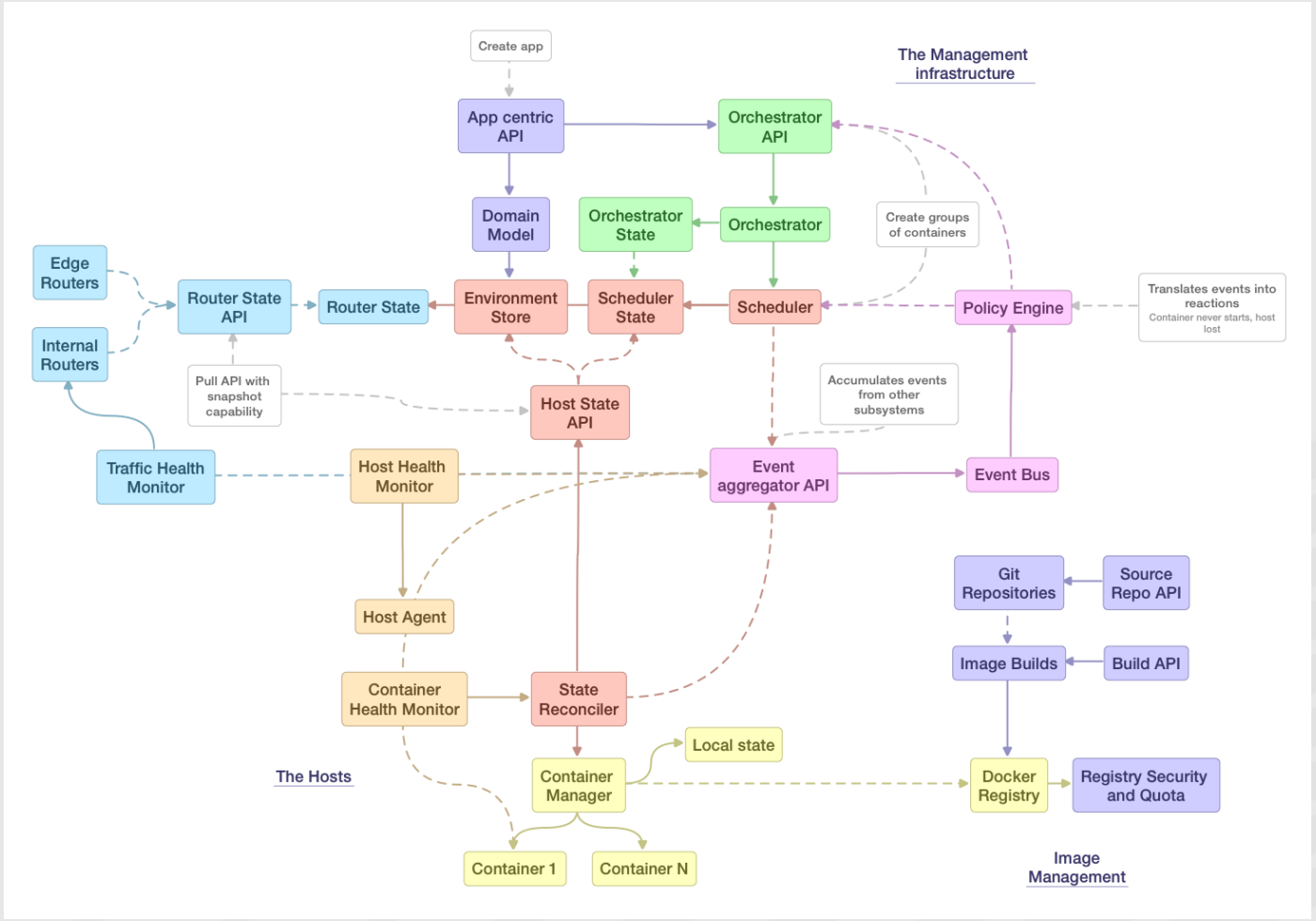
- Networking
  - App deployers should see flat networks
  - Define private vs public, internal vs external, fast vs slow
- Storage
  - Most components need *simple* persistent storage
  - Ensure storage is not coupled to the host
- Health
  - Every component should expose health information

Multi-Level Security

- Ensure containers “contain”
  - SELinux, user namespaces, audit
  - Decompose the Docker daemon over time
  - Fine grained security controls on SSH access
- Allow easy integration with existing security tools
  - Kerberos, system wide security, improved scoping of access
  - More customization possible
- Allow application network isolation

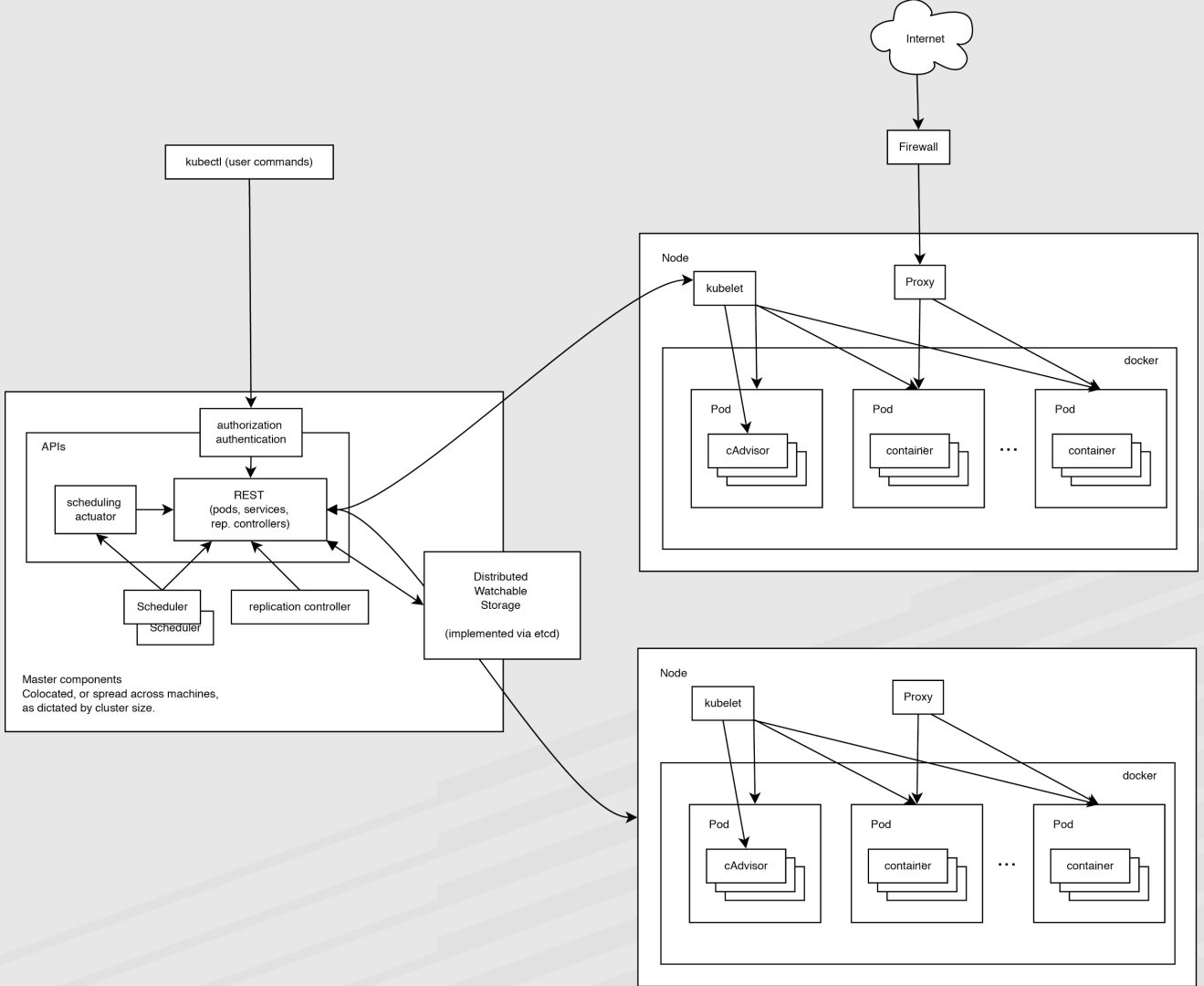
System Architecture

The moving pieces



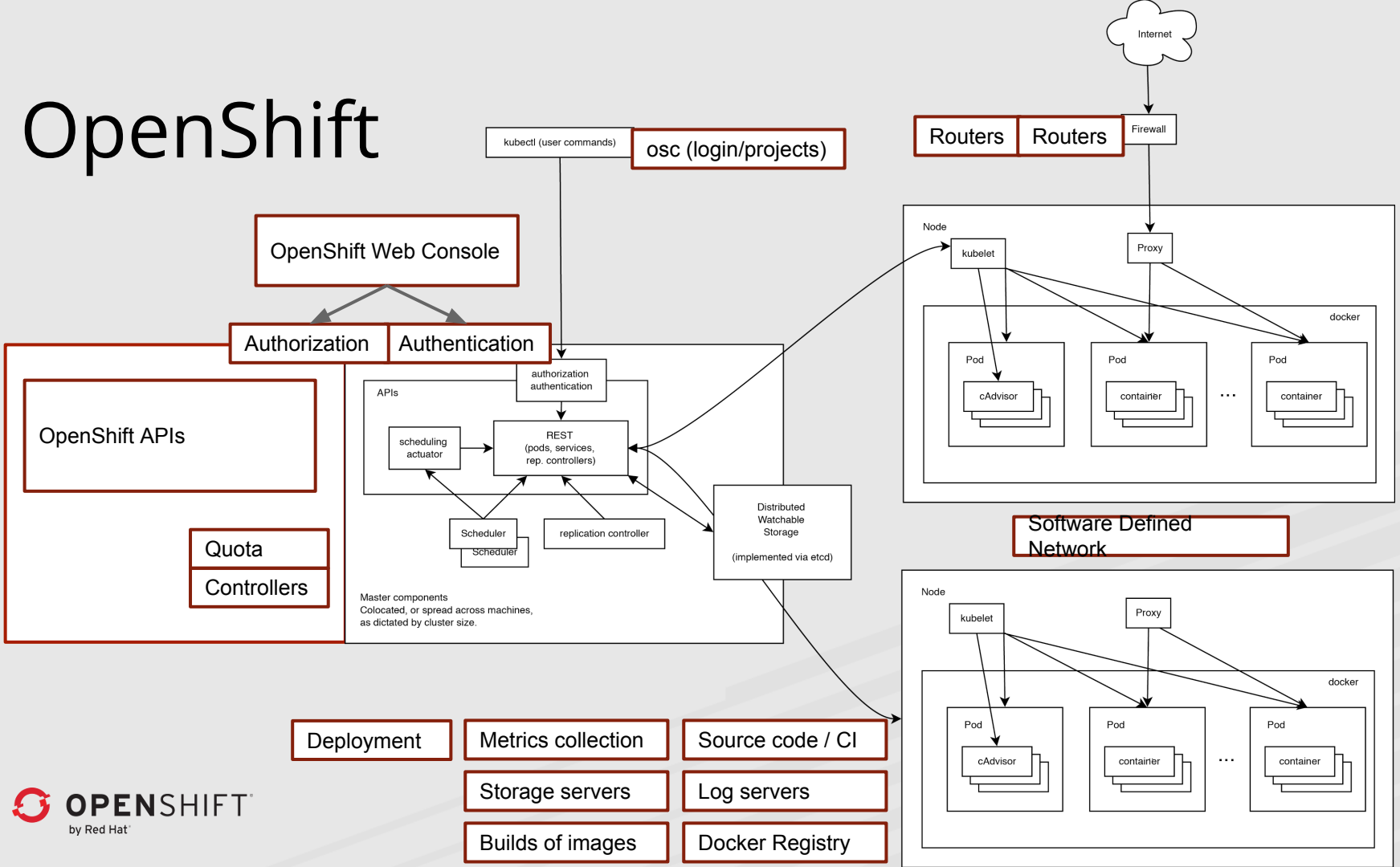
OPENSHIFT by Red Hat

Kube



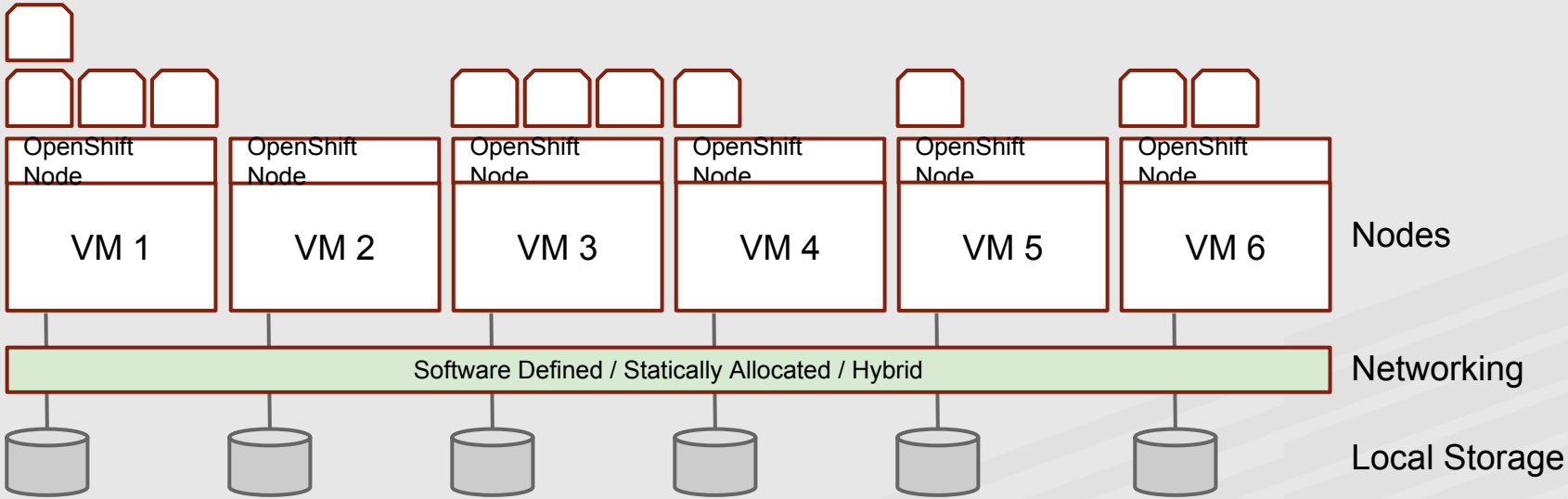
OPENSHIFT by Red Hat

OpenShift

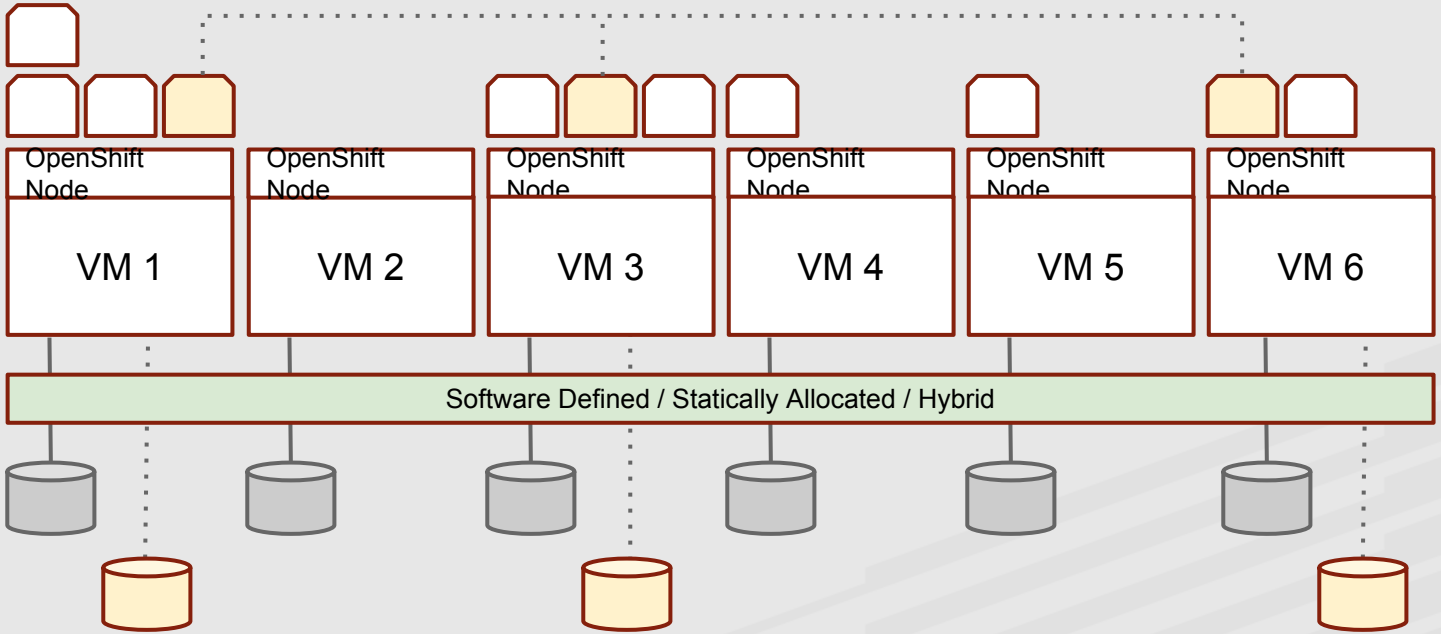


OPENSHIFT by Red Hat

OpenShift Infrastructure View



OpenShift Infrastructure View



Nodes

Networking

Local Storage

Persistent Network Storage

OPENSHIFT by Red Hat

What are the pieces?

- Docker
  - Container runtime and image distribution
  - Roll your own solutions for everything
- Kubernetes
  - Runtime and operational management of containers
- OpenShift
  - Lifecycle of applications - build, deploy, manage, promote
  - Manage tens to thousands of applications with teams

Docker Containers and Images

- Full access to the existing Docker ecosystem
  - Be able to use images from anywhere
- Operations and development share components
  - Customization still possible
  - Toolchains to support images
- Containers based on Linux Kernel tech
  - Continuing to work with upstream to improve security and reliability

Kubernetes

- Cluster Container Management
  - The substrate for running containers at scale
  - Contains just runtime and operational tools for containers
  - Composable system - only enough to enable other use cases
- OpenShift adds:
  - Ability to build, manage, and deliver app descriptions at scale
  - Turning source code into new deployable components
  - Moving from dev -> QA -> production

Kubernetes - Components

- Masters (OS2: brokers)
  - API (apiserver) - records client intent, displays current state
  - Schedulers (scheduler) - allocates pods onto hosts
  - Controllers (*-controller) - changes system from client intent
  - State Storage - etcd, potentially others in future
- Nodes (hosts / minions)
  - Agent (kubelet) - manages containers on each host
  - Proxy (kube-proxy) - local service routing and load balancing

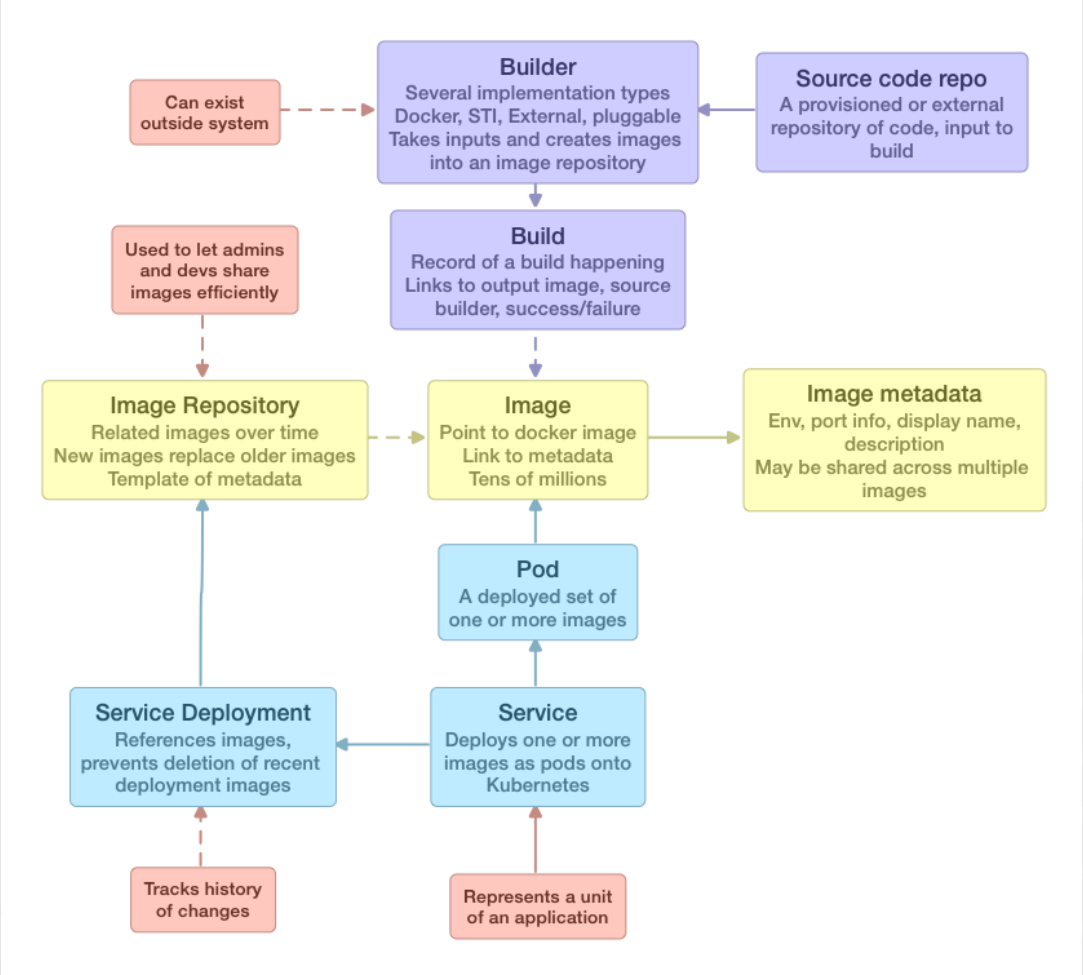
Kubernetes - Components

- Logging
  - common interface for moving data on and off nodes
- Events
  - communicate problems / changes back up to higher levels
- Node management
  - add, remove, and script host / addition removal
  - more control loops for health / problems

OpenShift - Components

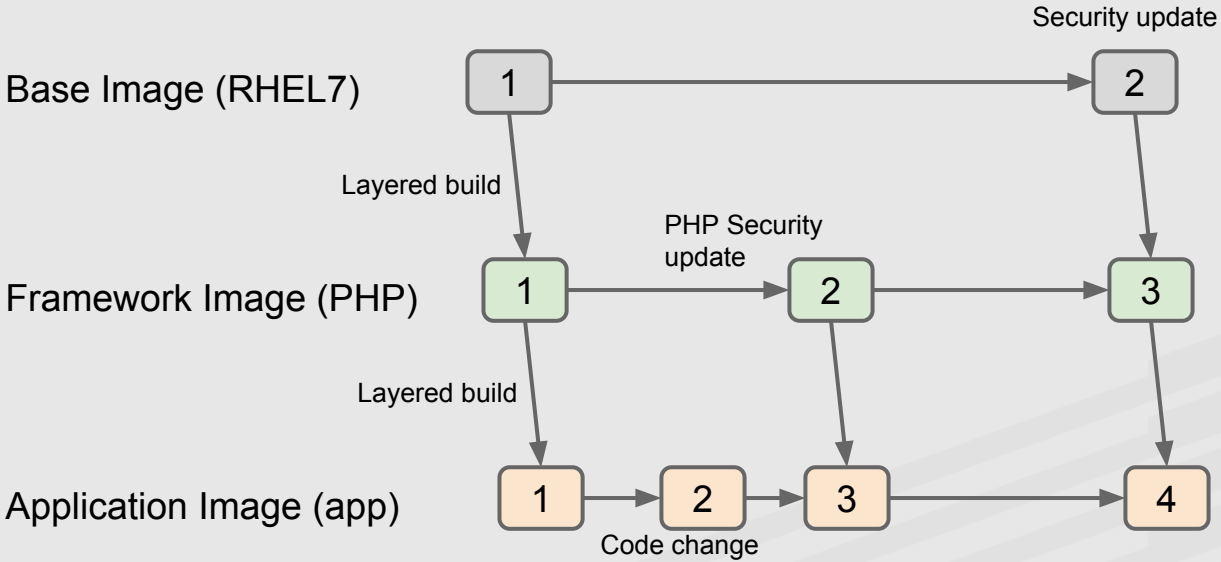
- Microservice based architecture
  - All components run on top of Kubernetes and provide additional services
  - Pieces are loosely coupled only core component is auth
- Offers an all-in-one mode for easy exploration
  - `docker run --net=host --privileged -v /var/run/docker.sock:/var/run/docker.sock openshift/origin start`
  - Includes Kubernetes, etcd, OpenShift in single process

Image Lifecycle



OPENSHIFT by Red Hat

How updates happen



Managing image promotion

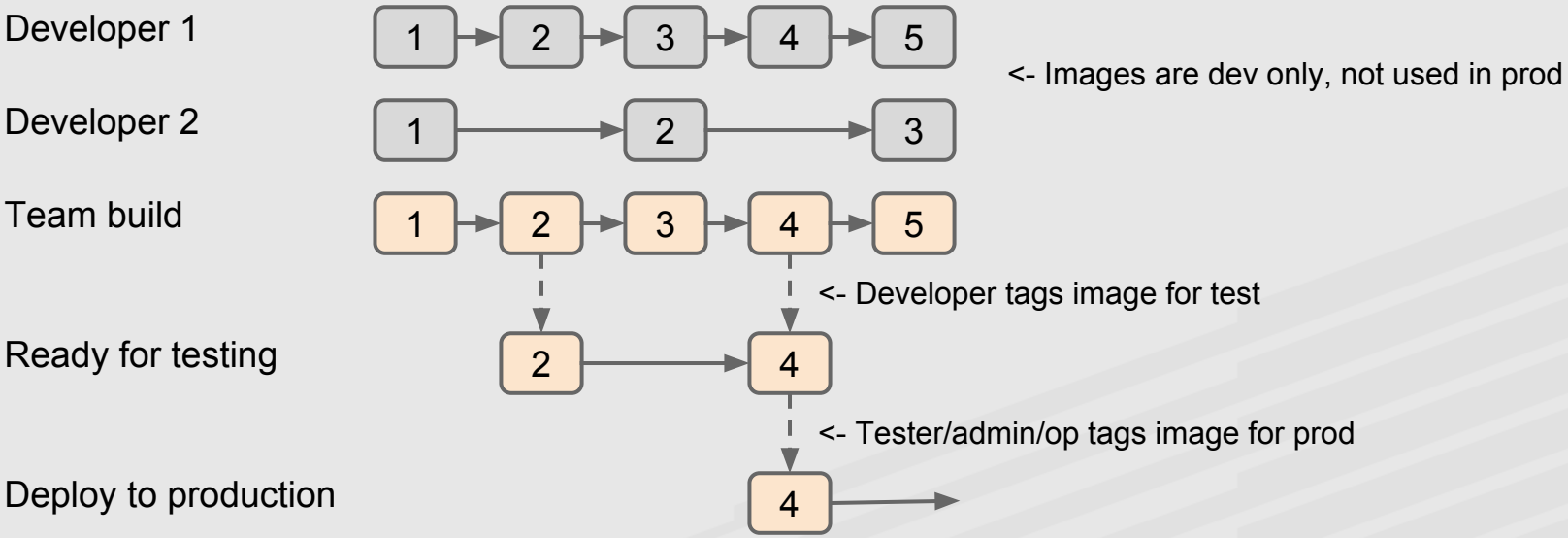


Image Lifecycle

- Image Registries
  - Places to store images so they can be accessed by Docker
  - Administrative tools for managing images at scale (quota, pruning, retention)
  - Content tracking, provenance, scanning of image data
    - Red Hat Satellite
- Building images
  - Need to be able to build thousands of new images for security updates

Image Lifecycle (cont.)

- Watching for changes to images
  - Allow teams and organizations to define “flows” for images
  - Trigger deployment automatically when an image is ready
- Testing images
  - Images need to be tested and validated in many environments
  - Much should be automated
- Promotion of images
  - Allow teams to combine process and flows

Network and Routing

- Allow containers to appear on the outside network
  - Declaratively expose a stable IP or port outside
  - Services with routable protocols (HTTP, TLS with SNI) can be aggregated efficiently behind proxies (HAProxy, Apache, Nginx)
- Segregate container traffic internally
  - Isolated based on team or topology structure
  - Allow integration with complex SDN (advanced scenarios)
- Allow containers to be auto-scaled based on traffic

Storage

- Allow easy integration with network storage
  - Block, Object, Shared filesystem (with SELinux)
  - Allow easy allocation / deallocation for operators
  - Add a fourth type similar to OSE 2.x for low cost hosting
- Reduce complexity on failover for persistent data
  - Operators can easily evacuate hosts
  - Focus is on keeping operational complexity low both on setup and runtime

Integrations with

- OpenStack
  - Similar to OSE 2.x - HEAT templates and autoscaling
  - Deeper keystone / ceilometer integration
  - Containerization integration w/ Docker and Nova - TBD
- Red Hat Atomic
  - RHEL distribution tuned for containers
  - Network and storage configurations

Concepts

OpenShift 3 Nouns

Pods and Containers

- Fundamental unit in the system
  - Pod is a group of related containers on the same system
  - Each container can be its own image with its own env
  - Pods share an IP address and volumes
- Pods are transient and not “special”
  - Pods should be able to be deleted at any time
  - Storage can be detached and reattached elsewhere
  - Different pods talk to each other through abstractions

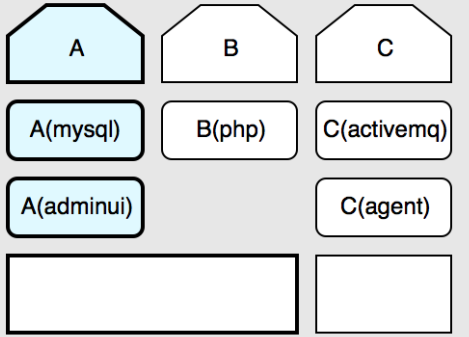
Pod Examples

- Single container - JBoss, MySQL, etc
- Web server and log parser
  - Web server container logs HTTP requests to disk
  - Log parser reads from disk and sends summary info elsewhere
- Apache security proxy and PHP app
  - Apache container listens on port 80 and uses mod_auth to check authentication, then proxies port 8080
  - PHP container listens on 127.0.0.1:8080 and serves a web app

Pod Examples (cont.)

- Wordpress pod with PHP and MySQL
  - PHP is visible externally, MySQL is not
  - Can’t be scaled up, but self contained and works
- Ruby worker application with local Redis
  - Ruby web application takes incoming HTTP requests and stores them in local Redis
  - Redis is queried by other servers for processing
  - Can be scaled out

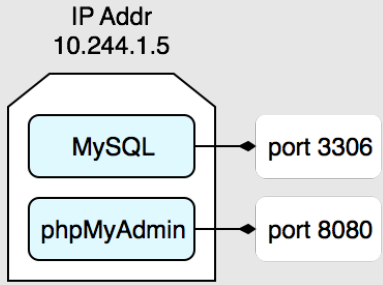
Pods (cont.)



Pods
Group of related containers placed onto the same host

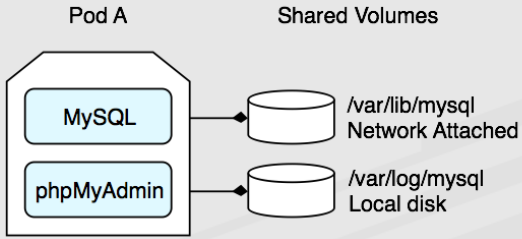
Docker Containers
Application processes and filesystem libraries

Hosts
Runs pods



Pod Networking
Each pod has an IP address that other pods can contact

Shared Ports
Each container must share pod ports. No conflicts allowed



Volumes per Pod
Each pod has a list of volumes that all containers access the same

Volume Types
Each volume can have different types, like local transient storage or network attached storage backed by Cinder, GCE, EBS, etc

OPENSHIFT by Red Hat

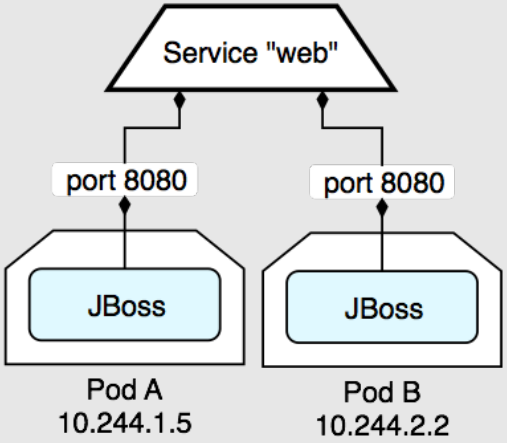
Connecting Pods

- Need a way for pod A to talk to pod B
  - Option 1: Hardcode IP address
  - Option 2: Query the server
- If there are 10 copies of pod A, which do you use?
  - Pick one randomly?
  - Load balance!
- What if it fails?
  - Want to have all copies of pod A talk to all copies of pod B

Services

- Abstract a set of pods as a single IP and port
  - Each host has a proxy that knows where other pods are
  - Simple TCP/UDP load balancing
  - No central load balancer (no SPOF)
- Creates environment variables in other pods
  - Like “docker link”, but across hosts
  - Service named “mysql” gets MYSQL_HOST and MYSQL_PORT

Services (cont.)



Services abstract other pods
A service is a TCP port that may transparently load balance other ports

Replication controllers copy pods
A controller ensures there are a certain number of copies of a pod, so if a host is lost another pod gets created.

Why Services?

- Stable endpoint for pods to reference
  - Allows list of pods to change dynamically
- Need a way to know where other pods are
  - By *name* (mysql is recognizable)
  - Simple service discovery
- Lots of other options for service discovery
  - The simplest thing that works at scale

Services (cont.)

- Improvements planned
  - Allow pods to alter how services are exposed (localhost:3306)
  - Point to external (outside) IPs and DNS
  - Internal private DNS (mysql.myapp.local)
  - Pluggable balancers like HAProxy
  - Bind service to external IP?
  - Use services between different security zones
- Fundamental “link” abstraction

What is an app?

- One or more set of pods
  - JBoss, MySQL, Redis, ActiveMQ, Backend Workers, Frontends
- Linked together by services
  - Edge router for myapp.com -> JBoss frontend pods
  - JBoss frontend pods -> **backend** service
  - **backend** service -> PHP pods
  - PHP pods -> **mysql** service
  - **mysql** service -> MySQL pod with attached storage

Deployments

- Define the lifecycle for a single image
  - Each deployment records a particular image and settings for that image at a point in time
  - Users create new deployments that describe new desired state
  - Sometimes, new images change details of deployment and those must be taken into account
- A record of the history of a single component

Templates and Config

- Config is declarative description of topology
  - Set of resources that describe a version of an app
  - Any object can be part of the description
  - Tools for applying config and ensuring reality matches config
- Templates let you parameterize config
  - For consumption by end users of standard patterns
  - Simple key/value substitution today

Templates and Config (cont.)

- Config is an active topic of iteration upstream
  - We plan to enable config as a core principle of the system
  - Additional concepts may be required to ease config use for end users
  - Deployments are stripped down and simplified Config

Building Images

- Allow infrastructure to build images
  - Everything depends on Docker images
  - Source to Image (STI) - artifacts or source into images
  - Integration with Jenkins and other build systems
  - Builds are run in containers under user resource limits
- Easy integration for existing build infrastructure
  - Push images into an image repository
  - Extend OpenShift with new builders

Source Code

- Easy integration with external source repos
  - GitHub webhooks, extensible to other patterns
  - Can easily connect builds with other systems via hooks as well
  - Future integration with other systems
- Simple high-density Git hosting OOTB
  - Easy to spin up new Git repositories for new projects
  - Lower operational cost than 2.x

Users, Teams, and Projects

- Allow cluster resources to be subdivided
  - A project controls access to a set of resources
  - Projects are allocated resources with hard and soft limits
  - Typically based on organizational boundaries
- Continue 2.x improvements on top of Kube
  - Teams, LDAP integration continue
  - Give additional control to operators with flexible policy
  - Simplify access control and authorization via OAuth

Quota and Usage

- Kubernetes brings finer grained resource control
  - Allows many dimensional specification of performance
  - Requires corresponding deeper quota and usage tracking
  - Policy and

Developer Experience

- Operators create images for teams to code on
  - Our JBoss environment
  - Custom MySQL tuning
- Also create templates for types of applications
  - Templates are blueprints for applications, can be complex
- Developers create from template or from scratch
- Code in their favorite toolchain

Developer Experience (cont.)

- Like 2.x, push code to a source repository
  - GitHub, external, or created in OpenShift
- Build runs and generates an image
- Image is deployed

- Lots of exciting changes coming...

30s Admin Experience

- Run OpenShift in 30 seconds or less
  - Download from Releases, install Docker
  - `openshift start`
- Reduce operational complexity for small deploys
  - Vagrant environment for default network setup
  - Deploy direct to IaaS (OpenStack, GCE, AWS)
- Future work
  - Run OpenShift/Kubernetes inside Docker

QUESTIONS?

GitHub http://github.com/openshift/origin

Clayton Coleman ccoleman@redhat.com

Dan McPherson dmphers@redhat.com

09/22/2014

TODO

- Better pod -> service -> pod diagram
- Storage diagram
- Deployment flow diagram
- Skill sets for ops
- Integrating operational tools