

RED HAT
SUMMIT

BOSTON, MA
JUNE 23-26, 2015

THE JOURNEY OF BRINGING EAP TO OPENSIFT V3

Aleš Justin, Marko Lukša
Red Hat Cloud Enablement
June 24, 2015

INTRODUCTION

About us & the CE team

- Aleš Justin (*ajustin@redhat.com*)
- Marko Lukša (*mluksa@redhat.com*)
- Cloud Enablement Team @ RedHat
 - Bring RedHat JBoss Middleware products to OpenShift v3
 - EAP
 - JWS
 - Fuse
 - JDG
 - ...

Project: Bring EAP to OpenShift

- EAP already available in OpenShift v2
- OpenShift v3 is a completely new environment, based on Kubernetes
- Prepare Docker images and templates:
 - standalone (single-node) EAP
 - clustered EAP systems
- Sounds simple, but really wasn't
 - OpenShift still being developed at fast pace
 - All of us new to Kubernetes
 - Not experts in computer networking
 - Mostly using VMs all the time (non-Linux users, etc)
 - Lots of possible paths (standalone/domain, logging, metrics, ...)

About the OpenShift v3 environment

- Docker
 - Package up your app with all its dependencies and the OS environment into an image
 - Image based on parent image
 - Layers
 - Creating new images:
 - manually (run image, make changes, commit as new image)
 - automatically (Dockerfile: FROM, ADD, RUN, CMD)
 - Push images to repository
 - Pull images from public repositories

About the OpenShift v3 environment

- **Kubernetes**
 - Orchestration system for Docker containers
 - Kubernetes Master & multiple Nodes (Minions)
 - Pods
 - Colocated group of containers
 - Resource sharing
 - Similar to a single server (single IP, port collisions, etc.)
 - Labels
 - (Key, Value) pairs
 - Label selectors
 - Services
 - Service discovery (ENV vars, DNS)
 - Replication controllers
 - Scaling
 - Flat network space

About the OpenShift v3 environment

- OpenShift v3
 - Complete DEVOPS system
 - Continuous delivery
 - Builds and Image Streams
 - Transform source code into runnable image
 - Docker build (Dockerfile)
 - Source-to-Image build
 - inject source code into Docker image and produce new Docker image)
 - incremental builds (no re-downloading of dependencies)
 - Build triggers: parent image change, source code change, generic web hooks
 - Deployments
 - ReplicationControllers
 - Triggers for creating a new deployment automatically (e.g. config change)
 - Strategy for transitioning between deployments
 - LifeCycle hooks
 - Routes (expose services to the outside world)
 - Templates (parameterizable set of resources)

Step 1: Identify requirements

- Brainstorming - initial list of subjects:
 - Service clustering
 - Configuration
 - Logging
 - Metrics
 - Auto-scaling
 - Single Sign-On
 - Testing
- General cloud requirement:
 - Need to have as few supporting containers as possible
 - Each supporting container must be as small as possible
 - Merge them all into single process

CLUSTERING

Service clustering

- Clustering for scalability
 - EAP is scalable already
 - Replicated sessions
 - EJBs
 - Cache
 - Replication controllers
- Clustering for high-availability
 - HAProxy
 - Kubernetes Services & Routes

Clustering - Multicast?

- EAP uses JGroups for clustering
- JGroups default: multicast UDP
 - If multicast available:
 - Very simple to set up - Marko's blog post
 - When not available:
 - Need to use unicast TCP
 - Existing methods of discovery
 - New mechanism specifically for Kubernetes/OS3

KubePing

- Initial implementation by Aleš
- Get list of pods/containers from Kube API
- Ping each container directly
 - Light / simple embedded server running
 - Get a hold of PingData
- Multiple EAP clusters
 - Label selectors
- Problems
 - Needs authorization to access K8s REST API
 - Needless complexity
 - Too implementation specific
 - Initially the only way (no DNS lookup of services)

DNSPing

- KubePing initially necessary (no DNS support in OS3 initially)
- Reduce coupling to the Kubernetes system
- Look up services through DNS A records
 - `<name>.<namespace>.svc.cluster.local`
 - returns the portal IP
 - for headless services returns A records for each endpoint
 - `<name>.<namespace>.endpoints.cluster.local`
 - always returns endpoints (for non-headless services also)
 - utilized by DNSPing
- SRV records
 - `<portname>.<protocol>.<name>.<namespace>.svc.cluster.local`

CONFIGURATION

Configuration

- Read-only config or modifiable during runtime?
 - Read-only atm → centralized place
- EAP Standalone or Domain Mode?
- Databases and other resources
 - Bundle all drivers
 - Template per different DB
 - Username / password in secrets
- Deploying apps (EAR, WAR, ...)
 - Docker build?
 - Source-To-Image

Standalone vs. Domain Mode

- Standalone mode
 - Simplicity
 - More “docker way” of doing things (cattle vs. pets)
 - Kubernetes replication controllers - the proper way of managing instances
- Domain mode
 - Well-known to existing EAP administrators
 - Centralized management policy
 - Centralized config for server groups
 - Lots of existing tools (CLI, Web Console, JON, ...)
 - Goes against Kubernetes/OpenShift model

Databases And Other Resources

- Multiple images (one containing each database driver)
 - explosion of number of images
- Single EAP image, but separate JSON application templates
 - eap6-postgresql-sti
 - eap6-mysql-sti
 - eap6-mongodb-sti
 - eap6-amq-sti
- Non-persistent vs. Persistent storage
- Configuration through template parameters
 - When creating from a template, some values entered manually, others auto-generated
 - Passed into the images through environment variables

LOGGING

Logging

- EAP defaults:
 - Log to STDOUT
 - Log to files
- The Docker Way: logging to STDOUT
 - `docker logs <container-id>`
 - `openshift cli log <pod>`
- Need centralized logging
 - get logs from lots of sources into a single log store

Centralized logging

- Lots (possibly hundreds) of containers
- Impossible to handle/look at them separately
- Benefits of seeing multiplexed front-end and back-end logs
- ELK stack:
 - Elastic
 - LogStash
 - Kibana
- But where do we take the logs from?
 - LogStash
 - LogSpout
 - FluentD

LogStash, LogSpout, FluentD

- **LogStash**
 - Grabs logs from various sources (stdin, files, etc.)
 - Filter log messages
 - Send them to e.g. Elastic through HTTP
- **LogSpout**
 - Grabs logs from STDOUT of all running Docker containers
 - Sends them to SysLog, Elastic and others
 - Problem: reads from Docker logs streams; downtime for LogSpout means missed logs
 - Solution: use FluentD instead
- **FluentD**
 - Docker streams its logs to disk, FluentD reads them from there
 - No missed logs

Multi-line log statements

- Each row of log is sent to Elastic as separate log message
- Problem: exception stacktraces
- Solution: log each log statement as a JSON object
 - Whole stacktrace added to Elastic as a single entity
 - Easier analysis, etc.
- New problem: hard to read log output to STDIN
 - Can't use *docker log <cid>* anymore (too unreadable)
 - Also can't use *oc logs <pod-id>*
 - Not really important - could log to files in the EAP-standard way

METRICS

Metrics

- Sources:
 - Containers (cAdvisor)
 - Low level; CPU, etc
 - JMX
 - JVM MBeans
 - Any MBean
 - DMR
 - Custom Management values
- Centralized storage in InfluxDB
- Visualization in Grafana

cAdvisor, jAdvisor → Heapster

- cAdvisor
 - already in OpenShift v3
- JMX
 - Jolokia
 - jAdvisor
- DMR
 - jAdvisor
- Moving it all to “heapster”

Heapster

- Collects metrics data from multiple sources (connects to Kubelets)
 - OS3 security increasing: work today, fail tomorrow
 - Problems:
 - Kubernetes REST Endpoint initially not secured (HTTP only)
 - Later moved from HTTP to HTTPS (initially HTTP hardcoded in Heapster)
 - Later also secured through client certificates and OAuth tokens
 - At the end, the Kubelets also secured in same way (another change needed in Heapster)
- Sends data to sinks
 - InfluxDB

SCALING

Scaling (manual)

- Scaling in Kubernetes
 - Replication controllers
 - Number of replicas
- Scale up
 - Not problematic in most cases
- Scale down
 - Problems!

Scaling down

- Stopping a container:
 - SIGTERM
 - Wait 10, 20, 30 seconds
 - SIGKILL
- Graceful shutdown
 - pre-stop hooks
 - HTTP get
 - Executable
 - Problems:
 - What if the Pre-stop hook fails during execution
 - What if the Kubelet fails
 - What if the whole server fails

Graceful shutdown

- Problem #1: tasks in progress
 - HTTP requests
 - Short-lived
 - Other tasks
 - Long-running tasks
- Problem #2: state
 - Non-replicated sessions
 - Transfer state to other nodes?

Auto-scaling

- Autoscaler in Kubernetes, OS3 or external autoscaler?
 - Imho, 99% covered with simple http(s) service monitoring
- Get metrics from where?
 - Directly from EAP
 - From InfluxDB → more generic
- What metrics to base scale-up & scale-down on?

SINGLE SIGN-ON

Single Sign-On

- OS3 allows you to use external OAuth providers
 - Rob Cernich (CE team) added support for using KeyCloak as an OAuth handler
 - OS3 is not meant to be an identity server
 - OS3 only provides authorization scopes specific to OS3
- Fabric8 provides KeyCloak as an installable application OOB
- EAP does not provide OAuth support OOB
 - WildFly 9
 - Will add SSO support into our EAP image later

DEMO

Demo #1 - full EAP

- Install the jboss-image-streams.json
- Install the eap-basic-sti template
- Install the eap-app-secret.json
- Create a new deployment from the eap-basic-sti template

Demo #2 - auto-scaling

- Simple “long” running app
 - Auto-scaling → custom Ascaler
 - Pre-stop monitoring → built-in app
 - Atm request count only
 - Pre-stop hooks → custom prestop-exec
 - Can be changed with HTTP get
- JMeter Test plan

TESTING

Testing?

- Arquillian support
 - Mocking OS3 behavior: build, push, deploy
- Testsuite
 - Using this Arquillian support

<https://github.com/jboss-openshift/ce-arq>

<https://github.com/jboss-openshift/ce-testsuite>

Not on Linux?

Project Jube (<https://github.com/fabric8io/jube>)

- Java based Kubernetes mock
 - It's all about Kubernetes REST API
 - And similar Kubernetes-like behavior
 - Replication, master election, ...
- Where is Docker?
 - Nope, no Docker here
 - Zip images
 - Lifecycle scripts

WRAP UP

Try it out yourself

- Set up OS3 on your own servers
 - Install Docker
 - Download and run OpenShift All-in-one server
 - <https://github.com/openshift/origin/releases>
 - `$ openshift start`
- Use our templates to deploy EAP
 - <https://github.com/jboss-openshift/application-templates>
 - `$ oc create -n openshift -f jboss-image-streams.json`
 - `$ oc create -n myproject -f eap/eap-basic-sti.json`
 - Open console at <https://localhost:8443/console> and click the “Create...” button

Thank you

- Aleš Justin (ajustin@redhat.com)
- Marko Lukša (mluksa@redhat.com)

Q & A

RED HAT SUMMIT

LEARN. NETWORK.
EXPERIENCE OPEN SOURCE.