About Me

Lakmal Warusawithana
Vise President, Apache Stratos
Director - Cloud Architecture, WSO2 Inc
lakmal@apache.org / lakmal@wso2.com
Twitter : lakwarus
Agenda

- Introduction to Apache Stratos
- Apache Stratos Architecture
- Does Docker Production ready?
- Introduction to CoreOS, Flannel, Kubernetes
- Apache Stratos 4.1 – Containerization and Composition Release
- Apache Stratos with Docker
  - Kubernetes Resources Used by Stratos
- Why Composite Application Support?
- Discuss few Apache Stratos features
- Demo - Docker, Kubernetes with autoscaling
Apache Stratos

- Apache Stratos is a highly-extensible Platform-as-a-Service (PaaS) framework that helps run Apache Tomcat, PHP, and MySQL applications and can be extended to support many more environments on all major cloud infrastructures.

- Stratos initially developed by WSO2 and last year donated to Apache Software Foundation.

- After successfully complete the incubating process, Stratos now graduated as Top Level Project.
Apache Stratos Layered Architecture

Apache Stratos 4.1.0 Layered Architecture

Applications

Application 1  Application 2  Application n

Cartridges

Any Pluggable Framework, Application, Data or Load Balancer Cartridges

Load Balancer

Stratos Load Balancer  HA PROXY  NGiNX  Identity Service  Logging Service  Monitoring/Metering Service

Message Broker

Stratos Manager

REST API  Artifact Distribution Coordinator  Metadata Service  Autoscaler  Complex Event Processor  CLI/Web UI

Cloud Controller

jclouds Client  Kubernetes Client  Mock IaaS Client

Infrastructure as a Service

Mock IaaS
Does Docker Production Ready?

- Docker network?
  - Deploying in Docker host cluster
- Can run enterprise apps in a single docker container?
- Problems of running enterprise apps in multiple docker containers?
  - File System sharing?
  - Network sharing?
  - Process space
  - How to identified an unit?
Apache Stratos 4.1 – Containerization and Composition Release

- Application Composition
- Containerization
  - Docker based cartridge support
  - integration with CoreOS
  - integration with Kubernetes
  - integration with flannel
  - integration with discovery service and build in docker registry support
What is CoreOS?

CoreOS Host

Docker Docker Docker Docker

Docker Docker Docker Docker

Docker SystemD Fleet etcd

Kernel
CoreOS Cluster

Local machine
- fleetctl
- etcdctl

systemd service files pool
Services

fleets

Host #1
- CoreOS host
- Docker containers
- etcd

Host #2
- CoreOS host
- Docker containers
- etcd

Host #3
- CoreOS host
- Docker containers
- etcd

etcd

© WSO2
What is Flannel?
What is Kubernetes?

- Kubernetes is a platform for hosting Docker containers in a clustered environment with multiple Docker hosts.
- Provides container grouping, load balancing, auto-healing, manual scaling features ... etc.
- Project was started by Google.
- Contributors == Google, CodeOS, Redhat, Mesosphere, Microsoft, HP, IBM, VMWare, Pivotal, SaltStack, etc.
Kubernetes with CoreOS
Apache Stratos L1 Architecture for Docker based Cartridges
Stratos Architecture with Docker Support
Kubernetes Resources Used by Stratos

- A Kubernetes Service is created for each transport/port mapping defined in the cartridge.
- Kubernetes Service is a load balancing service for Pods.
- A Kubernetes Pod is created for each member in a cluster.
- A Kubernetes Pod is a group of Docker containers.
- Kubernetes creates a separate Docker container for networking.
Why Composite Application Support?

- Real world applications are complex and require multiple heterogeneous service runtimes (Cartridges) to host the application.
- These Cartridges may have dependencies to each other - startup order, dependency ratio, dependent scaling, termination behaviors, data sharing.
- Capable of creating Cartridge group and it provides more flexibility to handle group behaviors such as group scaling, load balancing, etc.
- Capable of creating blueprint of an application runtime by using simple structured JSON payload.
Cartridge Group

Group

Startup Order = C1,G2
Dependent Scaling = C1,G2
Termination Behaviour = Terminate Dependents

Group G1

min=2, max=4

Cartridge C1

CPU Threshold = 75%

Group G2

min=1, max=2

Group Scaling = True
Startup Order = C3, C2
Dependent Scaling = C2,C3
Termination Behaviour = Terminate All

Cartridge C2

min=2, max=6

CPU Threshold = 60%

Cartridge C3

min=1, max=3

Memory Threshold = 80%

diagram - 01
Sample Group Definition

```json
{
    "name": "group2",
    "isGroupScalingEnabled": "true",
    "cartridges": [ "c2", "c3" ],
    "dependencies": {
        "startupOrders": [ "cartridge.c3,cartridge.c2" ],
        "scalingDependents": [ "cartridge.c3, cartridge.c2" ],
        "terminationBehaviour": "terminate-all"
    }
}
```
Multi-factored Auto Scaling

What is it?
- Scaling algorithm can use multiple factors, such as
  - Load average of the instance
  - Memory consumption of the instance
  - In-flight request count in LB
Multi-factored Auto Scaling...

- Capable of predicting future load
  - Real time analysis of current load status using CEP integration
  - Predict immediate future load based on CEP resulting streams
  - Predicting equation $s=ut + \frac{1}{2} at^2$
  - $s=\text{predicted load}$, $u=\text{first derivative of current average load}$, $t=\text{time interval}$, $a=\text{second derivative of current load}$

Why should one care?
- Maximise resource utilization
- Easy to do capacity planning
- Dynamic load based resource provisioning
- Optimizing across multiple clouds
Scalable and Dynamic Load Balancing

How Scalable it is?
- In theory infinite
  - horizontal scaling
  - limited by resource (instance capacity) availability

How Dynamic it is?
- Load Balancers are spawned dynamically
  - LB too is a cartridge
- In case of multi-cloud, multi-region, LB can scale per cloud/region
- Per service cluster LB
What is unique about Stratos

- Cartridge based LB model
- Can bring any third-party LB
  - HAProxy, nginx, AWS ELB
  - As easy as plugging into LB extension API
Smart Policies

What are the smart policies?
- Auto scaling
- Deployment

Auto scaling policy
- Define thresholds values pertaining scale up/down decision
- Auto Scaler refer this policy
- Defined by DevOps

Deployment policy
- Defined how and where to spawn cartridge instances
- Defined min and max instances in a selected service cluster
- Defined by DevOps based on deployment patterns
Smart Policies

Why should one care?
- Can provide cloud SLA

What are the advantages?
- Make DevOps life easy
  - help keep to SLA
- Make SaaS app delivery life easy
  - do not have to worry about availability in application layer
Composite Application Model and Policy Model
Cloud Bursting

What is it?
- Expanding/provisioning application into another cloud to handle peak load.

Why Should one care?
- Resource peak time can be off-loaded to third party clouds/resources

What is unique about it?
- Can off-load to any cloud
  - Private, Public and Hybrid
- Whole application can replicated into bursting cloud with all configuration
- Can migrate application into another cloud without downtime
Logging, Metering and Monitoring

What details are?
- Instance up/down time
- Each and every instances health status
  - application health, load average, memory consumption
- Application logs

Why should one care?
- Centralize view for all logging, metering and monitoring

What are the advantages?
- DevOps life easy
  - centralized log viewer
  - centralized dashboard
- Easy to throttling
Demo - Docker, Kubernetes with autoscaling
More Information !

- http://stratos.apache.org
- https://cwiki.apache.org/confluence/display/STRATOS/
- https://github.com/coreos/flannel
- https://www.youtube.com/watch?v=tsk0pWf4ipw