INTEL NOTICE & DISCLAIMER

No license (express or implied, by estoppel or otherwise) to any intellectual property rights is granted by this document.

Intel disclaims all express and implied warranties, including without limitation, the implied warranties of merchantability, fitness for a particular purpose, and non-infringement, as well as any warranty arising from course of performance, course of dealing, or usage in trade.

This document contains information on products, services and/or processes in development. All information provided here is subject to change without notice. Contact your Intel representative to obtain the latest forecast, schedule, specifications and roadmaps.

The products and services described may contain defects or errors known as errata which may cause deviations from published specifications. Current characterized errata are available on request.

Copies of documents which have an order number and are referenced in this document may be obtained by calling 1-800-548-4725 or by visiting www.intel.com/design/literature.htm.

Intel, the Intel logo, Intel® are trademarks of Intel Corporation in the U.S. and/or other countries.

*Other names and brands may be claimed as the property of others

Copyright © 2016 Intel Corporation.
AGENDA

BACKGROUND

SPARK ON KUBERNETES
- Why Spark-on-K8s
- How it works
- Current Issues

JD.COM CASE STUDY
- Architecture
- Network Choice
- Storage Choice

SUMMARY
BACKGROUND
Spark* on Kubernetes*(K8s) is a new project proposed by the companies including Bloomberg, Google, Intel, Palantir, Pepperdata, and Red Hat.

The goal is to bring native support for Spark to use Kubernetes as a cluster manager like Spark Standalone, YARN*, or Mesos*.

However, the code is under reviewing([SPARK-18278](https://issues.apache.org/jira/wiki/display/SPARK/SPARK-18278)) and expected to be merged on Spark 2.3.0 release.

*Other names and brands may be claimed as the property of others.
WHY CHOOSE SPARK ON KUBERNETES?

HETEROGENEOUS COMPUTING

CPU + GPU + FPGA

Customers are asking to use an unified cloud platform to manage their applications. Based on Kubernetes*, we can ease to set up a platform to support CPU, GPU, as well as FPGA resources for Big Data/AI workloads.

*Other names and brands may be claimed as the property of others.
System Technologies & Optimization (STO)
SPARK ON KUBERNETES
SPARK ON DOCKER SOLUTIONS

Solution1 - Spark* Standalone on Docker*
- Run Spark standalone cluster in Docker.
- Two-tiers resource allocation (K8s->Spark Cluster->Spark Applications).
- Less efforts to migrate existing architecture into container environment.

Solution2 - Spark on Kubernetes*
- Use native way to run Spark on Kubernetes like Spark Standalone, YARN, or Mesos.
- Single tier resource allocation (K8s->Spark Applications) for higher utilization.
- Must re-write the entire logical program for resource allocation via K8s.

*Other names and brands may be claimed as the property of others.
**SOLUTION1 - SPARK* STANDALONE ON DOCKER**

- **Kubelet**
  - Spark Master
  - Spark Slave Pod
    - App1 Executor
  - Spark Slave Pod
    - App2 Executor

- **Spark**
  - Submit App1
  - Submit App2

- **Kubernetes**
  - Master

- **App1**
  - Executor

- **App2**
  - Executor

*Other names and brands may be claimed as the property of others.

System Technologies & Optimization (STO)
SOLUTION2 - SPARK* ON KUBERNETES*

Step 1: Kubernetes Master
- Spark-Submit App1

Step 1: Kubelet
- App1 Executor Pod
- App2 Executor Pod

Step 3: Kubernetes Master
- Spark-Submit App2

Step 2: Kubelet
- App1 Driver Pod
- App1 Executor Pod

*Other names and brands may be claimed as the property of others.

System Technologies & Optimization (STO)
HOW TO RUN SPARK* ON K8S*

bin/spark-submit \
--deploy-mode cluster \n--class org.apache.spark.examples.SparkPi \n--master k8s://http://127.0.0.1:8080 \n--kubernetes-namespace default \n--conf spark.executor.instances=5 \n--conf spark.executor.cores=4 \n--conf spark.executor.memory=4g \n--conf spark.app.name=spark-pi \n--conf spark.kubernetes.driver.docker.image=localhost:5000/spark-driver \n--conf spark.kubernetes.executor.docker.image=localhost:5000/spark-executor \n--conf spark.kubernetes.initcontainer.docker.image=localhost:5000/spark-init \n--conf spark.kubernetes.resourceStagingServer.uri=http://$ip:31000 \nhdfs://examples/jars/spark-examples_2.11-2.1.0-k8s-0.1.0-SNAPSHOT.jar

*Other names and brands may be claimed as the property of others.
Spark* on Kubernetes* has released “v2.2.0-kubernetes-0.3.0” based on Spark 2.2.0 branch

**Key Features**

- Support Cluster Mode only.
- Support File Staging in local, HDFS, or running a File Stage Server container.
- Support Scala, Java, and PySpark.
- Support Static and Dynamic Allocation for Executors.
- Support running HDFS inside K8s or externally.
- Support for Kubernetes 1.6 - 1.7
- Pre-built docker images
**DATA PROCESSING MODEL**

**PATTERN 1:** Internal HDFS*

- **Virtual Cluster**
  - Docker1
  - Docker2

  **Host**

  Use HDFS as file sharing server. HDFS runs in the same host to give elasticity to add/reduce compute nodes by request.

  Please refer to Spark and HDFS.

**PATTERN 2:** External HDFS

- **Virtual Cluster**
  - Docker1

  **Host**

  Use HDFS as file sharing server. HDFS runs outside in a long-running cluster to make sure data is persisted.

  Please refer to PR-350

**PATTERN 3:** Object Store

- **Virtual Cluster**
  - Docker1
  - Object Store

  **Host**

  Launch a File Staging Server to share data between nodes. Input and Output data can put in an object store. Streaming data directly via object level storage like Amazon S3, Swift.

---

**Storage Plan for Spark* on K8s***

The design rule is based on "whether the data must be persisted".

**Spark.local.dir:**
For Spark Data Shuffling. Use Ephemeral Volume. Now it uses docker-storage with diff. storage backend. EmptyDir is WIP.

**File Staging Server:**
For sharing data such as Jar or dependence file between computing nodes. Now it uses docker-storage. Local Storage support in Persist Volume(PV) is WIP.

*Other names and brands may be claimed as the property of others.
The resources are allocated in the beginning and cannot change during the executors are running. Static resource allocation uses local storage (docker-storage) for data shuffle.

A WIP feature is to use EmptyDir in K8s for this temporary data shuffle.

*Other names and brands may be claimed as the property of others.
Dynamic resource allocation uses shuffle service container for data shuffle.

There are two implementations:
1st is to run shuffle service in a pod.
2nd is to run shuffle service as a container with an executor.

*Other names and brands may be claimed as the property of others.*
CURRENT ISSUES FOR SPARK ON K8S

• Spark* Shell for Client Mode is not ready yet.
• Data Locality Support
• Storage Backend Support
• Container Launch Time is too long
• Performance Issues
• Reliability

*Other names and brands may be claimed as the property of others.
DEMO
JD.COM CASE STUDY
**JD.COM’S MOON SHOT**

- JD has used K8s* as cloud infrastructure management for several years.
- JD would like to use K8s to manage all the computing resources including CPU, GPU, FPGA, ...etc.
- Target for all AI workloads; Using the same cluster for training/inference.
- Across multiple Machine Learning framework including Caffe, TensorFlow, XGBoost, MXNet, ...etc.
- To optimize workloads for different resource allocation.
- Multi-tenancy support by different user accounts and resource pool.

Reference: https://mp.weixin.qq.com/s?__biz=MzA5Nzc2NDAxMg%3D%3D&mid=2649864623&idx=1&sn=f476db89b3d0ec580e8a63ff781444a37

*Other names and brands may be claimed as the property of others.
MOON SHOT ARCHITECTURE

Applications
- Image Recognition
- NLP
- Security Solutions
- Finance
- Public Cloud

Computing Engine
- TensorFlow*
- Caffe*
- MXNet*
- XGBoost*
- BigDL
- MLib
- Spark
- SQL
- Streaming
- Deeplearning4j
- Spark

Container Cluster
- Docker* + Kubernetes*

Infrastructure
- CPU
- GPU
- FPGA
- Ethernet
- InfiniBand
- OMNI-Path
- SSD
- HDD

Management Center
- Authority Mgmt.
- Task Mgmt.
- Procedure Mgmt.
- Monitor Center
- Logging Center

*Other names and brands may be claimed as the property of others.

System Technologies & Optimization (STO)
NETWORK CHOICE BY JD.COM
TYPES OF CONTAINER NETWORK

1. Bridge
   Bridge is the default network (docker0) in Docker*. Linux bridge provides a host internal network for each host and leverages iptables for NAT and port mapping. It is simple and easy, but with bad performance.

2. Host
   Container shares its network namespace with the host. This way provides high performance without NAT support, but limits with port conflict issue.

3. Overlays
   Overlays use networking tunnels (such as VXLAN) to communicate across hosts. Overlay network provides the capability to separate the network by projects.

4. Underlays
   Underlays expose host interfaces directly to containers running on the host. It supports many popular drivers like MACvlan, IPvlan, ...etc. Some other ways via Underlay network are Direct Routing, Fan Networking, Point-to-Point.

*Other names and brands may be claimed as the property of others.

System Technologies & Optimization (STO)
**NETWORK SOLUTIONS**

**Flannel***
A simple and easy to configure layer 3 network fabric designed for K8s. It runs flanneld on each host to allocate subnet and uses etcd to store network configuration. Flannel supports several backends including VXLAN, host-gw, UDP, ...etc.

**Calico***
An approach to virtual networking and network security for containers, VMs, and bare metal services, which provides a rich set of security enforcement capabilities running on top of a highly scalable and efficient virtual network fabric. Calico uses BGP to set up the network and it also supports IPIP methods to build up a tunnel network.

**Weave***
Weave creates a virtual network that connects Docker containers across multiple hosts and enables their automatic discovery.

**OpenVSwitch***

**Others**
*Other names and brands may be claimed as the property of others.*
Felix - The Calico agent runs on each machine that hosts endpoints. It is responsible for programming routes and ACLs.

Orchestrator Plugin - Orchestrator-specific code that tightly integrates Calico into that orchestrator. It supports OpenStack Neutron, Kubernetes, ...etc.

etcd - A distributed key-value store to provide the communication between components as a consistent data store.

BIRD - a BGP client that distributes routing information.

BGP Route Reflector(BIRD) - an optional BGP route reflector for higher scale
## NETWORK PERFORMANCE TESTING

All scenarios use `ab` command to connect to nginx* server with different IP address. “`ab -n 1000000 -c 100 -H"Host: nginx.jd.local" 172.20.141.72:80/index.html “

<table>
<thead>
<tr>
<th>No.</th>
<th>Scenario</th>
<th>Concurrency #</th>
<th>Total Time(s)</th>
<th>Request per Second</th>
<th>Waiting Time(ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Client -&gt; Nginx</td>
<td>50</td>
<td>50.044</td>
<td>19982</td>
<td>0.05</td>
</tr>
<tr>
<td>2</td>
<td>Client -&gt; Nginx (with Proxy)</td>
<td>50</td>
<td>53.84</td>
<td>18573</td>
<td>0.054</td>
</tr>
<tr>
<td>3</td>
<td>Weave: Client -&gt; iptables -&gt; Weave -&gt; Pod</td>
<td>50</td>
<td>132.839</td>
<td>7527</td>
<td>0.133</td>
</tr>
<tr>
<td>4</td>
<td>Calico w/ IPIP: Client -&gt; iptables -&gt; Calico -&gt; Pod</td>
<td>50</td>
<td>111.136</td>
<td>8998</td>
<td>0.111</td>
</tr>
<tr>
<td>5</td>
<td>Calico w/o IPIP: Client -&gt; iptables -&gt; Calico -&gt; Pod</td>
<td>50</td>
<td>59.218</td>
<td>16886</td>
<td>0.059</td>
</tr>
</tbody>
</table>

JD.com decides to pick up Calico since Calico provides better performance than Weave and Calico can still provide tunnel method(via IPIP) to set up network.

*Other names and brands may be claimed as the property of others.

System Technologies & Optimization (STO)
NETWORK CONSIDERATIONS

• Physical Network Support including Router, Switch, ..etc.
• Network Isolation with Multi-tenancy
• Network Size
• Network Performance
STORAGE CHOICE BY JD.COM
Spark* Shuffle uses Ephemeral Volumes
- Docker* Storage: Use devicemapper
- Volumes(Ongoing): #439 Use EmptyDir

File Staging for jar file
- Local in Spark Executors(Docker Storage)
- Remote HDFS*
- Create a Staging Server Container
- Persistent Volumes(Ongoing): #306 Use PV

Input/Output Data
- Remote HDFS
- Remote GlusterFS*

*Other names and brands may be claimed as the property of others.
## TYPES OF EPHEMERAL VOLUMES

<table>
<thead>
<tr>
<th>Linux distribution</th>
<th>Supported storage drivers</th>
</tr>
</thead>
<tbody>
<tr>
<td>Docker CE* on Ubuntu*</td>
<td><code>aufs</code>, <code>devicemapper</code>, <code>overlay2</code>, <code>overlay</code>, <code>zfs</code></td>
</tr>
<tr>
<td>Docker CE on Debian*</td>
<td><code>aufs</code>, <code>devicemapper</code>, <code>overlay2</code>, <code>overlay</code></td>
</tr>
<tr>
<td>Docker CE on CentOS*</td>
<td><code>devicemapper</code></td>
</tr>
<tr>
<td>Docker CE on Fedora*</td>
<td><code>devicemapper</code>, <code>overlay2(experimental)</code>, <code>overlay(experimental)</code></td>
</tr>
</tbody>
</table>

*Other names and brands may be claimed as the property of others.
TYPES OF PERSISTENT VOLUMES

- GCE* PersistentDisk
- AWS* EBS
- AzureFile*
- AzureDisk*
- Fibre Channel
- FlexVolume*
- Flocker*
- iSCSI
- Ceph* RBD
- CephFS*
- Cinder*
- GlusterFS*
- vSphareVolume*
- Quobyte* Volumes
- HostPath
- VMware Photon*
- Portworx* Volumes
- ScaleIO* Volumes
- StorageOS*
- emptyDir

*Other names and brands may be claimed as the property of others.
DATA LOCALITY ISSUE

In cloud environment, compute and storage resource are separated. This could highlight data locality issue with performance drop.

Some possible solutions can help to resolve data locality issues

• Choose right workloads, most workloads only need to read data and write data at beginning and end phase.
• HDFS* on Kubernetes
• Alluxio*

*Other names and brands may be claimed as the property of others.
## DATA LOCALITY IMPACT

<table>
<thead>
<tr>
<th>Workloads</th>
<th>Types</th>
<th>Locality</th>
<th>Datasize</th>
<th>Cluster Size</th>
<th>Network</th>
<th>Execution Time</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Terasort</td>
<td>IO</td>
<td>Local</td>
<td>320GB</td>
<td>5</td>
<td>1Gb</td>
<td>2119.926sec</td>
<td>1x</td>
</tr>
<tr>
<td>Terasort</td>
<td>IO</td>
<td>Remote</td>
<td>320GB</td>
<td>5 Spark + 3 Hadoop</td>
<td>1Gb</td>
<td>4212.029sec</td>
<td>1.98x</td>
</tr>
<tr>
<td>Terasort</td>
<td>IO</td>
<td>Local</td>
<td>320GB</td>
<td>5</td>
<td>10Gb</td>
<td>500.198sec</td>
<td>1x</td>
</tr>
<tr>
<td>Terasort</td>
<td>IO</td>
<td>Remote</td>
<td>320GB</td>
<td>5 Spark + 3 Hadoop</td>
<td>10Gb</td>
<td>548.549sec</td>
<td>1.10x</td>
</tr>
<tr>
<td>Kmeans</td>
<td>CPU</td>
<td>Local</td>
<td>240GB</td>
<td>5</td>
<td>10Gb</td>
<td>1156.235sec</td>
<td>1x</td>
</tr>
<tr>
<td>Kmeans</td>
<td>CPU</td>
<td>Remote</td>
<td>240GB</td>
<td>5 Spark + 3 Hadoop</td>
<td>10Gb</td>
<td>1219.138sec</td>
<td>1.05x</td>
</tr>
</tbody>
</table>

Note1: This testing is using 5-nodes bare metal cluster.
Note2: 4 SATA SSD per Spark and Hadoop node
Note3: Performance may impact in different configuration including the number of disk, network bandwidth, as well as different platform.
STORAGE CONSIDERATION

- Application Types
- Storage for Persistent Data
- Storage for Processing Data
- Storage Performance
SUMMARY

- Spark* on K8s* provides a cloud native way to run Spark on Cloud which not only can get better resource utilization but also integrate with more big data services.

- JD.com’s Moon Shot uses K8s to create a heterogeneous cloud infrastructure, it can support both CPU and GPU for their AI workloads.

- Spark on K8s is still under developing and there are many issues/features are waiting to be fixed/implemented.

*Other names and brands may be claimed as the property of others.
Intel RDT integration

Intel DPDK for Software Define Network (SDN)

Intel HW Feature Enabling

Spark* on K8s* Feature Support

Enable more customers to use Spark on K8s