# Kubernetes for Big Data Workloads

Abhishek Tiwari

Citation: *A. Tiwari*, "Kubernetes for Big Data Workloads", Abhishek Tiwari, 2017. doi:10.59350/wh60e-4g784

Published on: December 27, 2017

Kubernetes has emerged as go to container orchestration platform for data engineering teams. Kubernetes has a massive community support and momentum behind it. In 2018, a widespread adaptation of Kubernetes for big data processing is anitcipated. Organisations are already using Kubernetes for a variety of workloads[1][2] and data workloads are up next. Kubernetes has been particularly successful with microservices and contemporary applications. In fact, if we go by current trends containerised microservices running on Kubernetes are the future.

## Benefits

Containerized data workloads running on Kubernetes offer several advantages over traditional virtual machine/bare metal based data workloads including but not limited to

- better cluster resource utilization
- portability between cloud and on-premises
- frictionless multi-tenancy with versioning
- simple and selective instant upgrades
- faster development and deployment cycles
- isolation between different types of workloads
- unified interface to all types of workload

## Key challenges

Over last few years, we have seen several attempts to run data workload in the containers especially distributed big data frameworks like Apache Hadoop, Apache Storm[3], and Apache Spark[4][5] without any software modifications. Obviously, containerized data workloads have their own challenges. Some of these challenges[6] includes,

- lack of smart schedulers for optimal resource utilization
- faster access to external storage and data locality (I/O, bandwidth)
- distributed stateful data workloads (Zookeeper, Cassandra)
- optimised container networking and security
- overall performance comparable to bare metal
- support for logging, monitoring, and observability
- overall characteristic of data workload varies (batch vs. stream)

[1] Workload Characteristics and Candidates for Containerization
[2] Running Workloads in Kubernetes
[3] Apache Storm cluster using Kubernetes and Docker
[4] Apache Spark cluster using Kubernetes and Docker
[5] Using Spark and Zeppelin to process big data on Kubernetes 1.2
[6] Lessons learned running Hadoop and Spark in Docker

As you can see there are a few interesting challenges to run big data workload on Kubernetes. The good news is all of these challenges including performance can be tackled and in several cases, there are new developments to overcome them.

## Performance

A recent performance benchmark completed by Intel and BlueData using the BigBench benchmarking kit has shown that the performance ratios for container-based Hadoop workloads on BlueData EPIC are equal to and in some cases, better than bare-metal Hadoop [7]. The workloads for both test environments ran on apples-to-apples configurations. Under the hood, the BlueData used several enhancements to boosts the I/O performance and scalability of container-based clusters. As illustrated below, a similar performance ratio was observed for container-based Spark cluster vs. bare-metal Spark cluster.

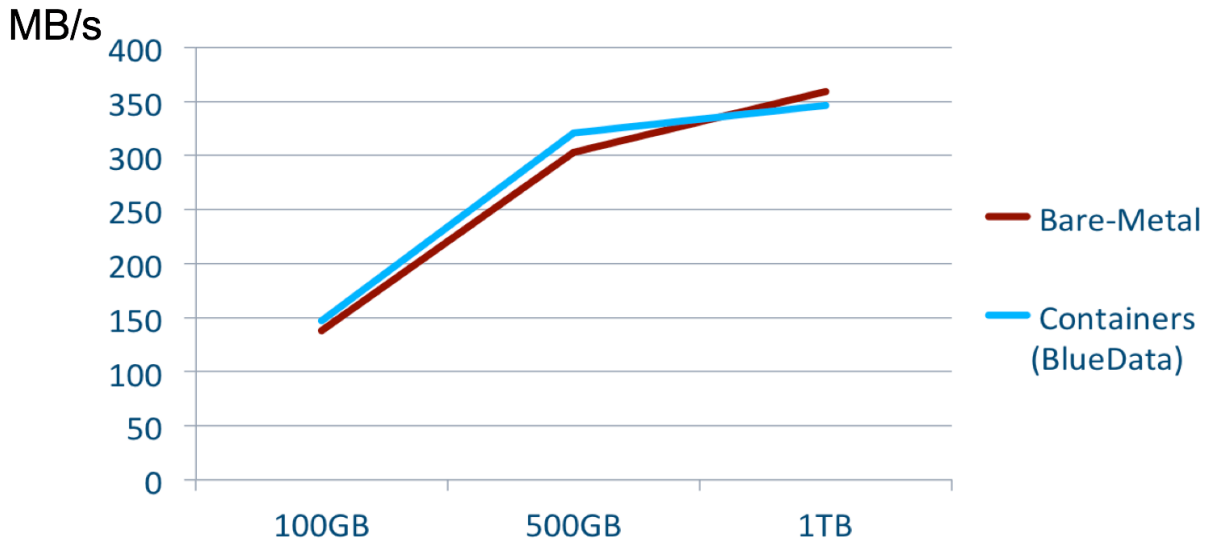


**Figure 1:** Spark on Docker: Performance. Credits BlueData

## Native frameworks

Rather than thinking data as a monolithic workload we can start considering it as a collection of containerized microservices. We are talking about new Kubernetes native big data frameworks. Kubernetes has all necessary primitives available to make it happen. Plus, Kubernetes offers us a verity of container patterns to play with. For instance, scatter/gather pattern can be used to implement a

[7] Bare-metal performance for Big Data workloads on Docker containers

MapReduce like batch processing architecture on top of Kubernetes. Similarly, event-driven stream data processing is a lot easier to implement as microservices running on top Kubernetes.

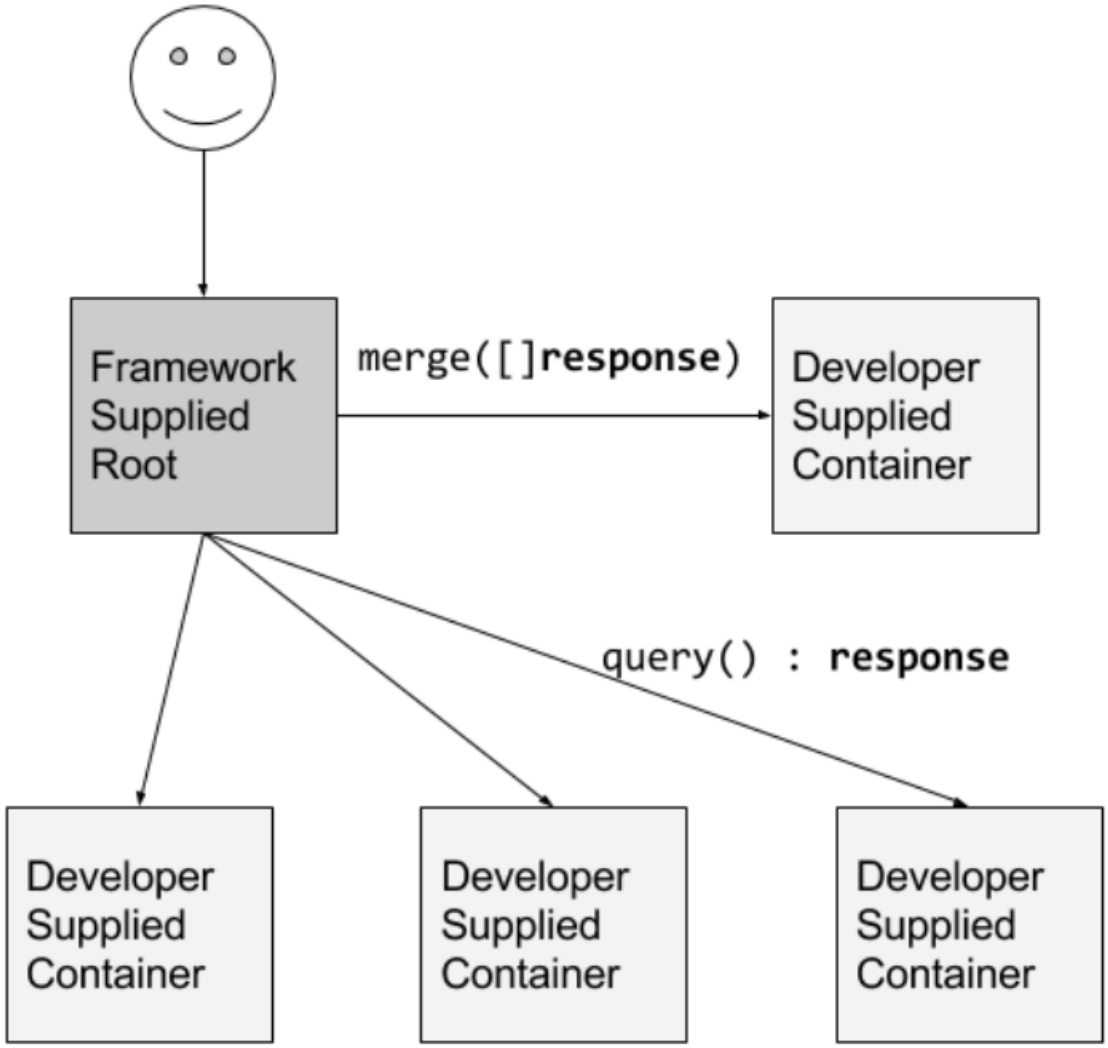


**Figure 2:** An example of the scatter/gather pattern: a reusable root container implements client interactions and request fan-out to developer-supplied leaf containers and to a developer-supplied merge container responsible for aggregating the results

A good example will be something like Pachyderm[8][9] - a relatively young project which implements a distributed Pachyderm File System (PFS) and a data-aware scheduler Pachyderm Pipeline System (PPS) on top of Kubernetes. Pachyderm uses default Kubernetes scheduler to implement

[8] Pachyderm: Building a Big Data Beast On Kubernetes
[9] Pachyderm: Data Pipelines

fault-tolerance and incremental processing. In addition, for FPS Pachyderm utilizes a copy-on-write paradigm which inspired by Git. Basically, Pachyderm is applying version control to your data as it's processed which processing jobs run on only the diff.

### Custom schedulers

A large number of companies are already running Kubernetes in production. In addition, you can run most of big data frameworks on Kubernetes but as mentioned in the previous section there are several challenges and lack of smart schedulers is one of them. To address this, the open source community is now trying to develop Kubernetes custom schedulers specifically optimised for big data workloads.

Kubernetes scheduler is responsible for scheduling pods onto nodes. Kubernetes ships with a default scheduler which provides a range of scheduling features. To schedule pods onto nodes, Kubernetes default scheduler considers several factors including individual and collective resource requirements, quality of service requirements, hardware constraints, affinity or anti-affinity specifications, data locality, inter-workload interference and so on. Using default scheduler's node affinity feature you can ensure that certain pods only schedule on nodes with specialized hardware like GPU, memory-optimised, I/O optimised etc. Similarly, pods affinity features allow you to place pods relative to one another.

Kubernetes allows you to run multiple schedulers simultaneously. To have more control over the scheduling of data workloads one can create custom schedulers. Apache Spark community is taking this approach to natively run Spark on Kubernetes[10][11][12][13]. It is important to note, Spark can already run on Kubernetes[14], but in a standalone mode. Spark with native Kubernetes scheduler has several advantages over standalone mode. For instance,

- Spark executors can be elastic depending on job demands since Spark can now communicate to Kubernetes to provision executors as required
- Spark can now leverage existing Kubernetes constructs like resource quota, namespaces, audit logging, etc.
- Spark can enforce strongly multi-tenant and multi-user isolation requirements by using namespaces, affinity, taints etc.

[10] Apache Spark enhanced with native Kubernetes scheduler back-end
[11] Support Spark natively in Kubernetes
[12] Support native submission of spark jobs to a Kubernetes cluster
[13] Apache Spark on Kubernetes
[14] Apache Spark cluster using Kubernetes and Docker

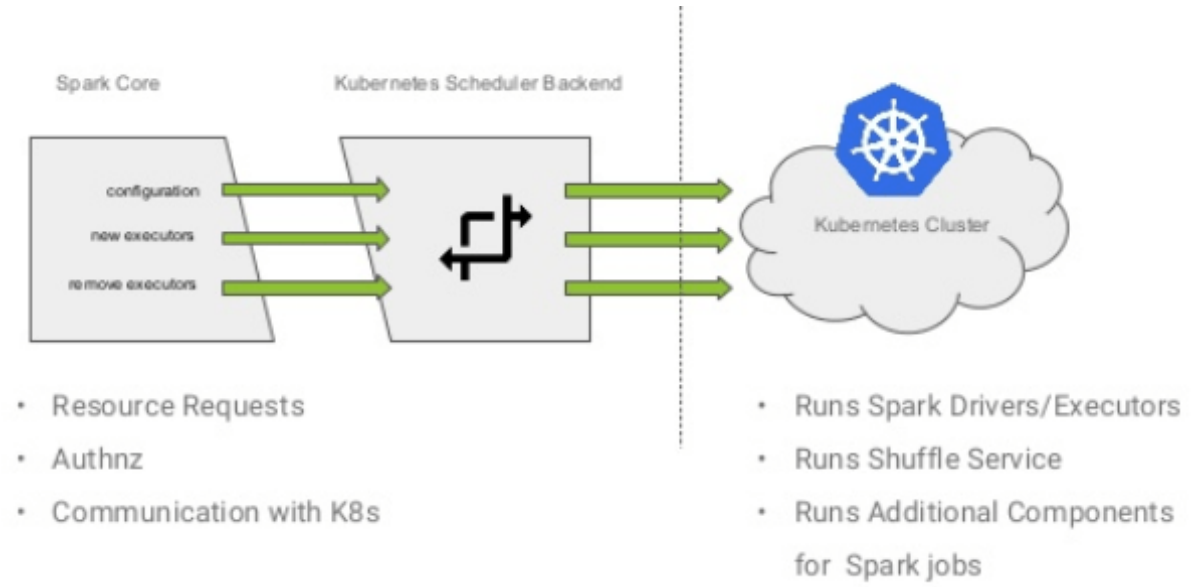


**Figure 3:** Apache Spark on Kubernetes. Image Credits Anirudh Ramanathan and Tim Chen.

Furthermore, the performance of Spark with native Kubernetes scheduler can be improved by running HDFS inside Kubernetes. This enables HDFS data locality by discovering the mapping of Kubernetes containers to physical nodes to HDFS datanode daemons[15]. Basically having HDFS in Kubernetes makes schedule data-aware.

Kubernetes custom scheduler is a relatively new feature which was released in version 1.6. It is possible to use YARN as Kubernetes custom scheduler. A few years back, Hortonworks tried to create a fork of Kubernetes with YARN scheduler[16] but at that time support for multiple schedulers was not available in Kubernetes.

Twitter Heron is another good example of this approach. Heron is a real-time, distributed stream processing engine developed at Twitter. It can be considered as a drop-in replacement for Apache Storm. Just like Apache Storm, Heron has a concept of topology. A topology is a directed acyclic graph (DAG) used to process streams of data and it can be stateless or stateful. Heron topology is essentially a set of pods that can be scheduled by Kubernetes.

Heron scheduler converts packing plan for a topology into pod definitions which is then submitted to Kubernetes scheduler via APIs[17]. A topology can be updated (scale up or down based on load) without having to build a new JAR to submit to the cluster.

[15] kubernetes-HDFS: HDFS cluster in Kubernetes
[16] A version of Kubernetes using Apache Hadoop YARN as the scheduler
[17] Experience porting Heron to Kubernetes

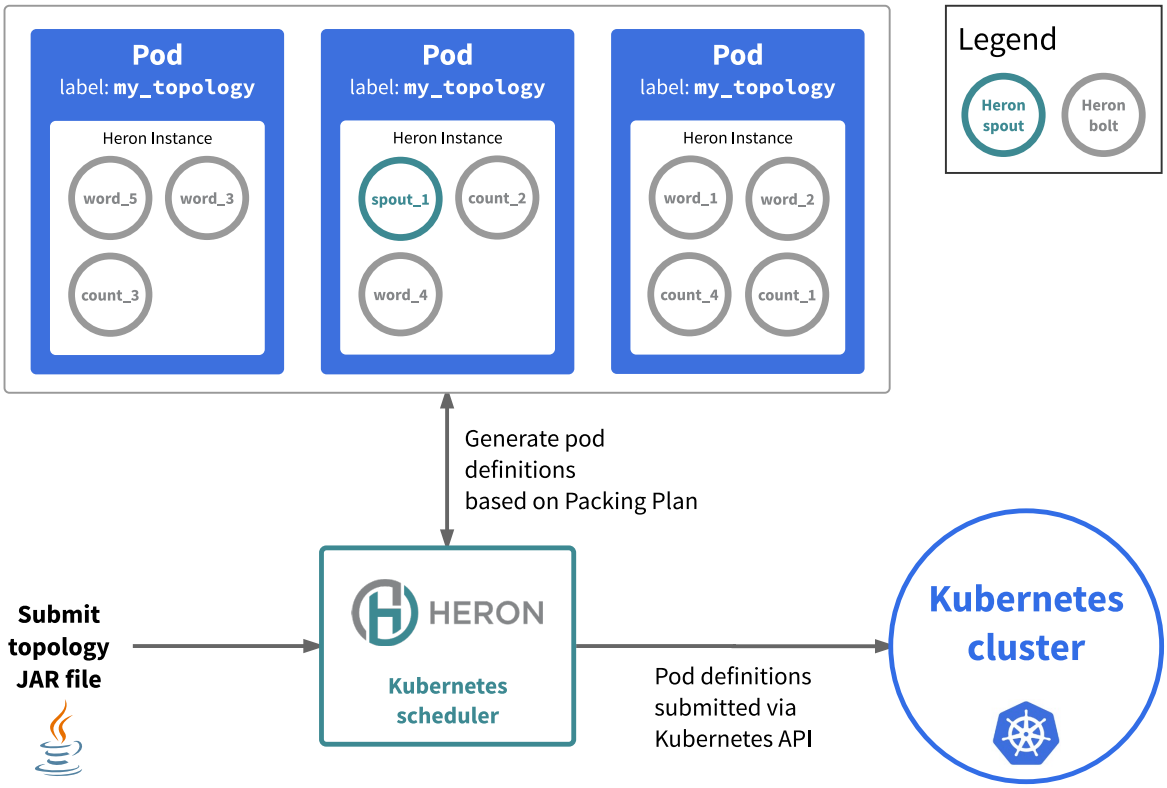


**Figure 4:** Heron on Kubernetes. Image credits Streamlio.

## Storage provisioning

Storage options have been another big roadblock in porting data workloads on Kubernetes particularly for stateful data workloads like Zookeeper, Cassandra, etc. But Kubernetes storage is evolving quite quickly. Recent Kubernetes updates provides extensive storage automation capabilities and options,

- automatically attach and detach storage, format disk, mount and unmount volumes per the pod spec so that pods can move seamlessly between nodes[18].
- direct access to raw block storage[19] without the abstraction of a filesystem for workloads that require consistent I/O performance and low latency.
- topology-aware volume scheduling[20] i.e. now a pod can influence where it is scheduled based on the availability of storage like must have local SSD storage.

[18] Dynamic Storage Provisioning
[19] Raw Block Consumption in Kubernetes
[20] Volume Topology-aware Scheduling

- ability to increase the size of persistent volume[21] after it has been provisioned while supporting various scenarios such as online/offline with data/without data.
- general availability of the StatefulSets[22][23] to manage pods with the state i..e. pods are created from the same spec, but they are unique hence not interchangeable

These new Kubernetes storage options have enabled us to deploy more fault-tolerance stateful data workloads on Kubernetes without the risk of data loss. For instance, by levering PersistentVolumes, a custom Cassandra Seed Provider, and StatefulSets we can provide a resilient installation of Cassandra[24].

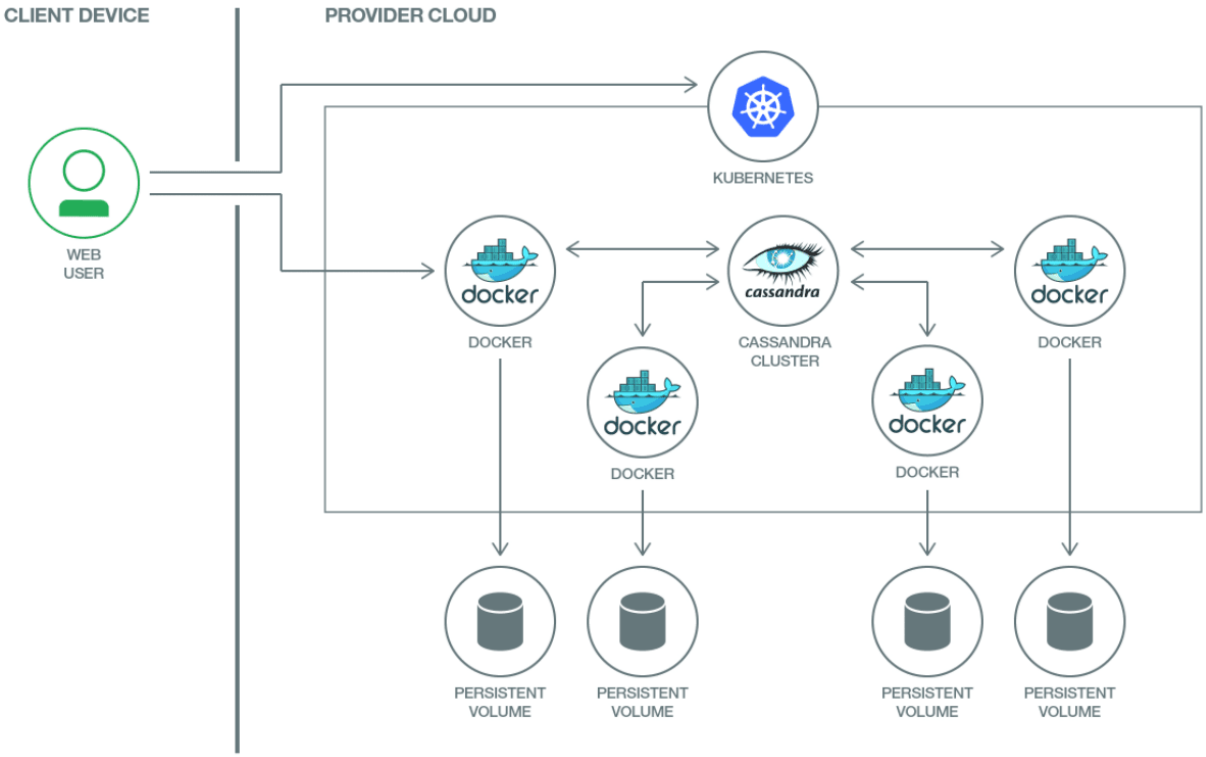


**Figure 5:** Scalable and resilient multi-node Cassandra deployment on Kubernetes Cluster using PersistentVolume and StatefulSets. Image credits IBM.

## Final thoughts

In this article, we covered several new developments required to productionize big data workloads on Kubernetes. In particular, we discussed custom schedulers and storage options to port existing

[21] Growing Persistent Volume size
[22] StatefulSets
[23] Fault Tolerant Stateful Services on Kubernetes
[24] Deploying Cassandra with Stateful Sets

big data frameworks on Kubernetes. These changes will have a big impact in 2018 and I anticipate Kubernetes will become defacto scheduler for big data workloads by effectively replacing YARN and Mesos. Lastly, I also think there is definitely a place for new Kubernetes native big data frameworks.