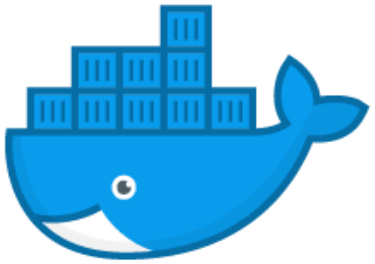
R&D Laboratory

LXC

+

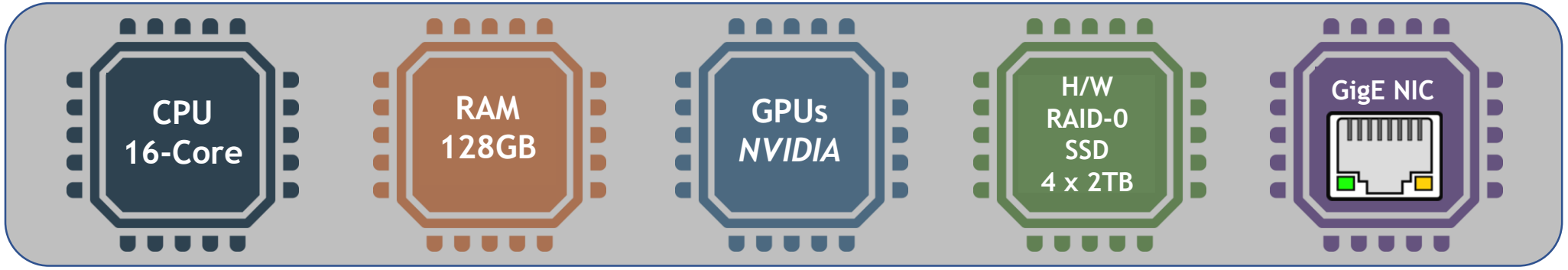
Kubernetes

+



docker

High-end server configured to assume arbitrary **application stacks** for **R&D** of client engagements. (© N. MILTON VEGA)



Fedora Host + LXC Guests (Fedora + Kubernetes + Docker-CE)

LXC O/S Containers (Virtual Private Servers)

- vps00
- vps01
- vps02
- vps03
- vps04
- vps05
- vps ...
- vps ...
- vpsN

k8s App Containers Kubernetes + Docker

- service-01
- service-02
- service-03
- service-04
- service-05
- service-06
- service ...
- service ...
- service-N

TEST / DEV STACK
- Storm Cluster
- Redis Cluster
- Cassandra Cluster
- Kafka Brokers
- Python-3 VENV

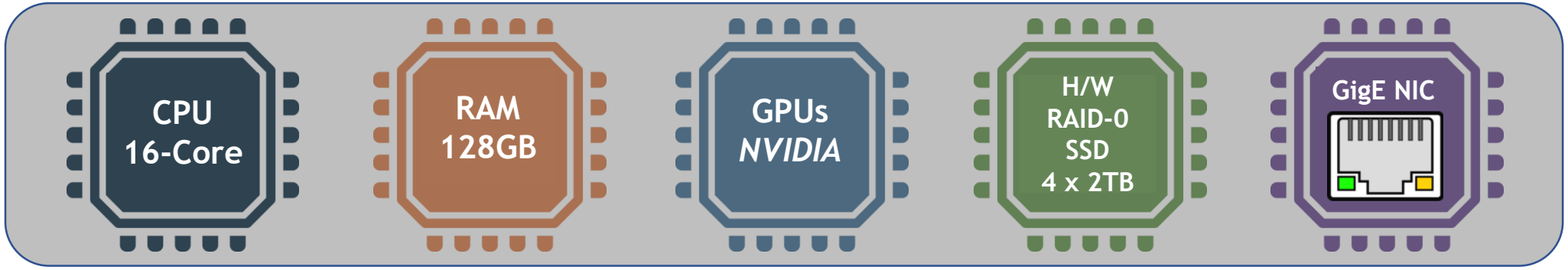
TEST / DEV STACK
- Cloudera Cluster
- CDH-5 / CDH-6
- CDH Mgt Console

TEST / DEV STACK
- Spark 2.x pySpark
- Kafka Brokers x 3
- ZooKeeper
- MongoDB

...

TEST / DEV STACK
- Tensorflow
- Scikit-Learn
- Pytorch
- Anaconda
- ...

High-end server configured to assume arbitrary **application stacks** for **R&D** of client engagements. (© N. MILTON VEGA)



Fedora Host + LXC Guests (Fedora + Kubernetes + Docker-CE)

LXC O/S Containers (Virtual Private Servers)

vps00 vps03 vps ...
vps01 vps04 vps ...
vps02 vps05 vps*N*

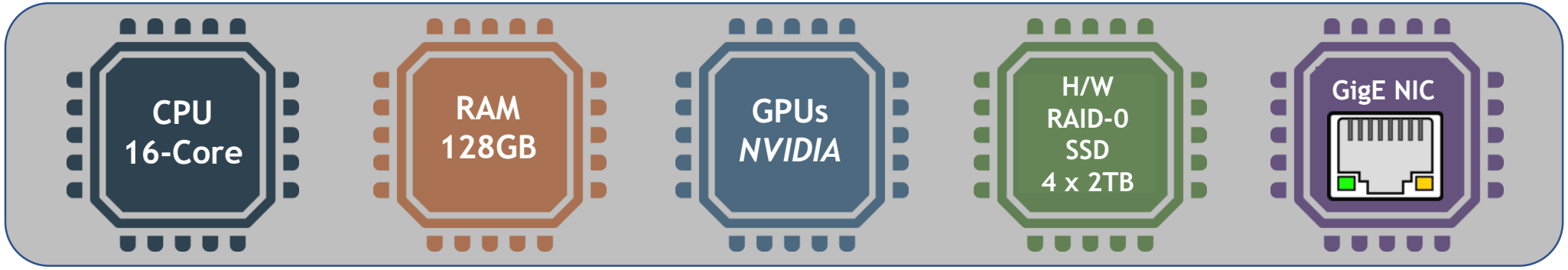
TEST / DEV STACK
- Storm Cluster
- Redis Cluster
- Cassandra Cluster
- Kafka Brokers

TEST / DEV STACK
- Cloudera Cluster
- CDH-5 / CDH-6
- CDH Mgt Console

LXC O/S Container Example Use-Case

Circa 2012, while architecting a **Real-Time Trade-Stream Anomaly Analytics** platform for **BofA/ML**, I used **Apache Storm** for **Complex Event Processing**. During the design I discovered a show-stopping deficiency with Storm and needed to reverse-engineer its behavior under failure scenarios; to see if a workaround was possible. At the time Storm had only a small GitHub page discussing it, and no books. And since my query to the Storm community about the issue went unanswered (http://bit.ly/1bsBooT), I used this platform to instrument behaviors myself. I devised a solution based on observations and published my results here: https://jupyter.ai/apache-storm/

High-end server configured to assume arbitrary **application stacks** for **R&D** of client engagements. (© N. MILTON VEGA)



Fedora Host + LXC Guests (Fedora + Kubernetes + Docker-CE)

LXC O/S Containers (Virtual Private Servers)

vps00 vps03 vps ...
vps01 vps04 vps ...
vps02 vps05 vps*N*

TEST / DEV STACK
- Storm Cluster
- Redis Cluster
- Cassandra Cluster
- Kafka Brokers

TEST / DEV STACK
- Cloudera Cluster
- CDH-5 / CDH-6
- CDH Mgt Console

LXC O/S Container Example Use-Case

Like **Apache Storm**, stacks requiring a proper O/S are deployed in **LXC Containers**. Another example of this is a **Cloudera Hadoop** cluster, where Docker containers are neither provided nor supported.

Six (qty. 6) LXC Containers serve this CDH use-case:
- vps00: **Master** node for big data services
- vps01: **Worker** node for shards / executors
- vps02: **Worker** node for shards / executors
- vps03: **Worker** node for shards / executors
- vps04: **Worker** node for shards / executors
- vps10: CDH SCM **Management** node

Issuing `lxc-start(1)` against these six (qty. 6) containers erect this CDH cluster on-demand.

Cloudera Manager Clusters Hosts Diagnostics Audits Charts Administration Search Support admin

Home Status All Health Issues Configuration All Recent Commands Add Cluster

PRISMALYTICS (CDH 6.1.0, Parcels)

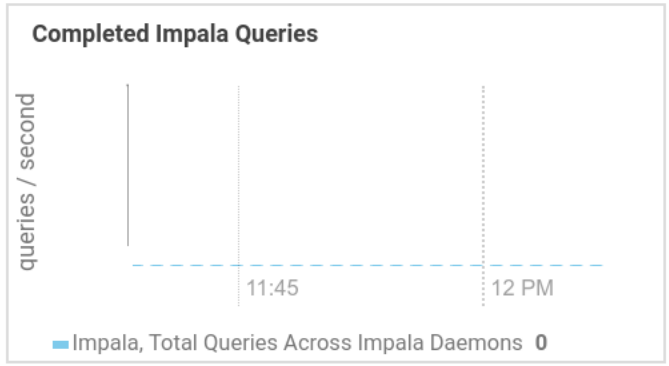
- 6 Hosts
- HBase
- HDFS
- Hive
- Hue
- Impala
- Oozie
- Spark
- YARN (MR2 In...
- ZooKeeper

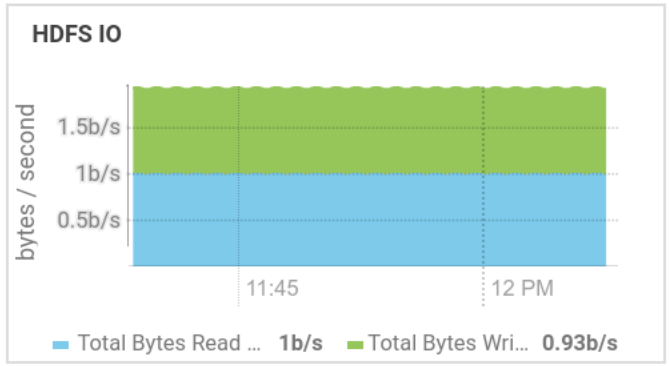
Cloudera Management Service

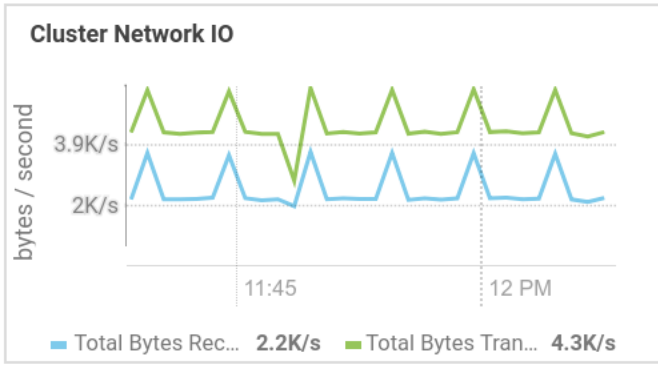
- Cloudera Man...

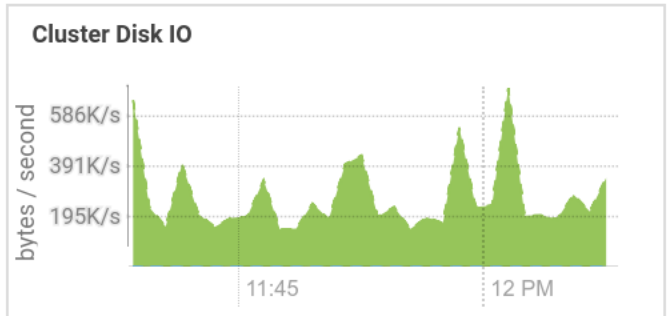
Charts

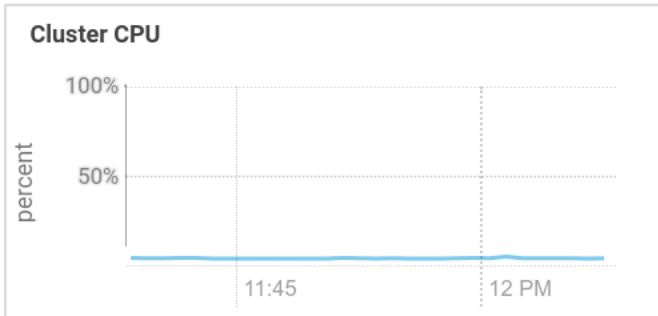
30m 1h 2h 6h 12h 1d 7d 30d





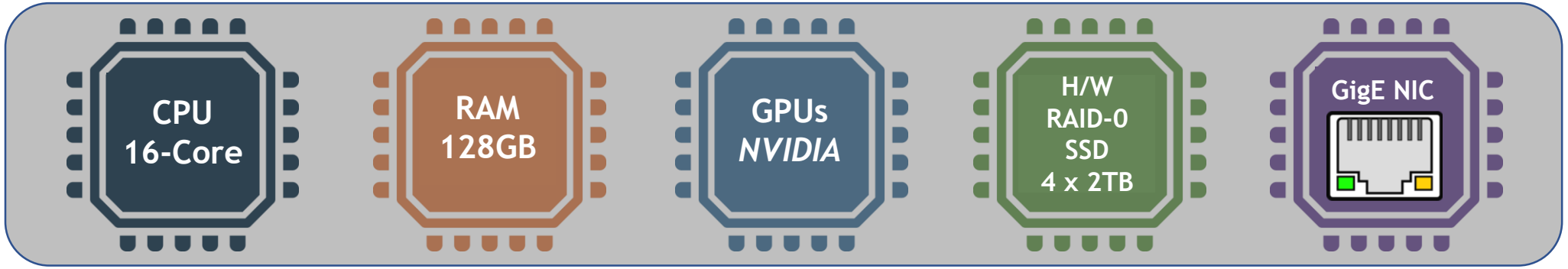






Feedback

High-end server configured to assume arbitrary **application stacks** for **R&D** of client engagements. (© N. MILTON VEGA)



Fedora Host + LXC Guests (Fedora + Kubernetes + Docker-CE)

k8s + Docker-CE Container Example Use-Case

Conversely, stack components with lightweight O/S dependencies are deployed in **k8s** + **Docker Containers** and run as coordinated services via **YAML** specs. For the stack shown, containers provide **Kafka**, **ZooKeeper** and **MongoDB** service endpoints that **Spark 2.x** connects to; thus erecting a streaming-data analytics pipeline on-demand. The service definition specifies three Kafka brokers, which is needed to simulate distribution of messages across topic-partitions by partition key. Tearing down a stack and replacing it with another is measured in minutes. Some results and Jupyter Notebooks derived from using these stacks are shared here: https://jupyter.ai

k8s App Containers
Kubernetes + Docker

service-01	service-04	service …
service-02	service-05	service …
service-03	service-06	service-*N*

TEST / DEV STACK
- Spark 2.x PySpark
- Kafka Brokers x 3
- ZooKeeper
- MongoDB
- Python-3 VENV

…

TEST / DEV STACK
- …
- …
- …
- …
- …