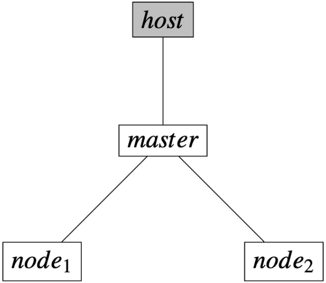
Analysis of computer clusters

Christian Faccio

March 28, 2025

Cluster Characteristics



- **One master node** connected to the Internet through NAT connection
- **Two worker nodes** connected to the master node with Internal Network (not directly to Internet)
- 2GB of RAM, 2 CPUs, 25GB of storage each

VM steps

1. Configuration of the master node
    - Adapters
    - Port Forwarding
    - (SSH connection to the host)
    - Network configuration
2. Creation of the worker nodes
3. DHCP and DNS configuration
4. Configuration of the Master node as Gateway
5. SSH on worker nodes
6. Creation of a distributed file system

Containers' steps

1. Creation of a `Dockerfile`
2. Creation of the `docker-compose.yaml` file
3. Start the containers

```
docker-compose up -d
ssh -i ssh_keys/id_rsa -p <port> user@localhost
```

Dockerfile

```
FROM ubuntu:latest

# Set non-interactive mode for apt-get
ENV DEBIAN_FRONTEND=noninteractive

# Install required packages
RUN apt-get update && apt-get install -y \
    openssh-server rsync iputils-ping \
    sysbench stress-ng iozone3 iperf3 \
    netcat-openbsd wget unzip hpcc \
    mpich vim \
    openmpi-bin openmpi-common openmpi-doc libopenmpi-dev \
    sudo \
    && rm -rf /var/lib/apt/lists/*

# Create SSH folder and set correct permissions
RUN mkdir -p /var/run/sshd /home/user/.ssh /shared \
    && chmod 700 /home/user/.ssh

# Create a new user 'user' with a home directory
RUN useradd -m -s /bin/bash user \
    && echo "user:userpassword" | chpasswd \
    && echo "user ALL=(ALL) NOPASSWD:ALL" >> /etc/sudoers

# Ensure SSH is configured for user
RUN sed -i 's/#PermitRootLogin prohibit-password/PermitRootLogin no/' /etc/ssh/sshd_conf
    && sed -i 's/UsePAM yes/UsePAM no/' /etc/ssh/sshd_config \
    && sed -i 's/#PubkeyAuthentication yes/PubkeyAuthentication yes/' /etc/ssh/sshd_conf
    && sed -i 's|#AuthorizedKeysFile.*|AuthorizedKeysFile .ssh/authorized_keys|' /etc/ss

# Copy SSH keys for user (passwordless login)
COPY ssh_keys/id_rsa.pub /home/user/.ssh/authorized_keys
COPY ssh_keys/id_rsa /home/user/.ssh/id_rsa

# Set correct permissions for SSH keys (user)
RUN chmod 600 /home/user/.ssh/id_rsa /home/user/.ssh/authorized_keys \
    && chown -R user:user /home/user/.ssh

# Expose SSH port
EXPOSE 22

# Switch to user
USER user
WORKDIR /home/user

# Start SSH service correctly with host key generation
CMD sudo ssh-keygen -A && sudo /usr/sbin/sshd -D -e && sudo chown -R user:user /shared &
```

Figure: `Dockerfile`

```yaml
services:
  master1:
    build: .  # Use the Dockerfile in the current directory to build the image
    container_name: master  # Set the container name to 'master'
    networks:
      - my_network  # Attach this container to the custom bridge network
    deploy:
      resources:
        limits:
          cpus: "2"  # Restrict the container to use a maximum of 2 CPU cores
          memory: 2G  # Limit the container's memory usage to 2GB
    ports:
      - "2220:22"  # Map port 2220 on the host to port 22 inside the container (SSH acce
    volumes:
      - shared_volume:/shared  # Mount shared volume for data exchange between containe
      - ./ssh_keys:/root/.ssh  # Mount pre-generated SSH keys for passwordless access
    tmpfs:
      - /shared:mode=777  # Create a temporary filesystem at /shared with full permissic

  node1:
    build: .  # Use the same Dockerfile for worker node
    container_name: node1  # Set the container name to 'node1'
    networks:
      - my_network  # Attach to the same network as the master node
    deploy:
      resources:
        limits:
          cpus: "2"
          memory: 2G
    ports:
      - "2221:22"  # Assign a different SSH port for node1
    volumes:
      - shared_volume:/shared  # Mount shared volume to enable data sharing
      - ./ssh_keys:/root/.ssh  # Use the same SSH keys for passwordless login
    tmpfs:
      - /shared:mode=777  # Temporary shared filesystem with full permissions

  node2:
    build: .  # Build using the same configuration
    container_name: node2  # Name this container 'node2'
    networks:
      - my_network
    deploy:
      resources:
        limits:
          cpus: "2"
          memory: 2G
    ports:
      - "2222:22"  # Assign another unique SSH port for node2
    volumes:
      - shared_volume:/shared
      - ./ssh_keys:/root/.ssh
    tmpfs:
      - /shared:mode=777

networks:
  my_network:
    driver: bridge  # Use a bridge network for inter-container communication

volumes:
  shared_volume:
    driver: local  # Use a local volume for persistent shared storage
```

Figure: `docker-compose.yaml`

Measuring performances

Different tests have been performed to measure the performances of the clusters:

- **HPCC**: Tests computation, memory access, and communication efficiency.
- **Iperf3**: Tests network performance between nodes.
- **Stress-ng**: Tests CPU, memory, and I/O performance under stress.
- **Sysbench**: Tests CPU and memory performance under stress.
- **IOzone**: Tests disk I/O performance.

```
mpirun -np 4 -hostfile hosts hpcc
```

Test	Unit	VMs	Containers
MPIRandomAccess	GUP/s	0.003	0.01
PTRANS (Wall)	s	0.550	0.42
StarDGEMM	Gflop/s	2.909	3.95
StarSTREAM Copy	GB/s	23.20	21.66
MPIFFT	Gflop/s	2.133	8.43
Avg. Ping Pong Bandwidth	GB/s	4.756	17.148
HPL	Gflop/s	10.64	12.32

Table: Main results of the HPCC tests

Overall, containers provide a more efficient execution environment for most HPCC workloads, especially in **memory access**, **computational efficiency**, and **communication latency**.

Iperf3

```
iperf3 -s #on the server
iperf3 -c <server_ip> #on the client
```

Category	Nodes	Transfer(GB)	Bitrate(GB/s)
VMs	Master-Node1	2.09	1.79
	Node1-Node2	2.15	1.84
Containers	Master-Node1	142	122
	Node1-Node2	140	120

Table: Main results of the Iperf3 tests categorized by type

Overall, containers are more efficient, with significantly **higher transfer rates** and **fewer network issues** compared to VMs.

Stress-ng

mpirun –hostfile hosts -np 4 stress-ng –cpu 2 –timeout 60s
mpirun –hostfile hosts -np 4 stress-ng –vm 2 –vm-bytes 1G
mpirun –hostfile hosts -np 4 stress-ng –io 2 –timeout 60s

	Bogo ops		**Bogo ops/s**	
Test	**VM**	**Containers**	**VM**	**Containers**
CPU	21532	24523	358.71	408.43
Memory	760052	7099658	12634.09	118275.80
I/O	2799634	505508	46659.86	8425.03

Table: Main results of the Stress-ng tests

Overall, containers demonstrate **better CPU and memory handling**, while VMs outperform containers in **disk I/O operations**.

Sysbench

```
mpirun --hostfile hosts -np 4 sysbench cpu
--cpu-max-prime=20000
```

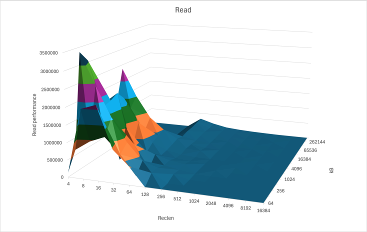
	Ops/s		Avg. Latency	
Test	**VM**	**Containers**	**VM**	**Containers**
CPU	2,198.45	3,303.00	0.46	0.31
Memory	19,327.09	41,471.36	0.05	0.03

Table: Main results of the Stress-ng tests (max values)

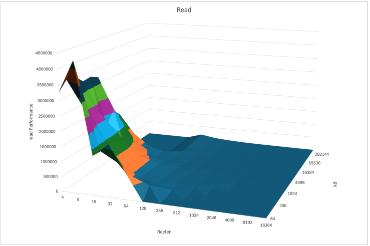
Overall, containers demonstrate better CPU and memory handling, with **higher throughput** and **lower latency** compared to VMs.

IOzone

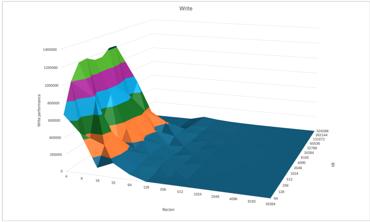
VM Read



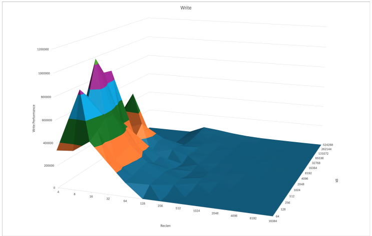
Containers Read



VM Write



Containers Write



iozone -+m /shared/machines.txt -f /shared/testfile -a -R -O

Conclusion

Containers generally provide better **performance** and **efficiency** for high-performance computing, networking, and I/O workloads. VMs still offer **stronger isolation**, but for resource-intensive tasks with heavy inter-process communication, containers tend to be **faster** and more **lightweight**.