Highly Available Container Orchestration

Author: David Vossel <dvossel@redhat.com>

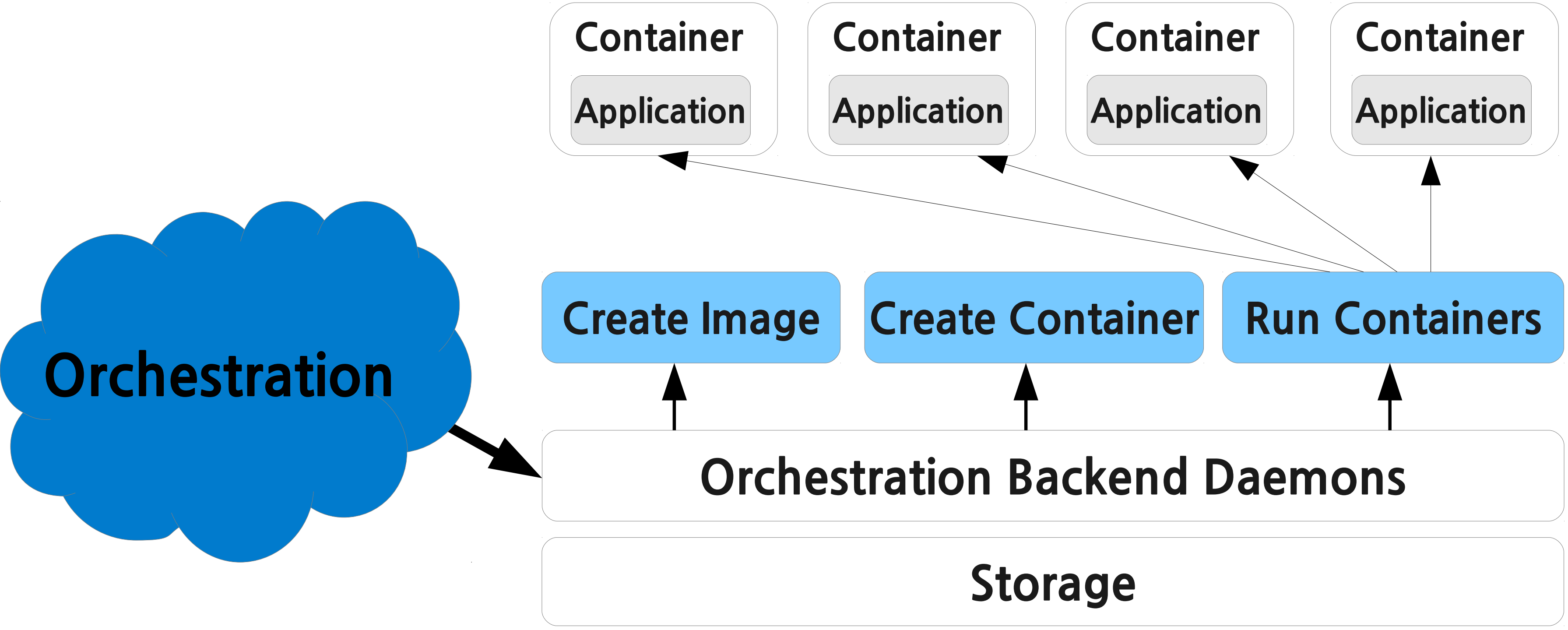
Date: 09/03/2014
Version: 2

Overview

- The Observation:
  - Container and VM orchestration tools are becoming common place, with new projects aiming to tackle orchestration in different ways popping up all the time.
- The Problem:
  - As these orchestration tools begin to be deployed in production environments, the concept of how to make these deployments Highly Available begins to be explored.
- The Solution:
  - This presentation aims to give a high level introduction of where the Pacemaker High Availability Stack fits in with container orchestration.

Why HA?

- To answer this, lets take a high level look at how container orchestration is generally done.
  - Orchestration front end executes commands on remote hosts using API supplied by backend daemons
  - Backend daemons expose API's for managing container images and running containers.

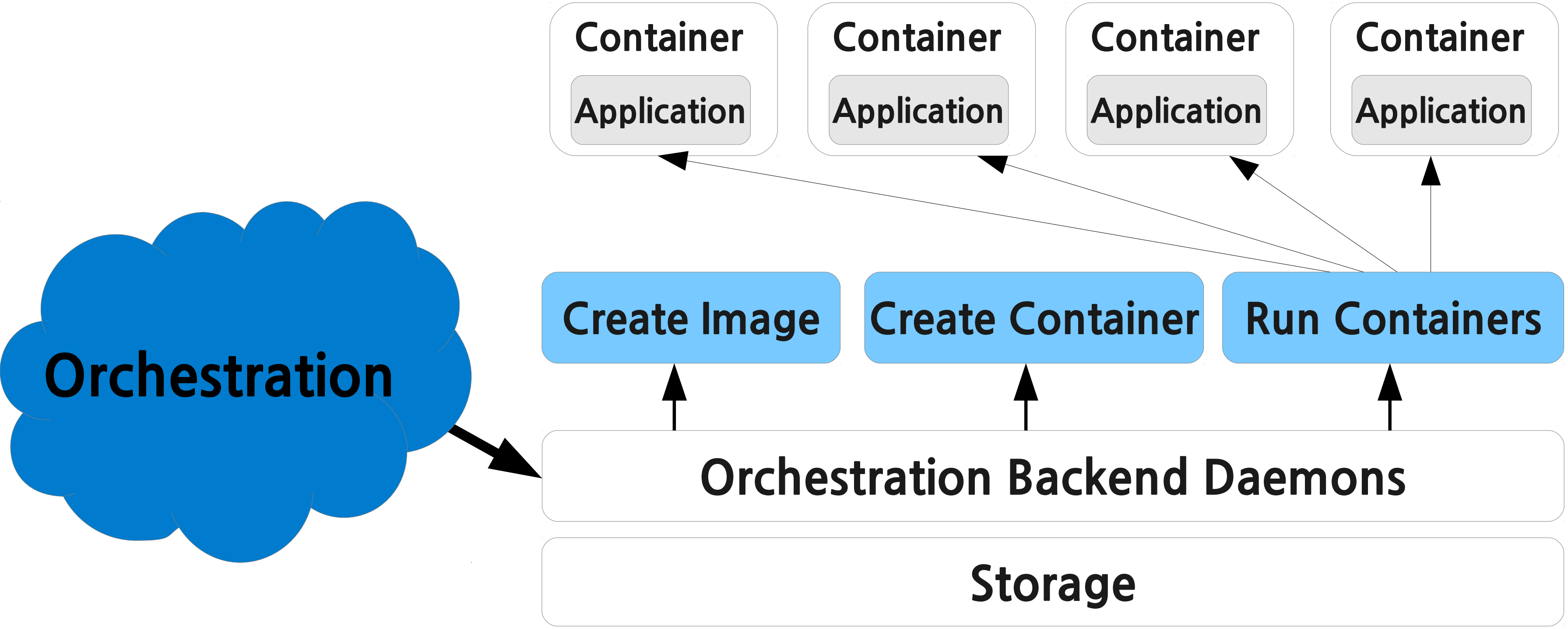


Side Note on Orchestration...

- We will not discuss orchestration specifics in this document. Below are the orchestration definitions as they apply to this discussion.

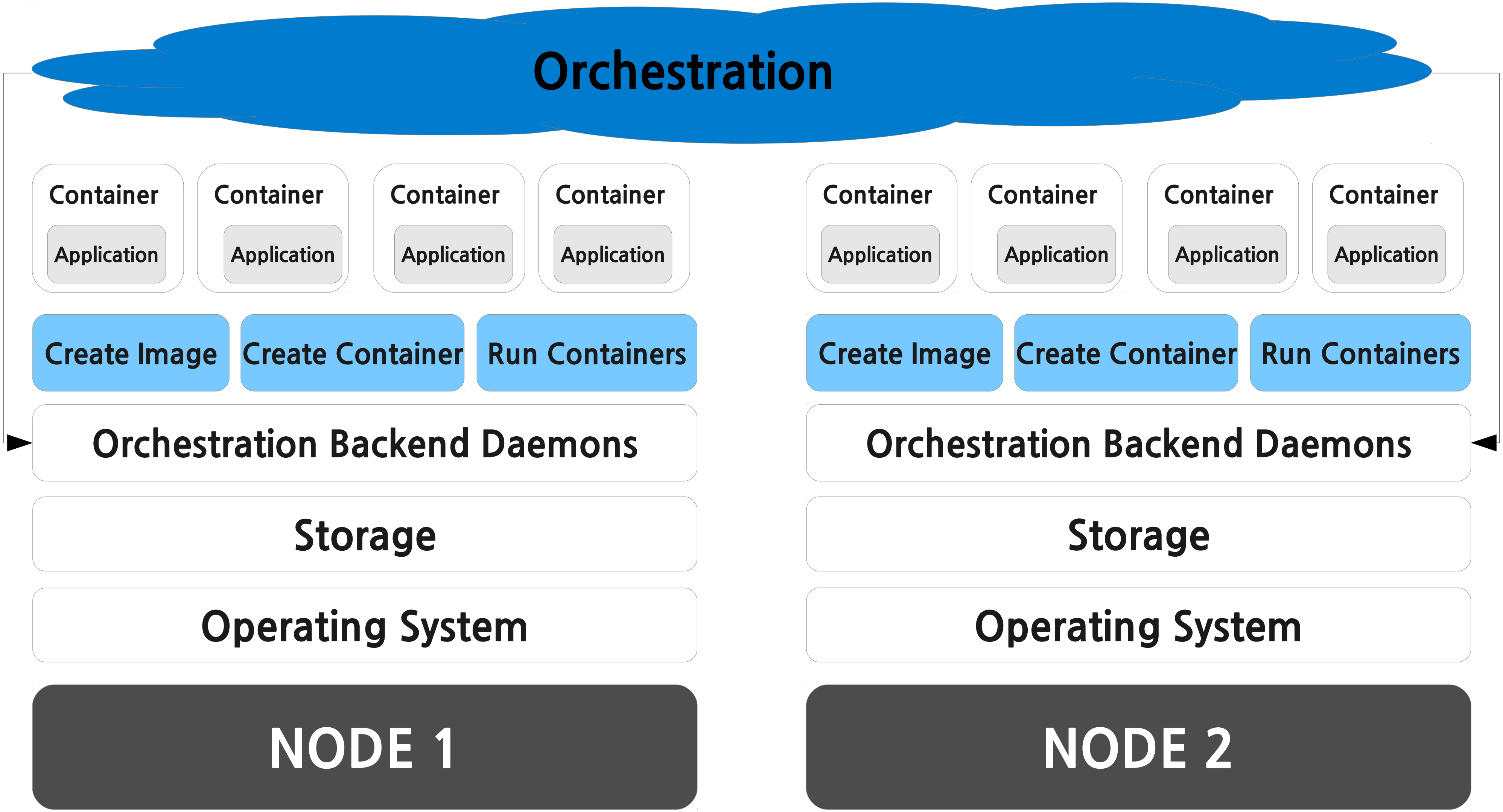
  Orchestration: Any method (automated or manual) by which containers are created and launched remotely.

  Orchestration Backend Daemons: Any local daemons required to execute the remote actions invoked during orchestration. This can involve REST API daemons, openstack components, libvirt, and even something as simple as ssh.



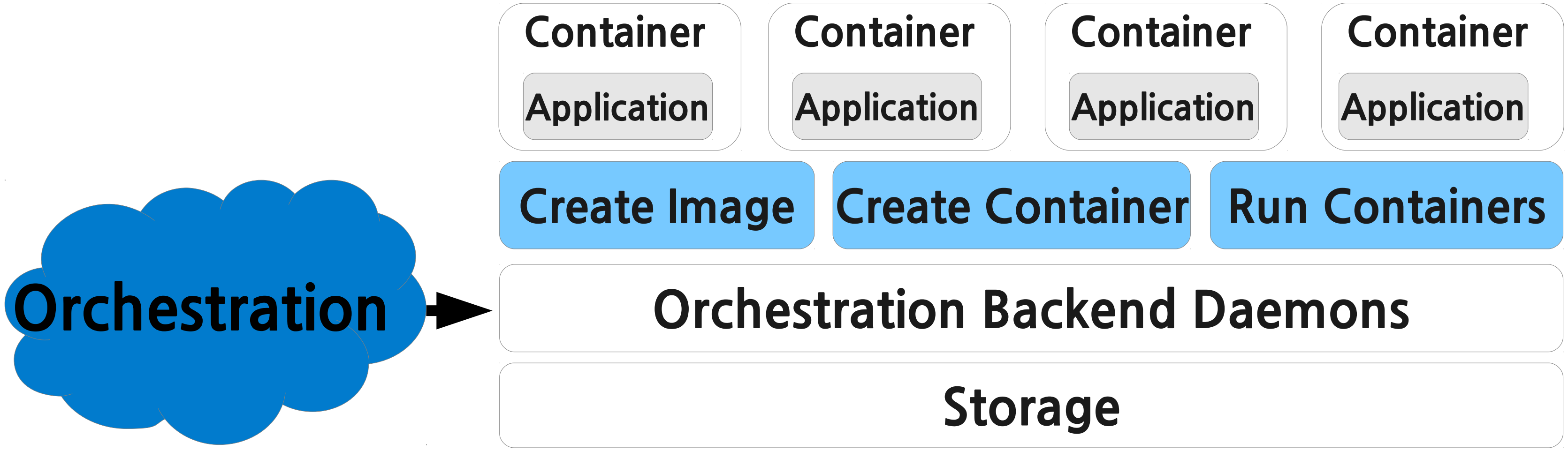
Big picture overview

- Without the HA stack the big picture looks like this, where remote orchestration handles container management across multiple nodes.



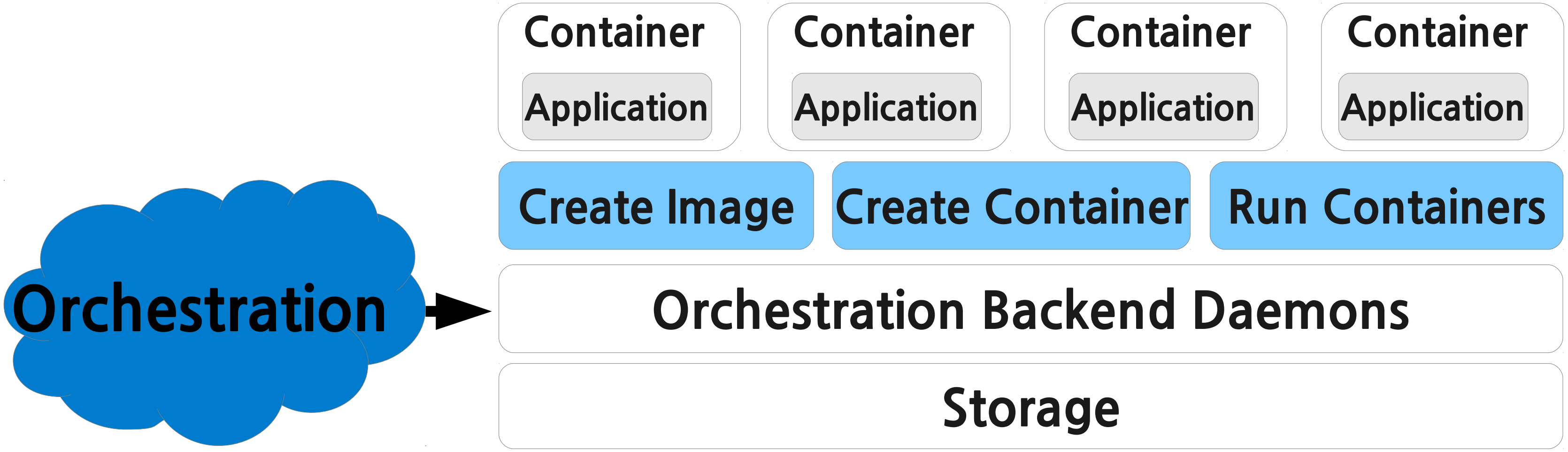
Problems...

- This model lacks fault tolerance and the ability to self heal.
  - No storage recovery.
  - Limited backend daemon recovery (systemd might auto-restart)
  - Limited container recovery.
    - No recovery of containers when backend daemons or lxc technology is down.
    - Lacks ability to monitor container applications.
  - No fencing



More Problems...

- Without a cluster manager, unpredictable scenarios arise
  - Backend API daemons running without container storage mounted
  - Orchestration backend API unavailable while containers are running.
  - Containers running with failed/degraded application inside.
  - Network communication between orchestration and remote backend API is down. Orchestration has no ability accurately determine the remote API is down without the use of fencing.



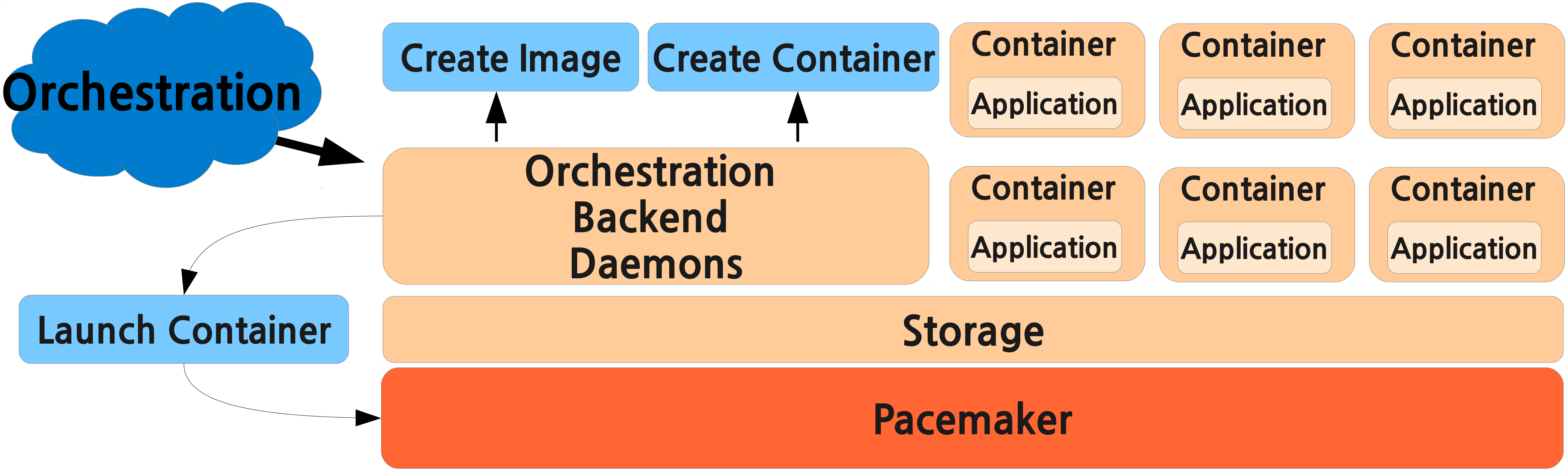
But my distributed container manager is different....

- As more orchestration tools emerge the developers of these tools realize the need for fault tolerance. As a result many begin build HA looking features into the distributed system.
- **Don't be fooled**. These are not HA solutions. HA must be a **holistic approach**. It is not possible to build a true HA solution without taking the entire system into account.
- There's one simple question that typically exposes a faulty HA solution immediately. **“Is the HA manager natively Highly Available itself?”** meaning is the tool being used to guarantee HA able to handle and react it's own internal faults in a predictable safe way as well as the system's the tool manages? The answer to this question nearly always a resounding NO. Why? Because true HA is very difficult.

Pacemaker to the Rescue!

- Pacemaker does not replace orchestration.... Instead pacemaker adds resilience to whatever orchestration system is already in use.

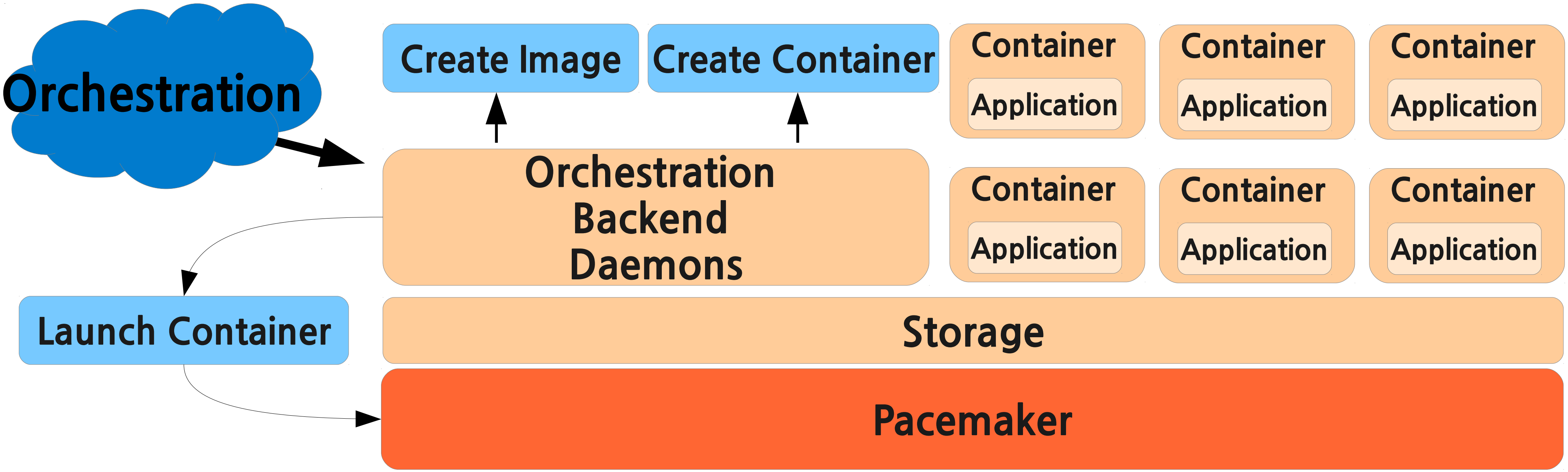
BLUE: Orchestration Actions
ORANGE: Pacemaker Managed Resources



Pacemaker to the Rescue!

- The storage, orchestration daemons, containers, and container applications are all managed as resources within the pacemaker cluster.
- Pacemaker introduces fault tolerance to the existing orchestration system by monitoring and recovering each component in a consistent and predictable way.
- Pacemaker itself is also highly available, which ensures consistent and predictable behavior regardless of where a fault may occur.

BLUE: Orchestration Actions
ORANGE: Pacemaker Managed Resources



Orchestration

Create Image

Create Container

Container
Application

Container
Application

Container
Application

Orchestration
Backend
Daemons

Container
Application

Container
Application

Container
Application

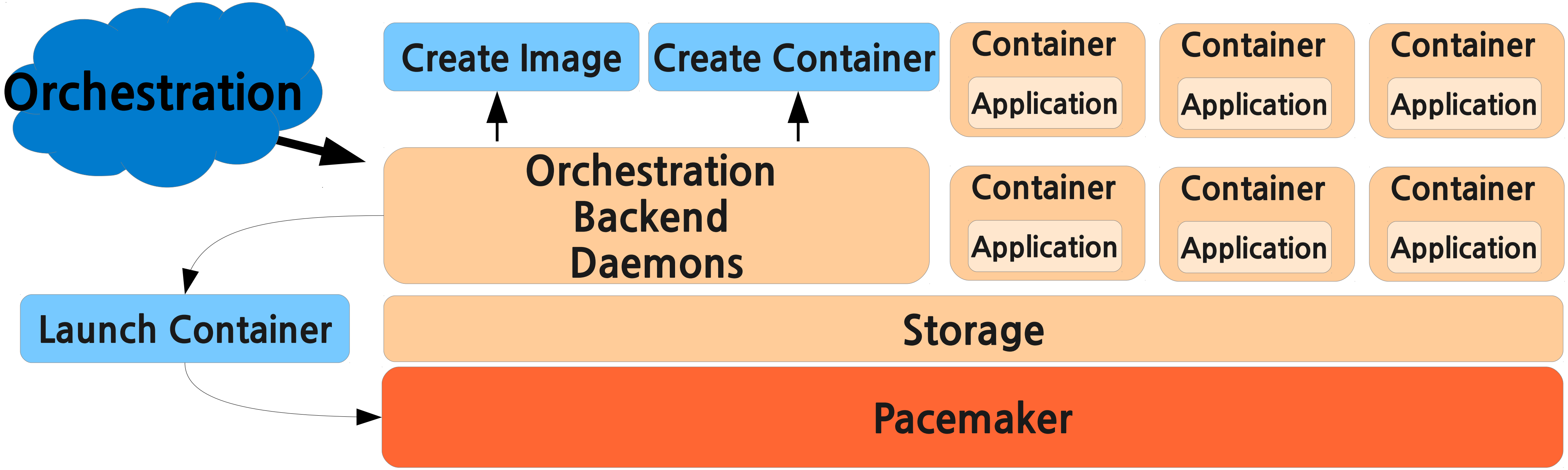
Launch Container

Storage

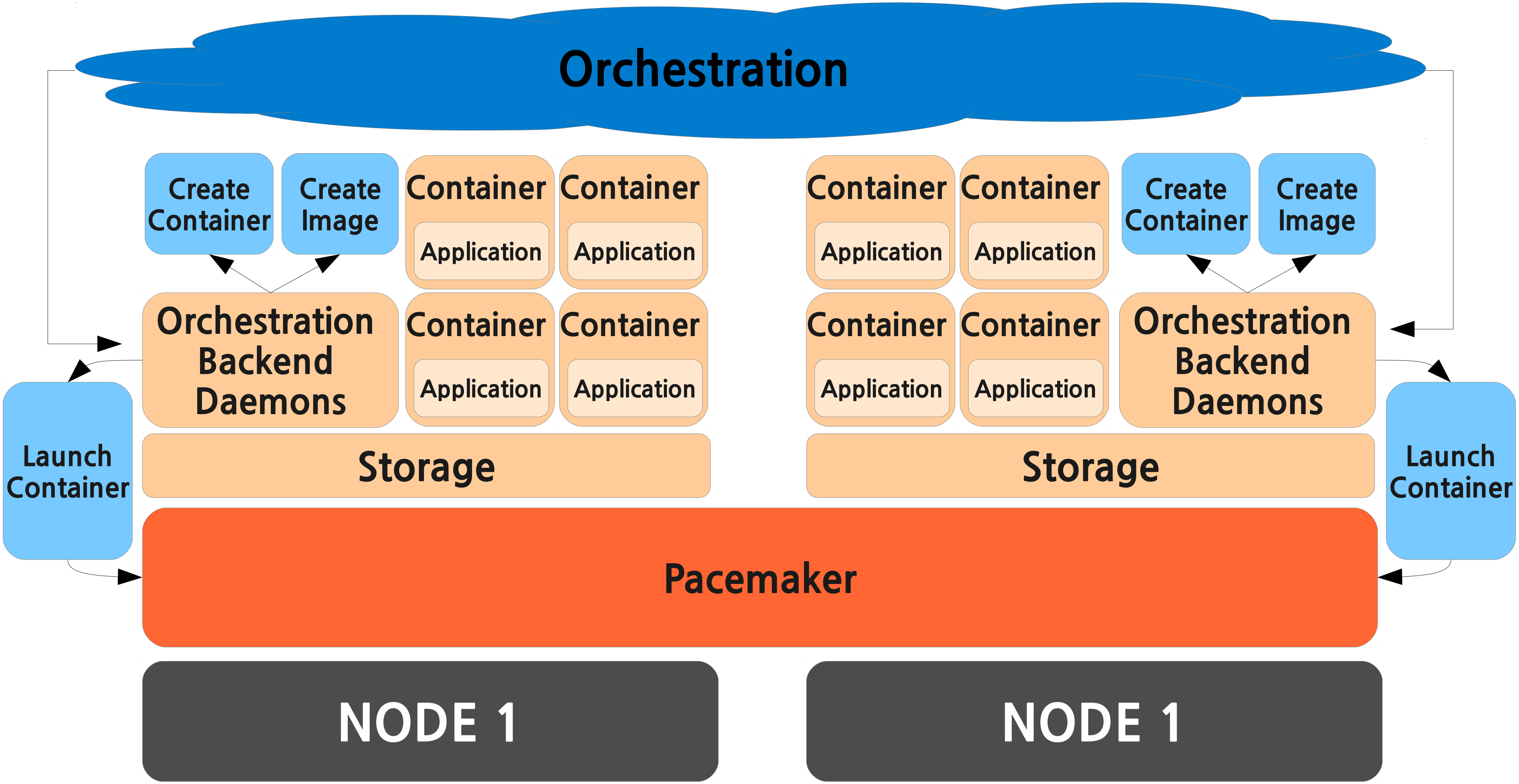
Pacemaker

Orchestration remains the same (almost)

- Orchestration still performs actions using backend API with one slight difference.
- “**launch container**” action is performed through pacemaker rather than by the backend API directly.
  - Pacemaker ensures storage dependencies are available before launch
  - Pacemaker monitors both the container and container's application for health.
  - Pacemaker automatically recovers and relocates failed container resources.
  - Pacemaker maintains control over containers even if API daemons are down.



Big Picture View: With HA



HA container Storage:

- When containers and virtual machines are involved, it is common to have these images available locally via some form of shared storage.
- Pacemaker can manage mounting shared storage through the use of the Filesystem OCF script that ships in the resource-agent's package.
- With the Filesystem resource-agent, pacemaker has the ability to both monitor the mounted filesystem is available as well is perform i/o health checks on the partition itself.

HA orchestration daemons:

- Pacemaker can already natively manage the system resources involved with container management and orchestration.
- The process is to hand over control of the daemons to Pacemaker rather than launching the daemons manually or at system initialization.
- Pacemaker supported resource standards. (Systemd, OCF, LSB, Upstart, Nagios)
- Pacemaker has the ability to understand complex resource ordering and colocation constraints. These resource constraints allow us to define resource dependencies such as (don't start this daemon until this shared storage partition is mounted).
- Resource constraints allow pacemaker to recover resources in a consistent and predictable manner after a failure. With constraints we can guarantee resources will always be stopped and started in the correct order.

HA Container Execution:

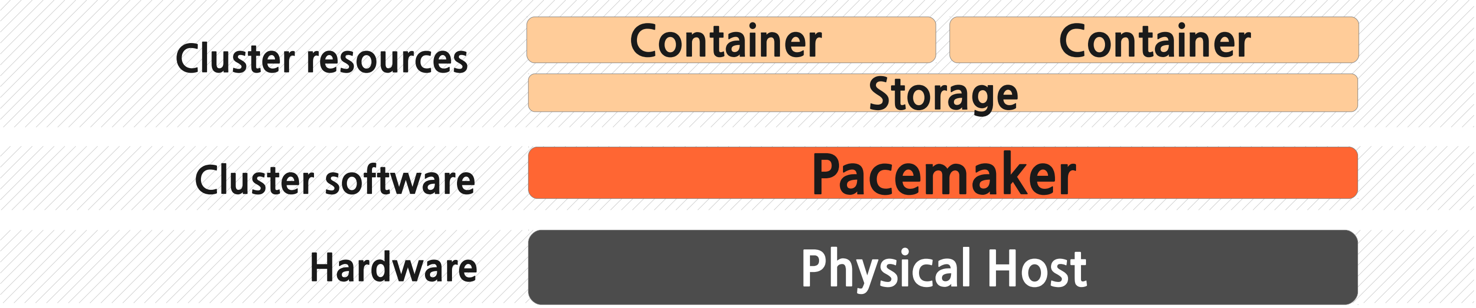
- After the container orchestration system creates a container, the orchestration must hand off the container to Pacemaker to launch and monitor.
- Pacemaker launches containers using custom OCF resource-agents scripts. Take for example a docker container, a 'docker' specific OCF script must be used.
- Unlike systemd unit files and traditional LSB style init scripts, OCF scripts allow parameters to be passed in to them.
- When pacemaker is told to manage resource instances using an OCF script, the resource instance along with the resource's parameters are stored in pacemaker distributed database (the CIB)
- The CIB itself is highly available ensuring that once a container is handed off to pacemaker to manage, that resource instance will remain persistent.

Extending HA to Container Applications

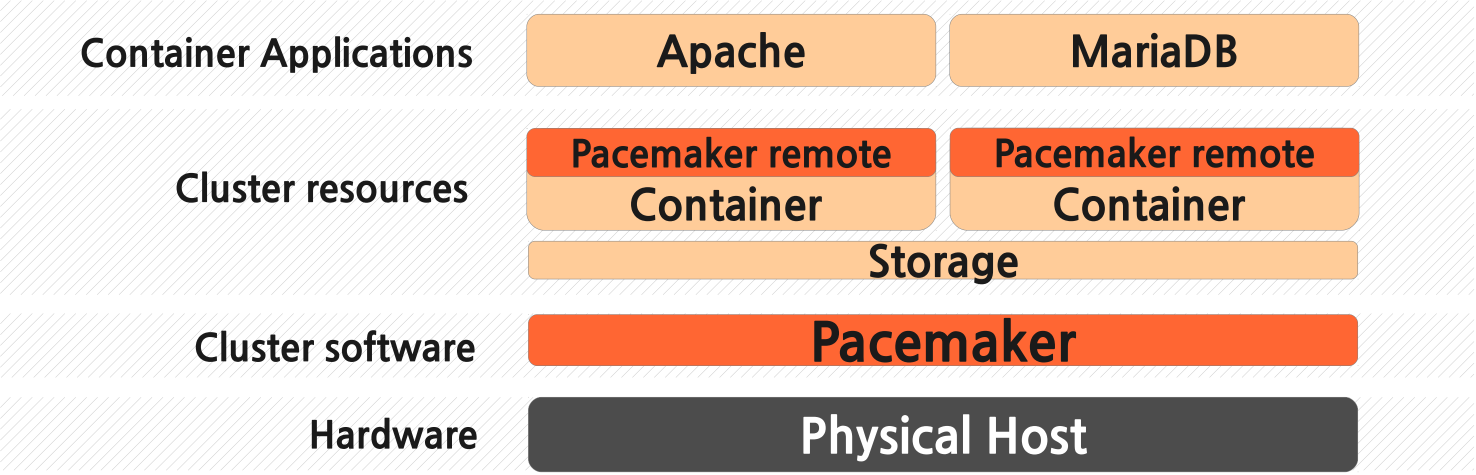
- One advanced feature Pacemaker offers is the ability to both manage executing containers as well as executing and monitoring the applications that run within the containers.
- This ability to manage containers' internal applications is achieved through the use of the **pacemaker_remote** daemon.
  - pacemaker_remote runs in the container's environment and extends pacemaker's resource management from a host node into container.
  - This allows pacemaker to manage resources within a container just as if the container were another HA node available to the system.
- A more in-depth overview of pacemaker_remote can be found here. http://clusterlabs.org/doc/en-US/Pacemaker/1.1-pcs/html-single/Pacemaker_Remote/index.html

Extending HA to Container Applications, cont....

- With pacemaker_remote we go from treating containers as black boxes, where the applications are hidden inside the containers invisible to the cluster manager...

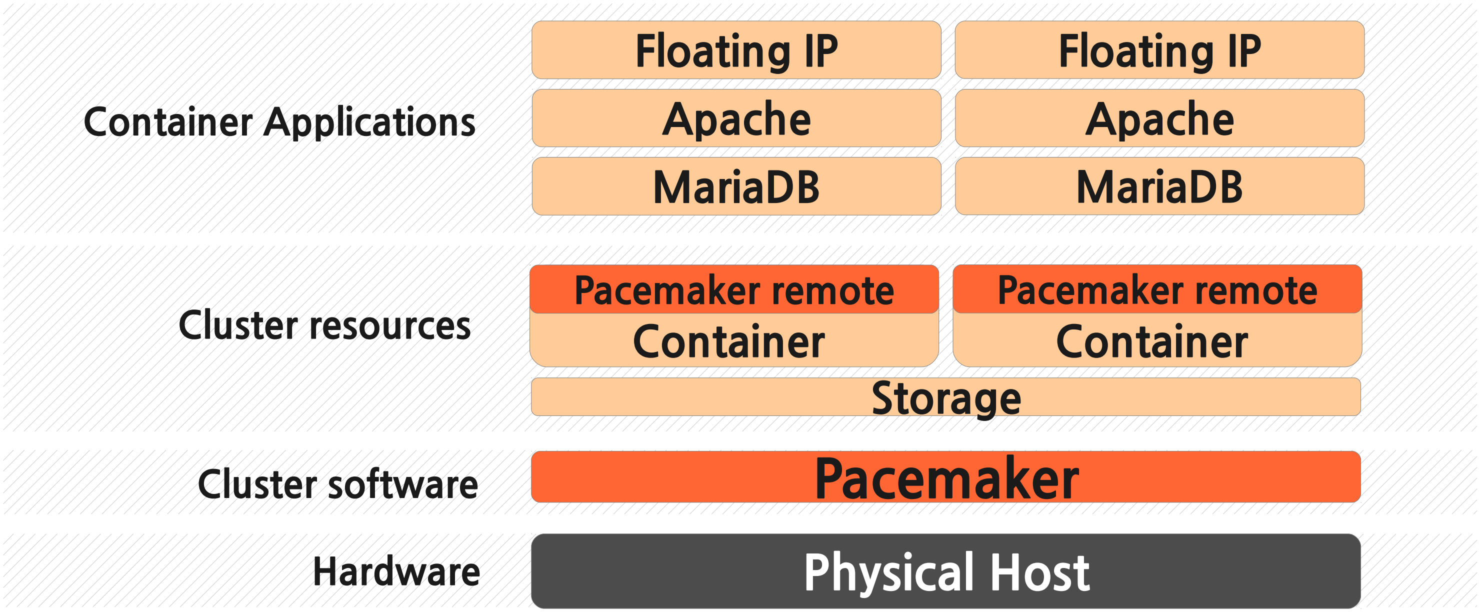


- To an environment where pacemaker is able to monitor both the containers and the applications running within the containers.



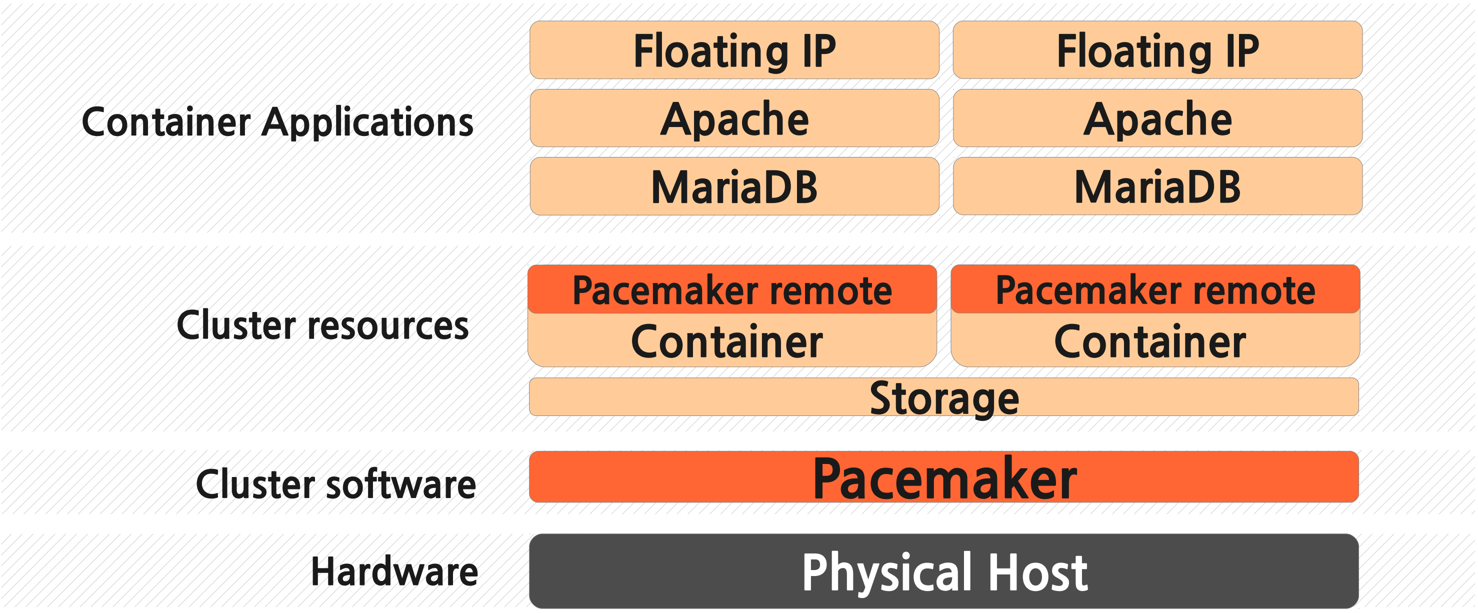
Extending HA to Container Applications, cont....

- By making the pacemaker_remote daemon the entry point for a container, and having pacemaker manage launching container applications through pacemaker_remote, multiple applications can be monitored in a single container as well as enforcing complex resource dependencies within a container.
- The figure below shows a container running pacemaker_remote launching multiple resource instances within each container. Start order and recovery strategies can be expressed for the resource instances running within the container just as if they were running on a physical cluster node.



Extending HA to Container Applications, cont....

- The ability to manage complex resource groups within a container using pacemaker_remote solves many of the problems encountered when trying to expose container to container communication. Resources that depend on one another can live within the same container.



Pacemaker Scaling

1000s of nodes and beyond!

- In the past, pacemaker was limited in the number of nodes it could scale to because of limits imposed by the corosync messaging layer of the stack. These limitations made it difficult for some deployments to extend past 16-20 nodes.
- Great improvements have been made to pacemaker's architecture allowing pacemaker to extend past the previous limitations to 100s and even 1000s of nodes.
- Pacemaker is able to scale beyond the corosync limitations through the use of pacemaker_remote on baremetal nodes.
- With the use of pacemaker_remote baremetal nodes, it is not a stretch to imagine a cluster consisting of 16 cluster-nodes coupled with 1000 pacemaker_remote baremetal nodes which collectively manage 1000s of containers.