# Container Orchestration

CS 4230

Jay Urbain, Ph.D.

## Topics

Microservices

Container Orchestration

Container Orchestration Tools

What is Kubernetes and How it (basically) works

Kubernetes Use Case and Case Studies

Next time: Kubernetes Architecture

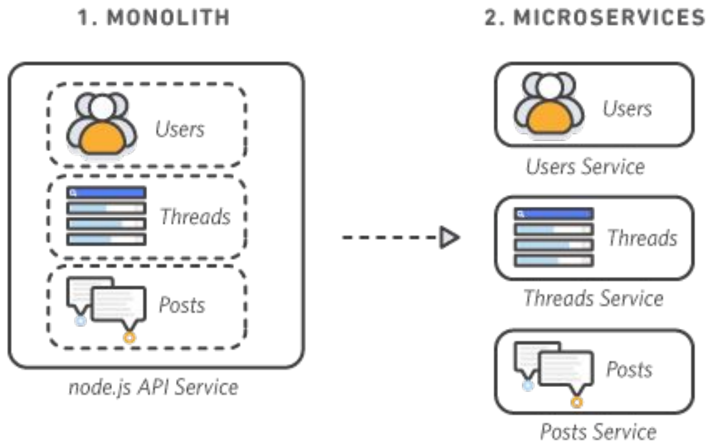
## What are Microservices?

## What are Microservices?

Architectural pattern and organizational approach to software development where software is composed of small independent services that communicate over well-defined APIs.

These services are owned by small, self-contained teams.

**What are the advantages?**

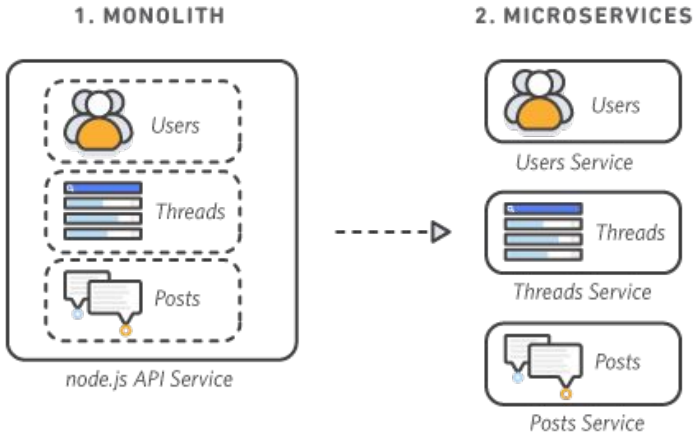


## What are the advantages of Microservices?

Microservices architectures make applications easier to scale and faster to develop, enabling innovation and accelerating time-to-market for new features.

They can be written in different programming languages and can use different technologies and tools, which allows for greater innovation and experimentation in the development process.

**What are the disadvantages?**

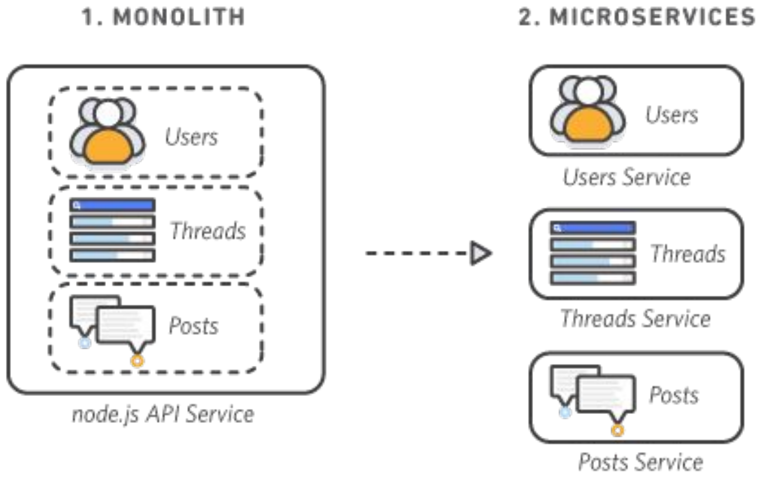


## What are the disadvantages of Microservices?

Complexity.

Implementing microservices can also introduce new challenges, such as the need for more complex infrastructure and increased complexity in monitoring and debugging the overall system.

Increase latency.



## What is a Monolithic Architecture?

## Monolithic vs. Microservices Architecture

Monolithic architectures:

- Processes are tightly coupled and run as a single service.
- If one process of the application experiences a spike in demand, the entire architecture must be scaled.
- Adding to or improving a monolithic application's features becomes more complex as the code base grows.
- This complexity limits experimentation and makes it difficult to implement new ideas.
- Monolithic architectures add risk for application availability because many dependent and tightly coupled processes increase the impact of a single process failure.

## Monolithic vs. Microservices Architecture

Microservices architecture

- An application is built as independent components that run each application process as a service.
- These services communicate via a well-defined interface using lightweight APIs.
- Services are built for business capabilities and each service performs a single function.
- Because they are independently run, each service can be updated, deployed, and scaled to meet demand for specific functions of an application.

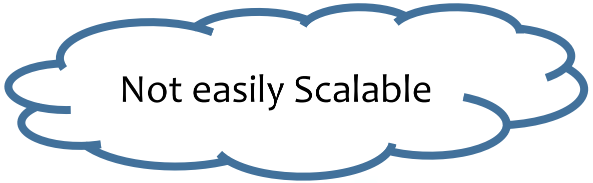
# Containers are Good…

- Both Linux Containers and Docker Containers
    - Isolate the application from the host

FASTER, RELIABLE, EFFICIENT, LIGHTWEIGHT, AND SCALABLE

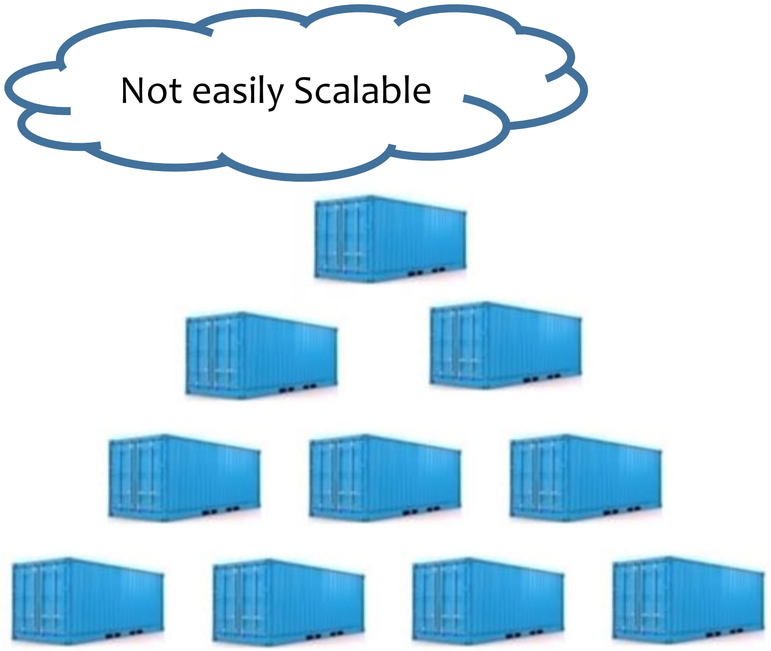
# Containers have Problems!

- Both Linux Containers and Docker Containers
  - Isolate the application from the host

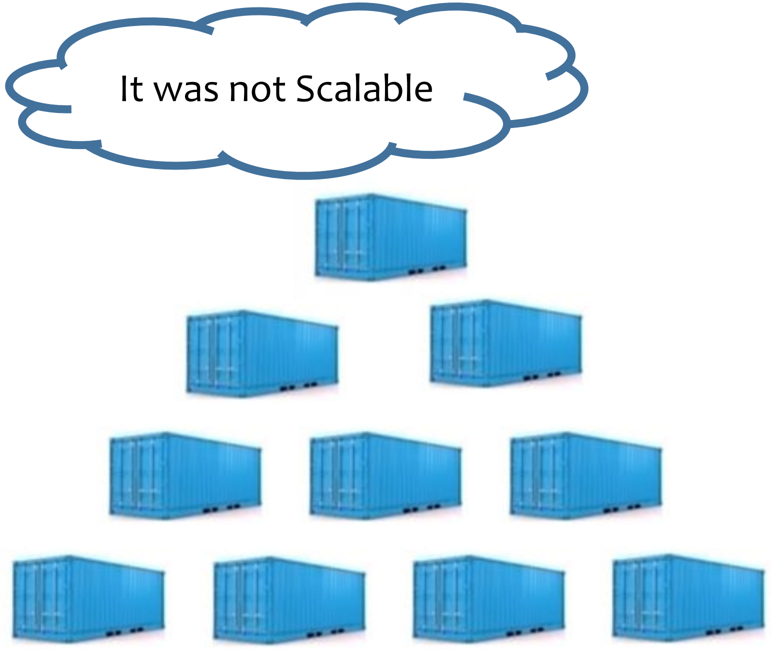


# Container Problems!

- Both Linux Containers and Docker Containers
  - Isolate the application from the host

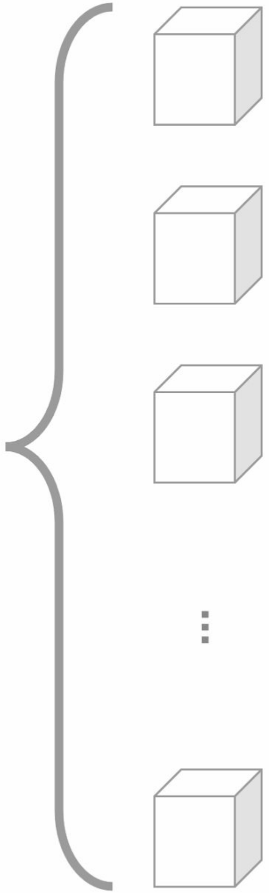


# Problems with Scaling up Containers



1. Containers could communicate with each other
2. Containers had to be deployed appropriately
3. Containers had to be managed carefully
4. Auto scaling was not possible
5. Distributing traffic was still challenging

run
**Services**
in
**Containers**



## *without* container orchestration

scaling up services
→ manual work
**increases**

fixing crashing nodes
→ manual work
**increases**

complexity for running
new stuff in production
**increases**

human cost for
running services
**increases**

scaling becomes
more and more
**difficult**

public cloud providers
bill more and more
**expensive**

## Why do we need container orchestration?

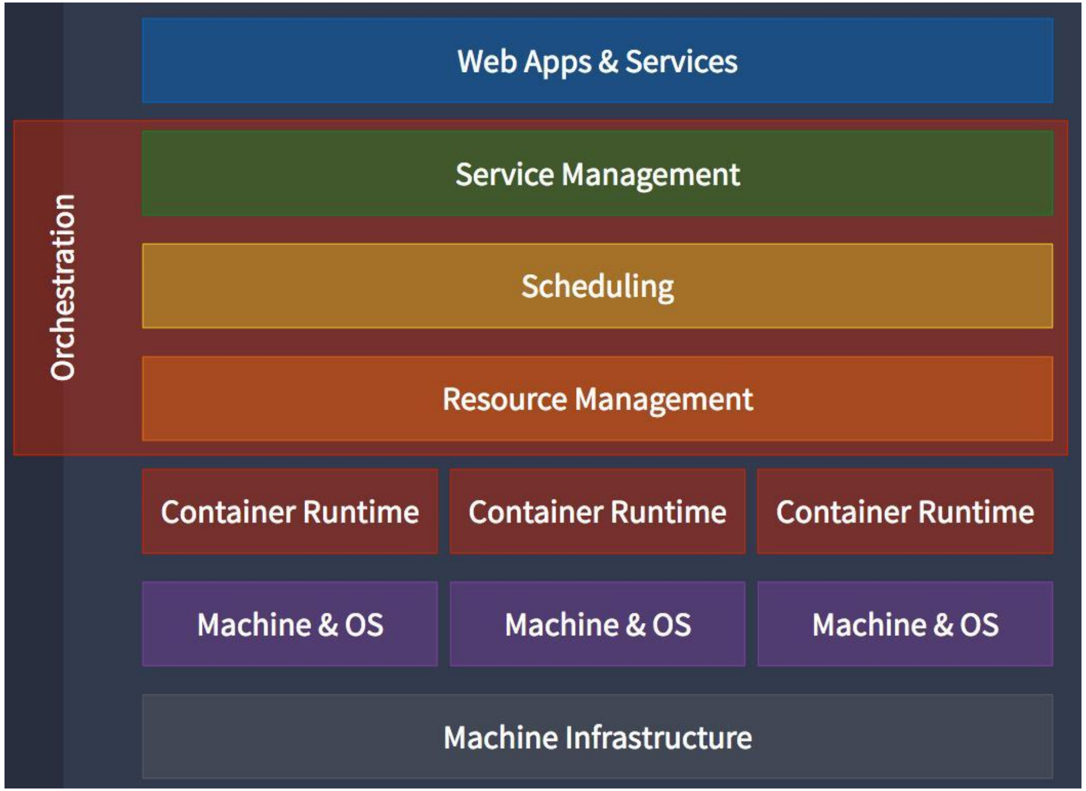
- Because containers are lightweight and ephemeral by nature, running them in production can quickly become a massive effort.
- Particularly when paired with microservices—which typically each run in their own containers—a containerized application might translate into operating hundreds or thousands of containers, especially when building and operating any large-scale system.
- This can introduce significant complexity if managed manually.
- Container orchestration is what makes that operational complexity manageable for development and operations—or DevOps—because it provides a declarative way of automating much of the work.

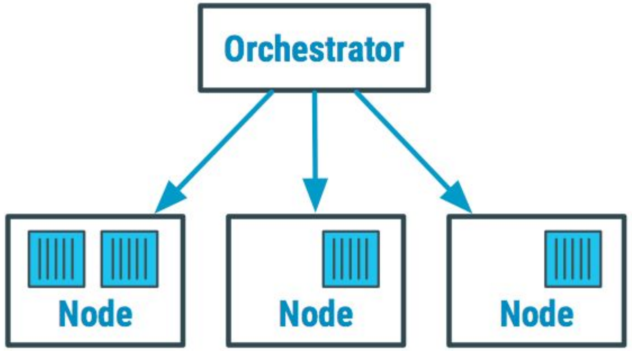
## Container Orchestration

## What is container orchestration?

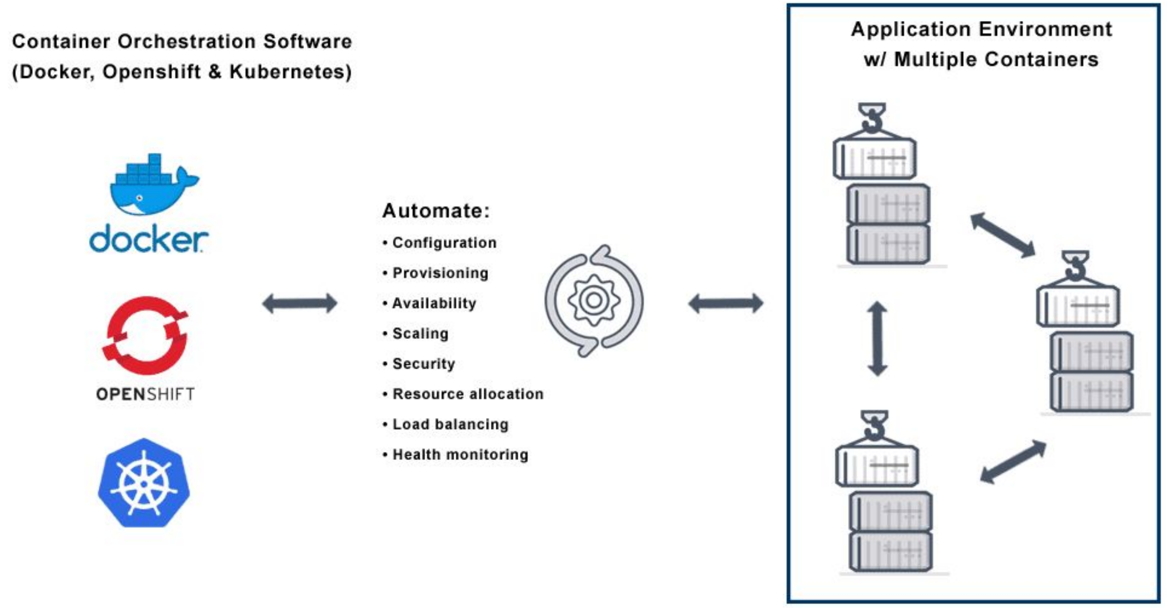
- Container orchestration is the automation of much of the operational effort required to run containerized workloads and services.
- This includes a wide range of things software teams need to manage a container’s lifecycle, including:
    - provisioning,
    - deployment,
    - scaling (up and down),
    - networking,
    - load balancing and more.

## Container orchestration automates and simplifies provisioning, and deployment and management of containerized applications.





- Container orchestration is the automatic process of managing or scheduling the work of individual containers for applications based on microservices within multiple clusters.
- The widely deployed container orchestration platforms are based on open-source versions like Kubernetes, Docker Swarm or the commercial version from Red Hat OpenShift.



## Container Orchestration

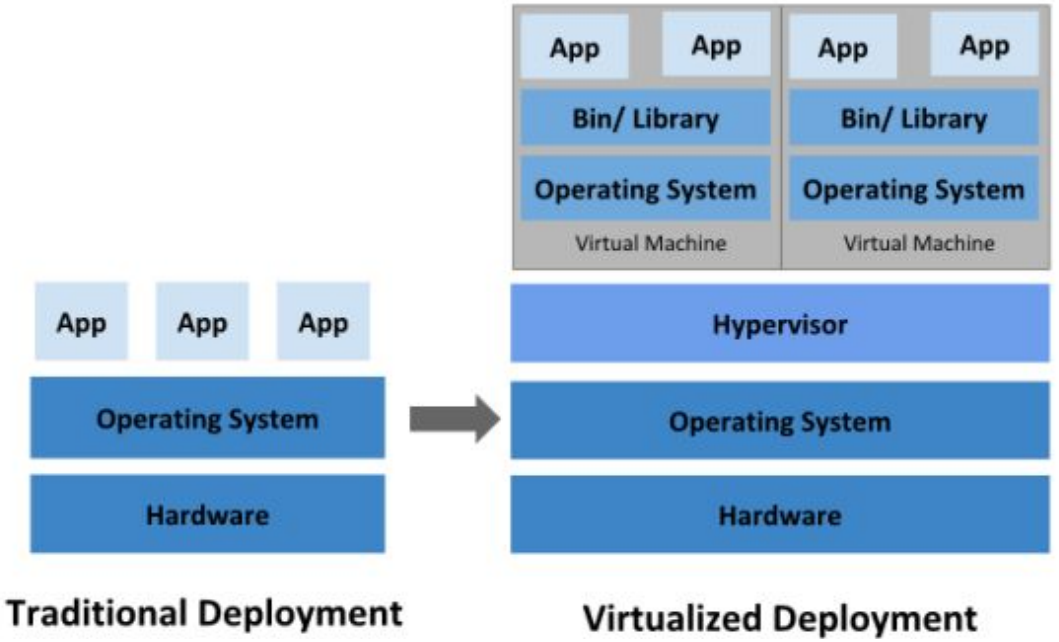
- Fault-tolerance
- On-demand scalability
- Optimal resource usage
- Auto-discovery to automatically discover and communicate with each other
- Accessibility from the outside world
- Seamless updates/rollbacks without any downtime.

## Why Do We Need Container Orchestration?

- Container orchestration is used to automate the following tasks at scale:
  - Configuring and scheduling of containers
  - Provisioning and deployments of containers
  - Availability of containers
  - The configuration of applications in terms of the containers that they run in
  - Scaling of containers to equally balance application workloads across infrastructure
  - Allocation of resources between containers
  - Load balancing, traffic routing and service discovery of containers
  - Health monitoring of containers
  - Securing the interactions between containers.

# Cloud Orchestration

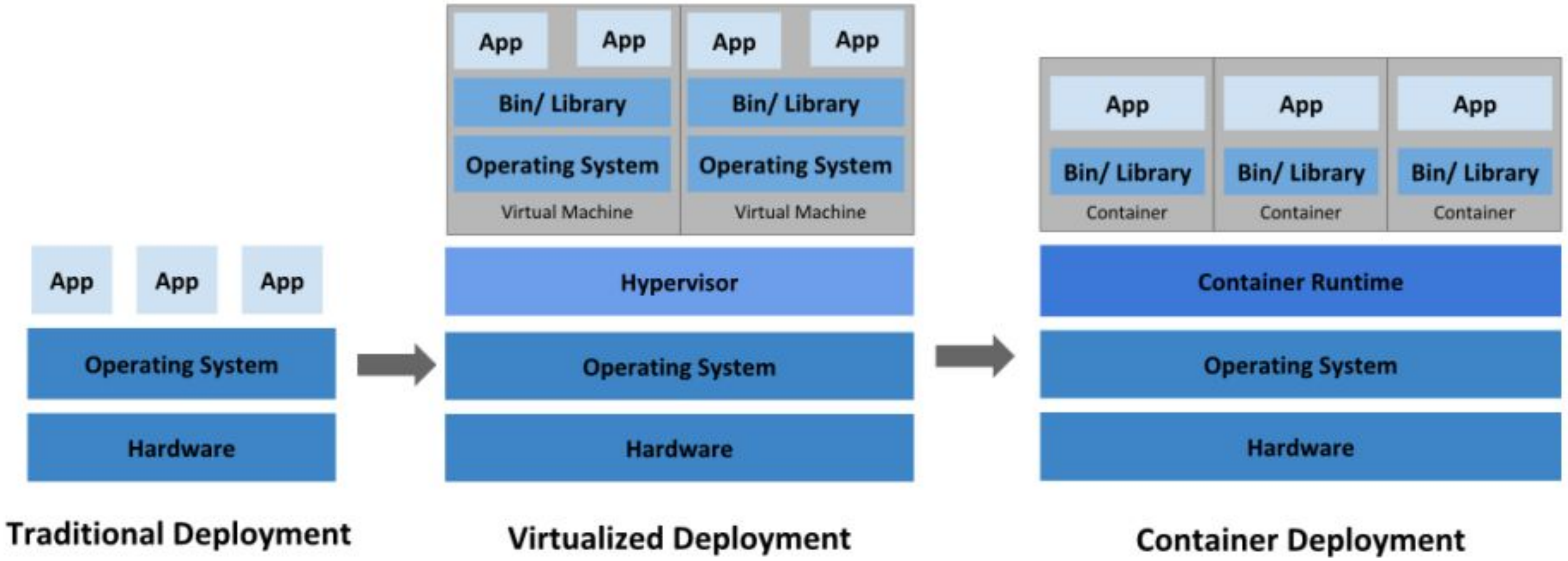
- Infrastructure-as-a-Service (IaaS)
  - Provisioning virtual machines from a cloud service provider (CSP)



aws Azure Google Cloud openstack.

# Container Orchestration

- Application containers
  - Lightweight OS-virtualization
  - Application packaging for portable, reusable software



## Container Orchestration

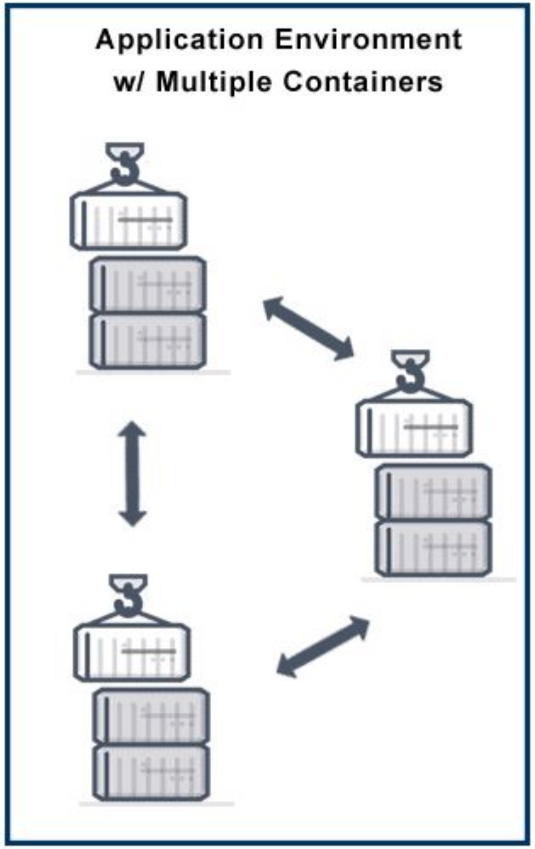
Container Orchestration Software
(Docker, Openshift & Kubernetes)

docker

OPENSHIFT

**Automate:**

- Configuration
- Provisioning
- Availability
- Scaling
- Security
- Resource allocation
- Load balancing
- Health monitoring



## What are the benefits of container orchestration?

- Container orchestration is key to working with containers, and it allows organizations to unlock their full benefits. It also offers its own benefits for a containerized environment, including:
- Simplified operations
  - Most important benefit of container orchestration and the main reason for its adoption.
  - Manages the complexity of Containers
- Resilience
  - Automatically restart or scale a container or cluster, boosting resilience.
- Added security
  - Keeping containerized applications secure by reducing or eliminating the chance of human error.

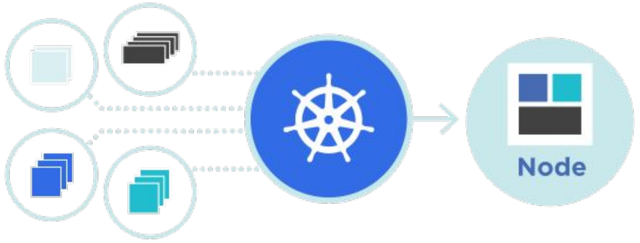
## What is Kubernetes

- Benefits (Works with all cloud vendors (Public, Private (on-premises), and Hybrid))

Kubernetes is an open-source container management tool which automates container deployment, container (de)scaling and container load balancing

**More about Kubernetes**

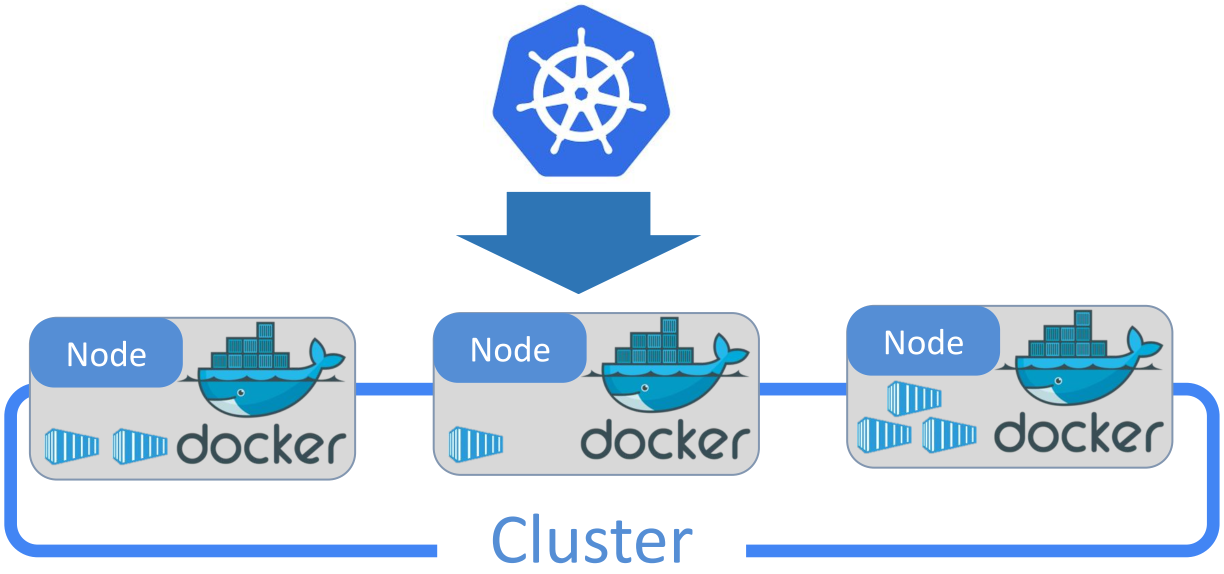
- Developed by Google and written in Golang with a huge community
- Can group 'n' containers into one logical unit for managing and deploying them

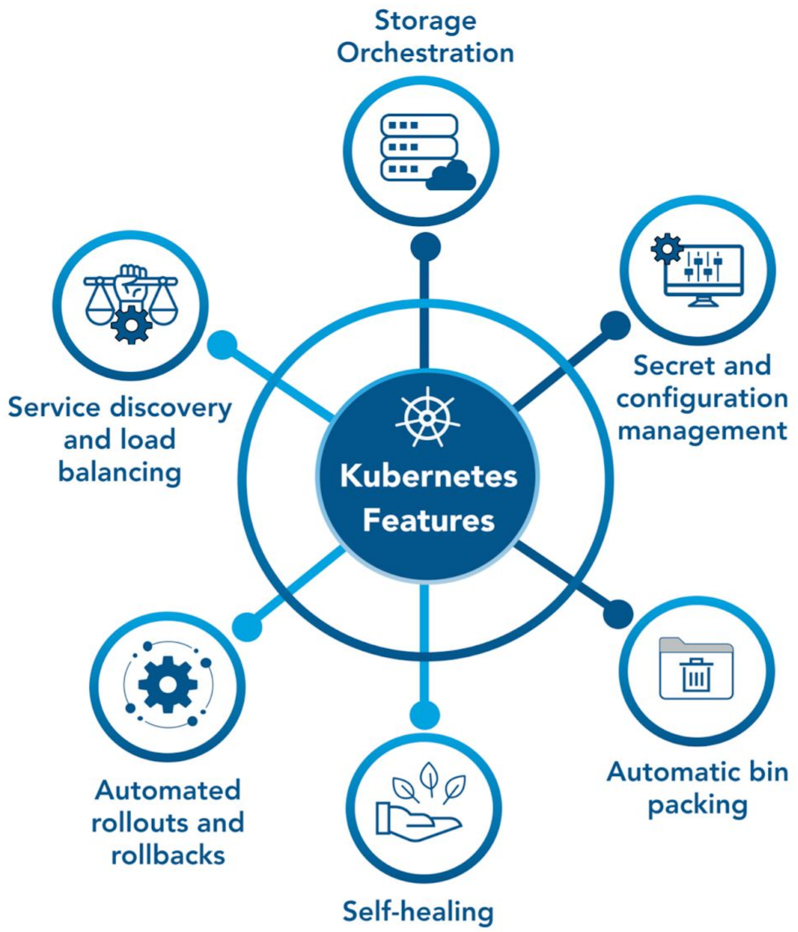


## Kubernetes

- Kubernetes is an open source container orchestration engine for automating deployment, scaling, and management of containerized application.
- Originally an open source project launched by Google and now part of the Cloud Native Computing Foundation (CNCF).
- Kubernetes is highly **extensible** and **portable**
- Kubernetes is considered **highly declarative**
- Kubernetes initial release (**7 June 2014)**
- Releases every 3 months

kubernetes


Node
docker
Node
docker
Node
docker
Cluster

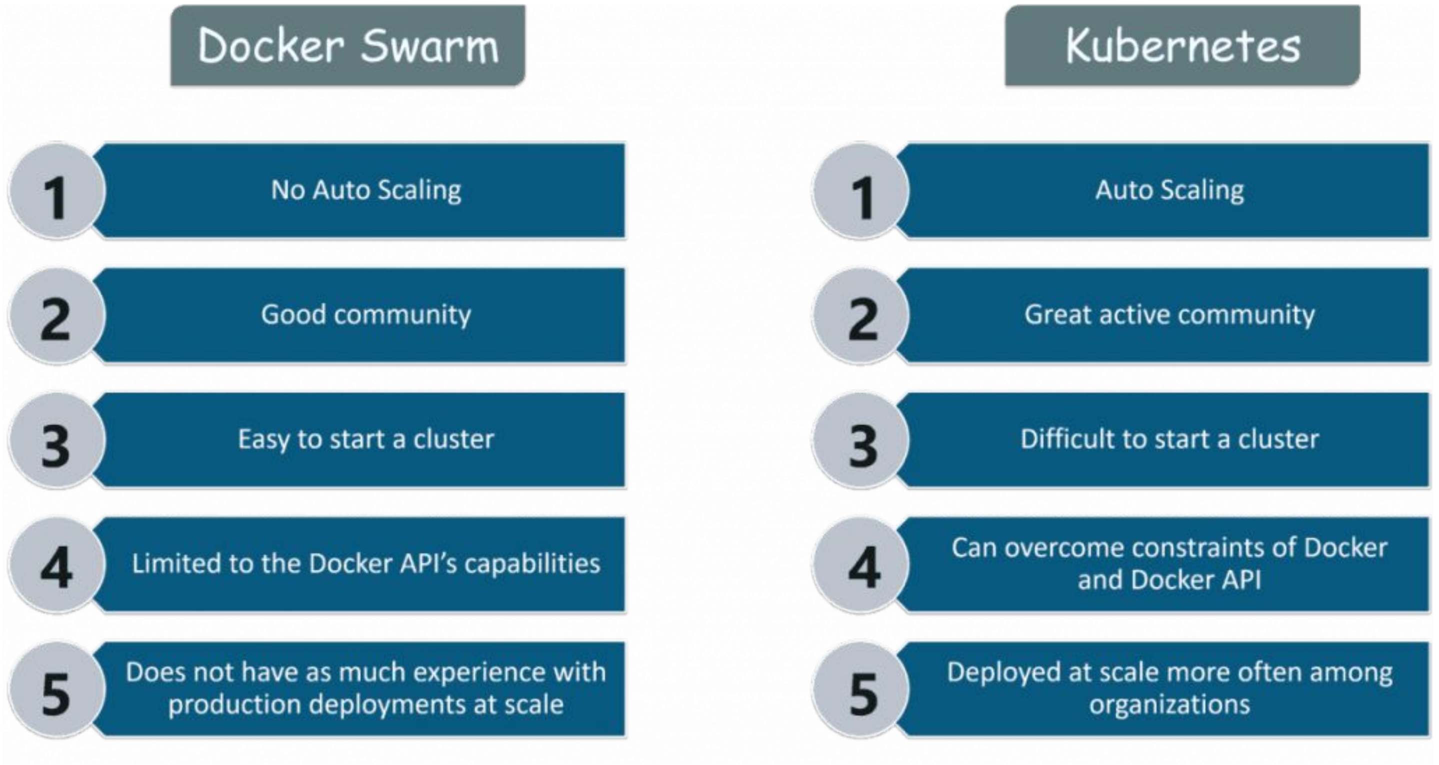

Storage Orchestration
Secret and configuration management
Automatic bin packing
Self-healing
Automated rollouts and rollbacks
Service discovery and load balancing
Kubernetes Features

## Kubernetes Myths

- Kubernetes is not:
  - To be compared with Docker
  - For containerizing apps
  - For apps with simple architecture

- Kubernetes is actually:
  - Robust and reliable
  - A container orchestration platform
  - A solution for scaling up Containers
  - Backed by huge community

# Kubernetes vs Docker Swarm



| | Docker Swarm | | Kubernetes |
|---|---|---|---|
| 1 | No Auto Scaling | 1 | Auto Scaling |
| 2 | Good community | 2 | Great active community |
| 3 | Easy to start a cluster | 3 | Difficult to start a cluster |
| 4 | Limited to the Docker API's capabilities | 4 | Can overcome constraints of Docker and Docker API |
| 5 | Does not have as much experience with production deployments at scale | 5 | Deployed at scale more often among organizations |

# Kubernetes vs Docker Swarm

| Features | Docker Swarm | Kubernetes |
|---|---|---|
| Installation and Cluster Configuration | Easy and Fast | Complicated and Time-consuming |
| GUI | Not Available | Available |
| Scalability | Scaling up is faster than K8S; but cluster strength is not as robust | Scaling up is slower but guarantees stronger cluster state |
| Load Balancing | Built-in load balancing technique | Requires manual service configuration |
| Updates and Rollback | Progressive updates and service health monitoring | Process scheduling to maintain services while updating |
| Data Volumes | Can be shared with any container | Only shared with containers in the same Pod |
| Logging and Monitoring | Only 3rd party logging and monitoring tools | Built-in logging and monitoring tools |

## Docker Compose vs Kubernetes

- Docker Compose and Kubernetes are both tools used for container orchestration, but they have some fundamental differences.
- Docker Compose deploys multi-container Docker apps to a single server.
- Docker Compose is a tool for defining and running multi-container Docker applications. It allows developers to define the services that make up an application in a YAML file and then start and stop those services with a single command.
- Kubernetes is a production-grade container orchestrator that can run multiple container runtimes, including Docker's, across multiple virtual or physical machines.

## Kubernetes Use Case and Case Studies

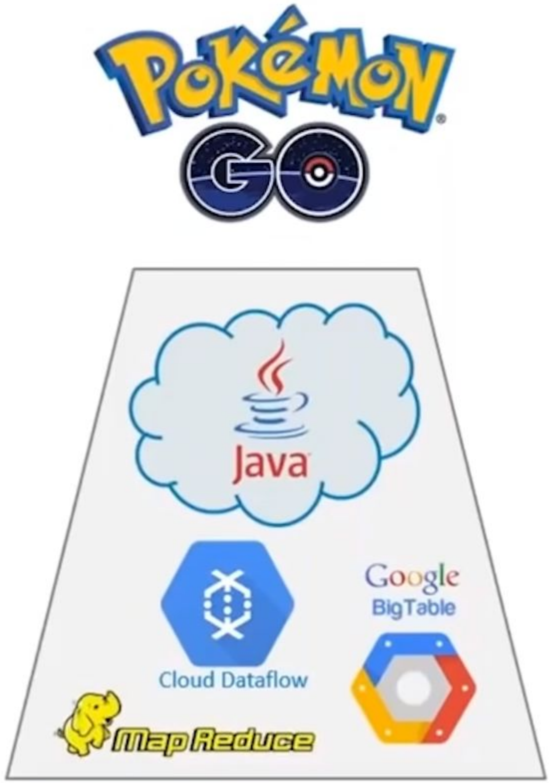
## Kubernetes Use Case

**Pokemon Go** is an augmented reality game developed by **Niantic** for Android and iOS devices

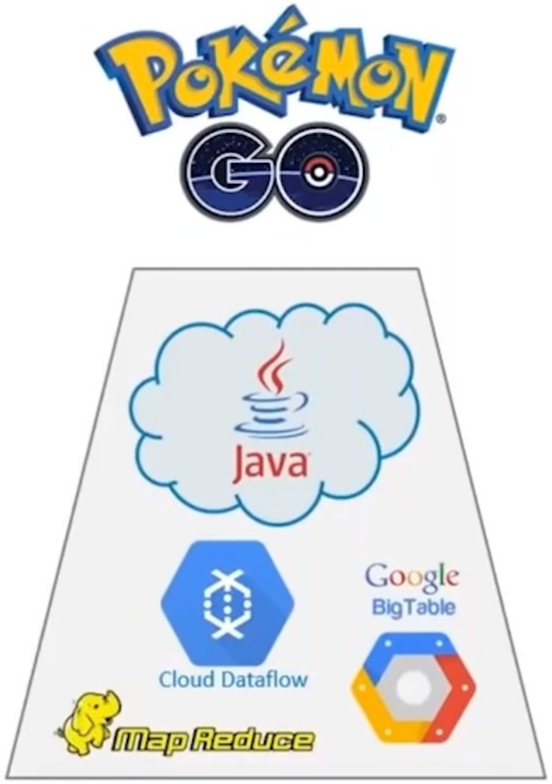
**Key Stats:**

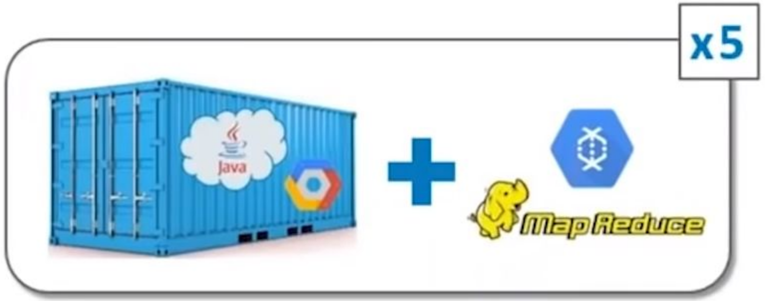
- 500+ million downloads, 20+ million daily active users
- Initially only launched in NA, Australia and New Zealand
- Inspired users to walk over 5.4 billion miles in a year
- Surpassed engineering expectations by 50 times

## Backend Architecture

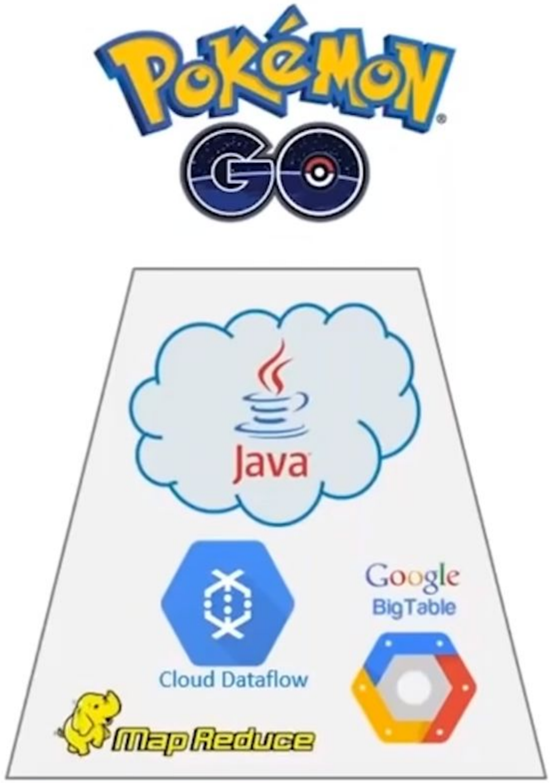


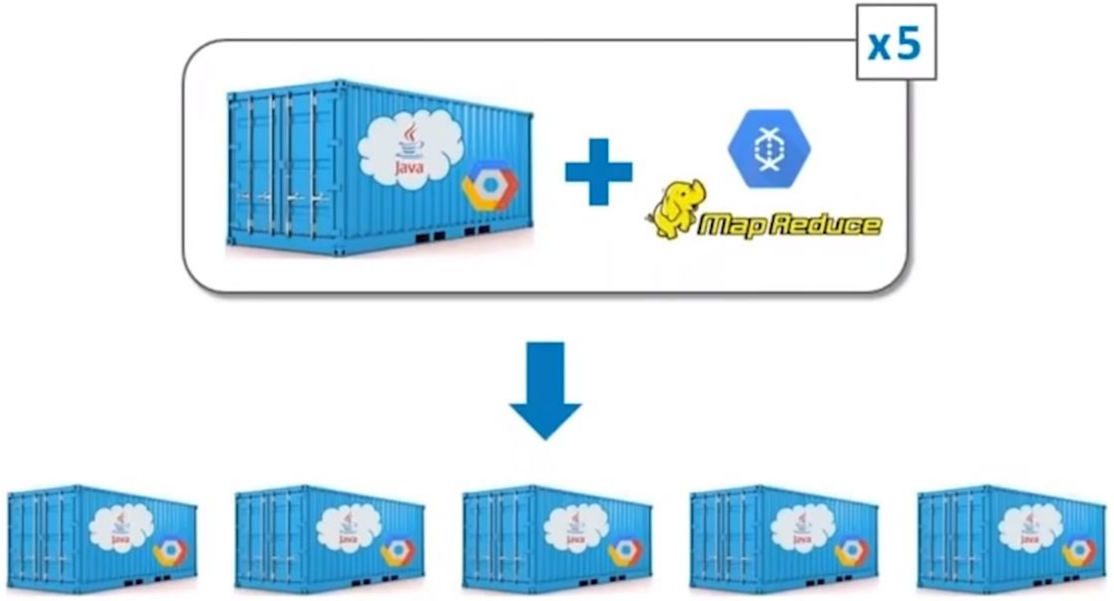
# MapReduce & Cloud Dataflow for scaling up





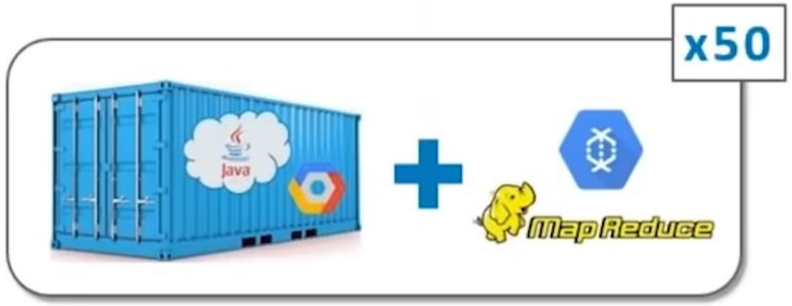
# Easy Scaling of Containers using Kubernetes





# Easy Scaling of Containers using Kubernetes

Challenge



- Biggest challenge from most applications is horizontal scaling
- For Pokemon Go, vertical scaling was an also a major challenge, because of real-time activity in gaming environment from millions of users
- Niantic were prepared for traffic disasters of up to x5 times

Solution

- Thanks to Kubernetes, Niantic were able to handle x50 times traffic

# Case Study: Booking.com

- Challenge
  - In 2016, Booking.com migrated to an OpenShift platform
    - which gave product developers faster access to infrastructure.
  - Kubernetes was abstracted away from the developers, the infrastructure team became a "knowledge bottleneck" when challenges arose.
    - Trying to scale that support wasn't sustainable.

## Case Study: Booking.com

- Solution
  - After a year operating OpenShift, the platform team decided to build its own vanilla Kubernetes platform—and ask developers to learn some Kubernetes in order to use it.
  - "This is not a magical platform," says Ben Tyler, Principal Developer, B Platform Track.
  - "We're not claiming that you can just use it with your eyes closed. Developers need to do some learning, and we're going to do everything we can to make sure they have access to that knowledge."

## Case Study: Booking.com

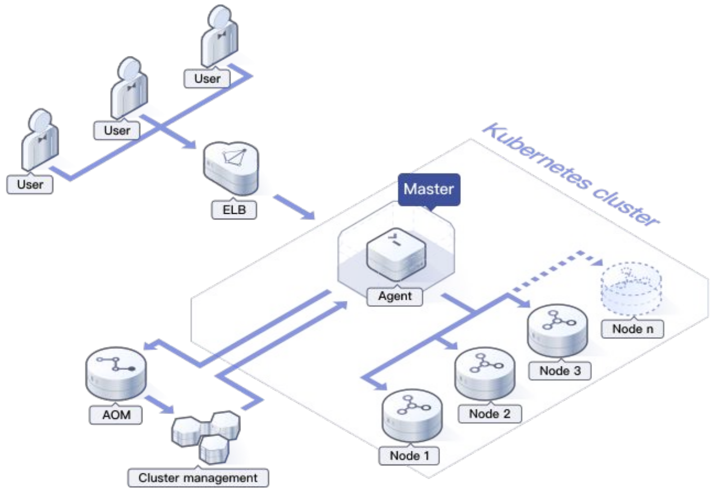
- Impact
  - Despite the learning curve, there's been a great uptick in adoption of the new Kubernetes platform.
  - Before containers, creating a new service could take a couple of days if the developers understood Puppet, or weeks if they didn't.
  - On the new platform, it can take as few as 10 minutes. About 500 new services were built on the platform in the first 8 months.

## Case Study: Huawei

- Challenge
  - In order to support its fast business development around the globe
    - Huawei has eight data centers for its internal I.T. department, which have been running 800+ applications in 100K+ VMs to serve these 180,000 users.
  - With the rapid increase of new applications
    - The cost and efficiency of management and deployment of VM-based apps all became critical challenges for business agility.

## Case Study: Huawei

- Solution
  - After deciding to use container technology
    - Huawei began moving the internal I.T. department's applications to run on Kubernetes.
    - So far, about 30 percent of these applications have been transferred to cloud native.



## Case Study: Huawei

- Impact
  - By the end of 2016, Huawei's internal I.T. department managed more than 4,000 nodes with tens of thousands containers using a Kubernetes-based Platform as a Service (PaaS) solution
  - The global deployment cycles decreased from a week to minutes
  - The efficiency of application delivery has been improved 10 fold
  - They saw significant operating expense spending cut, in some circumstances 20-30 percent