Kubernetes is an open source system for managing containerized applications across multiple hosts, providing basic mechanisms for deployment, maintenance, and scaling of applications.

Kubernetes is:
- **lean**: lightweight, simple, accessible
- **portable**: public, private, hybrid, multi cloud
- **extensible**: modular, pluggable, hookable, composable
- **self-healing**: auto-placement, auto-restart, auto-replication

"Kubernetes" is the Greek word for a ship's captain.
Large-scale cluster management at Google with Borg
Abhishek Verma†  Luis Pedrosa†  Madhukar Korupolu
David Oppenheimer  Eric Tune  John Wilkes
Google Inc.

Abstract

Google’s Borg system is a cluster manager that runs hundreds of thousands of jobs, from many thousands of different applications, across a number of clusters each with up to tens of thousands of machines.

It achieves high utilization by combining admission control, efficient task-packing, over-commitment, and machine sharing with process-level performance isolation. It supports high-availability applications with runtime features that minimize fault-recovery time, and scheduling policies that reduce the probability of correlated failures. Borg simplifies life for its users by offering a declarative job specification language, name service integration, real-time job monitoring, and tools to analyze and simulate system behavior.

We present a summary of the Borg system architecture and features, important design decisions, a quantitative analysis of some of its policy decisions, and a qualitative examination of lessons learned from a decade of operational experience with it.

Figure 1: The high-level architecture of Borg. Only a tiny fraction of the thousands of worker nodes are shown.

including with a set of qualitative observations we have made
Containers

A container image is a lightweight, stand-alone, executable package of a piece of software that includes everything needed to run it: code, runtime, system tools, system libraries, settings. Containerized software will always run the same, regardless of the environment. Containers isolate software from its surroundings and help reduce conflicts between teams running different software on the same infrastructure.
Containers can enable your company to pack a lot more applications into a single physical server than a virtual machine (VM) can. ... VMs take up a lot of system resources. Each VM runs not just a full copy of an operating system, but a virtual copy of all the hardware that the operating system needs to run.”
Container Cluster Orchestration

**docker**
- Package & run your app as containers
- Find existing container images from others
- Deploy your container on your laptop, server, or cloud

**Kubernetes**
- Container Cluster Orchestration Engine
- Declarative management hides complexity
- Open Source, Runs Anywhere

**Container Engine**
- Cluster-Oriented Container Service
- Full Google Cloud Platform Infrastructure
- Powered by Kubernetes
...back to Kubernetes
Primary concepts

**Container**: A sealed application package (Docker)

**Pod**: A small group of tightly coupled Containers
  example: content syncer & web server

**Controller**: A loop that drives current state towards desired state
  example: replication controller

**Service**: A set of running pods that work together
  example: load-balanced backends

**Labels**: Identifying metadata attached to other objects
  example: phase=canary vs. phase=prod

**Selector**: A query against labels, producing a set result
  example: all pods where label phase == prod
Kubernetes Architecture
An ocean of user containers

Scheduled and packed dynamically onto nodes
More than just packing and isolation

**Scheduling**: Where should my containers run?

**Lifecycle and health**: Keep my containers running despite failures

**Discovery**: Where are my containers now?

**Monitoring**: What’s happening with my containers?

**Auth**\{n,z\}: Control who can do things to my containers

**Aggregates**: Compose sets of containers into jobs

**Scaling**: Making jobs bigger or smaller
apiVersion: v1
kind: ReplicationController
metadata:
  name: backend-contr
  # these labels can be applied automatically
  # from the labels in the pod template if not set
labels:
  app: graphviz
  tier: backend
spec:
  replicas: 2
  # Pod Template
  template:
    metadata:
      labels:
        app: graphviz
        tier: backend
    spec:
      containers:
        - name: server
          # Uncomment this line to use the public docker hub image
          image: omerio/graphviz-server
          # my own private Container Registry image
          # image: gcr.io/gdg-apps-1090/graphviz-server
          # Arguments to the image entrypoint, more details here:
          # http://kubernetes.io/v1.1/docs/api-reference/v1/definitions.html#_v1_container
          args: ["8080"]
          resources:
            # The resources specification for each Pod, more details here:
            # http://kubernetes.io/v1.1/docs/design/resources.html#resource-specifications
            requests:
              cpu: 100m
              memory: 100Mi
          ports:
            # More details on the container ports here,
            # http://kubernetes.io/v1.1/docs/api-reference/v1/definitions.html#_v1_containerport
            # The port exposed by the container
            - containerPort: 8080

kubectl create -f backend-controller.yaml
kubectl create -f backend-service.yaml