

263-3855-00: Cloud Computing Architecture

Section 03: From Containers to Clusters

Swiss Federal Institute of Technology Zurich (ETH Zürich)

Last Edit Date: 05/31/2025

Disclaimer and Term of Use:

1. We do not guarantee the accuracy and completeness of the summary content. Some of the course material may not be included, and some of the content in the summary may not be correct. You should use this file properly and legally. We are not responsible for any results from using this file.
2. This personal note is adapted (derived) from the slides and codes from *Professor Ana Klimovic and people from ETH Zurich Systems Group*. Please [contact us](#) to delete this file if you think your rights have been violated.
3. This work is licensed under a [Creative Commons Attribution 4.0 International License](#).

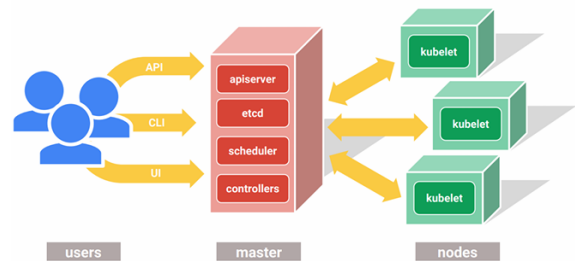
I. Cluster manager system architecture & API design

1. Cluster Manager Designs

- Centralized
 - A single manager responsible for the entire cluster
 - Examples: Kubernetes, Borg, Quasar
- Two-level
 - One global, central controller and multiple framework schedulers
 - Examples: Mesos, YARN
- Distributed
 - Multiple concurrently operating scheduling agents
 - Examples: Omega, Sparrow

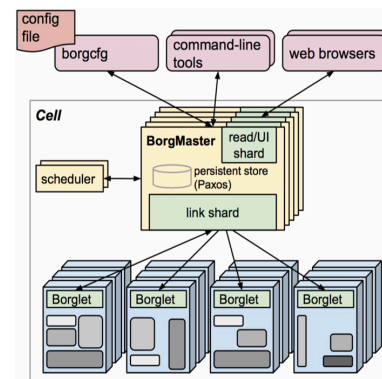
2. Kubernetes - container orchestration system

- Cluster management for containerized applications
 - Manages the container lifecycle
 - Schedule containers on (shared) hardware resources
 - Like an OS for your cloud cluster
- Kubernetes philosophy: user manage applications, not machines
- Kubernetes configurations are typically specified in YAML files, which is a human-readable data serialization language.
- Inspired and informed by Google Borg cluster manager



Google Borg Cluster Manager

- Cluster manager at Google that does
 - Efficient task-packing
 - Admission control
 - Machine sharing
 - Resource over-commitment
- Borgmaster
 - Main borgmaster process and scheduler
 - Five replicas
- Borglet



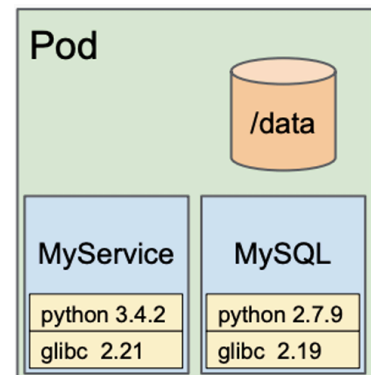
- Manage and monitor tasks and resources
- Borgmaster polls Borglet every few seconds

Container orchestration

- **Resource scaling:** make sets of container bigger or smaller
- **Resource allocation:** decide where my containers should run
- **Load balancing:** distribute traffic across a set of containers
- **Lifecycle and health:** keep my containers running despite failures
- **Naming and discovery:** find where my containers are now
- **Storage volumes:** provide data to containers
- **Logging and monitoring:** track what is happening with my containers
- **Debugging and introspection:** enter or attach to containers
- **Identity and authorization:** control who can do things to my containers

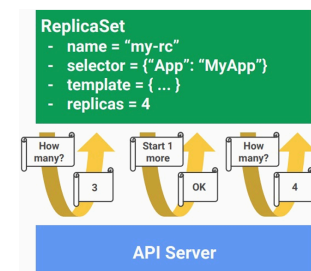
Kubernetes abstraction - Pods

- A pod is the unit of scheduling and migrate in Kubernetes
- Containers: handle package dependencies
- Pods: a bunch of containers with the same
 - **Lifecycle:** live together, die together
 - **Network:** same IP address, same routes, iptables
 - **Storage volumes:** can share data
 - Intended to run a common task



Kubernetes - keeping N pods running

- ReplicaSets abstractions: a layer on top of Pod API that ensures N copies of a pod are running
- Simple control loop:
 - Too few -> start some pods
 - Too many -> kill some pods



Kubernetes - controlling pod placement

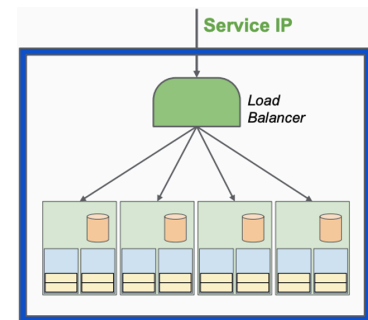
Labels and selectors

- Add a "label" (i.e., arbitrary metadata) to a Kubernetes cluster node
- Query / select for particular label values when specifying how containers should be deployed on nodes



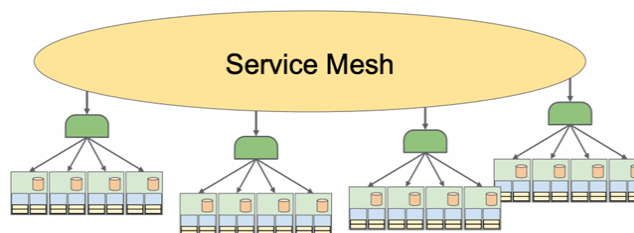
Kubernetes abstraction - services

- A group of pods that work together
- Load balancing among pod replicas
- Can have a stable virtual IP address and port (also a DNS name)
 - A virtual IP is managed by kube-proxy, a network proxy that runs on each node in the cluster, watching all services and updating iptables when backends change



Service mesh

- Designed to make it easier to connect, manage, and secure traffic between services running in containers
- Service mesh consists of
 - Data plane: proxies deployed as “sidecars” within each container on Kubernetes pods, which establish connections to other services
 - Control plane, configure how communication is managed, sophisticated load balancing, security policies, etc.
- Key advantage: change policies without changing application code



Quality of Service (QoS)

- Need to guarantee a certain performance to the application.
- Example of QoS objective, often called service level objective (SLO), support 10 M queries per second with 95th percentile latency under 10 ms
- QoS requirements drive resource provisioning decisions
- Need resource isolation
 - Application must not be able to affect each other’s performance

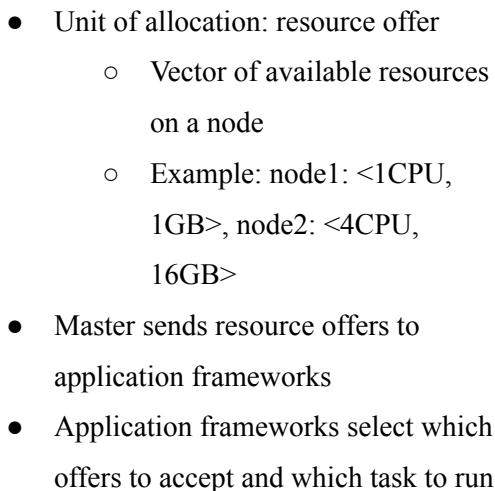
Kubernetes - resource allocation mechanisms

- Request
 - How much of a resource (CPU, RAM) the container is asking to use, with a strong guarantee of availability
 - Scheduler will not overcommit requests, do not schedule container until requested resources are available
- Limit
 - Max amount of a resource the container can access
- Consequences
 - Usage > request: resources might be available
 - Usage > limit: container is throttled or killed

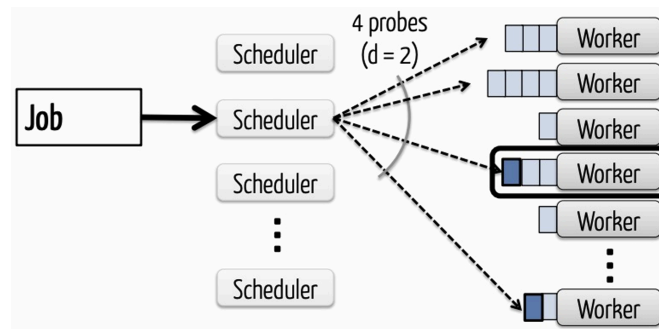
Kubernetes - QoS

- Defined in terms of requests and limits
 - Guaranteed: highest protection - request > 0 && limit == request
 - Burstable: medium protection - request > 0 && limit > request
 - Best effort: lowest protection - request == 0
- What does “protection” mean?
 - Highest protection = least likely to be evicted
 - Out of memory score
 - CPU scheduling

3. Mesos

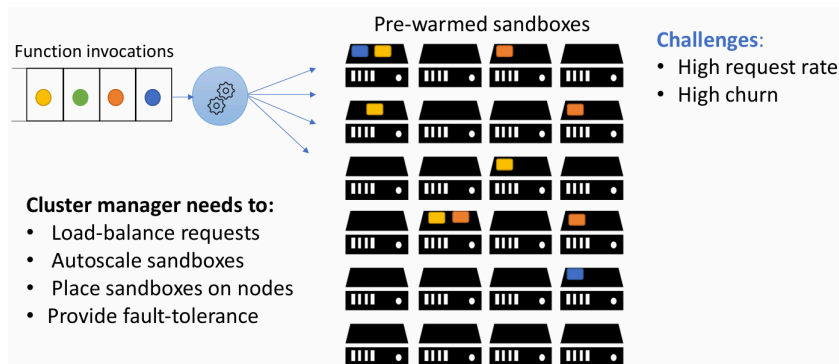


- Distributed scheduler architecture Each scheduler probes workers independently and queues up tasks on least-loaded workers.
- The power of two choices: instead of randomly assigning tasks to workers, probe 2 machines and place tasks on the least loaded of the two.
 - Problem: high overhead to do for one task at a time
 - Solution: probe machines and assign a batch of tasks each time
- Optimizations: late binding
 - Place m task reservations on the least loaded of $d * m$ workers
 - Workers place a reservation for the task at the end of the queue. When the reservation reaches the front of the queue, the worker notified the scheduler that placed the reservation. The scheduler assigns the job's task to the first m workers to reply, and replies to the remaining $(d - 1)m$ workers with a no-op.



II. Cluster management for serverless computing / FaaS

1. FaaS cluster orchestration



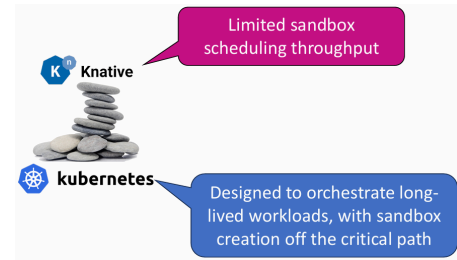
2. Current approach: retrofit legacy cluster manager

Add “serverless” features

- Autoscaling sandboxes based on request load
- Scale-to-zero support, buffering requests while wait for sandbox creation

Provides useful functionality

- Request load balancing
- Server and sandbox health monitoring
- Autoscaling sandboxes based on CPU / memory utilization
- Fault tolerance



3. A better approach: Dirigent

Dirigent is a clean-state cluster manager for FaaS that improves sandbox scheduling throughput by > 1000x vs. Knative-on-K8s.

III. Related readings

1. [Function as a Function](#)
2. [Dirigent: Lightweight Serverless Orchestration](#)