Kubernetes,
gVisor

Yiying Zhang
Cluster Management with Kubernetes

Please open the gears tab below for the speaker notes

Satnam Singh satnam@google.com

Work of the Google Kubernetes team and many open source contributors
Introduction to gVisor: Sandboxed Linux Container Runtime

EMMA HARUKA IWAO
Outline

- Kubernetes overview and interface
- Kubernetes internal architecture
- gVisor
We need 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

...
**Everything** at Google runs in containers:

- Gmail, Web Search, Maps, ...
- MapReduce, MillWheel, Pregel, ...
- Colossus, BigTable, Spanner, ...
- Even **Google’s Cloud Computing product GCE itself**: VMs run in containers
Open Source Containers: Kubernetes

Greek for “Helmsman”; also the root of the word “Governor” and “cybernetic”

- Container orchestrator
- Builds on Docker containers
  - also supporting other container technologies
- Multiple cloud and bare-metal environments
- Supports existing OSS apps
  - cannot require apps becoming cloud-native
- Inspired and informed by Google’s experiences and internal systems
- **100% Open source**, written in Go

Let users manage **applications**, not machines
Primary concepts

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

**Pod**: A small group of tightly coupled Containers

**Labels**: Identifying metadata attached to objects

**Selector**: A query against labels, producing a set result

**Controller**: A reconciliation loop that drives current state towards desired state

**Service**: A set of pods that work together
Pod

- a Kubernetes abstraction that represents a group of one or more application containers, and some shared resources for those containers
  - Shared storage, as Volumes
  - Networking, as a unique cluster IP address
  - Information about how to run each container, such as the container image version or specific ports to use
Node

- A node is a worker machine (either VM or physical machine)
- One pod runs on one node, one node can run multiple pods
- Nodes managed by control plane
Pods: Grouping containers

- Container Foo
- Container Bar

- Namespaces
  - Net
  - IPC
  - ..
Pods: Networking

Namespaces
- Net
- IPC
- ..
Pods: Labels

- Container Foo
- Container Bar
- Namespaces
  - Net
  - IPC
  - ..
Persistent Volumes

A higher-level abstraction - insulation from any one cloud environment

Admin provisions them, users claim them

Independent lifetime and fate

Can be handed-off between pods and lives until user is done with it

Dynamically “scheduled” and managed, like nodes and pods
Labels

Arbitrary metadata
Attached to any API object
Generally represent **identity**
Queryable by **selectors**
  • think SQL ‘select ... where ...’
The **only** grouping mechanism
Use to determine which objects to apply an operation to
  • pods under a ReplicationController
  • pods in a Service
  • capabilities of a node (scheduling constraints)
Selectors

App: Nifty  
Phase: Dev  
Role: FE

App: Nifty  
Phase: Dev  
Role: BE

App: Nifty  
Phase: Test  
Role: FE

App: Nifty  
Phase: Test  
Role: BE
Pod lifecycle

Once scheduled to a node, pods do not move
  • restart policy means restart **in-place**

Pods can be observed *pending, running, succeeded, or failed*
  • *failed* is **really** the end - no more restarts
  • no complex state machine logic

Pods are **not rescheduled** by the scheduler or apiserver
  • even if a node dies
  • controllers are responsible for this
  • keeps the scheduler **simple**

Apps should consider these rules
  • Services hide this
  • Makes pod-to-pod communication more formal
Outline

• Kubernetes overview and interface
• Kubernetes internal architecture
• gVisor
Control Plane Components

Architecture Overview
kube-apiserver

- Provides a forward facing REST interface into the Kubernetes control plane and datastore
- All clients and other applications interact with Kubernetes **strictly** through the API Server
- Acts as the gatekeeper to the cluster by handling authentication and authorization, request validation, mutation, and admission control in addition to being the front-end to the backing datastore
Monitors the cluster state via the apiserver and **steers the cluster towards the desired state**

- **Node Controller**: Responsible for noticing and responding when nodes go down.
- **Replication Controller**: Responsible for maintaining the correct number of pods for every replication controller object in the system.
- **Endpoints Controller**: Populates the Endpoints object (that is, joins Services & Pods).
- **Service Account & Token Controllers**: Create default accounts and API access tokens for new namespaces.
kube-scheduler

• Component on the master that watches newly created pods that have no node assigned, and selects a node for them to run on

• Factors taken into account for scheduling decisions include individual and collective resource requirements, hardware/software/policy constraints, affinity and anti-affinity specifications, data locality, inter-workload interference and deadlines
cloud-controller-manager

- **Node Controller**: For checking the cloud provider to determine if a node has been deleted in the cloud after it stops responding
- **Route Controller**: For setting up routes in the underlying cloud infrastructure
- **Service Controller**: For creating, updating, and deleting cloud provider load balancers
- **Volume Controller**: For creating, attaching, and mounting volumes, and interacting with the cloud provider to orchestrate volumes
etcd

- etcd: an atomic key-value store that uses Raft consensus
- Backing store for all control plane metadata
- Provides a strong, consistent and highly available key-value store for persisting cluster state
- Stores objects and config information
Reconciliation between declared and actual state
Control loops

Drive **current state** -> **desired state**

Act independently

APIs - **no shortcuts** or back doors

Observed state is truth

Recurring pattern in the system

Example: ReplicationController
Replication Controllers

#N

backend
production

backend
production
Replication Controllers

A type of controller (control loop)

Ensure N copies of a pod always running
- if too few, start new ones
- if too many, kill some
- group == selector

Cleanly layered on top of the core
- all access is by public APIs

Replicated pods are fungible
- No implied ordinality or identity

Other kinds of controllers coming
- e.g. job controller for batch
Node Components

Architecture Overview
kubelet

• An agent that runs on each node in the cluster. It makes sure that containers are running in a pod.

• The kubelet takes a set of PodSpecs that are provided through various mechanisms and ensures that the containers described in those PodSpecs are running and healthy.
kube-proxy

- Manages the network rules on each node.
- Performs connection forwarding or load balancing for Kubernetes cluster services.
Container Runtime Engine

- A container runtime is a CRI (Container Runtime Interface) compatible application that executes and manages containers.
  - Containerd (docker)
  - Cri-o
  - Rkt
  - Kata (formerly clear and hyper)
  - Virtlet (VM CRI compatible runtime)
Outline

• Kubernetes overview and interface
• Kubernetes internal architecture
• gVisor
“Containers do not contain” — Dan Walsh, 2014

- Still sharing the same kernel
- e.g., each container gets its own network interface, but uses the same Linux TCP/IP stack
- Share same device drivers
- Linux kernel represents a large attack surface
- cgroup accounting may not be accurate
Are System Calls Secure?

- The interface between containers and OS is system calls
- Linux x86_64 has 319 64-bit syscalls
- 2046 CVEs since 1999
Why can VMs be More Secure?

- Virtual machines
  - Independent guest kernels
  - Virtual hardware interface
    - Clear privilege separation and state encapsulation
  - But virtualized hardware interface is inflexible
    - e.g., can’t change number of virtualized cores at run time
    - and VM is heavy weight with large memory footprint
Sandboxing

- Rule-based sandboxing: reduce the attack surface by restricting what applications can access
  - e.g., AppArmor, SELinux, Seccomp-bpf
  - Rules can be fragile (not properly capture threats) and can’t prevent side channel attacks
gVisor

- Sandboxes untrusted applications
- Implements Linux system API in user space
  - 211 syscalls so far
  - Not direct port, not just filters
  - Runs unmodified Linux binaries
- Secure by default
  - No need to configure filters, policies
  - One “user-level kernel” per sandbox
- Written in Go, a memory/type-safe language
gVisor Architecture

- Two separate processes (communicated through IPC (9P))
  - Sentry: emulates Linux system calls in user space
  - Gofer: file access
- Most exploited syscalls: socket(2) and open(2)
  - Even if sentry is compromised, still can’t access files or open ports
- Network is handled by user-mode network stack in Sentry

![Diagram showing the gVisor Architecture]

- 211 Linux syscalls
  - 108 unsupported
- 55 Linux syscalls
- 52 Linux syscalls
Trapping System Calls

• Two modes supported

• ptrace
  • A debugging interface provided by Linux (PTRACE_SYSEMU is used to trap syscalls)
  • Sentry intercepts syscalls like a debugger attached to the application

• KVM
  • Sentry executes as a guest OS in a VM
  • Hardware virtualization support is required
gVisor Performance and Cautions

- 15MB memory usage
- 150ms startup time
- What it IS good for:
  - Small containers
  - Spin up quickly
  - High density
- What it’s NOT good for:
  - Trusted images (which can run on normal containers for better performance)
  - Syscall heavy workloads
  - Direct access to hardware, i.e. passthrough device support
  - Applications that use syscalls not supported by gVisor

Figure 3: **System Call Overhead.** The bars show the average latency for gettimeofday across 100M executions.
gVisor Usages in Google Cloud Platform

- Google App Engine is backed by gVisor
- Fast startup and low overhead are essential
- As long as Node.js works, perfect compatibility is not required