Append-only, randomly collected thoughts, remarks, notes, formulas, and a right amount of mathematics.

Explain to me as if this is five months after.

✳ ✳ ✳

MacVlan

Here is a good overview on what is Linux MacVlan and how docker uses its bridge mode.

MacVlan is a software feature in Linux kernel to attach virtual mac addresses to a physical layer 2 device. The new mac addresses are said to be the "child" of the parent network device, and can be used to configure slave (usually virtual) devices used in containers or VMs. MacVlan can run in multiple modes, the one supported in docker is the "bridge". This allows different child mac addresses to be able to routed to each other when they arrive at the parent device. This effectively turns the parent device into a switch ("bridge"), hence having the name.

Following this, one can setup containers using macvlan. Each container will be associated with a virtual interface with mac address, hence able to get an ip by the docker runtime. On the other hand, docker internal IP implementation does not allow for DHCP. On the host running containers, container can ping each other as they are using the bridge mode of the same physical interface macvlan, but cannot ping the host. To ping host, one need to create a link using an additional macvlan and give it an IP. This is the internal constraint in macvlan: child mac cannot be routed to the parent mac in that same network interface. In fact, it is filtered out in the first hop. As an interesting consequence, outside of the host, I can ping both the host and the container via the ip associated with that interface. So each mac will have an IP (ARP protocol), but one interface can have multiple mac address hence multiple ip via macvlan.

Linux Programming Interfaces Today

Traditional operating system course in a computer science class will introduce file system as a storage abstraction (file as storage access gateways), sockets and DNS as network abstractions (naming and transportation), and thread/process as a compute abstraction (sharing, CPU, memory, etc).

Virtualization technologies was a first step to cope with the increasing compute, storage and networking hardwares. Applications, however monolithic they tend to be, are mostly unable to saturate a single hosts' power. As a by product, they achieve better isolation and provide opportunities for better management.

Containerization technologies were almost as early as when virtual machines began the fun, but since they are highly operating system dependent (Google was among the first a few hacking and upstreaming the patches for `cgroup` and `namespaces`), they did not attract so much attention nor having a commercial success like other virtual machine vendors.

It seems like today due to the domination of the Linux kernel and huge success in the open source ecosystem, most of the applications backing the Internet are on Linux. This fuels the new programming abstractions people use to build applications. Knowing the mature support in containerization, as well as the requirements for scaling applications for Internet load, applications are built and deployed not in an operating system like they would be when running inside a virtual machine. Instead, they become containers and assumes little about the underlying storage nor networking.

Networking are getting more and more complicated in the world of containers. Applications are not only requiring an IP address to talk to each other or consume an API. Networks are further divided via the network namespaces like the Pod Abstraction in Kubernetes, traffics are usually load-balanced and routed to different backend services, and tracing, logging, health checking and service discoveries are ubiquitous in today's backend design. These are far beyond what a socket can offer. Tools like Envoy are built to create this "Service Mesh" abstraction and provide these services, as well as some existing tools like Nginx and application libraries.

What abstractions should storage systems provide? Local file system was how everything begins, then follows richer semantics like Key-Value and SQL. Access path to the storage layer is no longer a simple local syscall. Network hops, placement decisions and data sharding are all new considerations in a storage service. Open file for writing, list directories and files, these are no longer what applications need. Structured data are stored in a relational manner, and unstructured, large, streaming data are either in a blob/object store, or piped back and forth via a streaming service.