ECE 473/573
Cloud Computing and Cloud Native Systems
Lecture 06 Containerization

Professor Jia Wang
Department of Electrical and Computer Engineering
Illinois Institute of Technology

September 5, 2025

Outline

Reading Assignment

- This lecture: Containerization
- Next lecture (Mon. 9/8): 5

Outline

Advanced Virtualization Techniques

Containerization

Docker Introduction

Memory Ballooning

- Guest OS running inside a hypervisor may install a balloon driver to facilitate memory virtualization.
- The balloon driver communicates with the hypervisor to handle its requests on memory usage.
- When there is not enough host memory, the hypervisor will request the balloon driver to allocate more from the guest OS
  - That should not be backed by disk storage.
  - Guest OS will need to flush some memory to disk before giving them to the balloon driver – optimizations are possible for guest OS and application with such information.
  - The balloon driver doesn't actually use the allocated memory but allows the hypervisor to reclaim them.
- When host memory becomes available, the hypervisor will request the balloon driver to release those allocated memory.
  - Now the guest OS can allocate them to applications.

Device Virtualization

- Device Emulation
  - Hypervisor intercepts and translates all I/O requests on a device from multiple guest OSes.
  - Flexible and highly compatible.
  - Performance overhead, especially for high performance I/O.
- Device Pass-Through
  - Allow access to the device by a single guest OS.
  - Not flexible – guest OS needs specific drivers.
  - No performance overhead.
- Paravirtualization
  - Guest OS has knowledge of virtualization and collaborates with the hypervisor to access the device.
  - Better performance than device emulation.
  - Need modification to guest OS.
- SR-IOV (Single Root I/O Virtualization)
  - Hardware collaborates with hypervisor to improve efficiency when accessed from multiple guest OSes.
  - Similar performance as device pass-through.
  - Limited to supported hardware.

Outline

Applications and Virtual Machines

- Is it a good idea to run a single application per virtual machine?
  - Consider cloud native applications.
  - What about operation and maintenance?
  - What about isolation?
  - What about performance and utilization?
- Cloud native application architecture
  - Loosely coupled scalable microservices.
  - Use microservices from third parties to reduce developement time and cost.
  - Running multiple instances of the same microservice to meet performance demand.
- How to deploy microservices to VMs?
  - Need multiple VMs for scalability.
  - Without renting more than enough VMs.

Microservice Deployment Considerations

- Use scripts to automate the installation process.
  - Install microservices and dependencies.
  - Speedup installation by providing a pre-built system.
- Microservices may impact application performance differently.
  - Need to promptly start more instances for some microservices to meet rising demand.
  - Leverage overprovisioning to improve utilization by running some microservices on the same server.
- Services may need root privilege to access certain resources.
  - But there could be bugs or misconfigurations.
  - Running multiple microservices in a single VM is risky.

Nested Virtualization

- Run virtual machines within other virtual machines.
  - Rent a VM and deploy microservices into their own nested VMs
  - Isolation achieved and overprovisioning is possible.
- VM images are large because they include the whole OS.
  - Consume a lot of resource to transmit.
- Starting a VM need to boot the whole OS and is slow.
- Nested VMs introduce a lot of performance overhead.
  - Two hypervisors and two OSes for a single microservice.
  - Very difficult to optimize as only the inner OS can understand the behavior of the microservice.
- Can we optimize if the OS running in the VM and the OSes running in the nested VMs are the same?

Containerization

- Lightweight virtualization
  - Virtualize the OS kernel instead of the whole hardware system.
  - Guest OS shares the kernel with the host OS so they need to be similar, e.g. different versions and distributions of Linux.
- Container: a guest OS with a microservice or other applications running inside.
- Container image: a package to start a container.
  - Containerized guest OS
  - A file system including programs and data.
- Container runtime: manage containers and container images.
- Container orchestration: manage containerized microservices and applications across multiple (virtual) servers.

Containers vs. VMs

- Container images are smaller than VM images.
  - No need to include the whole guest OS – the kernel is available from the host.
- Starting a container is faster than starting a VM.
  - No need to boot the guest OS – the kernel is already running.
- Containers has less performance overhead.
  - Shared kernel means that processes in a container actually run on top of the host OS directly.
- Overprovisioning is more effective with containers.
  - Host OS has knowledge of processes in containers so can optimizes better than hypervisor.
- Containers are isolated by the host OS kernel.
  - Weaker than isolation provided by hypervisor.
  - Sufficient for most use cases where the containers serve the same application.

Additional Features

- With containerization technologies, virtualization technologies focuses more on emulation and isolation.
- Containerization provide additional features to simplify operation and maintenance of microservices.
  - Scripting to containerize microservices and applications.
  - Flexible system, storage, and network configurations.
  - Version control and rollbacks.
  - Pre-built container images.

Outline

Docker Ecosystem

- Docker: open-source container runtime.
  - With docker container and docker images.
- Docker registry: a centralized repository for docker images.
- Docker engine: the core runtime for running containers.
- Container orchestration platforms
  - Docker Compose: manage docker containers on a single host.
  - Docker Swarm: manage docker containers on multiple hosts.
  - Kubernetes (k8s): support docker and other container runtimes.

A Few Docker Commands

- Work with docker images.
  - `docker pull`: download a docker image from a registry.
  - `docker build`: create a docker image from a Dockerfile.
  - `docker images`: list docker images available locally.
- Work with docker containers.
  - `docker run`: start a docker container from a docker image.
  - `docker exec`: execute command in a docker container.
  - `docker ps`: list docker containers.

Dockerfile

- A script to create docker images in textual format.
  - Make docker images reproducible.
  - Can be easily managed in a version control system.
- Common contents
  - Base image: start from an image with existing guest OS and/or packages installed.
  - Guest OS scripts: additional package installations and OS configurations.
  - Environment setup: enable containers to interact with host OS for configurations, permissions, storage, networking, etc.

Starting a Docker Container

- `docker run` uses a lot of options to control how a docker container should start.
- `--name` specifies a name for the container so one can find it easily in `docker ps`.
- `-p` publishes port from the container so one can access networked services from the host OS or externally.
- `-v` maps a host directory to the container so files can be shared between the two.
- `-e` sets environment variables in the container to configure microservices and to pass sensitive information.
- `-it` allows one to interact with the running container via a terminal and `-d` prevents so.

Summary

- Containerization provides a lightweight virtualization solution.
- Guest OS in a container shares the kernel with the host OS.
- Containerize an application or a microservices by first creating a container image and then starting a container from it.