Hálózatba kapcsolt erőforrás platformok és alkalmazásaik

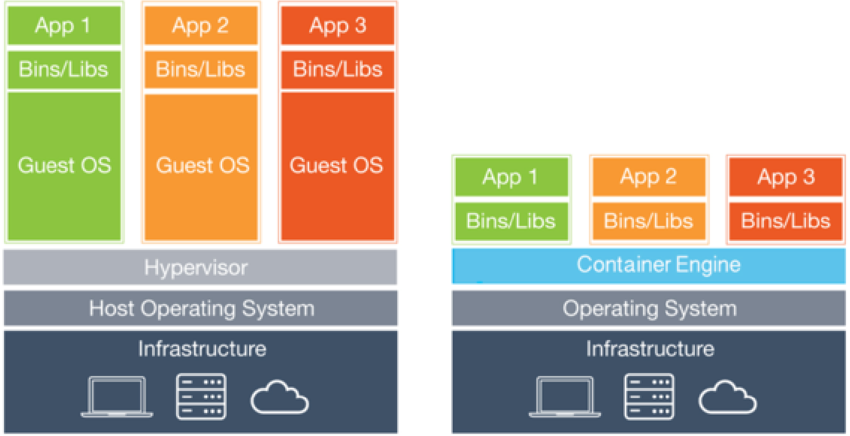
Maliosz Markosz

TMIT

2018

Containers

- Operating System-level virtualization
- Self-contained execution environments
  - with their own, isolated CPU, memory, block I/O, and network resources
  - share the kernel of the host operating system



Hypervisor-based Virtualization

Container virtualization

Containers

- Pros
  - lightweight, fast deployment time, portable, flexible
  - quick scaling
- Cons
  - security
    - runs a daemon that requires root
    - default user in container is root
  - lack the hardware isolation that VMs provide

Use of Containers 1/2

- Application packaging
  - with all of the parts it needs, such as libraries and other dependencies, and ship it all out as one package
- DevOps, Continuous Integration / Continuous Delivery

Use of Containers 2/2

- Microservices architecture
  - complex applications broken down into smaller, composable pieces which work together
    - divide and conquer
    - same concept: Service Oriented Architecture (SOA)
  - components can be scaled independently
  - ⇨ orchestration tools
  - contra: creates a whole another set of problems
    - understanding system as a whole, what's dependent on what
    - when one service fails, there is much higher possibility that it will cause a cascading failure which is far harder to trace

Linux Containers: Implementation

- Linux kernel features
  - cgroups (control groups): limiting and accounting resource usage (CPU, memory, disk I/O, network) for a collection of processes
  - namespaces: allow per-namespace mappings of resources (e.g. process IDs, mounts, user IDs, network interfaces, interprocess communication, filesystems), i.e. process isolation

A brief history

- 2000, FreeBSD jails
- 2001, Linux VServer
  - Linux kernel patch
- 2005, OpenVZ (Open Virtuozzo)
  - patched Linux kernel for virtualization, isolation, resource management and checkpointing
- 2006, Process Containers (Google) ⇨ cgroups
- 2008, Original Linux Containers: LXC, LXD
  - adding tools, templates, libraries for easy management
- 2013, Docker (⇦ 2008, dotCloud, Inc.)
  - utility that can efficiently create, ship, and run containers (high level view)
  - started with own container runtime environment
  - since Docker Engine 1.11 (2016) it is built on runC (a runtime based on Open Container Intiative technology) and containerd
- 2013, CoreOS rkt (rocket)
  - A Docker alternative

Windows Containers

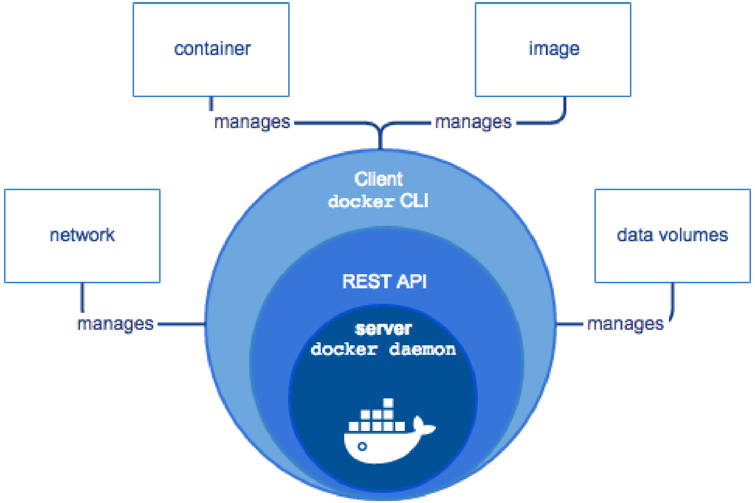
- Using native container technology in Windows
- Docker on/for Windows Server 2016 or Windows 10 Pro
- Types
  - Windows Server Containers
    - Process and namespace isolation
    - Kernel is shared with host
  - Hyper-V Containers
    - Runs a container in a VM
    - Kernel is not shared

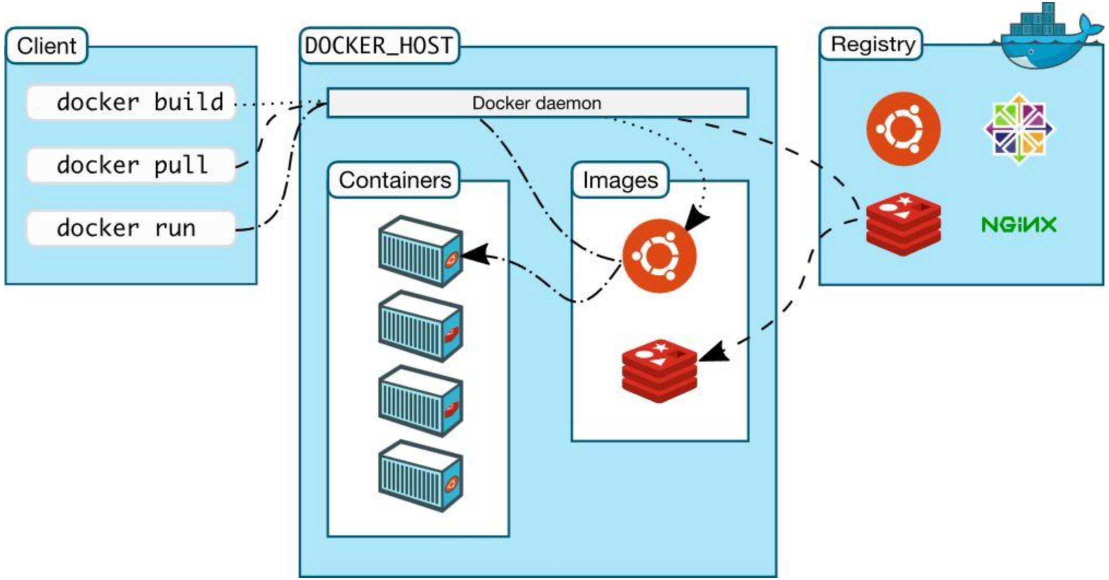
Docker

Docker terminology

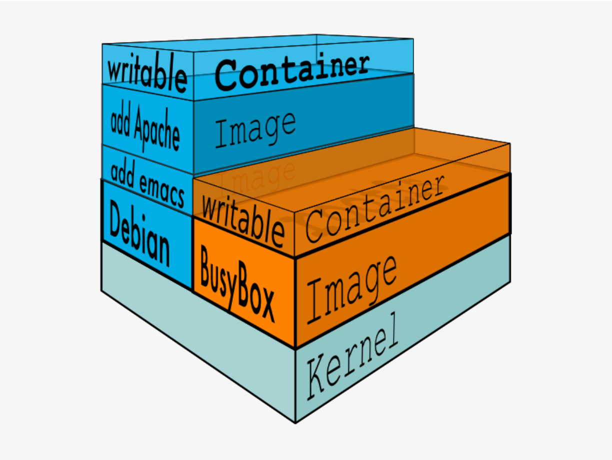
- Container: runtime instance of a Docker image
- Image: filesystem and parameters
- Registry: repository of images
  - Docker Hub
  - pull/push

Docker Architecture





Docker Images



- Read-only templates
- Consists of a series of layers
- Docker uses union file systems to combine these layers into a single image
- Image is defined in a Dockerfile
  - Starts from a base image (e.g. ubuntu, fedora, etc.)
  - Adding new layers by simple instructions
- Image specifies
  - container’s contents,
  - which process to run when the container is launched,
  - other configuration details

```
A Dockerfile:
FROM          ubuntu:14.04
RUN           apt-get update && apt-get install -y redis-server
EXPOSE        6379
ENTRYPOINT    ["/usr/bin/redis-server"]
```

Using Docker

- `sudo docker run -i -t ubuntu /bin/bash`
  - automatically downloads an Ubuntu image
  - creates a Docker container that just runs the bash shell
  - You’ll get dropped into a command prompt, like:

    ```
    root@4a2f737d6e2e:/#
    ```
  - running in a clean environment
  - very fast container start
- containers are ephemeral—changes to the container aren’t persistent
- for persistent storage: volumes

Container Orchestration

Container Orchestration – Single node

- Docker compose
  - running multi-container Docker applications
  - Compose file configures services

```
A docker-compose.yml:
version: '2'
services:
  web:
    build: .
    ports:
    - "5000:5000"
    volumes:
    - .:/code
    - logvolume01:/var/log
    links:
    - redis
  redis:
    image: redis
volumes:
  logvolume01: {}
```

Container Orchestration – Multi node

- Automating Linux container operations
  - ▫ Goals
    - Cluster together multiple hosts
    - Placement and Placement control
    - Affinity/anti-affinity
    - Network orchestration
    - High availability
    - Scaling
    - Load balancing
    - Rolling upgrades
  - ▫ Challenge: how to deploy and orchestrate containers at scale

Container Orchestration Tools

- Tools
  - On premise
    - Kubernetes (Google, 2014)
    - Docker Swarm
    - Apache Mesos / Marathon
    - …
  - Cloud Provider
    - Amazon ECS (Elastic Container Service)
    - Azure Container Service
    - Google Container Engine (built on Kubernetes)
    - …

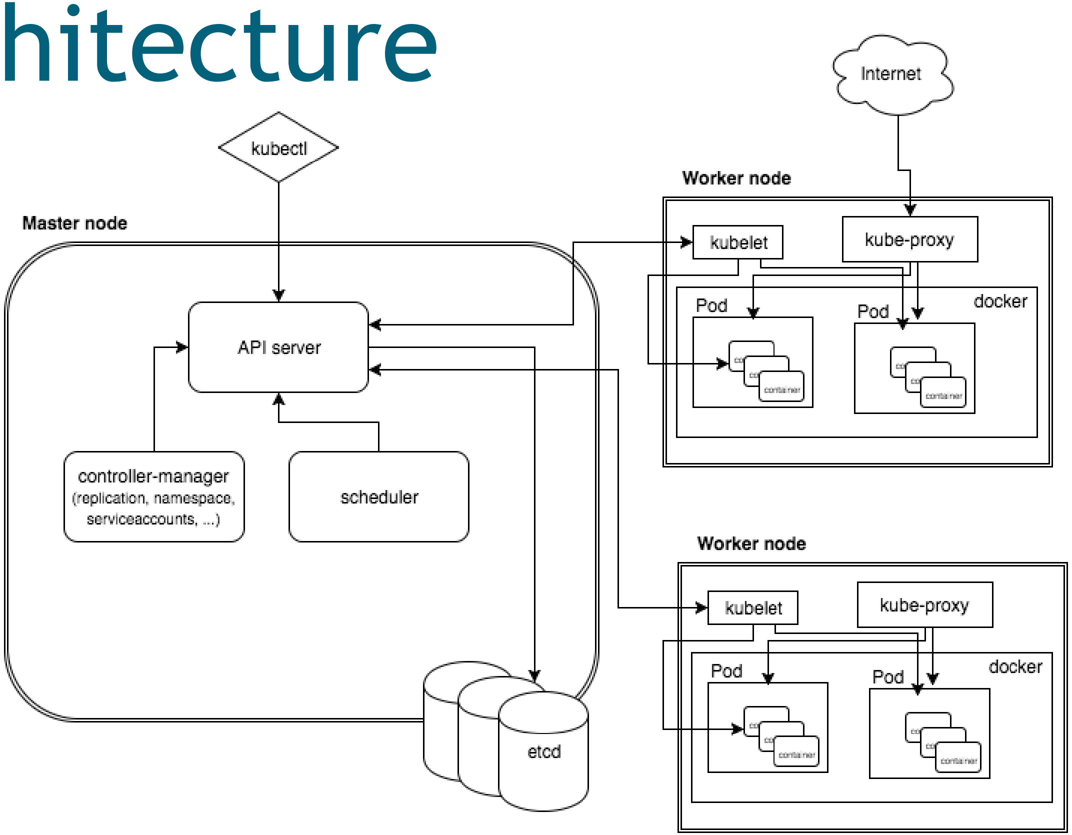
Kubernetes

Kubernetes

- Features
  - build application services that span multiple containers
  - schedule those containers across a cluster,
  - scale those containers,
  - manage the health of those containers over time
  - manage changes to existing containerized applications
  - fault-tolerant by allowing application components to restart and move across systems as needed
- Needs to integrate with networking, storage, security, telemetry and other services to provide container infrastructure
- This is all very useful when it comes to simple, stateless services that you can load balance across, and where all instances are completely identical
  - Things get a bit more complicated when you have stateful services, or when the micro-service itself is composed of multiple pieces

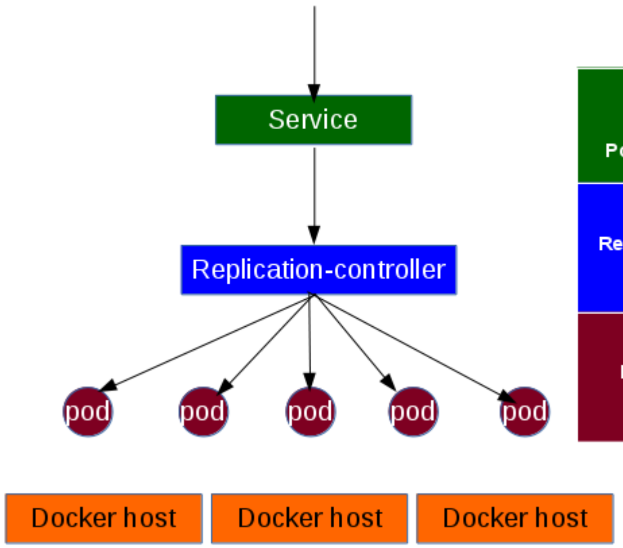
Kubernetes Architecture

- Pods add a layer of abstraction to grouped containers
- Supported container formats
  - Docker
  - rkt
  - runC
  - hypervisor-based



Kubernetes Services

- A Kubernetes Service represents load-balancing group of PODs



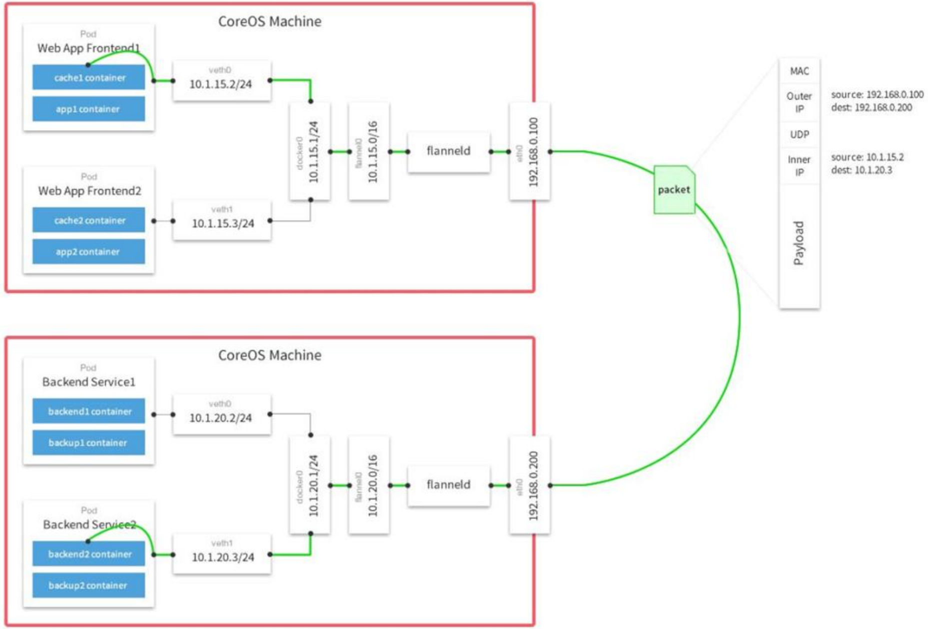
HTTP Port : 8000	MYSQL Port : 3306	WORDPRESS Port : 8001
Replicas = 2	Replicas = 1	Replicas =2
Pod x 2	Pod x 1	Pod x 2

Kubernetes Networking

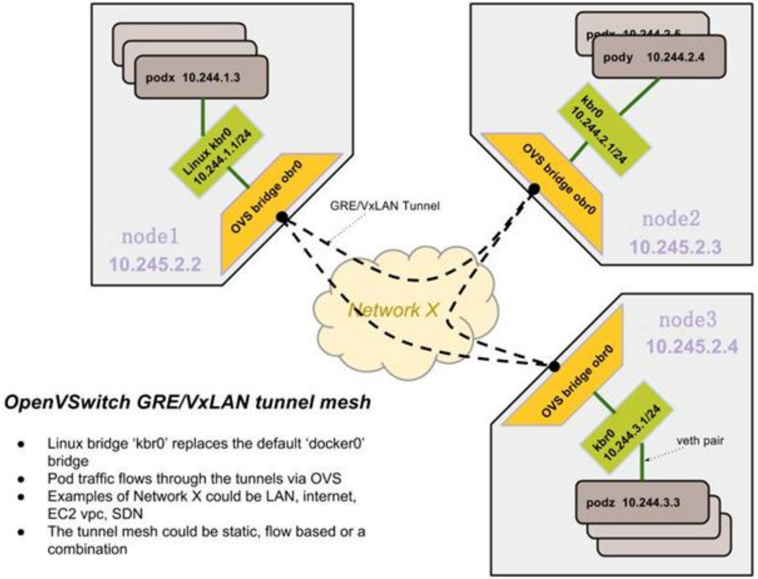
- Docker model: via virtual bridge
- Kubernetes model: applies IP addresses at the Pod scope
  - containers within a Pod share their network namespaces - including their IP address (reach each others ports on localhost)
  - inter-pod communication
  - many implemetation alternatives
    - Flannel, Contiv, Contrail, Linuxbridge, OpenVSwitch, …

Inter-pod communication

- flanel



- OVS



OpenVSwitch GRE/VxLAN tunnel mesh

- Linux bridge 'kbr0' replaces the default 'docker0' bridge
- Pod traffic flows through the tunnels via OVS
- Examples of Network X could be LAN, internet, EC2 vpc, SDN
- The tunnel mesh could be static, flow based or a combination

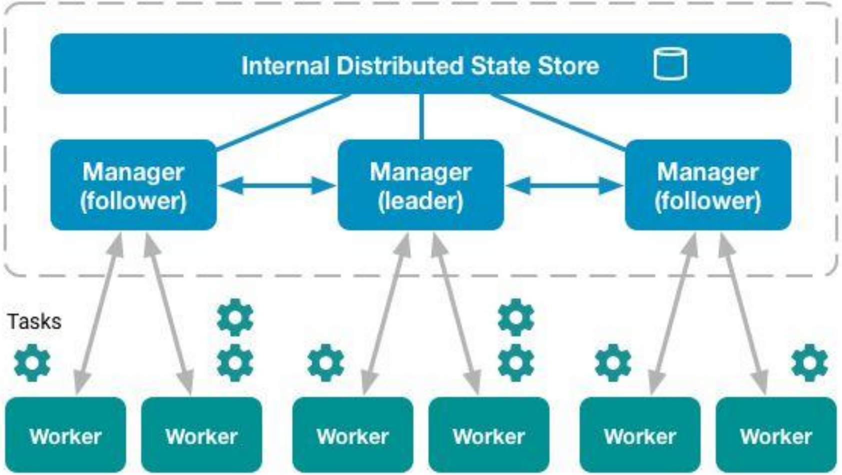
Docker Swarm

Docker Swarm Mode

- Docker Engine in swarm mode (since v1.12.0)
  - Cluster management
  - Scaling
  - Desired state reconciliation
  - Multi-host networking
  - Service discovery
  - Load balancing
  - Rolling updates
- Service: Central structure of the swarm system
  - Creating a service: specifying which container image to use and which commands to execute inside running containers

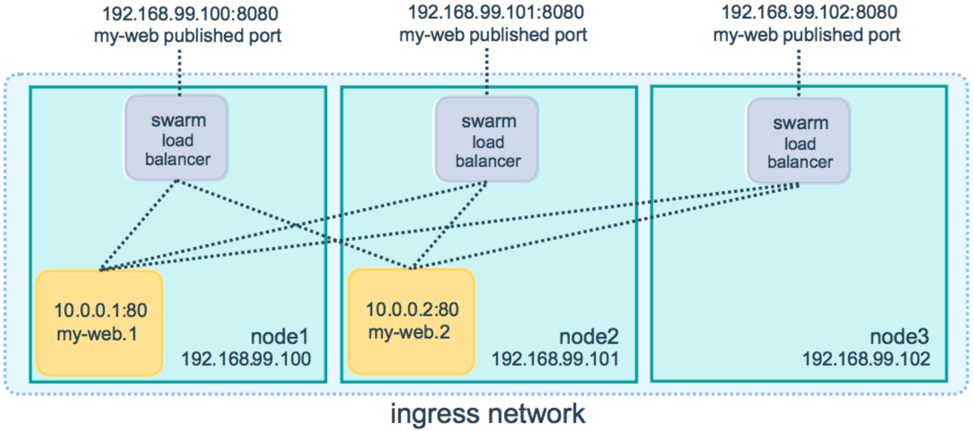
Docker Swarm Architecture

- Manager nodes
  - Maintain cluster state
  - Schedule services
  - Serving swarm mode API
  - Multiple managers for fault tolerance
- Worker nodes
  - Execute containers
  - By default managers are also workers



Swarm Mode Networking

- Swarm mode routing mesh
  - access port on any node, the swarm load balancer routes request to an active container



Containers and Cloud

- Hosts can come from several different sources, including physical servers, virtual machines or cloud providers
- VMs and containers co-exist
- Docker
  - primarily a Linux-based container packaging technology
  - Microsoft has adopted and partnered with Docker as its containerization packaging standard for Azure
  - Amazon ECS uses *Docker* images in task definitions to launch containers on *EC2* instances
- Google, 2014
  - everything at Google runs in a container
  - we start over 2 billion containers per week