# Ôn tập NT533 - Hệ tính toán phân bố

## Week 1 + 2: Course Introduction, Fundamentals

### Some keywords from the 5th Computer System Generation

- *The network is the computer*
- Distributed system: **Cluster, Cloud, Grid, P2P-Computing**
- Multicore processors and **parallel applications**
- Virtualization: **VMware, XEN, KVM (Kernel-based Virtual Machine), Docker**
- Open Source: **Linux**, BSD

## Client - Server

A system includes

- One or more **client** which use the services of the server and accesses data, stored at the server (consumers)
- A **Server** which provides services and/or data (producer)

The connection establishment is initiated by the clients

Communication works according to a protocol

- A client sends a request to the server
- The server responds with a reply

The client-server architecture consists of 2 layers and is called two-tier model (tier = layer)

Tasks in Client - Server Model

- Display (graphical) user interface
- Calculation of the (graphical) user interface
- Data processing
- Data management
- Data storage

Types of Clients

- Text-Terminals, X-Terminals
  - Only display the (graphical) user interface and transfer the user interaction to the server
  - Calculation of the (graphical) user interface, data processing and data storage, data management are tasks of the server
- Thin/Zero Clients
  - Calculate and display the graphical user interface
- Applet Clients
  - Calculate and display the graphical user interface and do a part of the data processing
  - The clients process the applications (applets) themselves
- Fat Clients
  - Only data management and data storage are located on the (file or database) server

Advantages of Thin Clients

- Low acquisition cost
- Reduced power consumption => reduced operating costs
- Little space consumption
- Reduced noise
- Central storage of data is more efficient and more secure
- Virtualization on the server
- Reduced effort for administration

Drawback of Thin Clients

- No 3D graphics performance
- Limited extensibility
- Users fear storing their data outside of their PC
- Server is a single point of failure and eventually a bottleneck

## Moore's Law

Published in 1965 by Gordon Moore

It is based of empirical observation

Moore originally meant the electronic components on of integrated circuit double every 12 months

Since the late 1970s, the packing density only doubles every 24 months

## Von Neumann Bottleneck

The data and control bus is increasingly becoming a bottleneck between the CPU and memory

The main memory and the bus system are key factors for the performance of a computer

The Von Neumann Architecture describes the structure of the general-purpose computer, which is not limited to a fixed program and has input and output devices

Main difference to modern systems: A single Bus to connect I/O devices directly with the CPU, is impossible today

Main memory is usually DRAM (Dynamic Random Access Memory)

The access time (or cycle time)

- DDR-400 SDRAM is 5 ns: 200 MHz
- DDR3-2400 SDRAM is 0.833 ns: 1200 MHz
- DDR4-4800 SDRAM is 0.417 ns: 2400

> *Note: 1 Hz =* $\frac{1}{seconds}$

To reduce bottleneck impact

- Caches is SRAM (Static Random Access Memory), its access speed is close to the CPU speed, and it can reduce the bottleneck impact

To increase bottleneck impact

- If multiple CPUs (or cores) share the main memory and thus share the memory bus

## Amdahl's Law

Published in 1967, named after Gene Myron Amdahl

Calculates the maximum expected acceleration of programs by parallel execution on multiple CPUs

According to Amdahl, the performance gain is limited mainly by the **sequential part** of the problem

A program can never be fully executed in parallel

Formula: $S = \frac{1}{1-P+\frac{P}{N}}$

> Source: Computer Organization | Amdahl's law and its proof - GeeksforGeeks

- S: speedup of the system
- P: proportion of the system that can be improved
- N: number of processors in the system

(Example): If a system has a single bottleneck that occupies 20% of the total execution time, and we add 4 more processors to the system, the speedup would be:

> Source: Computer Organization | Amdahl's law and its proof - GeeksforGeeks

- $S = \frac{1}{1-0.2+\frac{0.2}{4+1}}$
- S = 1.19
- This means that the overall performance of the system would improve by about 19% with the addition of the 4 processors.

Issues

- Amdahl's law does not take into account the cache and the effects, which are caused by the cache in practice
- In the optimal case, the entire data can be stored in the cache. In such a case (very rare), a super-linear SpeedUp may occur: $S_(p) = \frac{t_(s)}{t_(p)}$ , with
  - $S_(p)$ : Speedup Factor when using p CPU cores of a multiprocessor system
  - $t_(s)$ : execution time by using a single CPU core
  - $t_(p)$ : execution time by using p CPU cores
- The max. SpeedUp is usually p with p CPU cores (=⇒ linear SpeedUp)
- A super-linear SpeedUp is greater than p

Transferred to parallel computers, this means that with a growing number of CPUs, the problem size should grow too

The problem needs to scale with the number of CPUs

## Gustafson's Law

Gustafson's Law from John Gustafson (1988) says that a problem, which is sufficiently large, can be parallelized efficiently

Difference to Amdahl's law

- The parallel portion of the problem grows with the number of CPUs
- The sequential part is not limiting, because it gets more and more unimportant as the number of CPUs rises

Formula: $\text{scaled speedup} = N + (1 - N)s$

> Source: When Should I Use Parallel Programming? Gustafson's Law | Medium

- N: number of processors in the system
- s: the fraction of the computation that is serial (i.e., not parallelizable)

> *Note: In some documents, s is replaced with* $\alpha$

(Example): Suppose we have a program that is **70% parallel** and **30% sequential**, and we have **10 processors**

$\text{Scaled speedup} = N + (1 - N)s = 10 + (1–10) \times 0.3 = 7.3$

> **Note**
>
> Due to Gustafson's law - Wikipedia, s is the fraction time spent executing the serial parts, and p is the fraction time for parallel parts of the program. If the given fraction is s, then use the above formula, else use this formula: $S = 1 + (N - 1) \times p$

## Parallel Computers

Sequential operating computers which follow the Von Neumann architecture are equipped with

- A single CPU
- A single main memory for the data and the programs

For parallel computers, 2 fundamentally different variants exist

- Systems with shared memory
- Systems with distributed memory

For systems with shared memory

- The entire memory is part of a uniform address space, which is accessed by all CPUs
- Problem: Write operations of the CPUs must be coordinated

For systems with distributed memory

- It is also called *Cluster or Multicomputer*
- Each CPU can only access its own local memory
- The communication between the CPUs takes place via a network connection
- In a parallel computer, every single CPU and it’s local memory, are is independent node
- Nodes of the cluster can also be SMP systems

The symmetric multiprocessing (SMP) principle

- SMP allows to **dynamically distribute** the running processes to all available CPUs
- All CPUs can access the memory with the same speed

The asymmetric multiprocessing principle

- Each CPU must be assigned to a fixed task
- One or more CPUs run the operating system
- The other processes are distributed to the remaining CPUs
- Typically, the CPUs are identical

# Week 3 + 4: Cluster Computing, FaaS, Container, Function as a Service

## Cluster Computing

Clustering is parallel computing on systems with distributed memory

A cluster consists of at least 2 nodes

- Each node is an independent computer system
- The nodes are connected via a computer network
- Often, the nodes are under the control of a master and are attached to a shared storage
- Nodes can be ordinary PCs, containing commodity hardware, workstations, servers or supercomputers

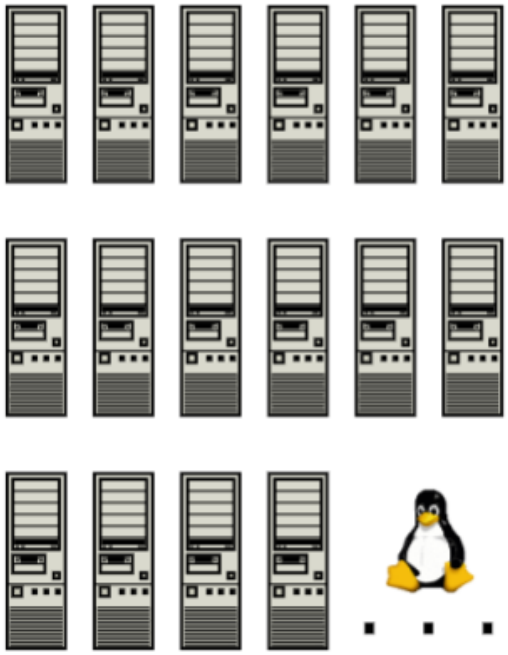
Clusters of workstations (COWs) or Network of Workstations (NOWs): The nodes are only available at specific times

Distinguishing Criteria of Clusters

- Structure: Homogeneous structure, Heterogeneous structure

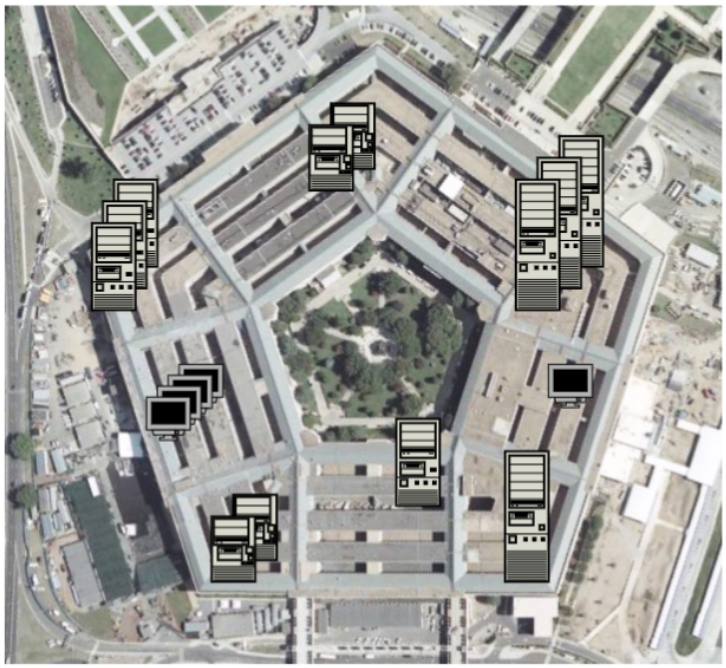
Heterogeneous structure

Homogeneous structure



- Installation concept: Glass-house, Campus-wide

- **Glass-house**
  - The cluster is located in a single room or server rack
- Advantages:
  - Fast access to all components for maintenance and troubleshooting
  - Nodes can be connected via high-performance networks
  - Increased protection against sabotage
- Drawbacks:
  - In case of a power failure or fire in the building, the operation of the entire cluster is at risk



- **Campus-wide**
  - The nodes are located in multiple buildings and spread across the site of the research center or company
- Advantages:
  - It is hard to destroy the cluster completely
- Drawbacks:
  - It is impossible to use high-performance computer networks
  - Often, the nodes contain different hardware components

- Fields of application: High Performance Clustering, High Availability Clustering, High Throughput Clustering
- Behavior in the event of failed nodes: Active/Passive-Cluster, Active/Active-Cluster

For High Availability Clustering

- Redundancy of nodes and their components are required
- Avoiding a single point of failure
- Redundancy means: A system contains components, which are not required for the functioning of the system, but they can take over the work of identical components in case of error
- By using redundant nodes, it is possible to emulate the technology and benefits of mainframes for a low price and a high level of flexibility is achieved
- The aim of HA clustering is to archive a high availability
- Not the availability of the nodes has top priority, but the availability of the offered services

The availability of a system: $\text{availability} = \frac{\text{mean uptime}}{\text{mean uptime} + \text{mean downtime}}$

Active/Passive-Cluster (also called: Hot-Standby-Clusters)

- During normal operation, at least a single node is in passive state
- Nodes in passive state do not provide services during normal operation
- If a node fails, a passive node takes over its services

- Failover = a node takes over the services of a failed node
- Benefit: The services must not be designed for cluster operation
- Drawback: Much potentially available performance remains unused in normal operation

Active/Active-Cluster

- All nodes run the same services
- All nodes are in active state
- If nodes fail, the remaining active nodes need to take over their tasks
- Advantage: Better distribution of load between nodes
- Drawback: Services need to be designed for cluster operation, because all nodes access shared resources (data!) simultaneously

Failover and Failback

- Failover: Ability to automatically transfer the tasks of a failed node to another node for minimizing the downtime. (i.e., Heartbeat for Linux)
- Failback: If failed nodes are operational again, they report their status to the load balancer and get new jobs assigned in the future

2 architecture of HA Clustering

- Shared Nothing Architecture: Distributed Storage, each node has its own storage resource
- Shared Disk Architecture: Shared Storage, all nodes have access to a shared storage

Distributed Replicated Block Device (DRBD): Free software to build up a network storage for Shared Nothing clusters, without an expensive Storage Area Network (SAN)

Split Brain

- If shared storage is used, each node tries to write on the storage
- If distributed storage is used, write requests cause inconsistent data on the nodes

## Libraries for Cluster Applications

Parallel Virtual Machine (PVM)

- **PVM is not a programming language!**
- Provides a uniform programming interface for the creation of a parallel computing unit with distributed memory
- Consists of **a daemon**, libraries and tools

- Especially suited for **heterogeneous** environments
- Focus: **not performance, but portability**

Message Passing Interface (MPI)

- **MPI is not a programming language!**
- Collection of functions (e.g. for process communication) to simplify the development of applications for parallel computers
- Contains **no daemon**
- Implements message-based communication (message passing)
- Especially suited for **homogeneous** environments
- Focus: **Performance and security**

MPI Functions

- **Broadcast Sending:** send same package to all node
- Scatter: divide packages into segments and send each segment to each node
- Gather: receive parts from node
- Allgather: each node send package to all nodes (become a network)
- Barrier: Blocks the execution of the calling process, until all processes in the communicator comm have called the barrier function

Gearman

- Framework for developing distributed applications
- Assigns one of 3 roles to every computer involved
  - **Client:** transfer jobs to the Job Servers
  - **Job Server:** assign jobs of the clients to the Workers
  - **Worker:** register themselves at Job Servers and execute jobs
  - Clients and workers access shared data
- Should only be used in **secure private networks**: The communication is **not encrypted and uses port 4730**, and no mechanism for the authentication of the systems is implemented

## Container Virtualization, Docker

Container Virtualization capsulate Applications in virtual environments!

Containers are more efficient than Hypervisor-based Virtualization or Paravirtualization!

Enables Container Virtualization

Docker Architecture

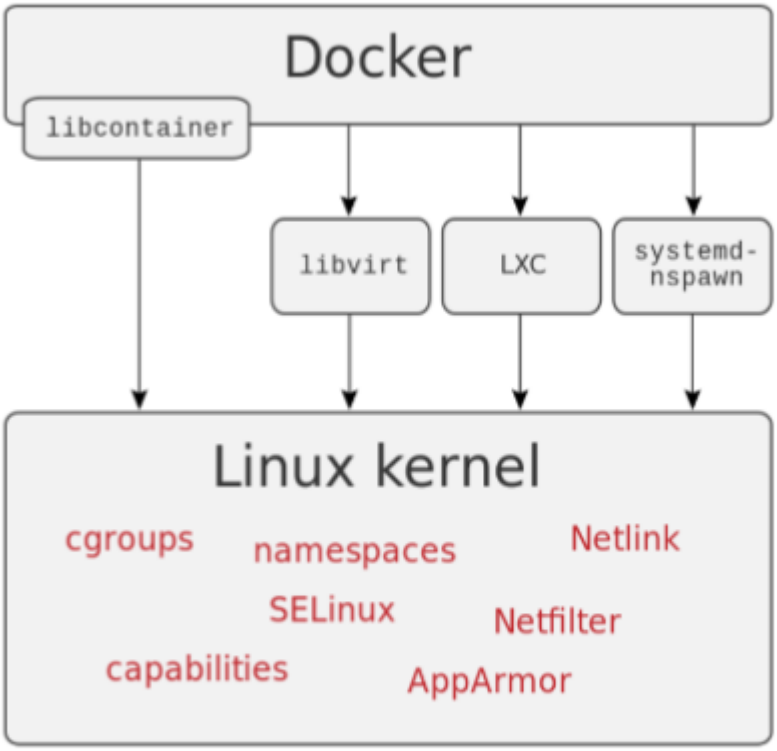
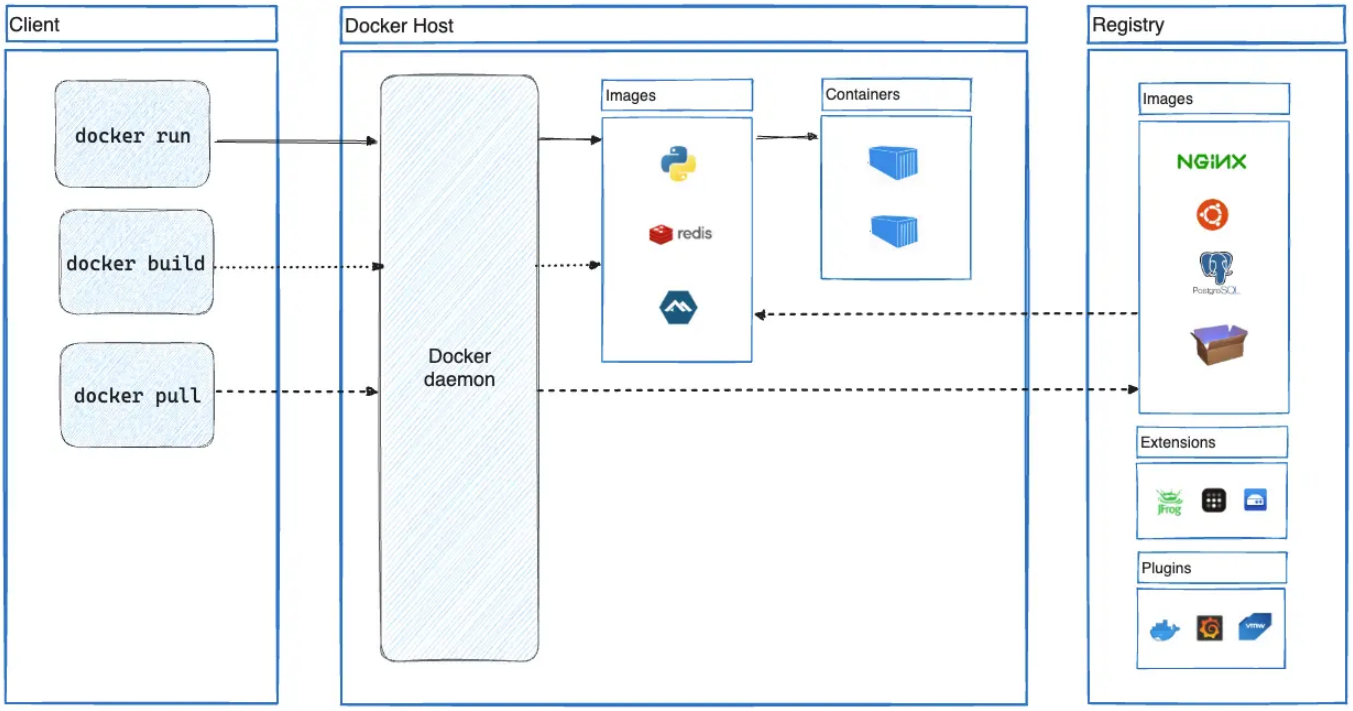



Figure: Docker Architecture

- Docker uses the Linux Kernel
- `libcontainer` creates containers
- `libvirt` manages Virtual Environments
- `LXC` will be replaced by `libcontainer`

Docker Application Architecture

> Source: Docker overview | Docker Docs




- **Docker Client:** The Docker client ( `docker` ) is the primary way that many Docker users interact with Docker. When you use commands such as `docker run` , the client sends these commands to `dockerd` , which carries them out. The `docker` command uses the Docker API. The Docker client can communicate with more than one daemon.
- **Docker Daemon:** The Docker daemon ( `dockerd` ) listens for Docker API requests and manages Docker objects such as images, containers, networks, and volumes. A daemon can also communicate with other daemons to manage Docker services.
- **Docker Object:**
  - **Containers:** Runnable Instance, Isolated from other containers
  - **Images:** Read-Only File, Defines an Application
- **Docker Registry:** A Docker registry stores Docker images. Docker Hub is a public registry that anyone can use, and Docker looks for images on Docker Hub by default. You can even run your own private registry.

Benefits

- Less resource consumption than OS Virtualization
- Isolation of Applications
- Fast deployment
- Perfect for testing purposes
- Containers can be restarted

## Function as a Service (FaaS)

Benefits

- Event-driven
- Scalable
- Fast deployment of code
- Payment per invocation

Popular FaaS: AWS Lambda, Google Cloud Functions, IBM Cloud Functions, Apache OpenWish,...

Generic Architecture

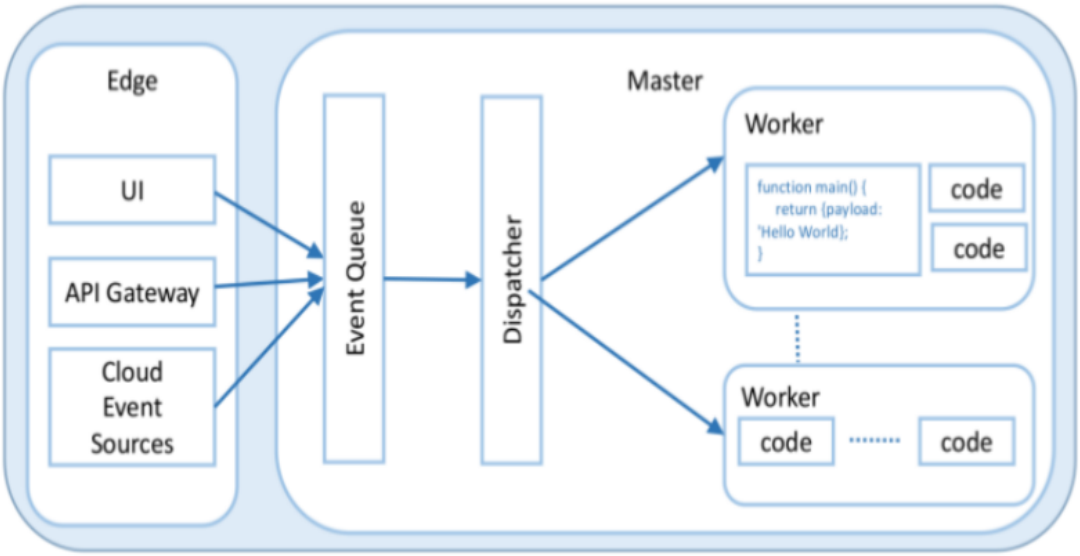



Figure: Generic FaaS Architecture [9]

- **Edge:** An UI for the management of functions, The general API for the implemented functions
- **Event Queue/Dispatcher:** Manages the triggered Events, Manages the scaling of invocations
- **Worker:** Execute the function invocations

## OpenFaaS

Architecture

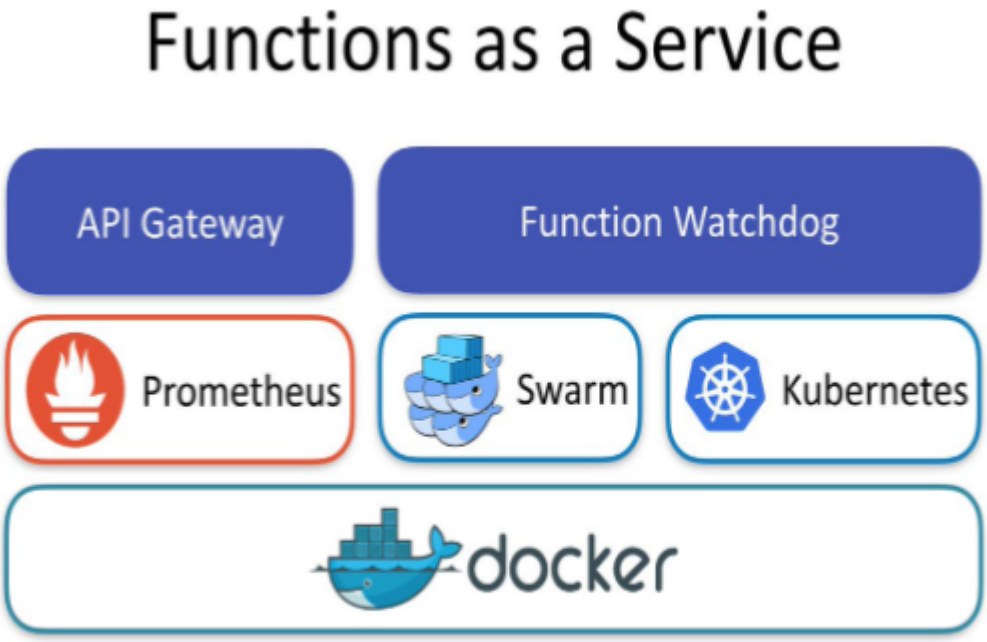



Figure: OpenFaaS Architecture [11]

- **API Gateway:** Provides a Route to the functions, UI for the management of functions, Scales functions through Docker
- **Function Watchdog:** Functions are added as Docker Images, Entrypoint for HTTP Requests
- **Prometheus:** Collects Metrics, Function Metrics can be inspected, Can be accessed through Web-UI
- **Docker:** Isolates Functions in Docker Images, Docker Swarm distributes functions, Kubernetes can be used to orchestrate Docker Instances

Benefits:

- Open Source
- Low resource consumption
- Deployment of functions
- Autoscaling
- Build in Monitoring and Metrics (Prometheus)

## Apache OpenWhisk

Architecture

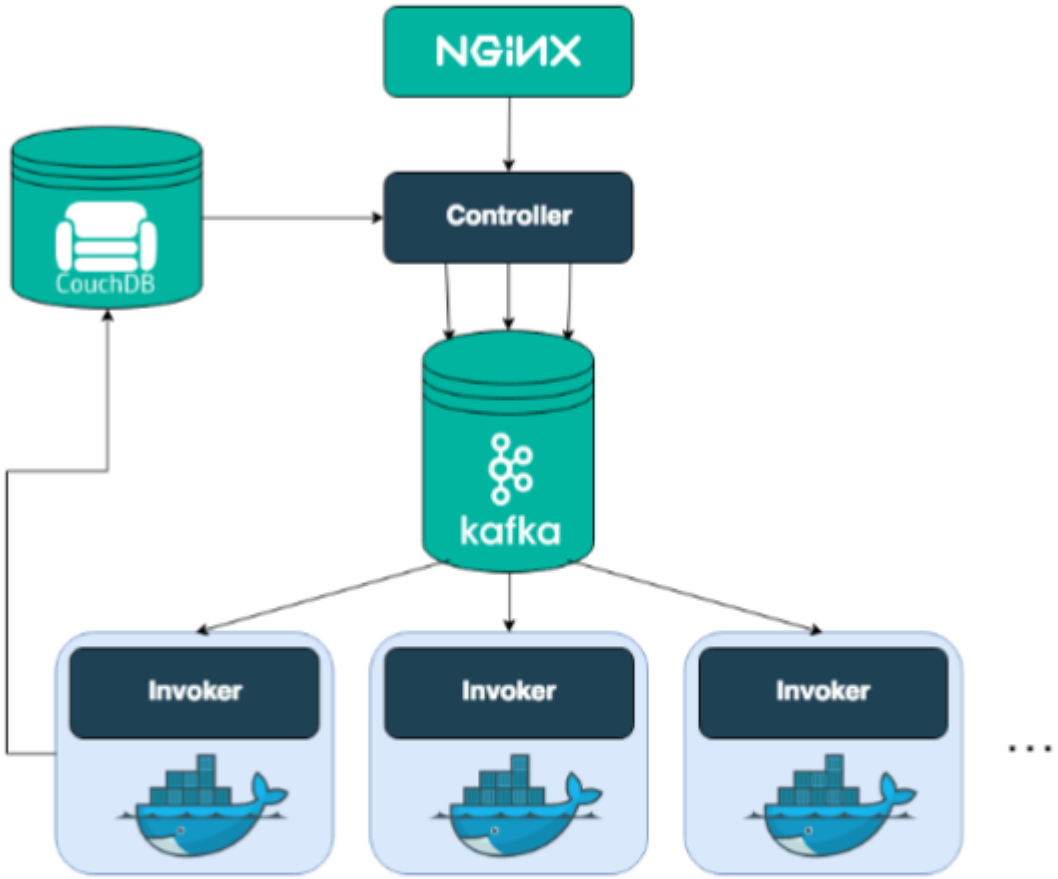



Figure: OpenWhisk Architecture

Source:
https://tinyurl.com/y7plrxbw

- **Nginx:** Loadbalancer for incoming requests, Forwarding requests to the controller
- **Controller:** Checks incoming requests, Controls the further action
- **Kafka:** Publish-Subscribe Messaging Service, Queues the requests
- **CouchDB:** Authentication of requests (permission checking), Stores information on the imported Functions
- **Invoker:** Docker Container(s) running the Function, Each Invoker can be paused for faster request fullfilment

Benefits

- Open Source Platform
- Functions can be deployed in a production ready environment (Apache)

## Week 5 + 6: Cloud Computing, Services and Concepts

## Week 7 + 8: Infrastructure Services (IaaS), Platform Services (PaaS)

## Week 9 - 10: Hadoop - MapReduce - HBase