Wrocław University of Science and Technology

# Large Scale Data Processing

## Lecture 3 – Data processing stack

dr inż. Tomasz Kajdanowicz, Roman Bartusiak, Piotr Bielak

November 25, 2019

HR EXCELLENCE IN RESEARCH

# Overview

- Concepts
  - Stateless vs Stateful application
  - High Availability
  - Continuous X
  - Software Updates
- Resource orchestration platforms
  - Hadoop
  - YARN
  - Zookeeper
  - Swarm
  - Kubernetes
  - Openstack
- Useful tools
  - Helm
  - Ansible

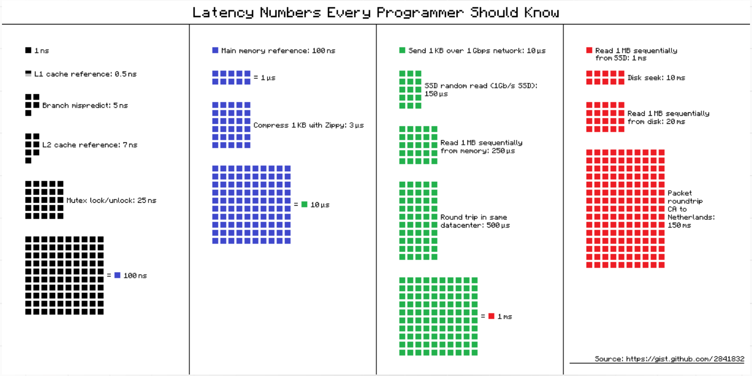
# Overview

- Concepts
  - Stateless vs Stateful application
  - High Availability
  - Continuous X
  - Software Updates
- Resource orchestration platforms
  - Hadoop
  - YARN
  - Zookeeper
  - Swarm
  - Kubernetes
  - Openstack
- Useful tools
  - Helm
  - Ansible

# Latency times

## Concepts

https://bit.ly/34jmK47



# Latency times

## Concepts

- L1 cache reference =0.5 ns
- Branch mispredict = 5 ns
- L2 cache reference = 7 ns
- Mutex lock/unlock = 25 ns
- Main memory reference = 100 ns
- Compress 1K bytes with Zippy = 3,000 ns = 3 µs
- Send 2K bytes over 1 Gbps network = 20,000 ns = 20 µs
- SSD random read = 150,000 ns = 150 µs
- Read 1 MB sequentially from memory = 250,000 ns = 250 µs
- Round trip within same datacenter = 500,000 ns = 0.5 ms
- Read 1 MB sequentially from SSD = 1,000,000 ns = 1 ms
- Disk seek = 10,000,000 ns = 10 ms
- Read 1 MB sequentially from disk = 20,000,000 ns = 20 ms
- Send packet CA->Netherlands->CA = 150,000,000 ns = 150 ms

# Stateless vs Stateful applications

## Concepts

- How to design apps?
- How to store user sessions?
- How to store user data?

# Stateless (1)

## Concepts (Stateless vs Stateful application)

- Application do not persist any data in their memory
- Requests are processed one by one
- Requests are processed independently (without context of previous requests)
- No session persisted in the app memory

# Stateless (2)

## Concepts (Stateless vs Stateful application)

- Session can be persisted in persistence solution
  - Accessible by other replicas of app
  - For example Redis or MongoDB
  - Can be scaled and replicated separately
- There is no need of sticky session usage
- App can be easily scaled horizontally
- Development process might be more complex

# Stateful (1)

## Concepts (Stateless vs Stateful application)

- Application can persist data in memory
- Requests can be processed in context of previous requests
- Session is persisted in app memory

# Stateful (2)

## Concepts (Stateless vs Stateful application)

- We need to use sticky session
- Scaling and availability is not so trivial
- Development might be easier

# High Availability

## Concepts (High Availability)

- characteristic of production level systems
- should be online (available) as much as possible
- SLA (Service Level Agreement) - *"nines"*

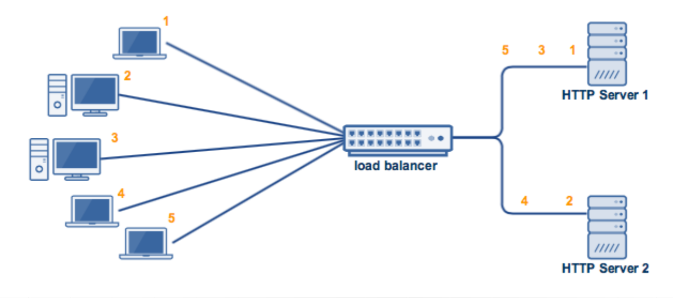
| Availability % | Downtime per year[note 1] | Downtime per month | Downtime per week | Downtime per day |
|---|---|---|---|---|
| 55.5555555% ("nine fives") | 162.33 days | 13.53 days | 74.92 hours | 10.67 hours |
| **90% ("one nine")** | **36.53 days** | **73.05 hours** | **16.80 hours** | **2.40 hours** |
| 95% ("one nine five") | 18.26 days | 36.53 hours | 8.40 hours | 1.20 hours |
| 97% | 10.96 days | 21.92 hours | 5.04 hours | 43.20 minutes |
| 98% | 7.31 days | 14.61 hours | 3.36 hours | 28.80 minutes |
| **99% ("two nines")** | **3.65 days** | **7.31 hours** | **1.68 hours** | **14.40 minutes** |
| 99.5% ("two nines five") | 1.83 days | 3.65 hours | 50.40 minutes | 7.20 minutes |
| 99.8% | 17.53 hours | 87.66 minutes | 20.16 minutes | 2.88 minutes |
| **99.9% ("three nines")** | **8.77 hours** | **43.83 minutes** | **10.08 minutes** | **1.44 minutes** |
| 99.95% ("three nines five") | 4.38 hours | 21.92 minutes | 5.04 minutes | 43.20 seconds |
| **99.99% ("four nines")** | **52.60 minutes** | **4.38 minutes** | **1.01 minutes** | **8.64 seconds** |
| 99.995% ("four nines five") | 26.30 minutes | 2.19 minutes | 30.24 seconds | 4.32 seconds |
| **99.999% ("five nines")** | **5.26 minutes** | **26.30 seconds** | **6.05 seconds** | **864.00 milliseconds** |
| **99.9999% ("six nines")** | **31.56 seconds** | **2.63 seconds** | **604.80 milliseconds** | **86.40 milliseconds** |
| **99.99999% ("seven nines")** | **3.16 seconds** | **262.98 milliseconds** | **60.48 milliseconds** | **8.64 milliseconds** |
| **99.999999% ("eight nines")** | **315.58 milliseconds** | **26.30 milliseconds** | **6.05 milliseconds** | **864.00 microseconds** |
| **99.9999999% ("nine nines")** | **31.56 milliseconds** | **2.63 milliseconds** | **604.80 microseconds** | **86.40 microseconds** |

# HA idea

## Concepts (High Availability)

How to achieve HA?

- replicate your service
- route all the traffic through a proxy / load balancer



Wrocław University of Science and Technology

# HA in real systems

## Concepts (High Availability)

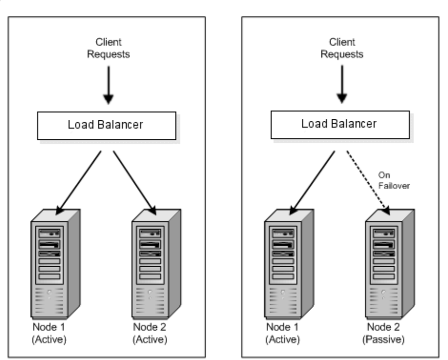
But in reality it's not that easy:

- implementation of health checks, liveness probes,
- retry mechanisms,
- how to automatically scale services?
- choose scaling criterion, number of fallback replicas,
- ensure fast application startup,

# Active-active vs active-passive

## Concepts (High Availability)

- two types of HA: **active-active**, **active-passive**
- for statefull application:
  - active-active is hard to maintain $\rightarrow$ the state of all instances must be synchronized along with processing of requests,
  - active-passive requires also state synchronization (easier, only one node receives requests),
- stateless applications - both HA types are easier to apply,



# Continuous X

## Concepts

- How to cooperate within teams?
- What we can do to make sure our software works and has high quality
- How to find out when something was broken?
- What if two people change dependant functionalities?

# Continuous Integration (1)

## Concepts (Continuous X)

Term comes from Extreme Programming by Kent Beck

- testing your code is important!,
- different kind of tests:
  - unit,
  - integration,
  - functional,
  - regression,
  - code quality,
  - security / bugs tests

# Continuous Integration (2)

## Concepts (Continuous X)

- manual vs automated tests,
- manual:
  - not perfect solution, but still used
  - e.g. done by testers to prepare automated tests
  - acceptance tests on client side, etc.
- automated:
  - utilize testing frameworks (e.g. in Python: unittest, pytest, flake8, in Scala: ScalaTest, Play),
  - should handle creation of proper testing environments (reproducible builds),
  - can be ran on developers machine,
  - after a developer pushes code to remote repository, the CI systems should build the code and run all tests against it (CI pipeline)

# Continuous Deployment (1)

## Concepts (Continuous X)

- What if continuous integration is not enough for you?
- What if you want to deploy your software as fast as possible after feature development?
- How to perform multiple deployments daily? (e.g. Facebook performs thousands of deployments per day)
- You need to be sure that your production environment is as fresh as it can be?

# Continuous Deployment (2)

## Concepts (Continuous X)

- Deploy your software as part of your build pipeline (to production or production-like environment)
- Create push-button deployment possibilities
- Automate as much as you can
- CI do not guarantee that your software will be deploy-able, CD does

# Continuous Deployment (3)

## Concepts (Continuous X)

- Create continuous integration pipeline with all its requirements
- Extend it to contain executable creation
- Create automation procedure that allows you to deploy your app/services
- Utilize appropriate updates procedure (canary etc.)
- Add deployment as part of build pipeline, or create special build for that

# Automation servers

## Concepts (Continuous X)

Software development automation servers

- **Jenkins**
- **GitLab CI**
- TeamCity
- **Zuul**
- Travis
- CircleCI

# Software Updates

## Concepts

- How to perform app updates without downtime?
- How to perform app updates to part of users
- How to make sure that app will be working correctly before full roll-out

# Software Update Types

## Concepts (Software Updates)

- Blue-Green
- Canary
- Rolling

# Data schema updates (1)

## Concepts (Software Updates)

- What about persistence?
- You need to separate schema from app upgrades
- On schema changes, you might need to do it in multi-phases

# Data schema updates (2)

## Concepts (Software Updates)

Field removal:

- Schema and app requires some field
- Change schema and app to make field optional
- Make upgrade
- Shut down app version that requires that field
- Change schema and app to make field absent

# Blue-green (1)

## Concepts (Software Updates)

- Load-balancer of proxy in front of app
- Two same environments
  - Green - One environment currently running
  - Blue - One that is updated
- Redirect traffic from green to blue after update
- All ok - blue becomes green, green becomes blue
- Problem? - redirect traffic back

# Blue-green (2)

## Concepts (Software Updates)

Update procedure

- Update blue environment
- Redirect traffic from green to blue
- All OK - blue becomes green, green can be again used as blue
- Problem? - redirect traffic back

Wroclaw University of Science and Technology

# Canary (1)

## Concepts (Software Updates)

- Roll-out updates only to small part of traffic
- You can roll-out only to part of your users
- At Facebook:
  - Gatekeeper
  - Roll-out using information about users
  - By age
  - By gender
  - etc.

# Canary (2)

## Concepts (Software Updates)

Update procedure

- Create app instance with new features
- Redirect part of traffic using some predicate to new instance
- All OK? - continue redirecting the traffic until 100% on new app
- Problem - roll-back to the old instances

Wrocław University of Science and Technology

# Rolling (1)

## Concepts (Software Updates)

- $N$ - number of running instances before update
- During the update, at most $N+1$ running instances
- At each step of update, replace one old version instance with new one

# Rolling (2)

## Concepts (Software Updates)

Update procedure

- Create new version instance
- All OK - Turn off one old version instance
- All OK - Repeat until you have only new version instances
- Problem? - Recreate old instances

# Overview

- Concepts
  - Stateless vs Stateful application
  - High Availability
  - Continuous X
  - Software Updates
- Resource orchestration platforms
  - Hadoop
  - YARN
  - Zookeeper
  - Swarm
  - Kubernetes
  - Openstack
- Useful tools
  - Helm
  - Ansible

Wrocław University of Science and Technology

# Overview

## Resource orchestration platforms (Hadoop)

- framework for processing big data
- not a brand name
- open source project
- effective for analytics, sometimes operations

| | **Operational** | **Analytical** |
|---|---|---|
| Latency | 1 ms - 100 ms | 1 min - 100 min |
| Concurrency | 1000 - 100,000 | 1 - 10 |
| Access Pattern | Writes and Reads | Reads |
| Queries | Selective | Unselective |
| Data Scope | Operational | Retrospective |
| End User | Customer | Data Scientist |
| Technology | NoSQL | MapReduce, BSP |

# Introduction

## Resource orchestration platforms (Hadoop)

Hadoop's core:

- HDFS
- MapReduce
- YARN
- Common

# HDFS Overview

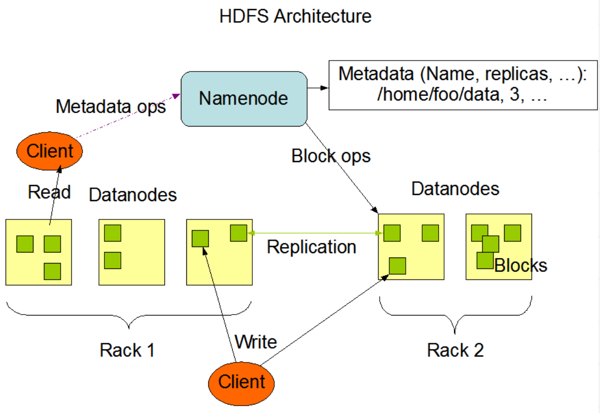
## Resource orchestration platforms (Hadoop)

- Hadoop Distributed File System
- heart of Hadoop technology
- manages how data files are divided and stored across the cluster
- data is divided into blocks; each server in the cluster contains data from different blocks
- some built-in redundancy
- provides file permissions and authentication
- has a master/slave architecture

# HDFS Architecture

HDFS cluster consists of:

- single NameNode - master server that manages the file system namespace and regulates access to files
- a number of DataNodes (usually one per node in the cluster) - manage storage attached to the nodes



# HDFS Operations

## Resource orchestration platforms (Hadoop)

The NameNode:

- executes file system namespace
- opening, closing, and renaming files and directories
- mapping of blocks to DataNodes

The DataNodes are responsible for:

- serving read and write requests from the clients
- perform block creation, deletion (NameNode instructions)

**Block**

- user data is stored in the files of HDFS; the file in a file system is divided into one or more segments and stored in individual data nodes
- file segments are called as blocks
- the minimum amount of data that HDFS can read or write is called a Block
- default block size is 64MB

# MapReduce

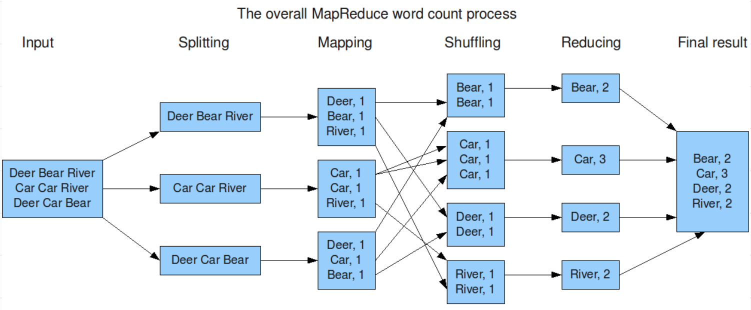
## Resource orchestration platforms (Hadoop)

MapReduce - a method for parallel processing on distributed servers

Stages of MapReduce:

1. map - process the input data from HDFS; input file is passed to the mapper function line by line; creates several small chunks of data
2. shuffle - transfers the map output from Mapper to a Reducer
3. reduce - takes the output of the mapper (intermediate key-value pair); output of the reducer is the final output in HDFS; usually aggregation or summation

# MapReduce example

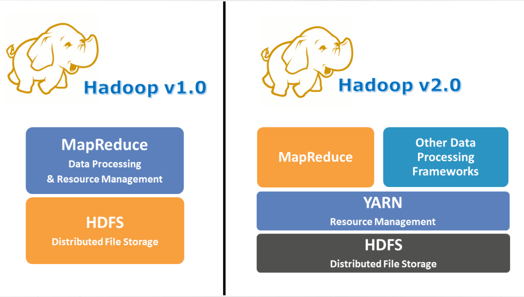



1. the input is first splitting into blocks
2. the splitted input is assined to [key,value] pair in mapper side
3. all the values are ordered in shuffling phase
4. reducer side sorting and grouping is done

# Hadoop YARN

## Resource orchestration platforms (YARN)

- not to be confused with Javascript's yarn package manager
- handles processing part in Hadoop
- YARN = Yet Another Resource Negotiator
- assigns resources to each application



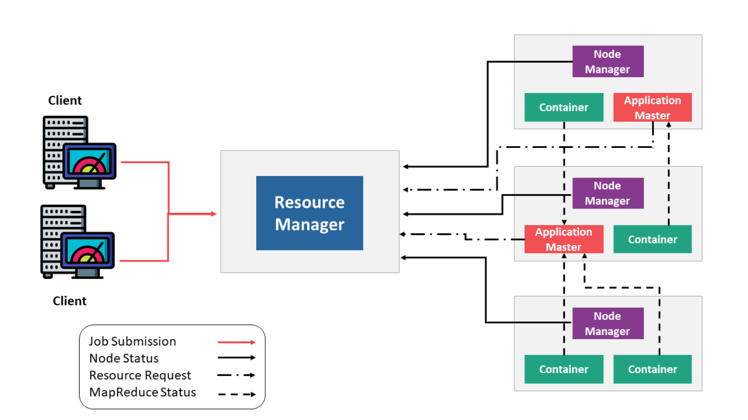
# YARN components

## Resource orchestration platforms (YARN)

- **Resource Manager**
  - communicates with the client;
  - tracks resources on the cluster;
  - coordinates node manager by assigning job according to requirement;
  - allocates memory in form of containers;
- **Node Manager**
  - monitoring of running app;
  - offers resources (memory and CPUs) as resource container;
  - every datanode $\rightarrow$ one node manager;
  - containers run tasks (including Application Masters);
- **Application Master**
  - every job is initialized by Application manager;
  - manages the global assignment of compute resources to applications (e.g. memory, CPU, disk, network, etc.)

# YARN diagram

## Resource orchestration platforms (YARN)



# Zookeeper

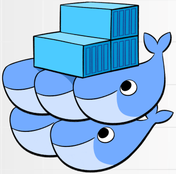
## Resource orchestration platforms (Zookeeper)

- not a resource orchestration tool
- often used along with Hadoop, HBase, Hive, Kafka etc.
- centralized service for:
  - maintaining configuration information,
  - naming,
  - providing distributed synchronization,
  - providing group services.

# Docker Swarm

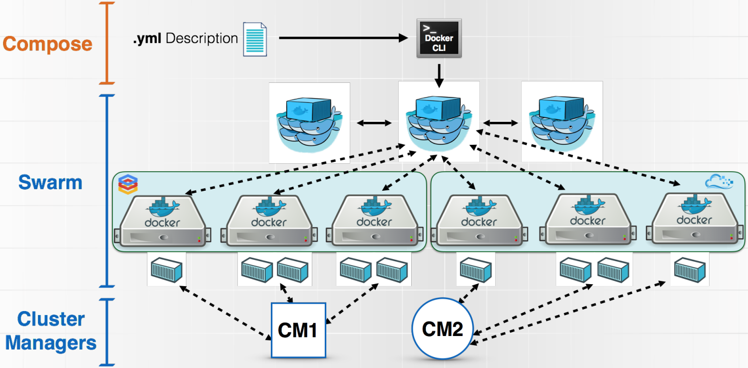
## Resource orchestration platforms (Swarm)

- container orchestration system created by Docker;
- allows to distribute Docker containers across multiple nodes;
- introduces *stacks* and services;
- resource reservations and limits;
- container scaling;
- rolling updates;
- does not provide advanced networking (overlay);
- very basic compared to Kubernetes;



# Docker Swarm

## Resource orchestration platforms (Swarm)



# Kubernetes

## Resource orchestration platforms (Kubernetes)

- Google
- Based on huge experience on handling distributed environments
- Open-source
- huge community

# Kubernetes

## Resource orchestration platforms (Kubernetes)

- Not so easy to deploy
- Many dependant services
- K3S
  - a simpler and easier in deployment and management
  - use SQLite, for big traffic might be a bottleneck
- For local development
  - Minikube
  - microk8s

# Architecture

## Resource orchestration platforms (Kubernetes)

Kubernetes has two types of nodes (each running multiple microservices):

- master (control plane) nodes:
  - API server,
  - Scheduler,
  - Controller Manager,
  - ETCD,
- worker nodes:
  - Kubelet,
  - Kube-proxy,
  - *Container runtime**

# Master nodes components (1)



https://kubernetes.io/docs/concepts/overview/components/

- API server (*kube-apiserver*):
  - exposes the Kubernetes API,
  - front end for the control plane,
  - designed to scale horizontally (via load balancers),
- Scheduler (*kube-scheduler*):
  - watches newly created pods that have no node assigned,
  - selects a node for them to run on,
  - individual and collective resource requirements, hardware/software/policy constraints, affinity and anti-affinity specifications, data locality, inter-workload interference and deadlines,

# Master nodes components (2)

## Resource orchestration platforms (Kubernetes)

https://kubernetes.io/docs/concepts/overview/components/

- CM (*kube-controller-manager*) – runs controllers:
  - 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,
- ETCD:
  - consistent and highly-available key value store,
  - used as Kubernetes' backing store for all cluster data,
  - can be run in standalone or cluster mode,

# Worker nodes components

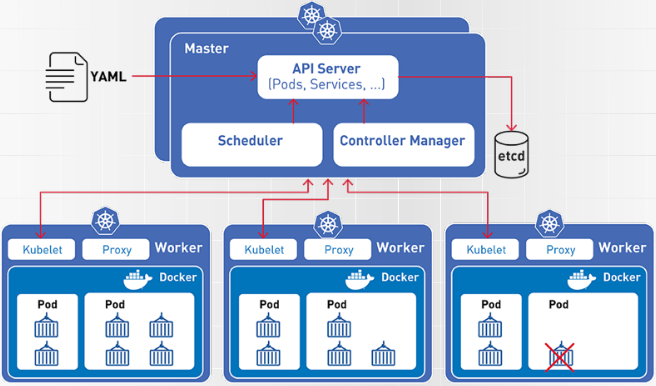
## Resource orchestration platforms (Kubernetes)

https://kubernetes.io/docs/concepts/overview/components/

- kubelet:
  - daemon running on each worker node,
  - makes sure that containers are running in a pod,
  - manages only containers created by Kubernetes,
- kube-proxy:
  - network proxy running on each worker node,
  - maintains network rules on nodes (ingress / egress),
  - utilizes OS packet filtering layer (if available)
- Container runtime – supported ones:
  - Docker, containerd,
  - cri-o, rktlet,
  - any implementation of the Kubernetes CRI (Container Runtime Interface).

# Architecture diagram

## Resource orchestration platforms (Kubernetes)



# Kubernetes objects

## Resource orchestration platforms (Kubernetes)

kubernetes defines multiple objects that an application can make use of:

- deployment
- service
- daemonset
- statefulset
- configmap
- secret
- persistentvolume(claim)

# Kubernetes manifests

## Resource orchestration platforms (Kubernetes)

Kubernetes object are created using YAML manifests:

```yaml
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
spec:
  selector:
    matchLabels:
      app: nginx
  replicas: 1
  template:
    metadata:
      labels:
        app: nginx
    spec:
      volumes:
        - name: nginx-sample-page
          configMap:
            name: nginx-sample-page
      containers:
      - name: nginx
        image: nginx:latest
        ports:
          - containerPort: 80
        volumeMounts:
          - name: nginx-sample-page
            mountPath: /usr/share/nginx/html/index.html
            subPath: index.html
```

# Kubernetes management



- most cluster actions are performed using `kubectl`,
- it allows to manage the cluster, as well as deploy application manifests,
- alternatively, for deployment you can use `helm` (we'll discuss it later),

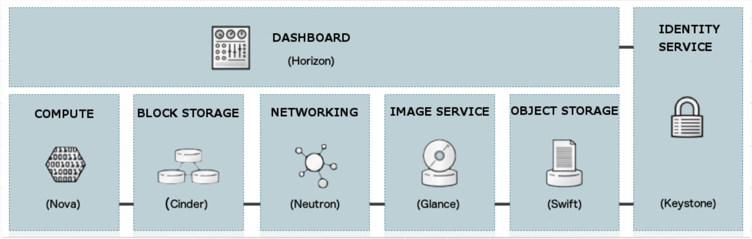
# Openstack

## Resource orchestration platforms (Openstack)

- most popular open source virtual machine orchestration platform;
- operates on VMs instead on containers;
- microservice based architecture;
- multiple specialized services;

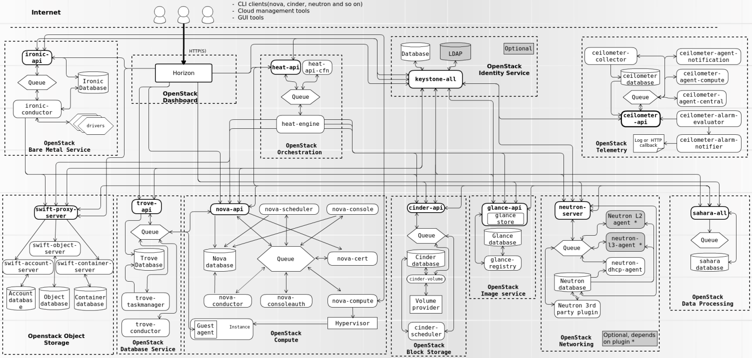
# Architecture (1)

## Resource orchestration platforms (Openstack)



# Architecture (2)

## Resource orchestration platforms (Openstack)



# Services

## Resource orchestration platforms (Openstack)

- **NOVA** (Compute)
- ZUN (Containers)
- **QINLING** (Functions)
- **IRONIC** (Bare Metal Provisioning)
- **CYBORG** (Hardware accelerators)
- SWIFT (Object store)
- **CINDER** (Block Storage)
- MANILA (Shared filesystems)
- **NEUTRON** (Networking)
- OCTAVIA (Load balancer)
- DESIGNATE (DNS)
- **KEYSTONE** (Identity)
- **PLACEMENT** (Placement)
- **GLANCE** (Image)
- HEAT (Orchestration)
- **MISTRAL** (Workflow)
- AODH (Alarming)
- **MAGNUM** (Container Orchestration Engine Provisioning)
- **SAHARA** (Big Data Processing Framework Provisioning)
- TROVE (Database as a Service)
- **MASAKARI** (Instances High Availability Service)
- MURANO (Application Catalog)
- **EC2API** (EC2 API proxy)
- HORIZON (Dashboard)
- and many more...

# Overview

- Concepts
  - Stateless vs Stateful application
  - High Availability
  - Continuous X
  - Software Updates
- Resource orchestration platforms
  - Hadoop
  - YARN
  - Zookeeper
  - Swarm
  - Kubernetes
  - Openstack
- Useful tools
  - Helm
  - Ansible

# Helm

## Useful tools

- How to manage Kubernates deployments?
- There are not variables in Kubernates manifests
- What to do if we want to rollback to previous version

# Helm

## Useful tools

- Package manager for K8S
- OpenSource, maintained by Google, Microsoft, Bitnami, ...
- Many ready to use charts in official repository
- Possibility to use variables during deployment
- Charts can have dependencies

# Helm

## Useful tools

- Easily find packages
- Easily create new packages
- Can operate on any K8S cluster
- Query the cluster to see installed packages
- Update, delete, rollback and check history of installed charts

# Helm

## Useful tools

- Basically it templates K8S manifests (Go templates)
- Thanks to that, it is independent from resources that you want to create
- Create resource YAML, utilize {{ }} to inject variables
- Variables can be provided as a file or during the deployment

# Helm

## Useful tools

Helm v2

- requires Tiler
  - server-side (on cluster) component
  - manages packages
- 2-way strategic merge path
- releases names are global
- release name is optional, if not provided will be generated

Helm v3

- no Tiler
- 3-way strategic merge path
- releases names are in namespaces
- release name is required
- library charts

# Directory structure

## Useful tools (Helm)

```
wordpress/
  Chart.yaml          # A YAML file containing information
                      #  about the chart
  LICENSE             # OPTIONAL: A plain text file containing
                      #  the license for the chart
  README.md           # OPTIONAL: A human-readable README file
  values.yaml         # The default configuration values
                      #  for this chart
  values.schema.json  # OPTIONAL: A JSON Schema for imposing
                      #  a structure on the values.yaml file
  charts/             # A directory containing any charts upon
                      #  which this chart depends.
  crds/               # Custom Resource Definitions
  templates/          # A directory of templates that,
                      #  when combined with values, will
                      #  generate valid Kubernetes manifest files.
  templates/NOTES.txt # OPTIONAL: A plain text file containing
                      #  short usage notes
```

# Example template

## Useful tools (Helm)

```yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: {{ .Release.Name }}-configmap
data:
  myvalue: "Hello World"
```

# Example template

## Useful tools (Helm)

```yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: {{ .Release.Name }}-configmap
data:
  myvalue: "Hello World"
  drink: {{ .Values.favoriteDrink }}
```

```
helm install --set favoriteDrink=monsterek ./mychart
```

# Helm

## Useful tools

- How to check if your template will work?
- dry-run will render template to STD
- you need to check it manually
- rendered template can be not accepted by kubernetes

# Introduction

## Useful tools (Ansible)

- open source automation tool,
- machine provision,
- configuration management,
- application deployment,
- YAML-based declarative language,
- agentless (uses SSH / Powershell),

ANSIBLE

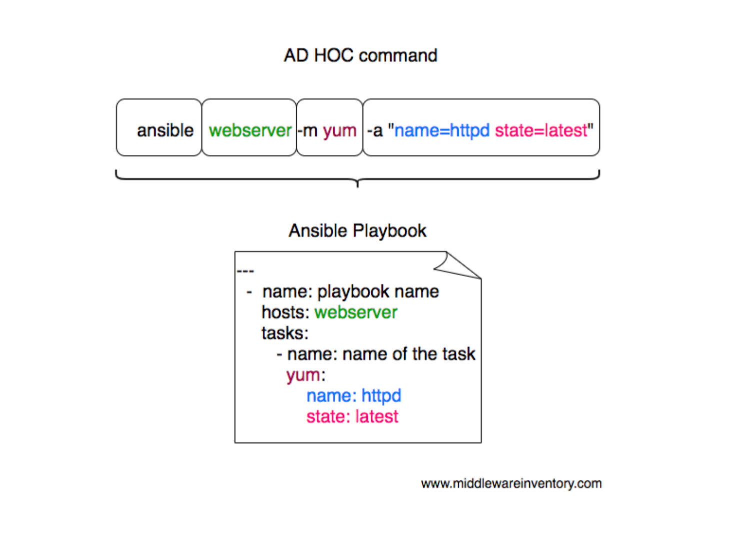
# Characteristics

## Useful tools (Ansible)

- simple & minimalistic (YAML-based language & Jinja templates),
- consistency (of created enviroments),
- security (no dedicated agent, SSH used for connections),
- reliability (**idempotent** playbooks),

# Command vs playbook

## Useful tools (Ansible)



# Example playbook

## Useful tools (Ansible)

```yaml
---
- name: Install nginx
  hosts: all
  become: true

  tasks:
  - name: Add epel-release repo
    yum:
      name: epel-release
      state: present

  - name: Install nginx
    yum:
      name: nginx
      state: present

  - name: Insert Index Page
    template:
      src: index.html
      dest: /usr/share/nginx/html/index.html

  - name: Start NGiNX
    service:
      name: nginx
      state: started
```

# Concepts (1)

## Useful tools (Ansible)

- playbooks:
  - define steps to build environments,
  - can be divided into multiple files (readability, reusability),
  - roles, vars, group_vars etc.
- modules:
  - define actual actions executed by Ansible,
  - examples given on previous slide,
  - standalone,
  - can be written in most scripting languages (Python, Bash, Perl, Ruby etc.),
  - should follow **idempotent** rule,

# Concepts (2)

## Useful tools (Ansible)

- inventory file:
  - description of nodes that can be accessed by Ansible,
  - INI or YAML format,
  - IP addresses or hostnames,
  - when necessary SSH keys and users can be provided,
  - nodes can be assigned to groups,

# Inventory file example

## Useful tools (Ansible)

```
# Consolidation of all groups
[hosts:children]
web-servers
offsite
onsite
backup-servers

[web-servers]
server1 ansible_host=192.168.0.1 ansible_port=1600
server2 ansible_host=192.168.0.2 ansible_port=1800

[offsite]
server3 ansible_host=10.160.40.1 ansible_port=22 ansible_user=root
192.168.6.1

# You can make groups of groups
[offsite:children]
backup-servers

[onsite]
server5 ansible_host=10.150.70.1 ansible_ssh_pass=password

[backup-servers]
foo.example.com
```

# Large Scale Data Processing

## Lecture 3 – Data processing stack

dr inż. Tomasz Kajdanowicz, Roman Bartusiak, Piotr Bielak

November 25, 2019