Cloud, Semantic Web

PA160, spring 2022

Martin Kuba, ÚVT MU
makub@ics.muni.cz

Outline

- Scalability
- Cloud Computing
  - IaaS
  - PaaS
  - SaaS
- Deploying apps to cloud
  - Containers, Docker, Kubernetes
  - Infrastructure as Code, Ansible
  - Continuous Integration / Continuous Delivery
- Semantic Web
  - expressing semantics
  - RDF, OWL, ontologies
  - Semantic Web Services

Load spike problems

- on 9/11 2001 news sites like cnn.com or bbc.com could not handle the load
- recent debacles of Czech e-government
  - highway e-vignetes (2×10^6 users)
  - on-line census (10^7 users)
- for scale:
  - Facebook has 1.85×10^9 daily users
  - 70,000 Google searches per second
  - seznam.cz has 2.5×10^6 users monthly, 6×10^5 daily

Scalability

- property of a system to handle a growing amount of work by adding resources to the system
- scaling **vertically** means adding resources to a single computer (CPUs, memory)
- scaling **horizontally** means adding more computers to a distributed software application

How to scale

- make **modular** and **loosely-coupled** systems
- use **caches** wherever possible
- use **CDN** - Content Delivery Network - for static files like images, videos, CSS, JS
  - routes traffic to the nearest or fastest datacenter
- use **load balancers** between clients and servers
- for database scaling use **replication** and/or **sharding** (horizontal partitioning of tables)
- Google search used Map-Reduce
- automatically add or remove computers from a **computing cloud** (elasticity)


Cloud Computing

Cloud Computing

- use of computing resources (hardware and software) that are delivered as a service over a network
- in 1960's called "utility computing"
- the retailer Amazon was dealing with peak traffic during Christmas seasons
- in 2006 Amazon released AWS - **Amazon Web Services**
  - EC2 - Elastic Compute Cloud
  - S3 - Simple Storage Service

Cloud definition

- **cloud computing** is a general term for anything that involves delivering hosted services over the Internet
- definition by NIST (National Institute of Standards and Technology, U.S. Department of Commerce): *Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction.*
- five essential characteristics: on-demand self-service, broad network access, resource pooling, rapid elasticity and measured service

Cloud Service Models

- Software as a Service (SaaS)
- Platform as a Service (PaaS)
- Infrastructure as a Service (IaaS)

SaaS - Software as a Service

- best known to computer users, the only cloud they directly use, provides **device independence**
- on-demand access to software, either to downloadable code executed on client computers, or through remote API calls to code executed on servers
- examples:
  - web mail – GMail, Outlook.com, email.cz
  - maps - Google Maps, Google Earth, mapy.cz
  - file services – Dropbox, Google Drive, OneDrive, pCloud
  - video libraries – YouTube, Vimeo, Netflix
  - music libraries - Spotify, Apple Music, Amazon Music
  - office tools – Google Docs, Microsoft Office 365
  - communication tools – zoom.us, Google Meet, WebEx
  - business software – Salesforce, NetSuite

PaaS - Platform as a Service

- platform is a software environment used to develop and run applications
- not visible to end users, targeted to application developers and maintainers delivering their SaaS applications
- the environment includes concrete programming languages, their specific libraries, and additional services like SQL, noSQL, newSQL storage
- examples:
  - Amazon Elastic Beanstalk (Go, Java, Node. js, PHP, Python, Ruby)
  - Google App Engine (Go, Java, Node.js, PHP, Python)
  - Microsoft Azure Websites (Java, Node.js, PHP, Python, .NET)
  - Heroku (Go, Java, Node.js, PHP, Python, Ruby, Scala, Clojure)

IaaS - Infrastructure as a Service

- provides a virtual data center
- IaaS provider provides **virtual machines** (VMs) with complete operating systems
- many VMs can be hosted on a single physical machine running **hypervisor** software (Xen, KVM, VMware)
- resources hired from an IaaS cloud can be used directly (e.g. on-demand movie rendering) or as a layer under a PaaS or SaaS cloud

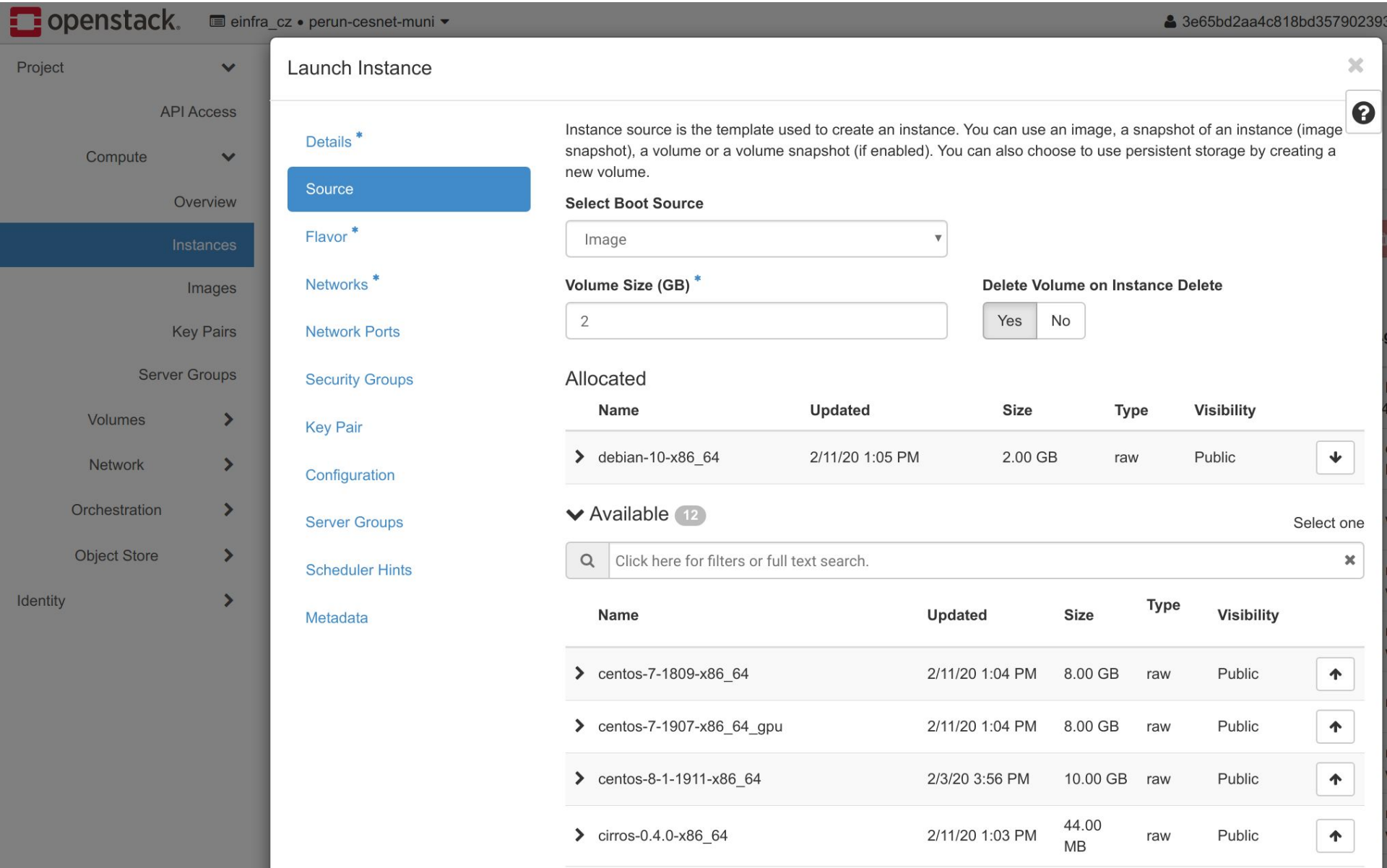
IaaS Providers Provide

- **disk images** with pre-installed popular operating systems (various versions of Linux, MS-Windows)
- **networking services -** virtual local area networks, virtual private networks, IP addresses, firewalls, load balancers, domain name service (DNS)
- **storage services** - virtual block storage, file storage, object storage, relational database storage, noSQL storage, tape archive storage, content delivery network (CDN)

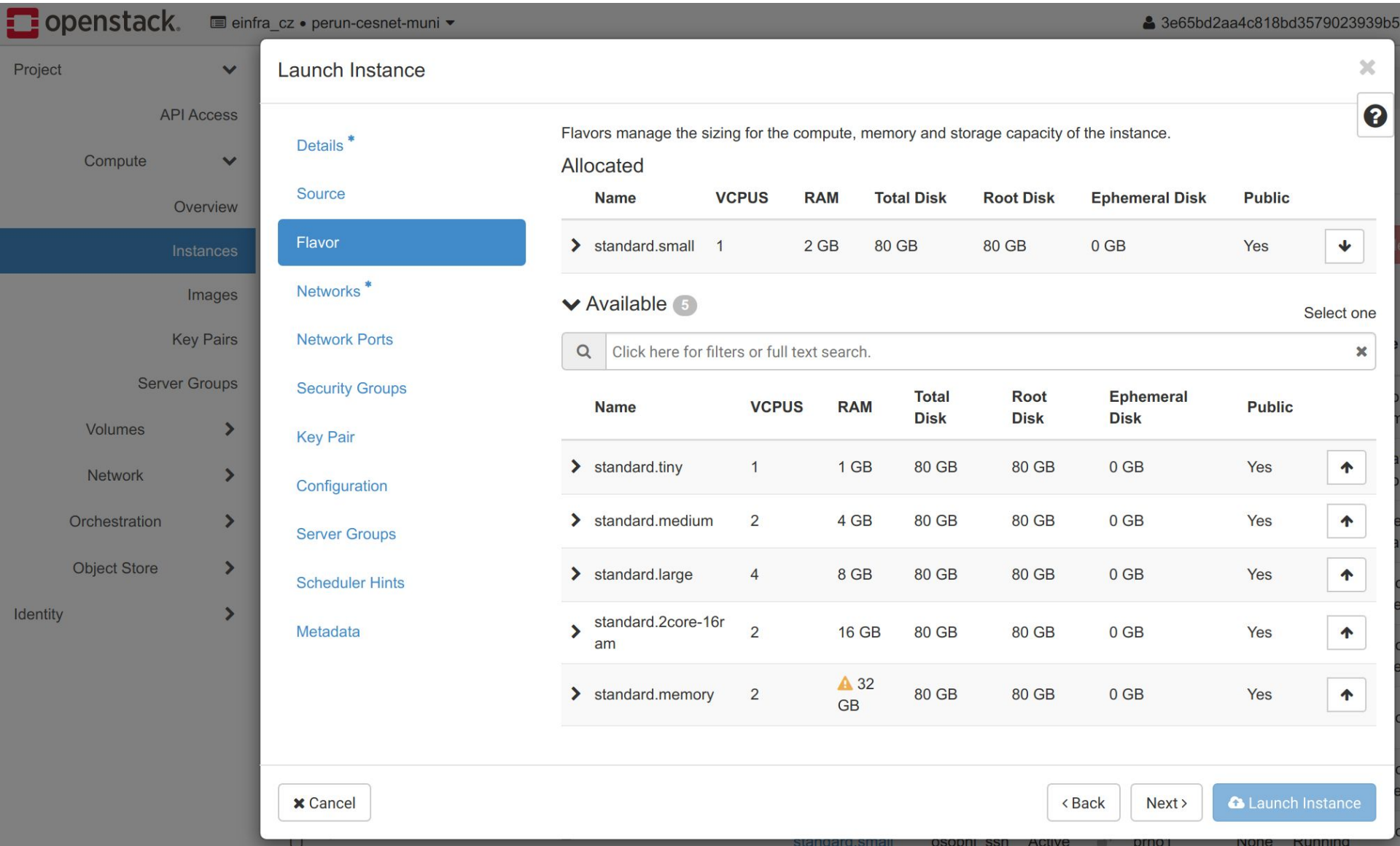
IaaS Examples

- providers:
  - Amazon Elastic Compute Cloud
  - Google Compute Engine
  - Microsoft Azure
  - IBM Cloud
  - Digital Ocean
- software:
  - OpenStack (used at https://cloud.muni.cz/)
  - VMware vSphere (used at https://vcenter.ics.muni.cz/)
  - OpenNebula (used at https://stratus.fi.muni.cz/)
  - Apache CloudStack
  - Eucalyptus

OpenStack - creating a new VM



OpenStack - creating a new VM (2)



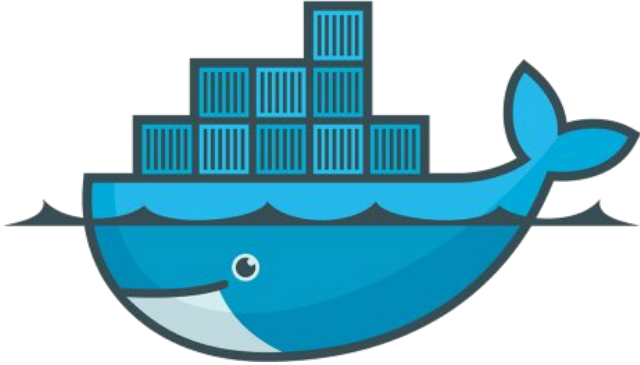
Cloud Service Models Summary

- **Software-as-a-Service** model provides on-demand access to software
- **Platform-as-a-Service** model provides on-demand software environment for deploying applications
- **Infrastructure-as-a-Service** model provides on-demand resources from a virtual data center

Serverless computing

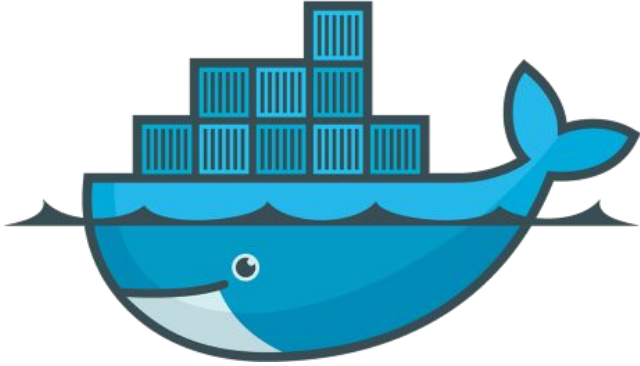
- also **Function-as-a-Service - FaaS**
- execute code as an event driven function without managing the underlying server resources
- FaaS extends PaaS with complete abstraction from the underlying infrastructure through a pay-per-execution billing model
- examples:
  - AWS Lambda (introduced 2014)
  - Google Cloud Functions (2016)
  - Microsoft Azure Functions (2016)
- hard to debug, strong vendor lock-in

Docker containers



- real Virtual Machines have some overhead
- Docker is a tool for deployment of software in so-called containers
- a **container** is an isolated environment with complete system libraries, running inside a hosting OS
- a container is in principle a chroot directory with cgroups and namespaces, with exportable directories and TCP ports for linking to other containers
- a container **image** can contain e.g. OS Ubuntu 20.04 with Apache web server, but can run on any Linux, MacOS X or Windows host

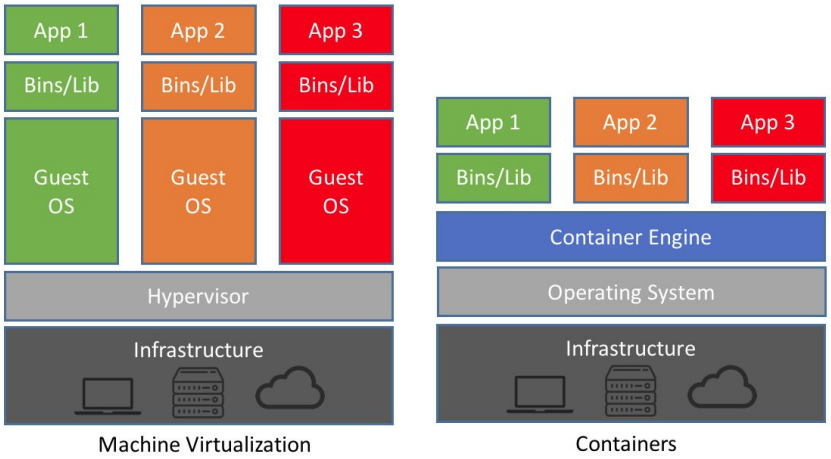
Docker containers (2)



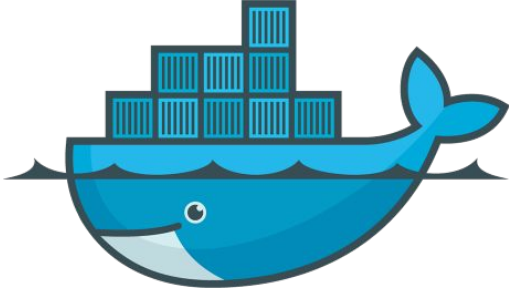
- Docker container images are like .deb or .rpm packages, but OS-independent
- anybody can create a new container image by modifying another container image
- running containers can be linked using TCP ports (e.g. Apache+PHP -> PostgreSQL)
- containers are ephemeral (short-lived), disposable
- containers do not store persistent data
- persistent data can be stored outside of containers using mounted directories (called *volumes*)

Containers vs. Virtual Machines

- each VM has its own kernel, containers share a kernel from the host
- a VM runs many processes (daemons, e.g. sshd, crond), a container only one process
- update of a container is done by replacing it, in VM by making permanent changes in it



Docker Container Registries



- DockerHub at https://hub.docker.com/
  - the default registry preconfigured in Docker
  - official containers for famous software (e.g. PostgreSQL, Apache, Ubuntu, Debian, OpenJDK,…)
  - image names without any prefix, e.g.
    - `postgres:14`
    - `httpd:2.4.52`
    - `ubuntu:22.04`
- GitHub, GitLab
  - each git repository has associated Docker registry
  - image names contain registry, e.g.
    - `docker.pkg.github.com/cesnet/perun/perun_rpc:3.9`
    - `registry.gitlab.ics.muni.cz:443/perun/perun_docker/perun_rpc:3.9`

Kubernetes

- container-orchestration system
- platform for automating deployment, scaling, and operations of application containers across **clusters of hosts**
- provisions both (stateless) containers and their persistent data
- released by Google in 2015

Pets and cattle

- machines can be treated in two ways
- **pets** - you are emotionally attached to them, because you have spent a lot of effort setting them up, you feel terrible when losing them, you give them names
- **cattle** - use them, then kill them when not needed; numbered, not named
- cloud machines are cattle

Infrastructure as Code (IaC)

- **managing computer data centers through machine-readable definition files**
- no interactive configuration tools
- necessary when managing hundreds of machines
- definitions may be in a version control system (e.g. git)
- tools like Ansible, Puppet, SaltStack, Chef
- declarative approach defines the desired state of machines

Push and Pull Configuration Strategies

- **Push** - configurations of machines are changed from outside after an event
  - used by Ansible
  - scales poorly to huge numbers of machines
- **Pull** - a machine downloads and applies a configuration periodically
  - Puppet downloads from a central server
  - ansible-pull downloads from a git repository
  - scales well
  - may be a problem when managing differing machines

Idempotent actions

- **idempotent -** can be run repeatedly with the same result as when run just once
- machine configuration actions should be idempotent
- if a machine already is in the desired state, nothing is changed
- e.g. adding a line to a file is not idempotent
- adding a line to a file only if it is not yet present is idempotent

Ansible

- configuration management tool written by RedHat in Python
- configuration is written in a YAML-formatted file called a **playbook**
- a playbook contains one or more **plays**
- a play declares
  - a list of managed machines (**hosts**)
  - the user making the changes (**remote_user**)
  - variables (**vars**)
  - executed actions (**tasks**)
  - handling of events (**handlers**) (e.g. service restarts)

Ansible - playbook example


```yaml
- name: "Ansible playbook for configuring an email address for the root account"
  hosts: all
  remote_user: root
  vars:
    root_email_address: "makub@ics.muni.cz"

  tasks:
    - name: "install mail transfer agent software"
      package:
        name: sendmail
        state: present

    - name: "set root mail alias"
      lineinfile:
        path: /etc/aliases
        regexp: "^root:"
        line: "root: {{ root_email_address }}"
      notify: "run newaliases"

  handlers:
    - name: "rebuild the data base for the mail aliases file"
      command: newaliases
      listen: "run newaliases"
```

Ansible - executed playbook


```
$ ansible-playbook -i cloud13.perun-aai.org,cloud14.perun-aai.org,cloud15.perun-aai.org playbook_ukazka.yml

PLAY [Ansible playbook for configuring an email address for the root account] ***********************************

TASK [Gathering Facts] ***********************************************************************************
ok: [cloud13.perun-aai.org]
ok: [cloud14.perun-aai.org]
ok: [cloud15.perun-aai.org]

TASK [install mail transfer agent software] **************************************************************
ok: [cloud13.perun-aai.org]
changed: [cloud14.perun-aai.org]
changed: [cloud15.perun-aai.org]

TASK [set root mail alias] *******************************************************************************
changed: [cloud15.perun-aai.org]
changed: [cloud14.perun-aai.org]
ok: [cloud13.perun-aai.org]

RUNNING HANDLER [rebuild the data base for the mail aliases file] ****************************************
changed: [cloud14.perun-aai.org]
changed: [cloud15.perun-aai.org]

PLAY RECAP ***********************************************************************************************
cloud13.perun-aai.org      : ok=3    changed=0    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
cloud14.perun-aai.org      : ok=4    changed=3    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
cloud15.perun-aai.org      : ok=4    changed=3    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
```

Ansible features

- highly customisable by extensions
- in addition to physical and virtual machines, it can also modify Docker containers
- can manage Docker servers
- can manage (create, destroy) virtual machines in a cloud (OpenStack, Amazon, Azure, Google, and many others)
- can manage networks and network services

CI/CD - Continuous Integration / Delivery / Deployment

- **CI** - the practice of frequently integrating new or changed code with the existing code repository
- used in combination with **automated tests**
- detects errors as quickly as possible, reduces integration problems
- builds triggered by every commit to a repository
- **C. Delivery** - changes are automatically bug tested and uploaded to a repository
- **C. Deployment** - automatically releasing a developer's changes from the repository to production

Famous CI/CD Tools

- **Travis CI** - since 2011, service for CI of projects hosted at GitHub, activated by file `.travis.yml`
- **GitLab CI** - since 2016, for own projects, most flexible, activated by file `.gitlab-ci.yml`
- **GitHub CI** - since 2019, for own projects, activated by files `.github/wokflows/*.yml`

GitLab CI/CD

- **CI/CD pipeline** consists of **stages** (build, test, deploy) which consist of **jobs**
- jobs are simple linux shell commands that return non-zero status when they fail
- a pipeline is run by a **GitLab Runner** inside a specified Docker container
- a GitLab Runner can run anywhere, on a dedicated server or on user's notebook
- built packages (e.g. .jar files) or containers are uploaded to a **registry**

Happy CI/CD scenario

1. a developer pushes a commit into a git repo
2. CI runs automated tests
3. CI builds a new version of a container with the software
4. CD uploads the container to registry
5. CD deploys the container into Kubernetes cluster
6. new version of the software is running

Hands-on experience:

- example of CI/CD of a web service https://github.com/martin-kuba/muni-pa160-chat-service
- example of Ansible deployment of the web service https://github.com/martin-kuba/muni-pa160-ansible


Semantic Web

Semantic Web

- idea introduced by Tim Berners Lee (inventor of WWW) in 2001
- *“The Semantic Web is not a separate Web but an extension of the current one, in which information is given well-defined meaning, better enabling computers and people to work in cooperation”*
- web instead of platform for distributed presentations would be platform for distributed knowledge

Semantics

- semantics = meaning
- semantic in SW means *machine-processable*
- semantic continuum (Uschold 2003)
  a. implicit semantics in the minds of humans
  b. explicit informal semantics (text description in natural language, e.g. HTML specification)
  c. formal semantics for humans (in formal language processed by humans, e.g. modal logic)
  d. formal semantics for machine processing
- the goal is to create robotic decision-making devices
- a form of Artificial Intelligence

Michael Uschold - Where Are the Semantics in the Semantic Web?

Pump: "A device for moving a gas or liquid from one place or container to another."

(pump has (superclasses (...))

Shared Human Consensus	Text Descriptions	Semantics Hardwired and Used at Runtime	Semantics Processed and Used at Runtime
Implicit	**Informal (Explicit)**	**Formal (For Humans)**	**Formal (For Machines)**

Figure 1. Semantic Continuum.

Semantics can be implicit, existing only in the minds of the humans who communicate and build web applications. They can also be explicit and informal, or they can be formal. The further I move along the continuum, the less ambiguity there is, and the more likely it is to have interoperable, robust, and correctly functioning web applications.

Syntax vs Semantics

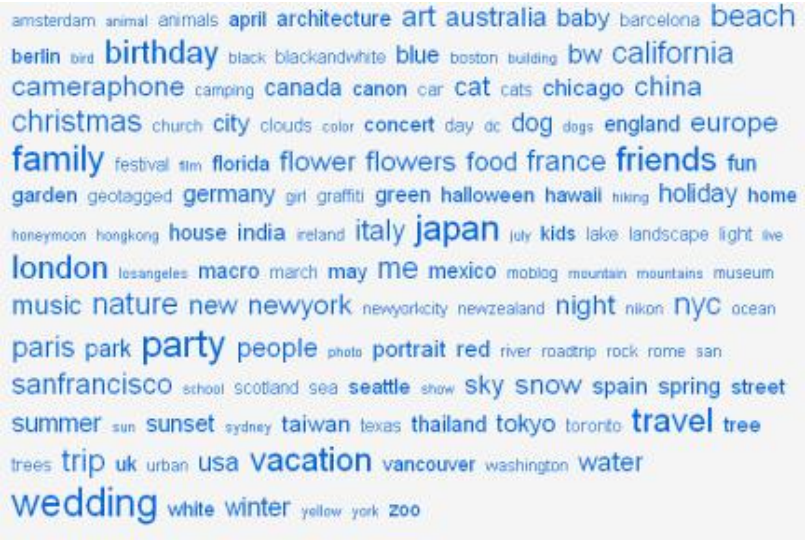
- **syntax** refers to grammatical structure, defines allowed strings of symbols
- **semantics** refers to the meaning of the vocabulary symbols arranged with that structure
- **Syntax** describes the way to construct a correct sentence. For example, "*this water is triangular"* is syntactically correct.
- **Semantics** relates to the meaning. "*this water is triangular*" does not mean anything, though the grammar is ok.

Expressing Semantics

- folksonomies
- microdata, JSON-LD
- RDF triples and RDF Schema vocabularies
- OWL-DL ontologies for automated reasoning

Folksonomies

- keyword metadata as **tags**
- e.g. an image of a dog may be tagged with tags *dog*, *collie* or *pet*
- (+) low entry barrier, no user training
- (-) no synonym control, flat structure
- tag clouds - tag size proportional to usage



Microdata

- competing Microdata, Microformats, RDFa
- nesting semantics within existing content on web pages
- Microdata provides JavaScript API
- Microdata use namespace-qualified vocabularies predefined at data-vocabulary.org or schema.org
- supported by Google search engine
- opposite vision than in 2000:
  - XML with CSS or XSLT - semantic markup with presentational metadata
  - HTML5 with Microdata - presentational markup with semantic metadata

Comparison of Microdata and others

Microformat:

```
<div class="vcard">
<span class="fn">Bob Smith</span>
<span class="title">engineer</span> at <span class="org">ACME Corp</span>.
<span class="adr">
<span class="locality">Albuquerque</span>,
<span class="region">NM</span>
</span>
</div>
```

Microdata:

```
<div itemscope itemtype="http://data-vocabulary.org/Person">
<span itemprop="name">Bob Smith</span>
<span itemprop="title">engineer</span> at <span itemprop="affiliation">ACME Corp</span>.
<span itemprop="address" itemscope itemtype="http://data-vocabulary.org/Address">
<span itemprop="locality">Albuquerque</span>, <span itemprop="region">NM</span>
</span>
</div>
```

RDFa:

```
<div xmlns:v="http://rdf.data-vocabulary.org/#" typeof="v:Person">
<span property="v:name">Bob Smith</span>
<span property="v:title">engineer</span> at <span property="v:affiliation">ACME Corp</span>.
<span rel="v:address">
<span typeof="v:Address">
<span property="v:locality">Albuquerque</span>, <span property="v:region">NM</span>
</span>
</span>
</div>
```

JSON-LD

- Microdata are deprecated since 2016
- Microdata DOM API was deprecated since 2018 in Mozilla, removed since Firefox 49
- JSON-LD (JavaScript Object Notation for Linked Data) replaces them
- W3C Recommendation since 2014
- Google parses JSON-LD and uses them in searches

Example of JSON-LD

```
<script type="application/ld+json">
{
  "@context": "https://schema.org",
  "@type": "Organization",
  "url": "http://www.example.com",
  "name": "Unlimited Ball Bearings Corp.",
  "contactPoint": {
    "@type": "ContactPoint",
    "telephone": "+1-401-555-1212",
    "contactType": "Customer service"
  }
}
</script>
```

RDF - Resource Description Framework

- statements about web resources
- triples *subject-predicate-object*
- subject and predicate are URIs
- object can be a URI or a data value
- **reification** - an RDF statement is assigned a URI and treated as a resource
- producers and consumers of RDF statements must agree on the semantics of the resource identifiers, conveyed by some controlled vocabulary

RDF Schema

- tool for defining controlled vocabularies
- defines
  - classes of things
  - properties (binary predicates)
  - subsumption relationships (subclasses, subproperties)
  - `rdf:type` - resource is an instance of a class
- **SPARQL** (SPARQL Protocol and RDF Query Language) is an SQL-like language for querying RDF graphs
- *entailment rules* allow to entail e.g. that when a resource is in a particular class, then it is also in all its superclasses

RDF Schema example

- RDFS can define two classes:
  - Person
  - Student as subclass of Person
- a RDF statement may state that a resource representing John Doe is of rdf:type Student
- by entailment, John Doe is also a Person

OWL

- Web Ontology Language defined by W3C
- **ontology** is a term from artificial intelligence
- ontology is "*an explicit (written) formal conceptualization*", used for capturing knowledge about some domain of interest
  - "**conceptualization** is an abstract simplified view of some selected part of the world, containing the objects, concepts, and other entities that are presumed of interest for some particular purpose and the relationships between them"
- OWL 1 released in 2004, OWL 2 in 2009
- two different (incompatible) semantics
  - RDF based - OWL Full
  - DL (Description Logics) based - OWL DL

Types of logic

- propositional (Boolean) logic
  - formulae made of atomic propositions with values **true** or **false**, and logical connectives like **negation** ($\neg A$), **and** ($A \wedge B$), **or** ($A \vee B$) and **implication** ($A \rightarrow B$)
  - sound, complete and decidable in finite time
- predicate logic
  - adds predicates, quantifiers, terms
  - formulae look like $\forall x \exists y(P(x) \rightarrow Q(f(y)))$
- first order predicate logic
  - quantifiers can range only over elements of sets
  - sound and complete, but **not decidable**
- description logics
  - logics designed to be as expressive as possible while retaining decidability

OWL DL

- **D**escription **L**ogics is a decidable fragment of First Order Predicate Logic (FOPL) plus decidable extensions
- **reasoners** - software able to entail (decide) complete inferable knowledge in finite time
- **OWL DL ontology** is a set of assertions about:
  - classes
  - individuals
  - properties (binary relations)
    - object properties (between two objects)
    - data properties (between object and data literal)
- can use SWRL (Sem. Web Rule Language)

Example of OWL DL ontology

Prefix(:=<http://goo.gl/kWFWj#>)

Prefix(xsd:=<http://www.w3.org/2001/XMLSchema#>

Ontology(<http://goo.gl/kWFWj>

Declaration(Class(:Person))

Declaration(Class(:MarriedPerson))

Declaration(NamedIndividual(:Martin))

Declaration(NamedIndividual(:Lenka))

Declaration(ObjectProperty(:hasSpouse))

Declaration(DataProperty(:hasEmail))

SymmetricObjectProperty(:hasSpouse)

FunctionalObjectProperty(:hasSpouse)

ClassAssertion(:Person :Lenka)

ClassAssertion(:Person :Martin)

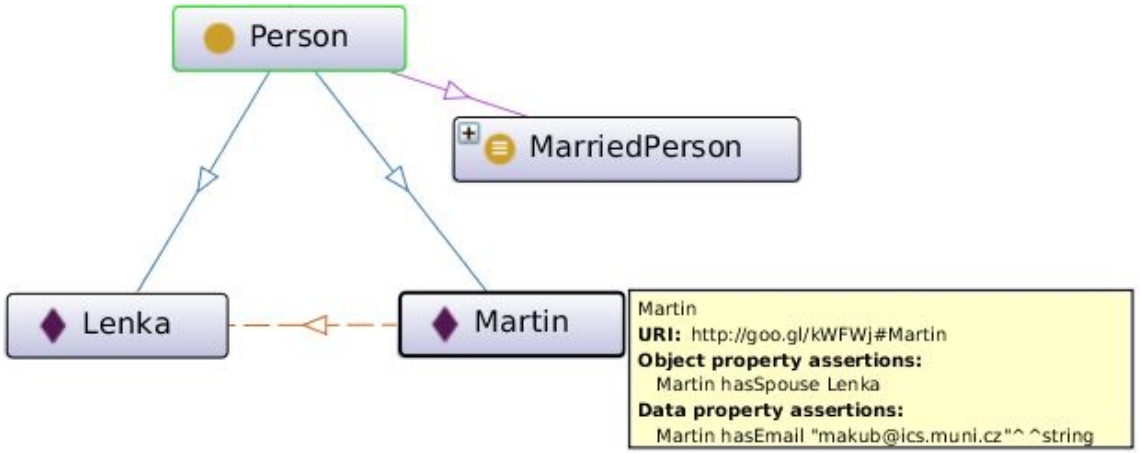
DifferentIndividuals(:Martin :Lenka)

ObjectPropertyAssertion(:hasSpouse :Martin :Lenka)

DataPropertyAssertion(:hasEmail :Martin "makub@ics.muni.cz"^^xsd:string)

SubClassOf(:MarriedPerson :Person)

EquivalentClasses(:MarriedPerson ObjectSomeValuesFrom(:hasSpouse :Person))



OWL DL Tools

- ontology editor with GUI - Protege
  - https://protege.stanford.edu/
- reasoners
  - Pellet
  - HermiT
  - FACT++
  - Stardog
- Java API for OWL - OWL API
  - https://github.com/owlcs/owlapi/

Limits of OWL DL

- DL is based on FOPL $\forall x \exists y(P(x) \rightarrow Q(f(y)))$
- thus DL cannot express:
  - fuzzy expressions - “It **often** rains in autumn.”
  - non-monotonicity - “Birds fly, penguin is a bird, but penguin does not fly.”
  - propositional attitudes - “Eve **thinks** that 2 is not a prime number.”
  - modal logic
    - possibility and necessity - “It is **possible** that it will rain today.”
    - epistemic modalities - “Eve **knows** that 2 is a prime number.”
    - temporal logic - “I am **always** hungry.”
    - deontic logic - “You **must** do this.”
- Transparent Intensional Logic (TIL)
  - can express anything that can be said
  - has no calculus or reasoning algorithms

Semantic Web Services

- research efforts OWL-S, WSDL-S, WSMO
- semantics can enhance discovery
  - on the semantic continuum move it from b) to d)
  - e.g. search for "getHardDriveQuote" can find also "getQuoteForHardDrive" (synonym) and "getSCSIDriveQuote" (subsumed term)
- web service semantics
  - **Data semantics** - it defines meaning of the data, i.e. inputs and outputs of operations
  - **Functional semantics** - it defines meaning of the operations, i.e. how they transform input to output
  - **QoS semantics** - it provides meaning for quality aspects, like price, availability, level of trust etc. Service selection may be based on such characteristics.
  - **Execution semantics** - it provides details like preconditions and effects of service invocation, conversation patters of service invocation etc

That’s it.

Thank you for your attention