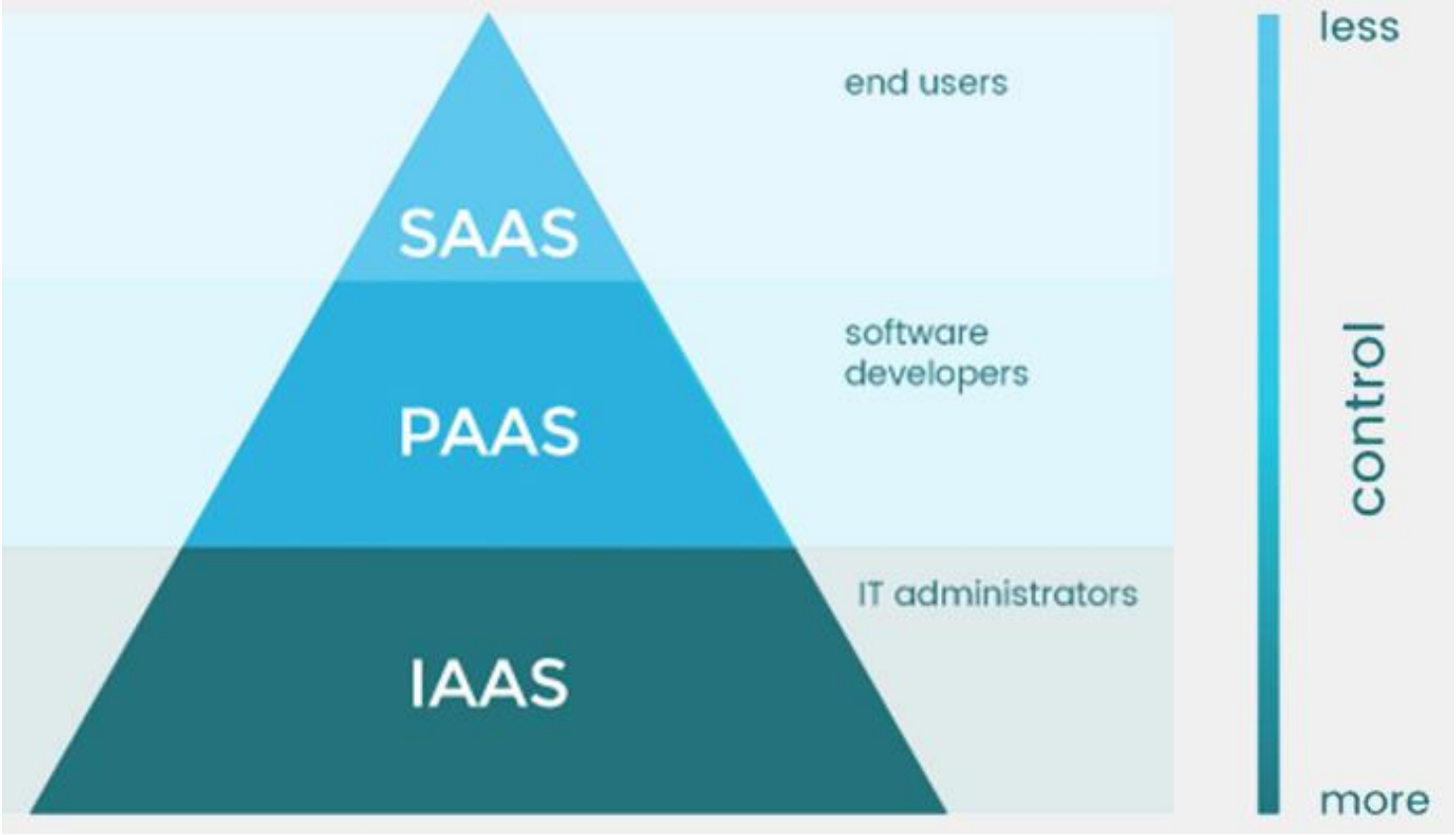
TETRA OpenCloudEdge

Virtual meeting June 2020

Contents

- Recap cloud concepts
- Current infrastructure
- Kubernetes concepts
- Deployed Kubernetes applications
- First experiences with OpenStack
- Conclusion and future work

Cloud concepts



IaaS building blocks: Compute, Storage, Network

Our cloud infrastructure servers

3x HPE ProLiant DL325 gen 10

- 1U single rack unit
- AMD EPYC 7302P **16 core** processor
- 64 GB Registered RAM
- 8 Small Form Factor (2,5”) SATA storage
  - 1TB SSD and 3x 2TB HDD
- 4x 1 Gbps Ethernet

An **uneven** amount of servers is preferable for distributed application to achieve *quorum*

Distributed storage

Ceph distributed & redundant storage

- Object storage
- Block storage
- File storage

Open source software-based solution

Can be back-end for *OpenStack Cinder, Glance, Swift*

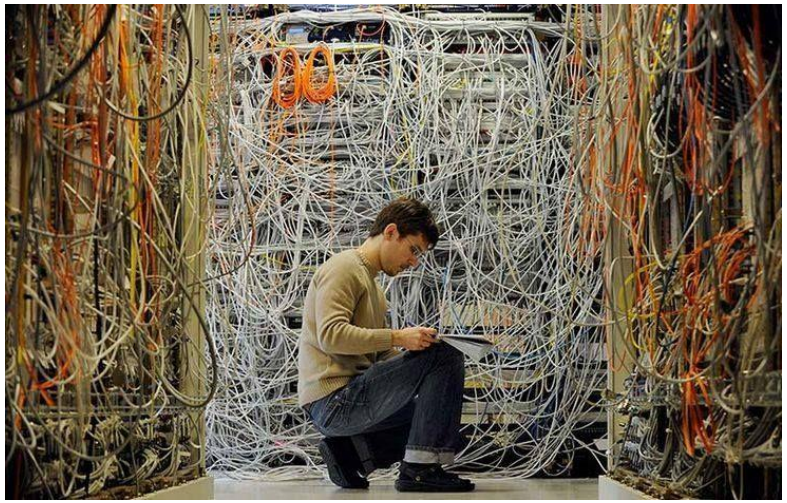
13 TB redundant storage

Combination of slow (HDD) and faster* (SSD) storage

Networking

Only 1x 1 Gbps network interface in use per server

BUT: 4x 1 Gbps per server available



We will buy additional network equipment

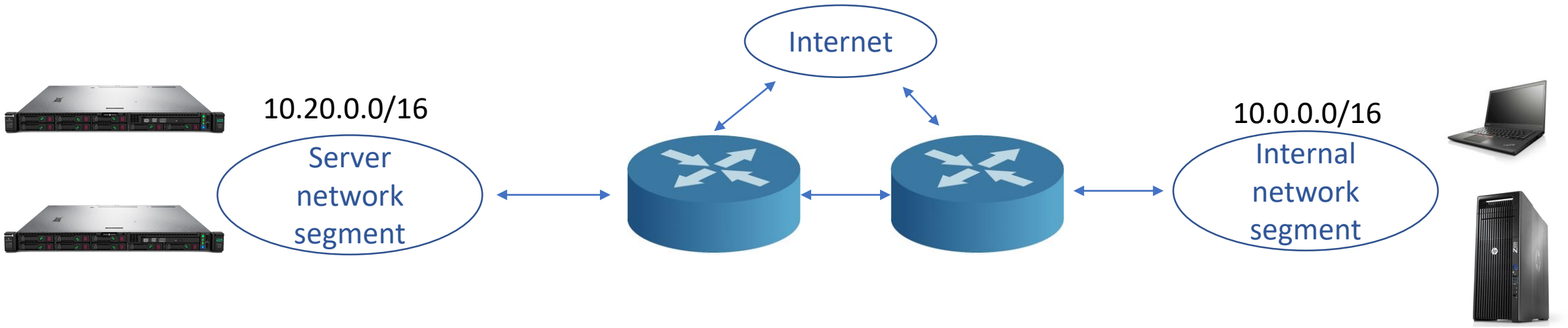
Possible improvements

- IEEE 802.3ad link aggregation
- Distinct network segments for public / cloud-internal / distributed storage

Networking

Servers and regular (edge) devices are on different network segments

- Cloud-internal network between different network segments is tricky
- Cloud-internal overlay network using IP-in-IP or VXLAN



No public IPv6 available: use of private ranges on *fc00::/7*

Might migrate to a separated network segment

- Better network test environment completely under our control
- allows experimenting with BGP features

Integrating workstations and “edge” devices

Inside VUB-ETRO network

- Workstation computers
  - Machine with **RTX 2070** GPU for CUDA workloads
- Raspberry Pi as IoT gateway
- Provides Low latency, high bandwidth with our private cloud servers

At researcher’s home network (connected via VPN)

- High latency, low bandwidth with our private cloud servers
- Low latency, high bandwidth inside the home network
- Significant difference cloud-edge regarding network performance

Interaction with the public cloud

Future work

No foreseeable problems

- Our private cloud infrastructure can get public IPv4 addresses
- Bi-directional communication possible

Which public cloud providers would we look at according to you?


Cloud environments and their host OS

OpenStack, Kubernetes, etc. need an underlying host OS (or hypervisor)

RHEL/CentOS 7 rather dated (7/2014)

- Newer hardware might not perform optimally

RHEL/CentOS 8 new at time of testing (9/2019)

- Not yet all software and libraries compatible (1/2020)

OpenStack *Train*: ~~CentOS 8 support~~

OpenStack *Ussuri*: CentOS 8 support ✔

First get to know **Kubernetes** while waiting on *Ussuri* release

What is Kubernetes?

Kubernetes (K8s) = Container orchestration platform

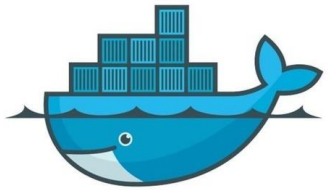
kubernetes

Automates deployment, scaling and management of *containerized* apps

Open source and initially developed by Google

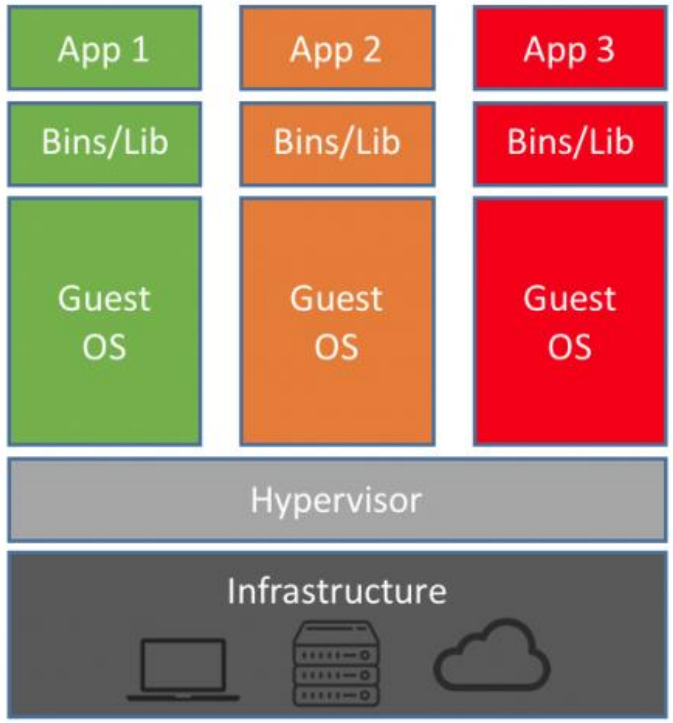
Currently very popular in the cloud ecosystem

Requires an external *container* engine: e.g. Docker

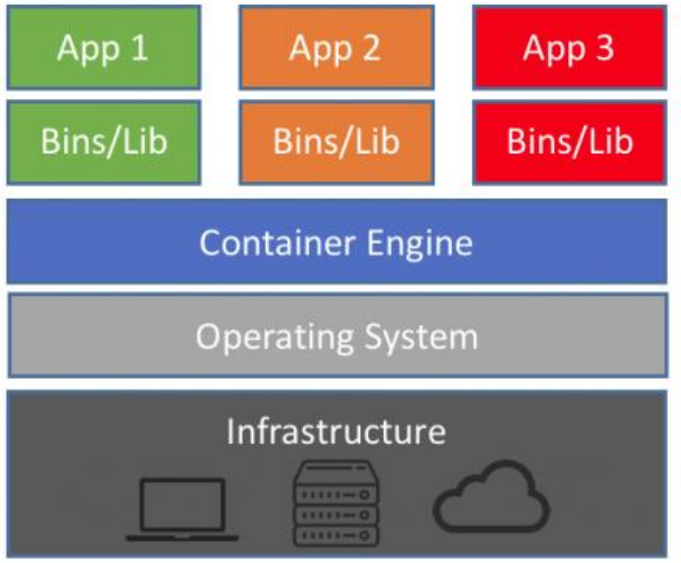


docker

Virtualization or Containerization?



Machine Virtualization



Containers

openstack.

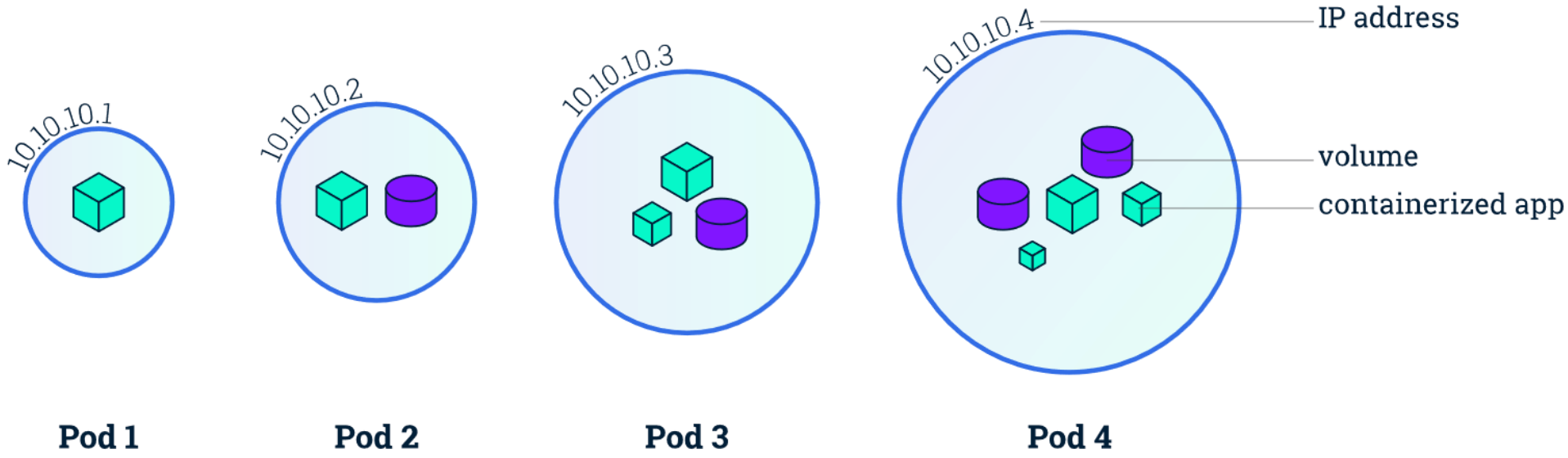
vmware

docker + kubernetes

Kubernetes basic concepts

Pods → building blocks. They *can* contain

- Compute: one or more containerized apps
- Storage: volumes can be mounted
- Network: cloud-internal IP address



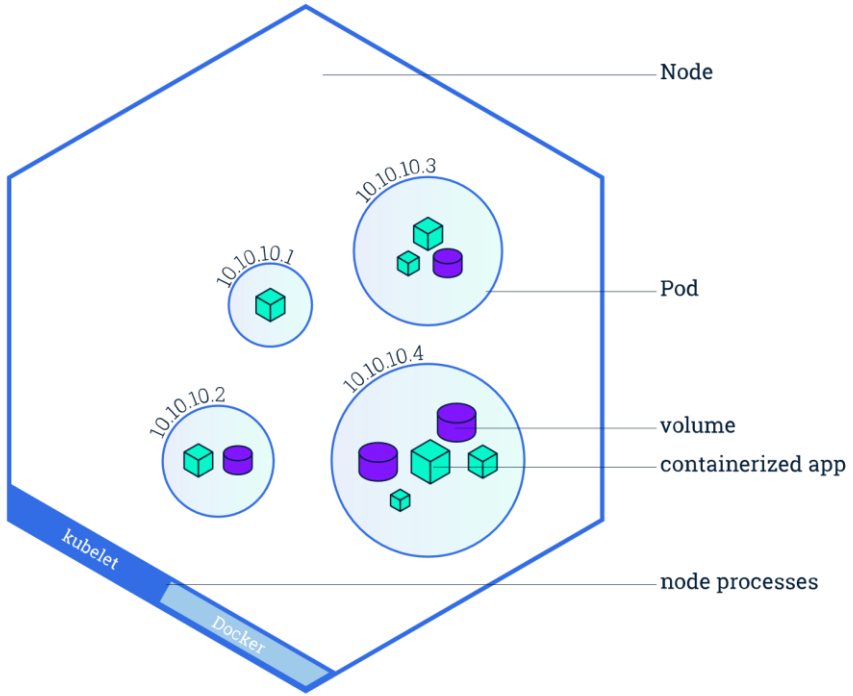
Kubernetes basic concepts

Nodes → machines running Kubernetes

- Nodes run pods

Worker nodes run *kubelet* and Docker

- Remotely managed by *control-plane*



Control-plane nodes manage and schedule the cluster

- Control-plane requires *quorum* for decision making

Kubernetes basic concepts

Deployments → pod orchestration

- Describe how to deploy & maintain pods and their amount of replicas

> *"I want **X** pods of **version Y** of **application Z** up and running using **these** resource constraints."*

Pods are automictically (re)deployed in case of update or problem

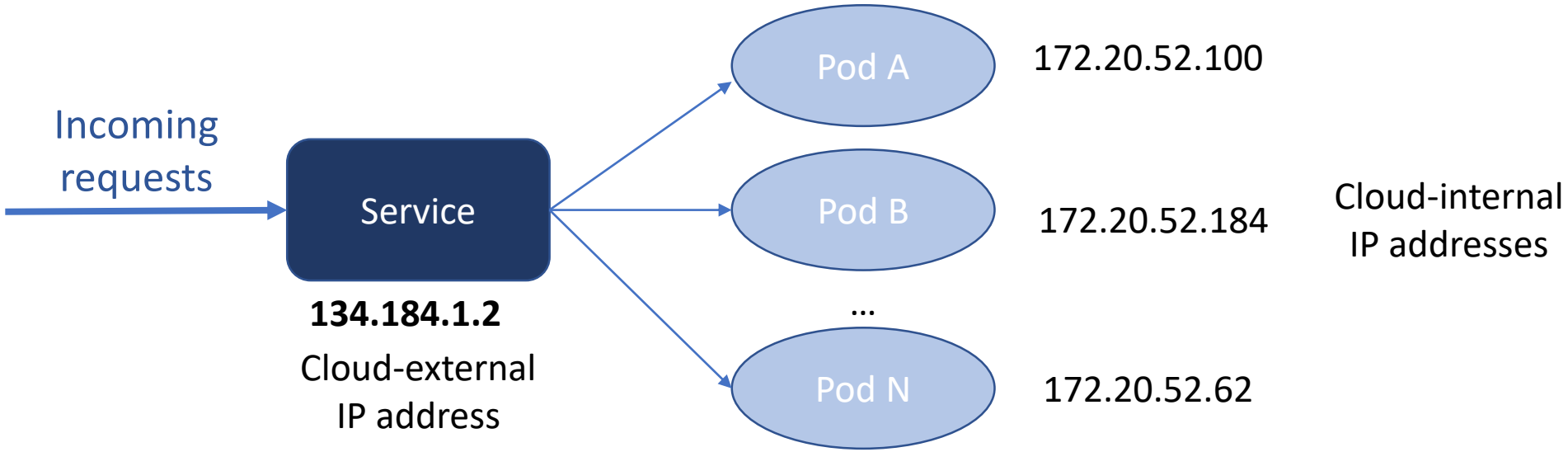
Horizontal **Autoscaling** can be included to adapt the replica count based on their resource metrics.

Kubernetes basic concepts

Services → persistent access points to sets of homogenous pods

Why?

- Persistent access point: pods can be short-living and have dynamic IP
- Load balancing between available pods
- Can facilitate cloud-external access



Kubernetes basic concepts

Namespaces are used to create separate environments

- Different applications/deployments
- Different users

Role-Based Access Control (RBAC) policies provide **Authorization**

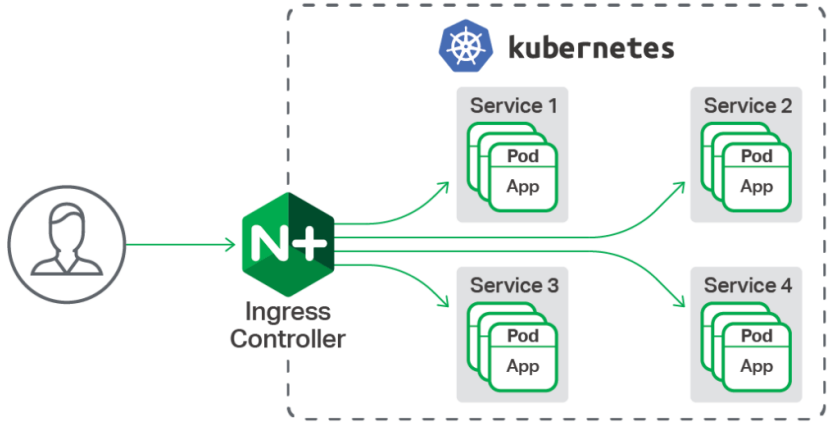
- On namespace level: Roles + RoleBindings
- On k8s cluster level: ClusterRoles + ClusterRoleBindings

Clients can **Authenticate** with K8s API via ***ServiceAccount***
or via external method (X509 Certificate, OpenStack Keystone, etc.)

Kubernetes basic concepts

Ingress: HTTP(S) reverse proxy server for incoming connections

- Allows having multiple (web)applications on the same IP:port combination by using DNS name
- TLS termination and certificate management



Storage via **PersistentVolume** and **PersistentVolumeClaim** abstractions

- PV → Infrastructure side PVC → Application side
- Can be automated via **StorageClass** resource

Kubernetes and virtualization

Kubernetes → containerization

- Containers use kernel *cgroups* for isolation
- Isolated but still remains a shared operating system
- *What about bugs and vulnerabilities?*

Projects like *KubeVirt* and *Kata Containers goal:*

Kubernetes → virtualization

Kubernetes Applications

Self-hosted Docker registry

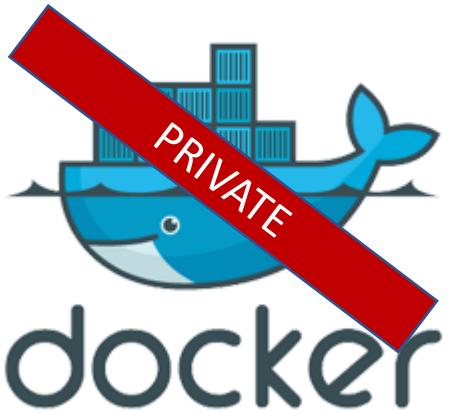
Containerized applications work via **images**

- Prebuilt from base images with application-specific changes applied
- Public registry (image repository) available on https://hub.docker.com

Host private registry on the K8s cluster

- Local → low delay, high bandwidth
- Keep development internal

K8s manages high-availability of the registry



Object detection on images

GPU-accelerated (CUDA) training of model (via COCO dataset)

Webapp with OpenCV backend (on CPU) detects objects

- Horizontal autoscaling adapts compute based on load metrics

HTTP POST request

HTTP POST response

ETROpy online programming environment

Web application for students to make programming exercises

- Generic approach allows many languages: Python, Java, C, C++, C# (and more)
- No control over uploaded source code under evaluation

Platform must be able to safely execute potentially malicious code

Use the benefits of containerization

- Short-lived containers (pods) in isolated environment
- Images provide suitable environments (compiler, interpreter, libraries)

ETROpy online programming environment

ETROpy webapp manages code-validation pods in *etropy* namespace

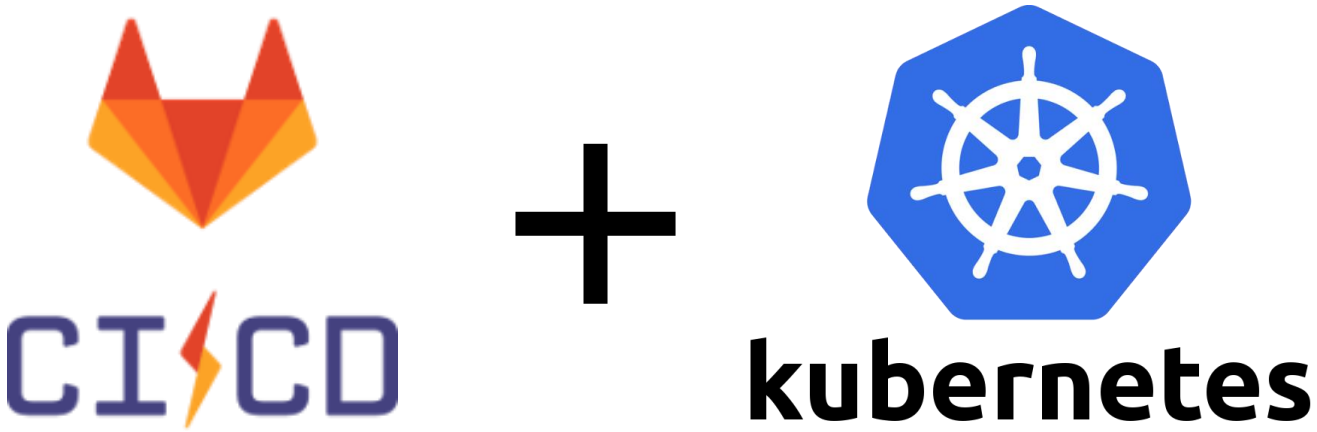
- Webapp uses K8s client API (Python)
- Authentication & authorization via *ServiceAccount* and RBAC

1. Create pod from image with suitable environment
2. Transfer source code
3. Compile source code if required; return on failure
4. Validate use-cases sequentially:
   a) Transfer use-case input
   b) Run program with use-case input; return on failure or timeout
   c) Retrieve use-case output and validate
5. Terminate pod, process results

Continuous integration and deployment

For our current cryptographic research we evaluate:

- The scalability of the cryptographic technique
- The practical feasibility and means of implementation



Classic application architecture

academic interest:
client and service

The stack

Mobile application

Java

Rust

Android

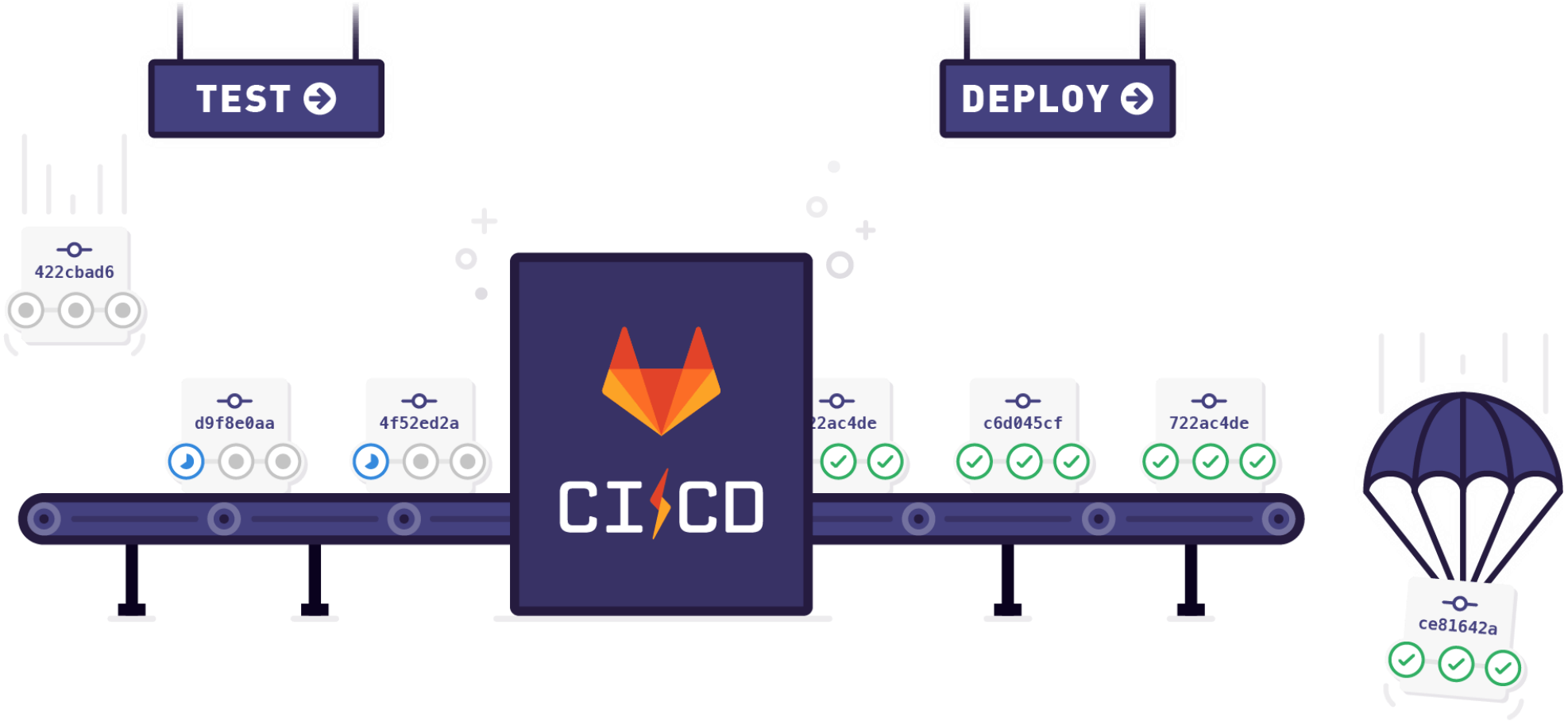
REST service

Actix Web + Diesel

Rust

Docker

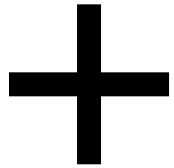
Gitlab CI



.gitlab-ci.yml

```
build:nightly:
  <<: *rust-nightly
  stage: build
  script:
    - cargo build --all-features
    - cargo build --release --all-features
```

Gitlab CI on top of Kubernetes



Gitlab jobs operate (typically) in Docker.

Kubernetes provides a logic partner.

https://docs.gitlab.com/runner/install/kubernetes.html

Docker-in-Docker (dind)

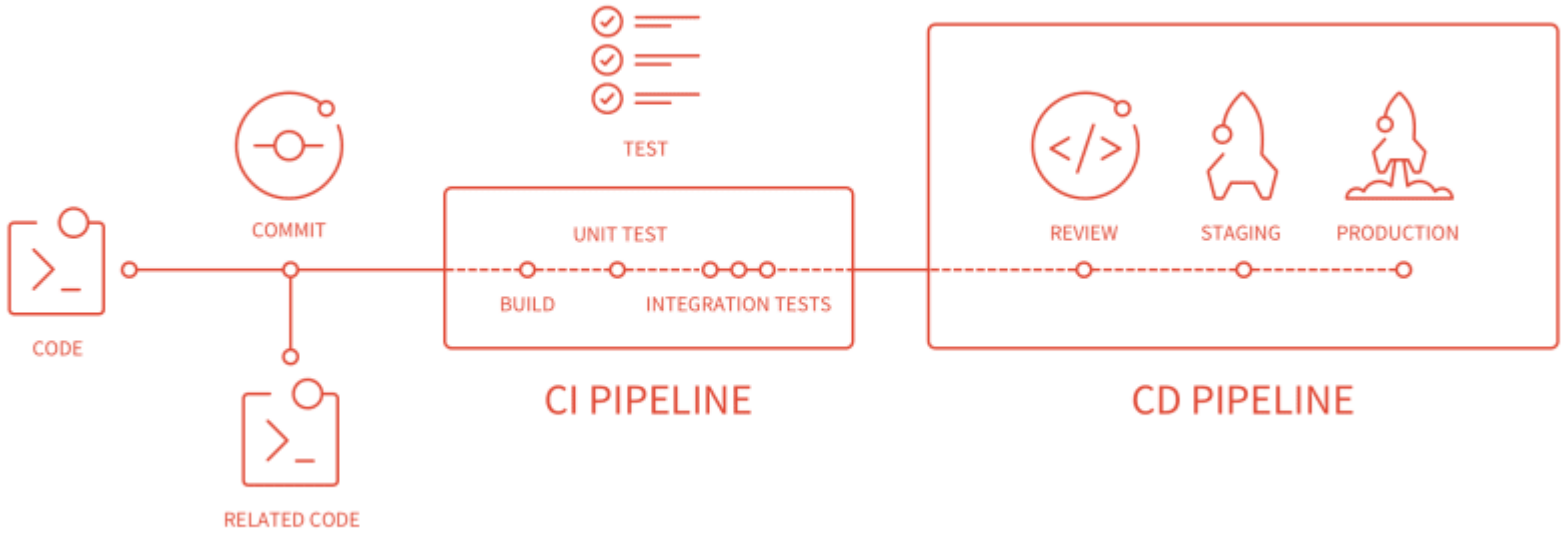
A Gitlab job building a Docker container e.g.:

```
build:docker:
  <<: *dind
  stage: build
  script:
    - docker build .
```

This requires Docker-in-Docker, or a 3rd party build-tool (e.g. Kaniko).

Towards Kubernetes continuous deployment

Gitlab provides Kubernetes integration in recent versions,
allowing the deployment of built images.



OpenStack Ussuri current experiences

Open-source cloud platform

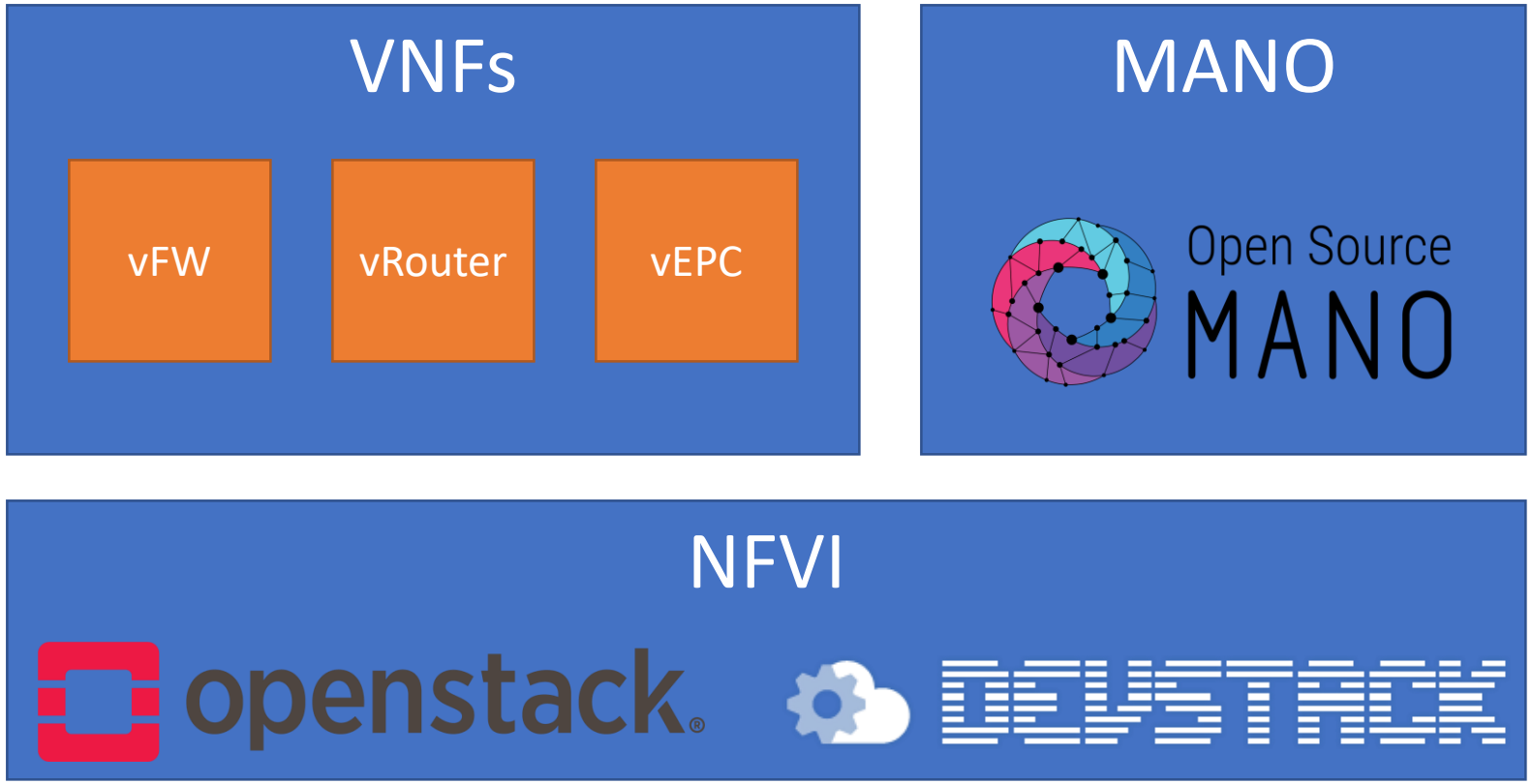
Released May 2020

DevStack

- script to bootstrap single-node OpenStack
- used for development of OpenStack services
- tested on Ubuntu 18.04

Future work: other deployment options

Network Function Virtualization



Conclusion

- Introduction of our physical ‘cloud’ infrastructure
- OpenStack evaluation postponed while waiting on *Ussuri* version
- Impressed with Kubernetes operation, performance, userbase
  - Probably suffices for majority of SME use-cases
  - We have introduced basic concepts and several deployed applications
- Isolation Containerization ↔ Virtualization might remain an issue for some

Future work

- Add 3 GPUs to the K8s cluster: accelerate cryptographic operations
- Kubernetes integration with Gitlab
- Extension of network infrastructure and IPv6 evaluation
- OpenStack ↔ KubeVirt (or similar technology)
- Interaction with edge and public cloud

Points to discuss

- Kubernetes ↔ OpenStack
- Hybrid cloud - preferences
- Use-case suggestions
- Other suggestions

Next meeting: Hands-on workshop in December 2020 (TBD)