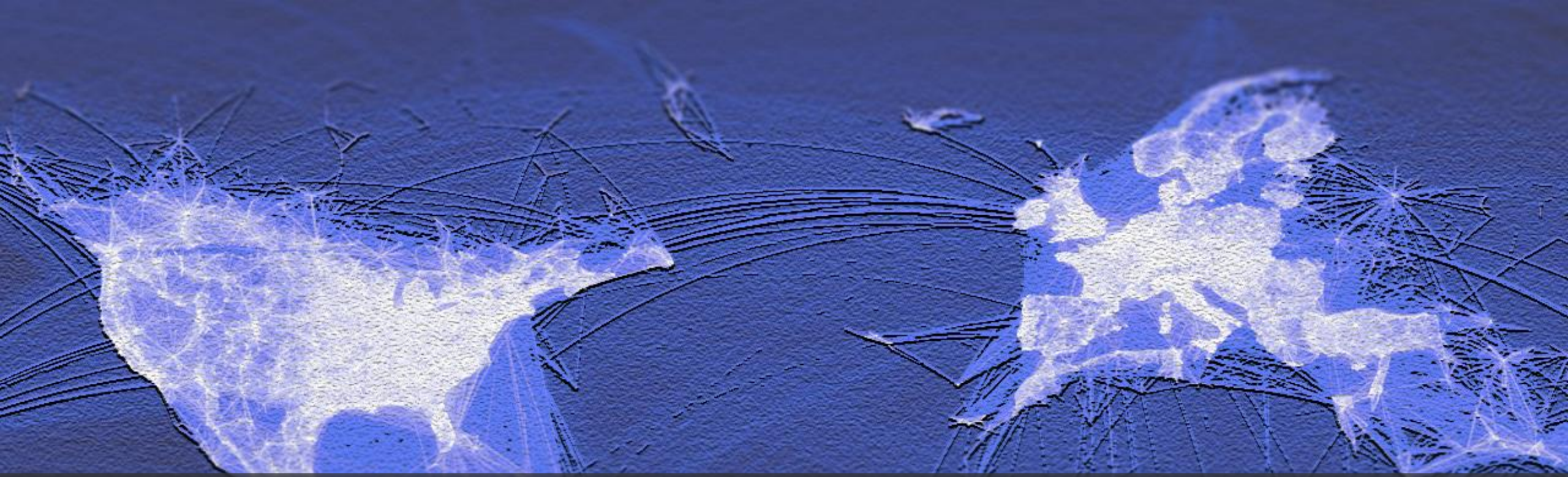


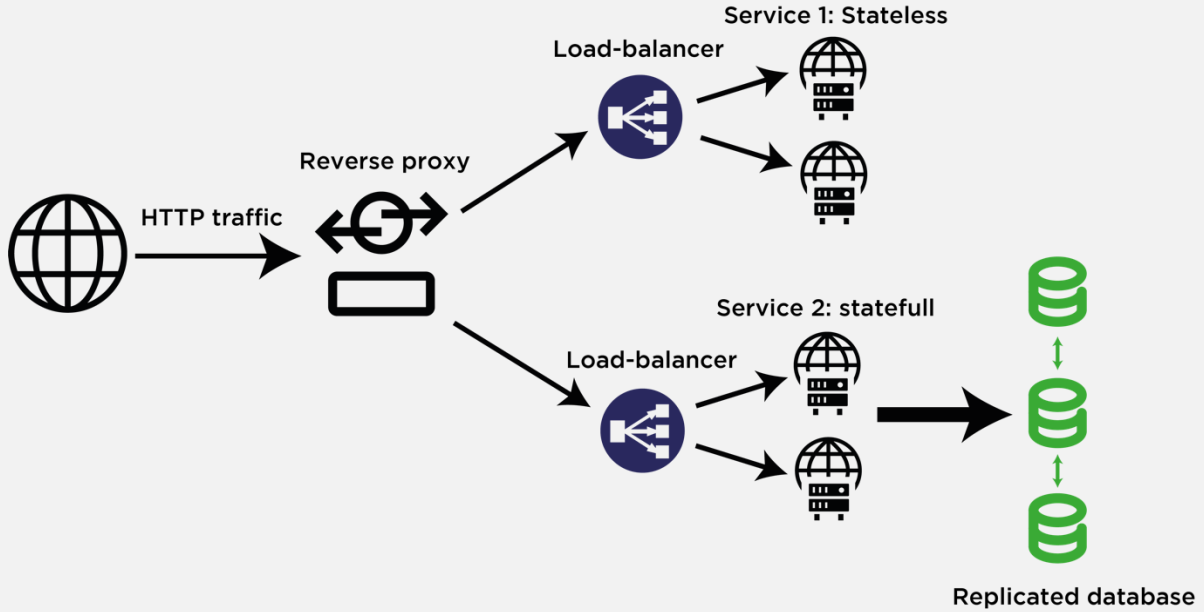
Introduction to Kubernetes

Introduction

Vincent Gauthier
Associate Professor
@vincentgauthier
http://complex.luxbulb.org
SAMOVAR/Telecom SudParis/Institut Polytechnique de Paris

NET 4255: PROJECT OVERVIEW

Project Overview



1. The first part of the project (**on your machine**)
   - Develop all the microservices you need with Docker first.
   - Create a docker Compose a toy model of your web service with the micro-services developed previously (web server, database, load balancer).
2. The second part of the project (**on Kubernetes cluster**)
   - Instantiate your previously developed toy model on a Kubernetes cluster.
   - Scale your toy to make it scalable and redundant.

NET 4255: How does it work !

- This is a long-term project that will span the duration of the class.
- The project template is available at the following URL:

https://classroom.github.com/a/eE0kbWc8

  - Git clone the project template into your own repository.
  - When you complete a task, commit your code and answers to your Git repository.
- Complete each of the required tasks in **order**!
- You'll be graded individually, and **we expect you to complete the project individually**, but you can work in groups and help each other.

NET 4255: How to get grade

When you finish a question/challenge, **call an instructor**

- Show your Git repository and explain what you did.
- Show the execution of your code to validate it with the instructor.
- Answer each questions if needed.
- If all goes well, you will receive some points for each validated question.
- Each question should be validated before moving on to the next.

NET 4255: RULES

The secondary purpose of this class is to develop your ability to learn new techniques **on your own**.

- The instructors are here to guide you, not to debug your code.
- **Your ability to learn on your own will also be graded**.
- Don't ask how:
  - "docker bla bla"
  - "kubectl bla bla"
  - It doesn't work, what should I do? ....
- **RTFM**: There are many resources on the web and there are several good ways to answer the questions of the project.
- So, help yourself and learn to learn.
- When you leave the class, please shutdown your Pods (container) to minimize the cost of running the cluster
- You have quota in the Kubernetes cluster with 2 VCPU and 2Go of memory so pace yourself when you deploy things

Introduction to Kubernetes Architecture

MICROSERVICES

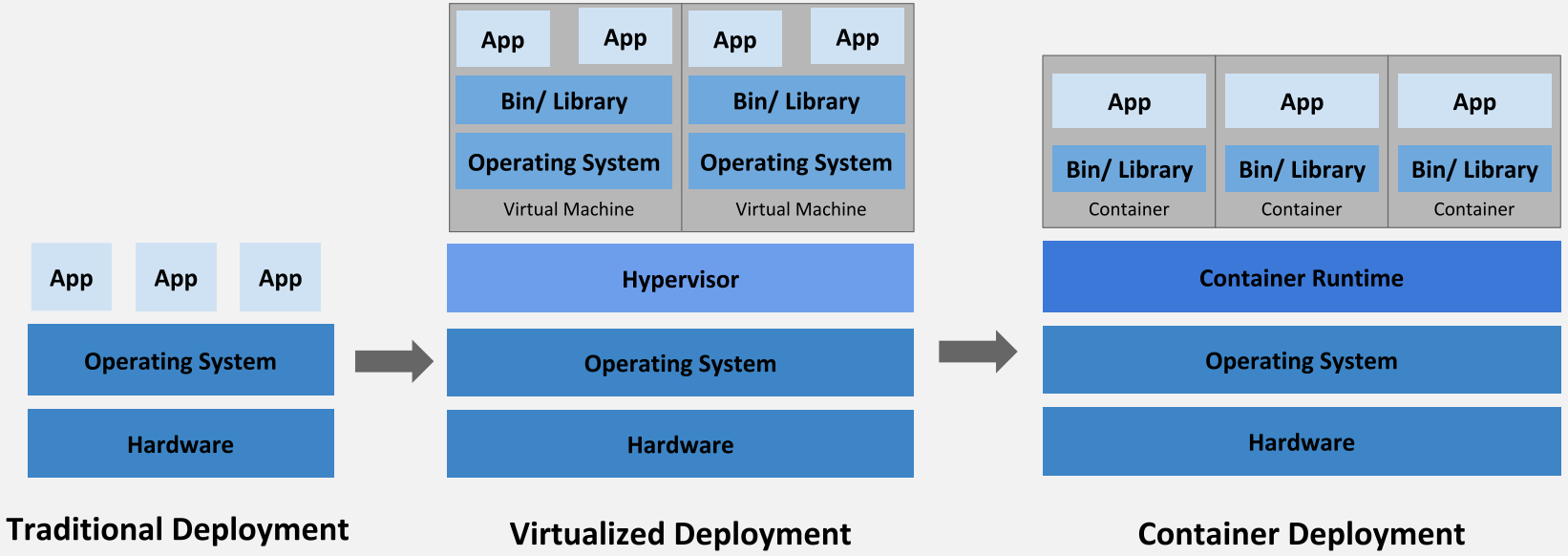
- Microservices can be deployed individually on separate servers provisioned with fewer resources - only what is required by each service and the host system itself, helping to lower compute resource expenses.

- Microservices-based architecture is aligned with Event-driven Architecture and Service-Oriented Architecture (SOA) principles, where complex applications are composed of small independent processes which communicate with each other through Application Programming Interfaces (APIs) over a network. APIs allow access by other internal services of the same application or external, third-party services and applications.

- Although the distributed nature of microservices **adds complexity to the architecture**, one of the greatest **benefits of microservices is scalability**. With the overall application becoming modular, each microservice can be scaled individually, either manually or automated through demand-based autoscaling.

What is a container?

- Containers are an application-centric method to deliver high-performing, scalable applications on any infrastructure of your choice. Containers are best suited to deliver microservices by providing portable, isolated virtual environments for applications to run without interference from other running applications.
- A container image bundles the application along with its runtime, libraries, and dependencies, and it represents the source of a container deployed to offer an isolated executable environment for the application. Containers can be deployed from a specific image on many platforms, such as workstations, Virtual Machines, public cloud, etc.



CONTAINER ORCHESTRATORS

With enterprises containerizing their applications and moving them to the cloud, there is a growing demand for container orchestration solutions. While there are many solutions available, some are mere re-distributions of well-established container orchestration tools, enriched with features and, sometimes, with certain limitations in flexibility. Although not exhaustive, the list below provides a few different container orchestration tools and services available today:

Container orchestrator beyond Kubernetes

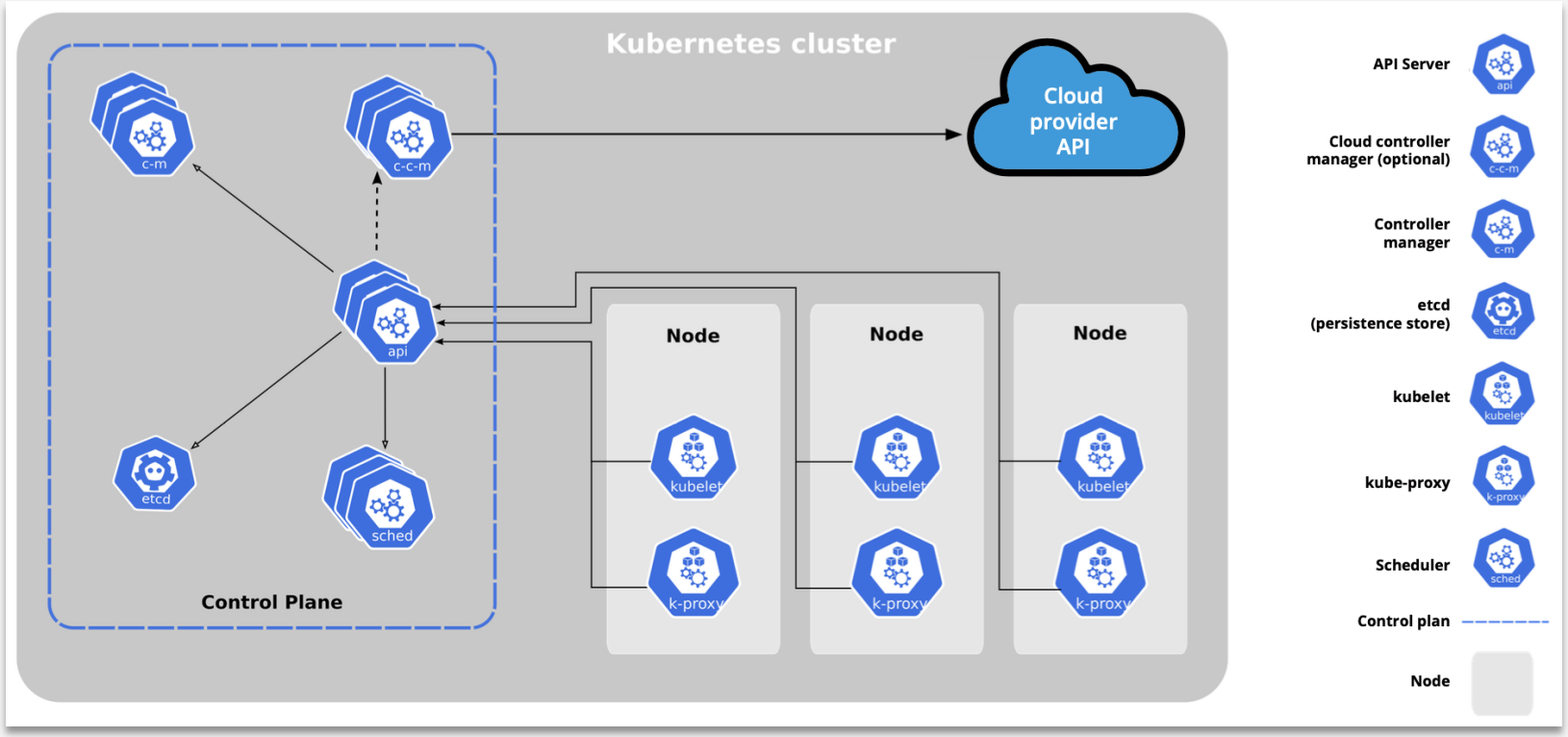
- **Amazon Elastic Container Service**: Amazon Elastic Container Service (ECS) is a hosted service provided by Amazon Web Services (AWS) to run containers at scale on its infrastructure.
- **Azure Container Instances**: Azure Container Instance (ACI) is a basic container orchestration service provided by Microsoft Azure.
- **Azure Service Fabric**: Azure Service Fabric is an open source container orchestrator provided by Microsoft Azure.
- **Kubernetes**: Kubernetes is an open source orchestration tool, originally started by Google, today part of the Cloud Native Computing Foundation (CNCF) project.
- **Marathon:** Marathon is a framework to run containers at scale on Apache Mesos and DC/OS.
- **Nomad:** Nomad is the container and workload orchestrator provided by HashiCorp.
- **Docker Swarm**: Docker Swarm is a container orchestrator provided by Docker, Inc. It is part of Docker Engine.

KUBERNETES VS CONTAINERS

At a very high level, Kubernetes is a cluster of compute systems categorized by their distinct roles:

- One or more control plane nodes
- One or more worker nodes (optional but recommended).

Main difference:
The role of the **kube-scheduler** is to assign new workload objects, such as pods encapsulating containers, to nodes - typically worker nodes. During the scheduling process, decisions are made based on current Kubernetes cluster state and new workload object's requirements.



Control Plane: API Server - Key-Value Data Store

API Server

All the administrative tasks are coordinated by the kube-apiserver, a central control plane component running on the control plane node. The API Server intercepts RESTful calls from users, administrators, developers, operators and external agents, then validates and processes them. During processing the API Server reads the Kubernetes cluster's current state from the key-value store, and after a call's execution, the resulting state of the Kubernetes cluster is saved in the key-value store for persistence. The API Server is the only control plane component to talk to the key-value store, both to read from and to save Kubernetes cluster state information - acting as a middle interface for any other control plane agent inquiring about the cluster's state.

Key-value Store

etcd is an open source project under the Cloud Native Computing Foundation (CNCF). *etcd* is a strongly consistent, distributed key-value data store used to persist a Kubernetes cluster's state. New data is written to the data store only by appending to it, data is never replaced in the data store. Obsolete data is compacted (or shredded) periodically to minimize the size of the data store. Out of all the control plane components, only the API Server is able to communicate with the *etcd* data store.

Introduction to Kubernetes Building Block

NAMESPACE/PODS

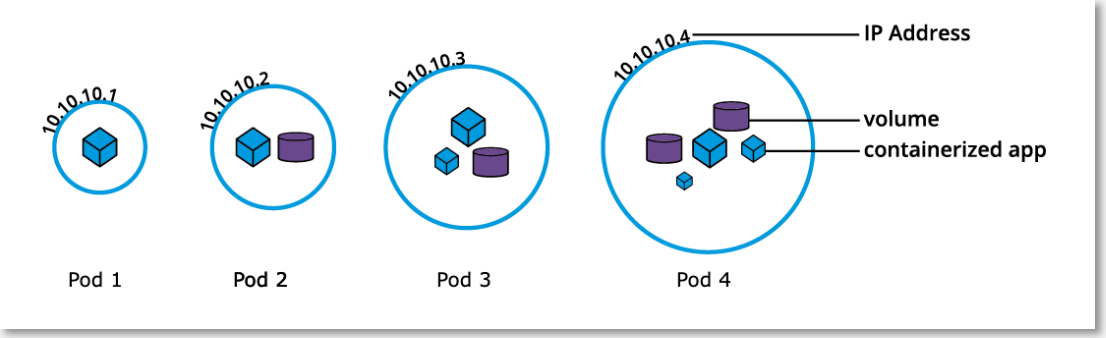
Namespace

If multiple users and teams use the same Kubernetes cluster, we can partition the cluster into virtual sub-clusters using Namespaces. The names of the resources/objects created inside a Namespace are unique, but not across Namespaces in the cluster.

Pods

A Pod is the smallest Kubernetes workload object. It is the unit of deployment in Kubernetes, which represents a single instance of the application. A Pod is a logical collection of one or more containers, enclosing and isolating them to ensure that they:

- Are scheduled together on the same host with the Pod.
- Share the same network namespace, meaning that they share a single IP address originally assigned to the Pod.
- Have access to mount the same external storage (volumes) and other common dependencies.



Network Communication

Container-to-Container Communication Inside Pods

- Making use of the underlying host operating system's kernel virtualization features, a container runtime creates an isolated network space for each container it starts. On Linux, this isolated network space is referred to as a network namespace. A network namespace can be shared across containers, or with the host operating system. When a grouping of containers defined by a Pod is started, a special infrastructure Pause container is initialized by the Container Runtime for the sole purpose of creating a network namespace for the Pod. All additional containers, created through user requests, running inside the Pod will share the Pause container's network namespace so that they can all talk to each other via localhost.

Pod-to-Pod Communication Across Nodes

- In a Kubernetes cluster Pods, groups of containers, are scheduled on nodes in a nearly unpredictable fashion. Regardless of their host node, Pods are expected to be able to communicate with all other Pods in the cluster, all this without the implementation of Network Address Translation (NAT). This is a fundamental requirement of any networking implementation in Kubernetes. The Kubernetes network model aims to reduce complexity, and it treats Pods as VMs on a network, where each VM is equipped with a network interface - thus each Pod receiving a unique IP address. This model is called "IP-per-Pod" and ensures Pod-to-Pod communication, just as VMs are able to communicate with each other on the same network.

External-to-Pod Communication

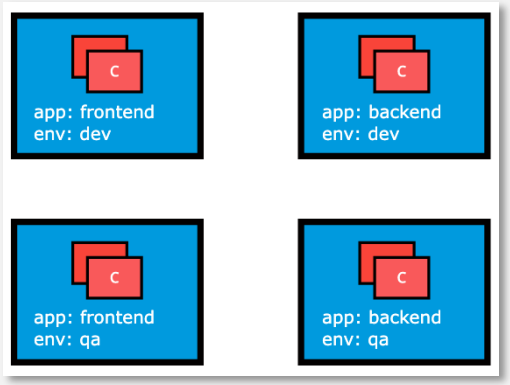
- A successfully deployed containerized application running in Pods inside a Kubernetes cluster may require accessibility from the outside world. Kubernetes enables external accessibility through Services, complex encapsulations of network routing rule definitions stored in iptables on cluster nodes and implemented by kube-proxy agents. By exposing services to the external world with the aid of kube-proxy, applications become accessible from outside the cluster over a virtual IP address and a dedicated port number.

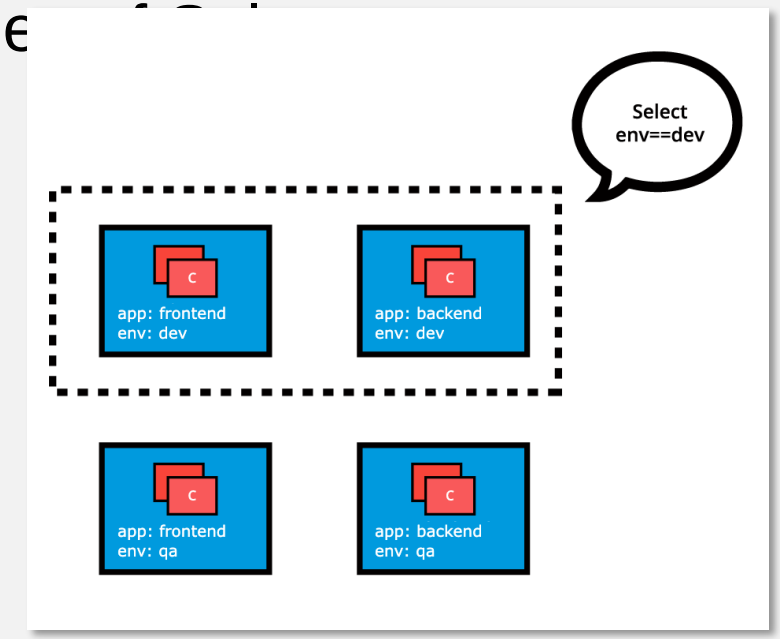
Proxy - kube-proxy

- The kube-proxy is the network agent which runs on each node, control plane and workers, responsible for dynamic updates and maintenance of all networking rules on the node. It abstracts the details of Pods networking and forwards connection requests to the containers in the Pods. The kube-proxy is responsible for TCP, UDP, and SCTP stream forwarding or random forwarding across a set of Pod backends of an application, and it implements forwarding rules defined by users through Service API objects.

LABELS

- Labels are key-value pairs attached to Kubernetes objects (e.g. Pods, ReplicaSets, Nodes, Namespaces, Persistent Volumes). Labels are used to organize and select a subset of objects, based on the requirements in place. Many objects can have the same Label(s). Labels do not provide uniqueness to objects. Controllers use Labels to logically group together decoupled objects, rather than using objects' names or IDs.
- Controllers, or operators, and Services, use label selectors to select a subset of objects. Kubernetes supports two type

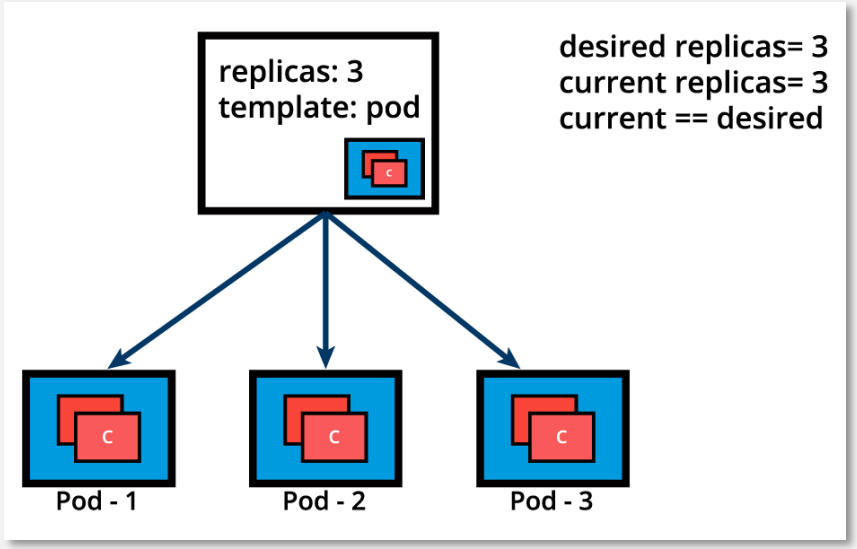


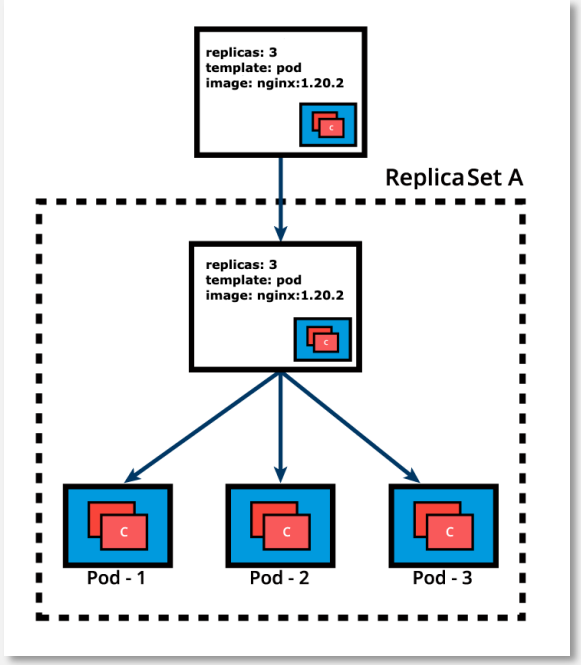


REPLICATSET/DEPLOYMENTS

When a single instance of an application is running there is always the risk of the application instance crashing unexpectedly, or the entire server hosting the application crashing. To avoid such possible failures, we can run in parallel multiple instances of the application, hence achieving high availability. The lifecycle of the application defined by a Pod will be overseen by a controller - the ReplicaSet. With the help of the ReplicaSet, we can scale the number of Pods running a specific application container image. Scaling can be accomplished manually or using an autoscaler.

we graphically represent a ReplicaSet, with the replica count set to 3 for a specific Pod template. Pod-1, Pod-2, and Pod-3 are identical, running the same application container image, being cloned from the same Pod template. For now, the current state matches the desired state.





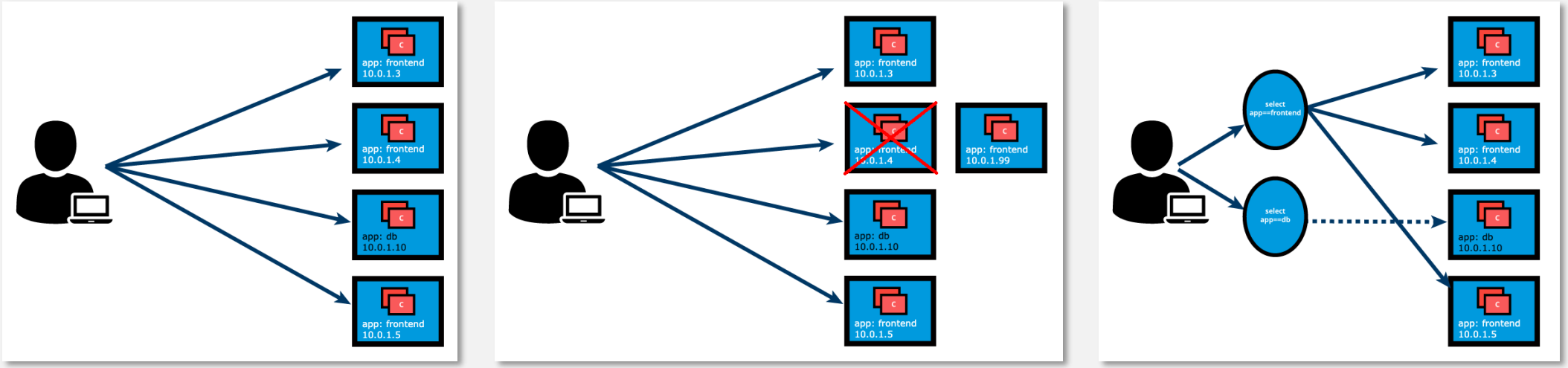
DaemonSets

DaemonSets are operators designed to manage node agents. They resemble ReplicaSet and Deployment operators when managing multiple Pod replicas and application updates, but the **DaemonSets** present a distinct feature that enforces a **single Pod replica to be placed per Node**, on all the Nodes. In contrast, the ReplicaSet and Deployment operators by default have no control over the scheduling and placement of multiple Pod replicas on the same Node.

SERVICES

A containerized application deployed to a Kubernetes cluster may need to reach other such applications, or it may need to be accessible to other applications and possibly clients.

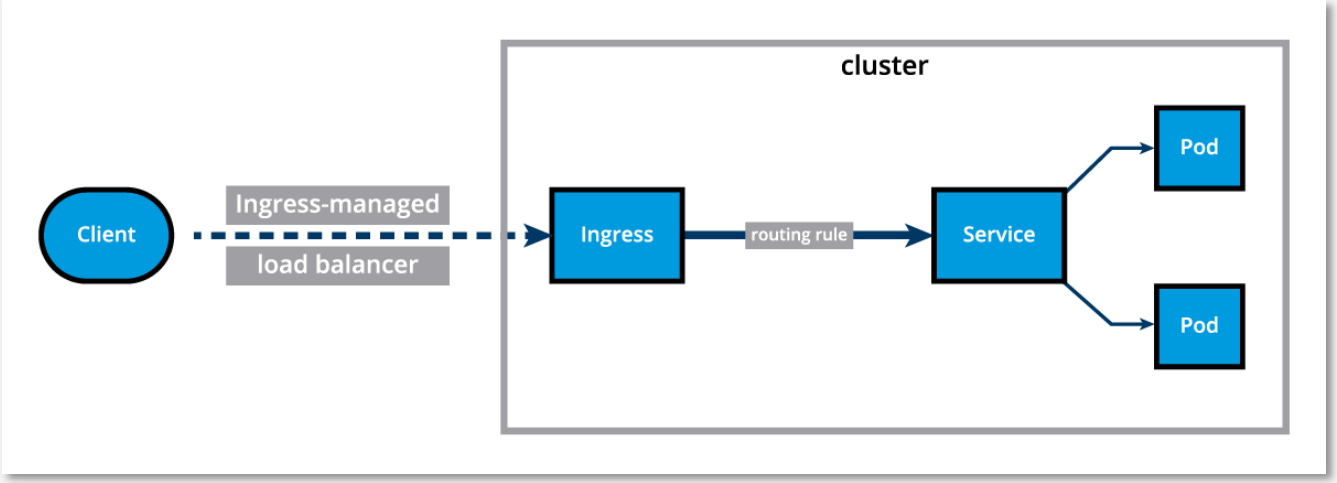
This mechanism is called a **Service**, and it is the recommended method to expose any containerized application to the Kubernetes network. The benefits of the **Kubernetes Service** becomes more obvious when exposing a multi-replica application, when multiple containers running the same image need to expose the same port.



INGRESS

With Services, routing rules are associated with a given Service. They exist for as long as the Service exists, and there are many rules because there are many Services in the cluster. If we can somehow decouple the routing rules from the application and centralize the rules management, we can then update our application without worrying about its external access. This can be done using the **Ingress** resource.

- According to kubernetes.io: "An Ingress is a collection of rules that allow inbound connections to reach the cluster Services".
- To allow the inbound connection to reach the cluster Services, Ingress configures a Layer 7 HTTP/HTTPS load balancer for Services and provides the following:
  - TLS (Transport Layer Security)
  - Name-based virtual hosting
  - Fanout routing
  - Loadbalancing
  - Custom rules.



Word of caution

- Not all applications are well suited to run in Kubernetes environment
  - For instance: Database system is cumbersome to run in a cloud native environment (getting easier)
- Kubernetes is more useful for stateless type of application
- Kubernetes add a layer of complexity in the system design
- It is harder to debug a cloud native application than a standalone application

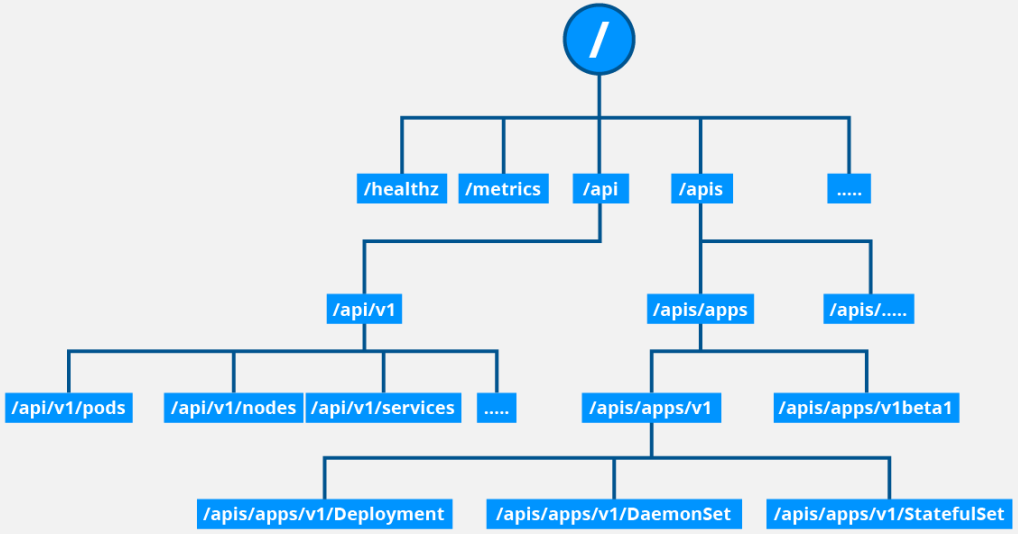
How to interact with Kubernetes cluster

KUBECTL COMMAND

kubectl is the Kubernetes Command Line Interface (CLI) client to manage cluster resources and applications. It is very flexible and easy to integrate with other systems, therefore it can be used standalone, or part of scripts and automation tools. Once all required credentials and cluster access points have been configured for ***kubectl***, it can be used remotely from anywhere to access a cluster.

The main component of the Kubernetes control plane is the API Server, responsible for exposing the Kubernetes APIs. The APIs allow operators and users to directly interact with the cluster. Using both CLI tools and the Dashboard UI, we can access the API server running on the control plane node to perform various operations to modify the cluster's state.

- Core group (/api/v1)
  - This group includes objects such as Pods, Services, Nodes, Namespaces, ConfigMaps, Secrets, etc.



EXAMPLE OF KUBECTL COMMANDS

List all node in the kubernetes cluster

```
$ kubectl get nodes
NAME                                          STATUS   ROLES    AGE   VERSION
gke-net4255-gke-net4255-gke-e1b8268b-4ckq     Ready    <none>   18h   v1.27.4-gke.900
```

List all pods in the kubernetes cluster

```
$ # show running pods in the namespace "vgauthier"
$ kubectl get pods --namespace=vgauthier -o wide
NAME                      READY   STATUS    RESTARTS   AGE     IP             NODE                                          NOMINATED NODE   READINESS GATES
webnodb-89cc76d75-ljcmk   1/1     Running   0          2m49s   10.152.1.119   gke-net4255-gke-net4255-gke-e1b8268b-4ckq   <none>           <none>
```

EXAMPLE OF DEPOYMENT

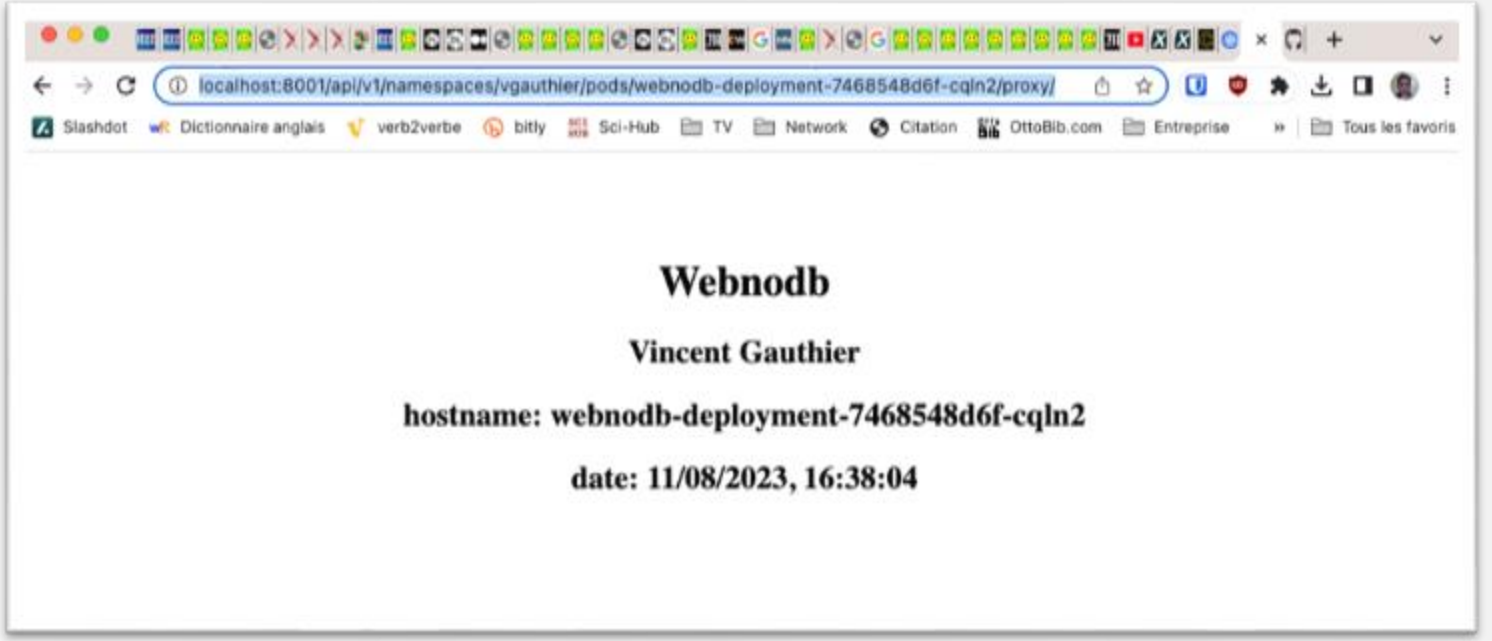
```
$ kubectl get deployments -n vgauthier
NAME                 READY   UP-TO-DATE   AVAILABLE   AGE
webnodb-deployment   3/3     3            3           107s
$ kubectl get pods -n vgauthier
NAME                                  READY   STATUS    RESTARTS   AGE
webnodb-deployment-7468548d6f-cqln2   1/1     Running   0          114s
webnodb-deployment-7468548d6f-gbzjg   1/1     Running   0          114s
webnodb-deployment-7468548d6f-l8vdc   1/1     Running   0          114s
$ kubectl get rs -n vgauthier
NAME                            DESIRED   CURRENT   READY   AGE
webnodb-deployment-7468548d6f   3         3         3       2m36s
```

DESCRIBE

```
$ kubectl describe deployment  webnodb-deployment -n vgauthier
Name:                   webnodb-deployment
Namespace:              vgauthier
CreationTimestamp:      Wed, 08 Nov 2023 17:23:16 +0100
Labels:                 app=webnodb
Annotations:            deployment.kubernetes.io/revision: 1
Selector:               app=webnodb
Replicas:               3 desired | 3 updated | 3 total | 3 available | 0 unavailable
StrategyType:           RollingUpdate
MinReadySeconds:        0
RollingUpdateStrategy:  25% max unavailable, 25% max surge
Pod Template:
  Labels:  app=webnodb
  Containers:
   webnodb:
    Image:        vgauthier/webnodb:latest
    Port:         5000/TCP
    Host Port:    0/TCP
    Environment:  <none>
    Mounts:       <none>
  Volumes:        <none>
Conditions:
  Type           Status  Reason
  ----           ------  ------
  Available      True    MinimumReplicasAvailable
  Progressing    True    NewReplicaSetAvailable
OldReplicaSets:  <none>
NewReplicaSet:   webnodb-deployment-7468548d6f (3/3 replicas created)
Events:
  Type    Reason             Age    From                   Message
  ----    ------             ----   ----                   -------
  Normal  ScalingReplicaSet  5m42s  deployment-controller  Scaled up replica set webnodb-
deployment-7468548d6f to 3
```

CHECK YOUR PODS DEPLOYEMENT

```
$ kubectl proxy
Starting to serve on 127.0.0.1:8001
```



How to interract with Docker

Docker File

- A file to manage the creation of the container:
  - FROM - Selection of the base image
  - RUN - Automatically execute commands when creating the container
  - COPY - Copy files from the local machine to the container
  - ENV - Set an environment variable
  - WORKDIR - Change the working directory
  - EXPOSE - Expose a port, 5000 for HTTP
  - CMD - Execute a command after the container is launched
  - …

```
# syntax=docker/dockerfile:1

FROM ubuntu:22.04
COPY . /app
RUN make /app
CMD python /app/app.py
```

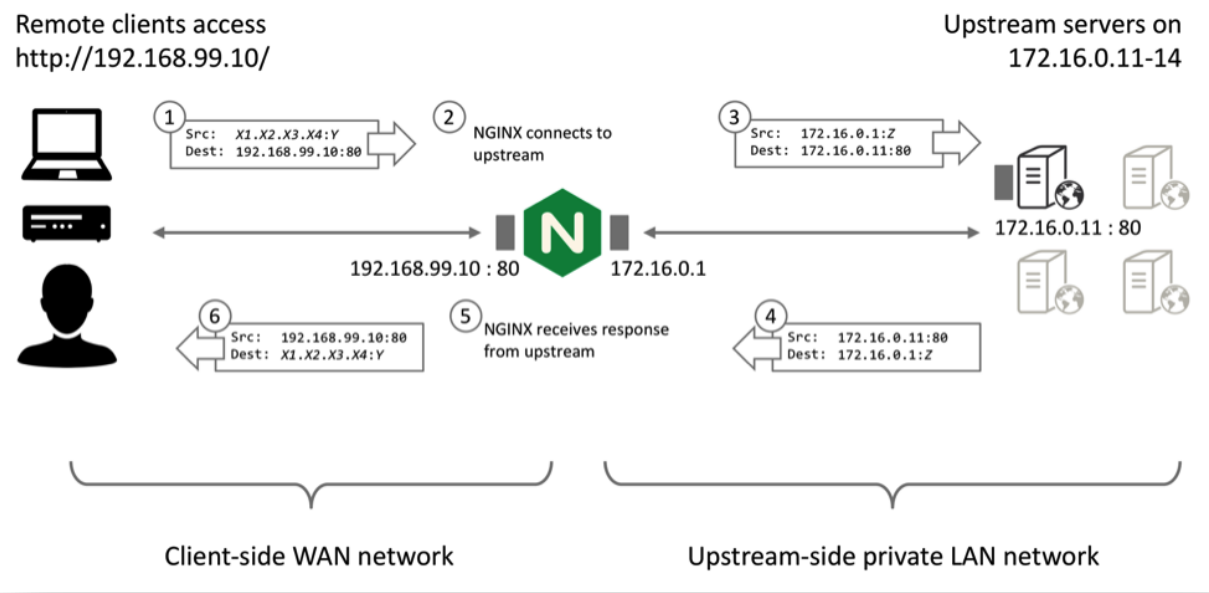
Docker Compose

- A file to manage the deployment of ONE or MANY containers:
  - SERVICES – List of containers to deploy
  - IMAGE – Choose which image to deploy
  - PORTS – Expose and map network ports between the container and the host machine.
  - VOLUME – Set the volume for each container - stockage
  - …

```
version: '3'
services:
 mon-site-web:
 container_name: mon-container
    image: nginx:latest
    ports:
      - "8080:80"
    volumes:
      - ./index.html:/usr/share/nginx/html/index.html
```

End of lecture

LOAD BALANCER



1. Remote clients make TCP connections or send UDP datagrams directly to the Load balancer/reverse proxy at its published IP address and port. NGINX Plus terminates the TCP connection or UDP session and reads the request data within.
2. The load balancer Plus then makes a new connection (or reuses an existing, idle connection) to the selected (load-balanced) upstream server.
3. When NGINX Plus writes the request to the upstream server, the connection originates from NGINX Plus's internal IP address.
4. When the upstream server responds to the request, it writes data to the NGINX Plus internal IP address.
5. NGINX Plus receives the response data on the upstream-side connection. It may process or modify the response (for example, apply compression to an HTTP response).
6. NGINX Plus then writes the response data on the client-side connection.

CREDITS

- **LinuxFoundationX: Introduction to Kubernetes**

  https://www.edx.org/learn/kubernetes/the-linux-foundation-introduction-to-kubernetes

- Kubernetes: Up and Running, 2017.