Software supply chain security for the cloud

Agathe Blaise (Thales), Jacopo Bufalino (CNAM)

Summary

- **Comparison:** Virtual Machines and Containers
  - Notions of virtualisation
  - Notions of system and application containers (Docker)
  - Security of containers

- **Security of cloud-computing** environments
  - Security best practices in Kubernetes
  - Kubernetes networking security
  - Role-based access control (RBAC)
  - Seccomp & AppArmor

Virtual Machines and Containers

{OPEN}

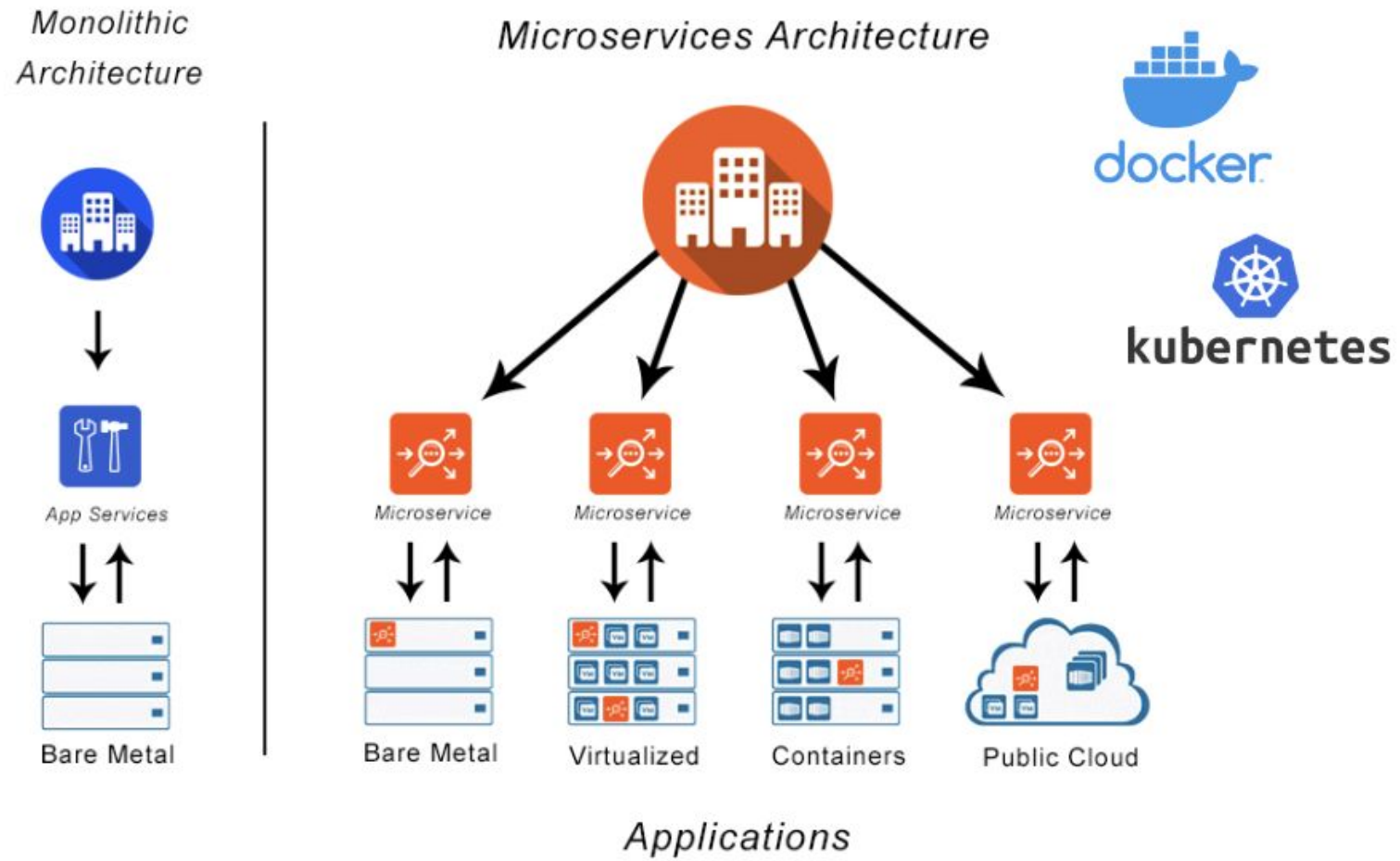
Virtual Machine (VM) vs. Container

Virtual Machines (VMs)

- Emulate both the HW layers and the OS (heavier but more secure)

Containers

- Emulate only the OS layer (lightweight)



{OPEN}

Containerization concept

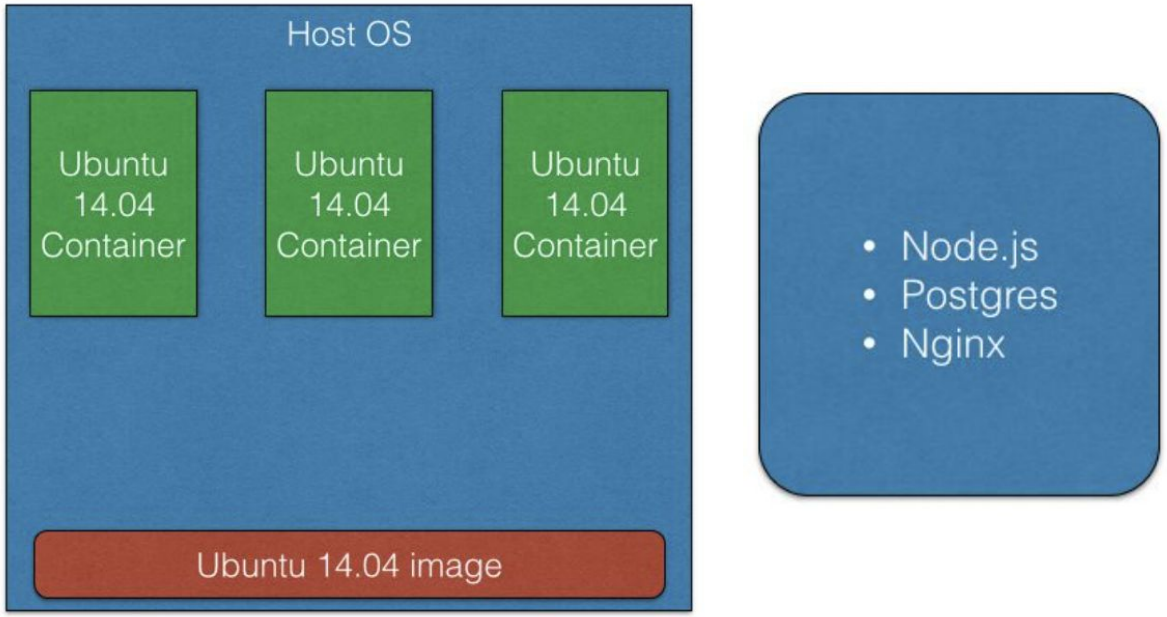
- Products of operating system virtualization
  - Provide a **lightweight virtual environment** that groups and isolates a set of processes and resources, such as memory, CPU, disk, … from the host
  - **Isolation** guarantees that any process inside the container cannot see any process or resource outside the container
  - However: Containers share the **same kernel** as the **host** system

- Use cases for containers
  - OS/system containers
  - Application containers

{OPEN}

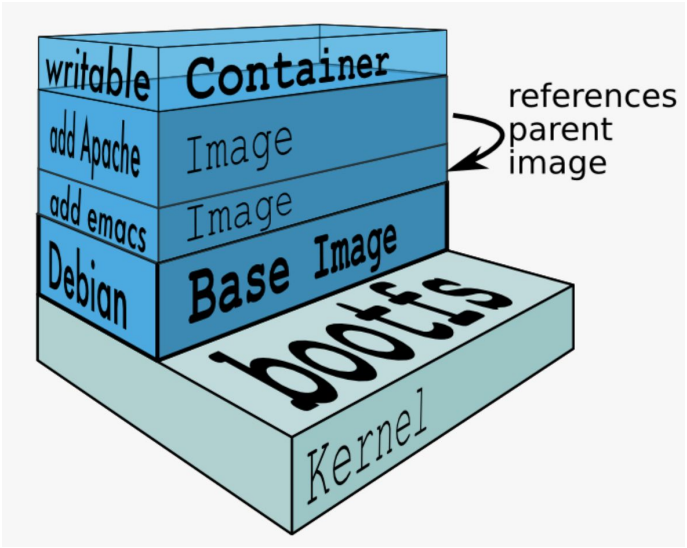
OS / System containers

- Virtual environments that share the kernel of the host operating system
  - Useful to provide user space isolation
  - Similar to the usage of VMs
  - Examples: LXC, OpenVZ, Linux Vserver, BSD Jails, Solaris Zones



Application containers

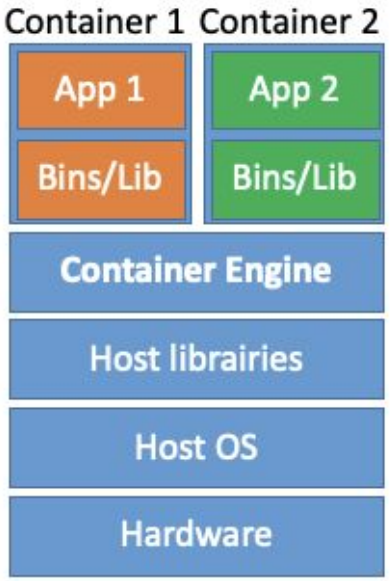
- Designed to package and run a single service
  - Useful to run a single process – running an application with multiple containers (one by application)
  - Combination of layers from instructions
  - Example: Docker, Rocket



{OPEN}

Containerization concept

- Share a single host OS and relevant librairies, drivers or binaries
- Linux namespaces (LXC), Docker containers, Rocket containers, etc..
- Applications containers vs. system containers



However:

- Acceleration of the development cycle (*DevOps – CI/CD*)
- Increase in complexity of the application stack (mostly web services and their frameworks)
- Security sacrified in the rush to the market

- R. K. Barik, R. K. Lenka, K. R. Rao and D. Ghose, "Performance analysis of virtual machines and containers in cloud computing," *ICCCA*, 2016, pp. 1204-1210.
- T. Combe, A. Martin and R. Di Pietro, "To Docker or Not to Docker: A Security Perspective," in *IEEE Cloud Computing*, vol. 3, no. 5, pp. 54-62, Sept. 2016.

Kubernetes orchestration

Kubernetes (K8s)

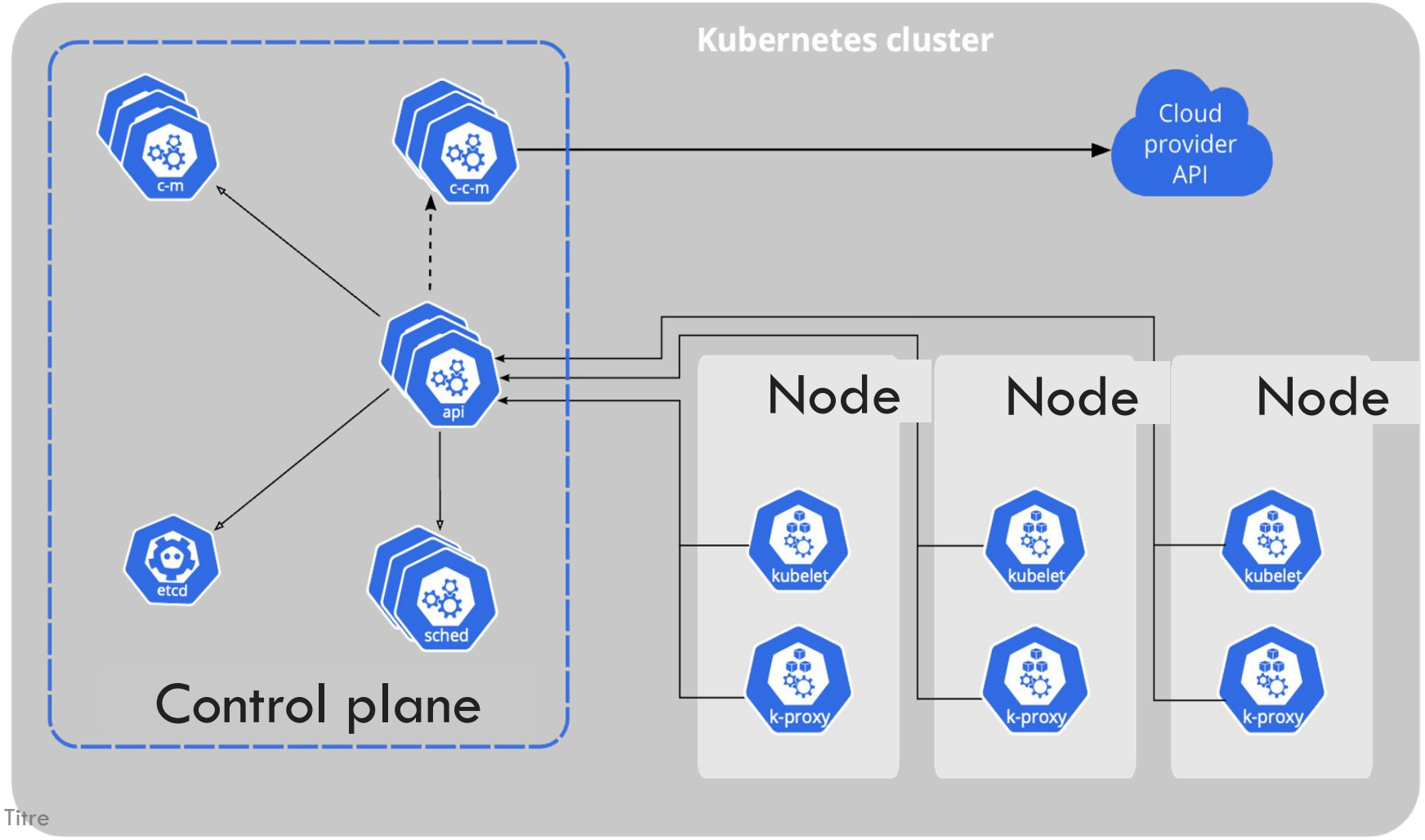
- Open-source container orchestration tool
- Released by Google in 2014
- Easy to deploy, scale, connect, manage multiple containers
- Kubernetes **cluster**: a set of master and worker nodes
  - **Master** node: brain keeping the cluster up and running
  - **Worker** nodes: execute workloads (e.g., application containers)
  - Usually, master and worker nodes are replicated for high availability

Kubernetes (K8s)

Different types of objects:

- **Container**: unit of software that bundles all resources (source code, libraries, other dependencies) required to run an application
- **Pod**: minimum deployable unit in a cluster that contains one or more (similar) containers with shared storage and network resources
  - Used for scalability
- **Deployment**: collection of pods deployed on worker nodes for scalability and performance reasons

Kubernetes Architecture



API server

Cloud controller manager *(optional)*

Controller manager

etcd (persistence store)

kubelet

kube-proxy

Scheduler

Control plane

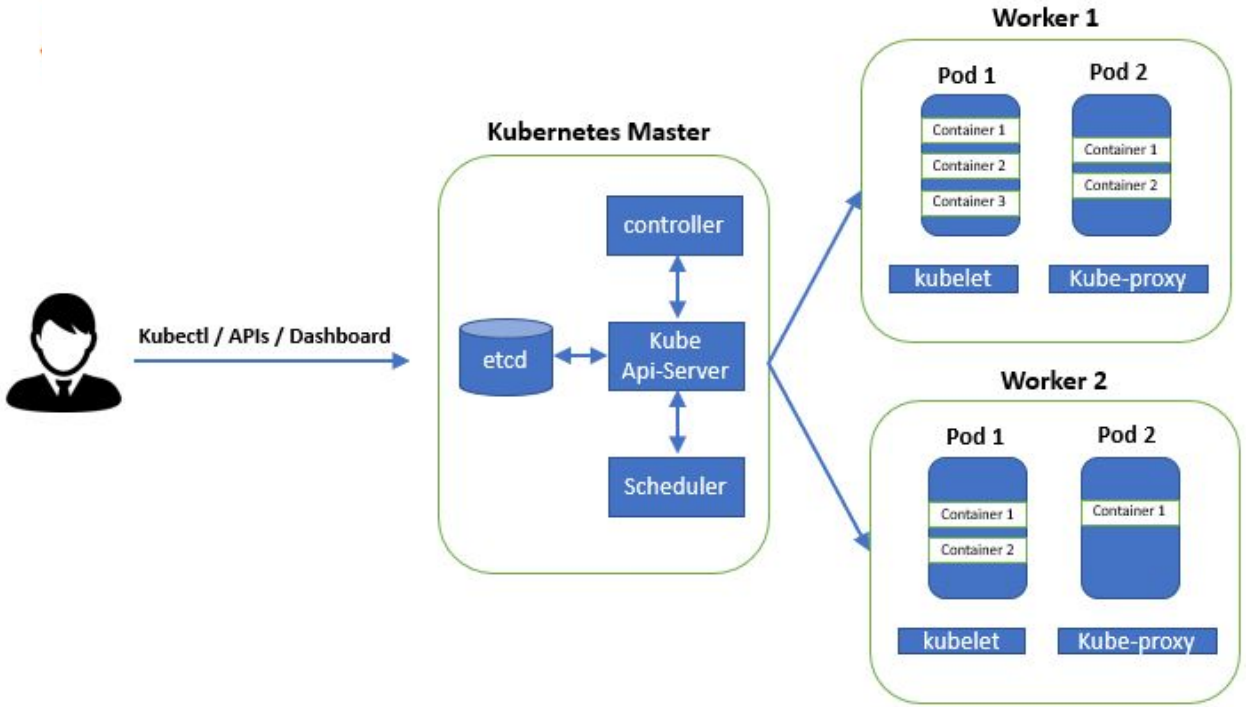
Node

{OPEN}

Kubernetes Architecture

Master Node :
- **Controller:** ensures the correct number of resources are available in the cluster.
- **etcd:** key-value store.
- **Scheduler:** assigns pods to nodes.
- **Kube-apiserver:** exposes the Kubernetes API.

Worker Node :
- **Kube-proxy:** manages network rules and forwards traffic.
- **Kubelet:** ensures the operation of containers within pods.



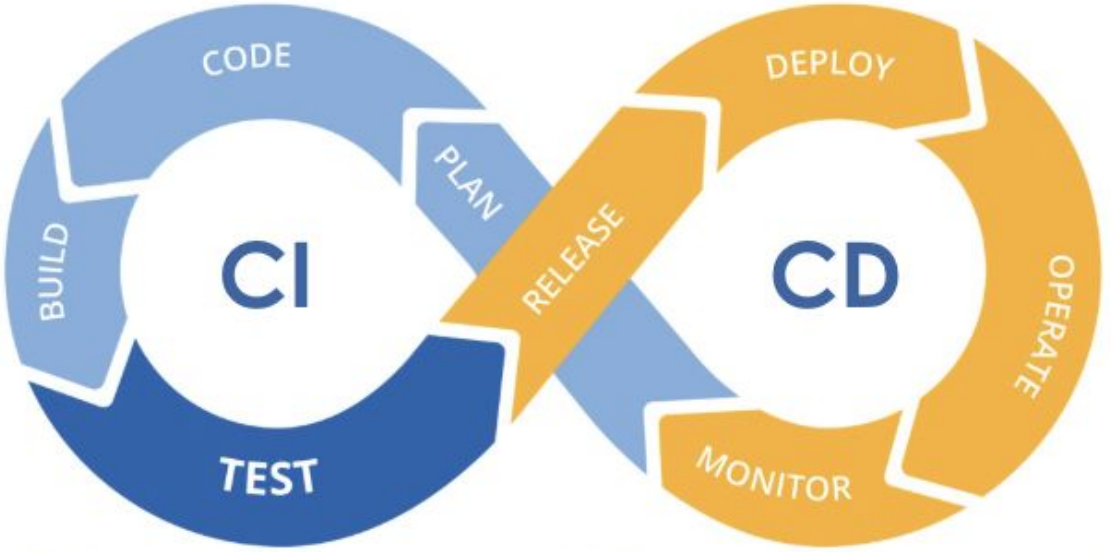
{OPEN}

Foundations of Kubernetes security

DevOps for Kubernetes

4) Kubernetes manifests managing containers and other resources

1) Application code within the container



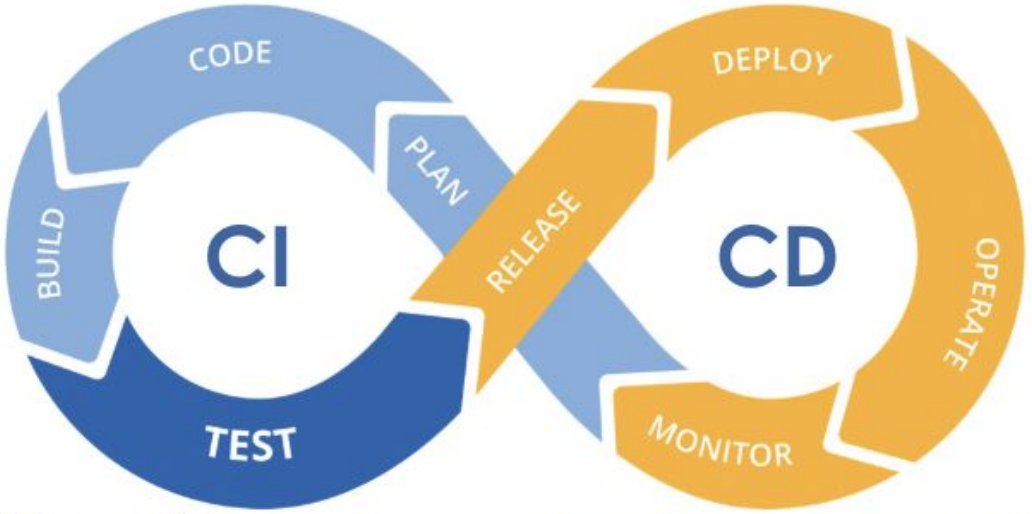
2) Building the container image

5) Deploy to production

3) Run unit tests, generate artifacts

6) Data collection and logging, real-time monitoring

DevOps for Kubernetes



1) Application code within the container
/!\ insecure code, CVEs

2) Building the container image
/!\ vulnerable base images, unsafe Dockerfiles

3) Run unit tests, generate artifacts
/!\ tampered dependencies, unverified artifacts

4) Kubernetes manifests managing containers and other resources
/!\ insecure configuration, overprivileged pods, exposed services

5) Deploy to production
/!\ supply chain attacks

6) Data collection and logging, real-time monitoring
/!\ undetected breaches, zero-day attacks

Best practices in Kubernetes security

→ We should define **a set of best practices** for Kubernetes supply-chain:

1. Networking

Network Policies to restrict pod-to-pod communication.

2. Authentication & Authorization

Role-based Access Control (**RBAC**) and **service accounts**

3. Cryptography

TLS encryption for K8s API traffic, secrets stored in **etcd** with encryption

4. Secrets management (client credentials, application-specific secrets)

Secrets **never logged or stored** outside of explicit, user-supplied locations

5. Multi-tenancy isolation:

Pod Security Policies, AppArmor and **Seccomp** profiles for syscall filtering

Kubernetes networking

{OPEN}

Kubernetes networking

Several types of Kubernetes Services binding to Pods:

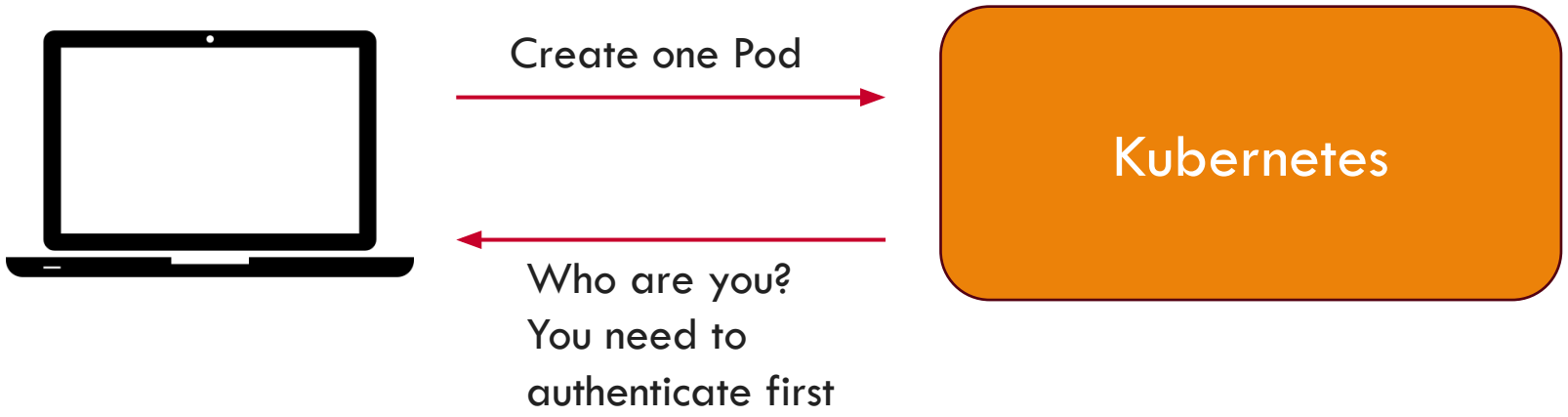
- **ClusterIP** (by default): exposes the Service only inside the cluster, making it accessible to other Pods but not from external clients.
- **NodePort**: exposes the Service on a static port (30000-32767) on each node's IP, allowing external access to the cluster using NodeIP:NodePort.
- **LoadBalancer**: provisions an external load balancer (typically from a cloud provider) to expose the Service to the Internet. It assigns a public IP to distribute traffic to backend Pods
- **Ingress**: not a service but an API object that manages HTTP(S) traffic routing. It provides a single entry point for multiple Services, supports **path-based and host-based routing**, and can handle SSL termination.

Refer to Jacopo's slides (14.03.2025) for more details

Role-based Access Control (RBAC)

Authentication in Kubernetes

Authentication is **the process of verifying a user's identity** before granting them access to a system of resource



/!\ Kubernetes does not manage the user accounts natively

Normal users **cannot** be added to a cluster through an API call

~~`kubectl create user alice`~~

{OPEN}

Authentication in Kubernetes

Kubernetes supports several authentication methods:

- Client Certificates
- Static Token Authentication
- Service Account Tokens

{OPEN}

Authentication in Kubernetes

- Static Token File

API server reads bearer tokens from a file provided

```
root@control-plane:~# cat /root/token.csv
Dem0Passw0rd#,bob,01,admins
```

Minimum of 3 columns: token, user name, user uid

Passed to "--token-auth-file"

```
[Service]
ExecStart=/usr/local/bin/kube-apiserver --advertise-address=165.22.212.16 --etcd-cafile=/root/certificates/ca.crt --etcd-cert
file=/root/certificates/etcd.crt --etcd-keyfile=/root/certificates/etcd.key --etcd-servers=https://127.0.0.1:2379 --service-a
ccount-key-file=/root/certificates/service-account.crt --service-cluster-ip-range=10.0.0.0/24 --service-account-signing-key-f
ile=/root/certificates/service-account.key --service-account-issuer=https://127.0.0.1:6443 --token-auth-file /root/token.csv
```

Authentication in Kubernetes

- X509 certificates

Uses the client certificates for authentication

client-certificate-data encoded in base 64

Can be decoded → in the form of a **X509 certificate**

Contains informations about:

- Common Name
- Organization
- Locality, State & Country
- Validity
- …

Authentication in Kubernetes

Kubernetes clusters have two categories of users:

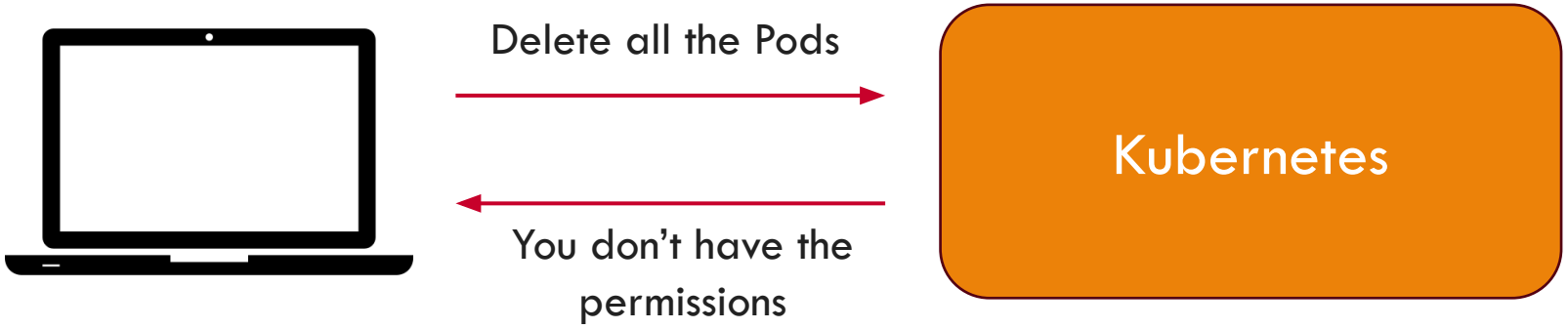
Normal users (for humans)
Certificate-based authentication
primarily used

Service accounts (for apps)
Using service accounts and appropriate tokens
→ will be later studied

{OPEN}

Authorization in Kubernetes

Authorization is the **process of determining what an authenticated user or entity is allowed to do**

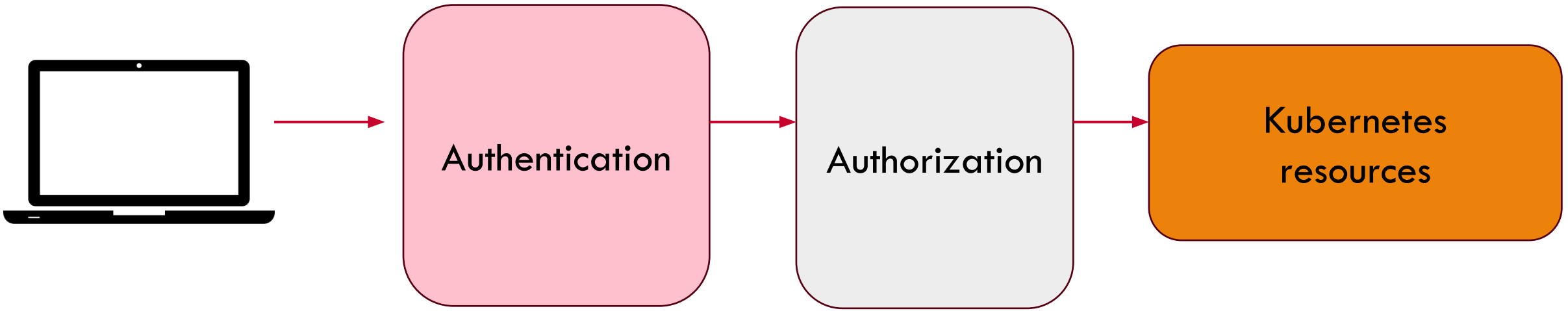


{OPEN}

Authorization in Kubernetes

Kubernetes **authorization** takes place **following authentication**

A client making a request must be authenticated (logged in) before its request can be allowed



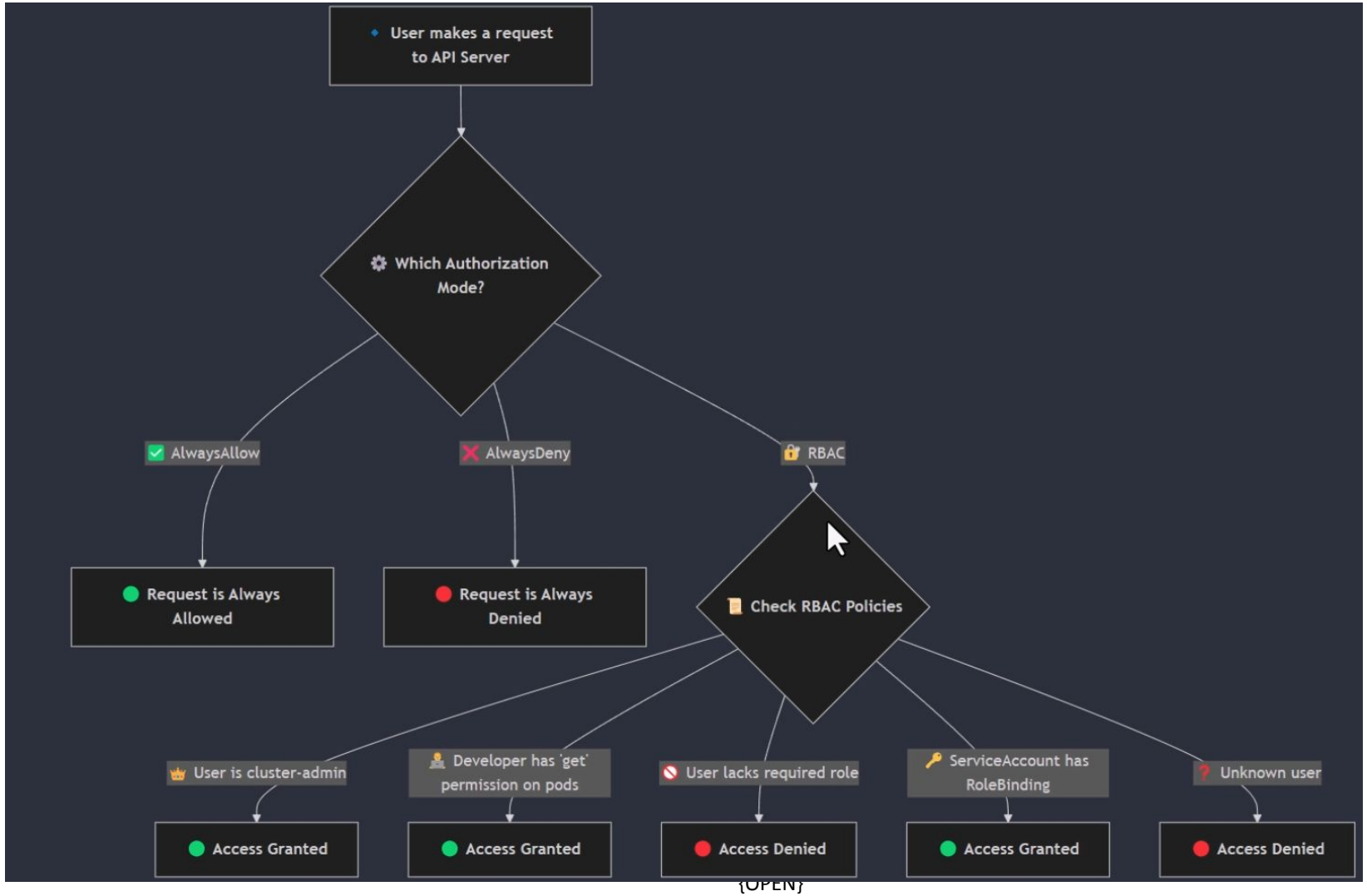
{OPEN}

Authorization in Kubernetes

The Kubernetes API server may authorize a request using one or several authorization modes. Some of them include:

Authorization Mode	Description
AlwaysAllow	This mode allows all requests, which brings security risks. Use this authorization mode **only for testing**.
AlwaysDeny	This mode blocks all requests. Use this authorization mode **only for testing**.
Role-Based Access Control (RBAC)	Defines set of permissions based on which access is granted. Recommended for **production environments**.

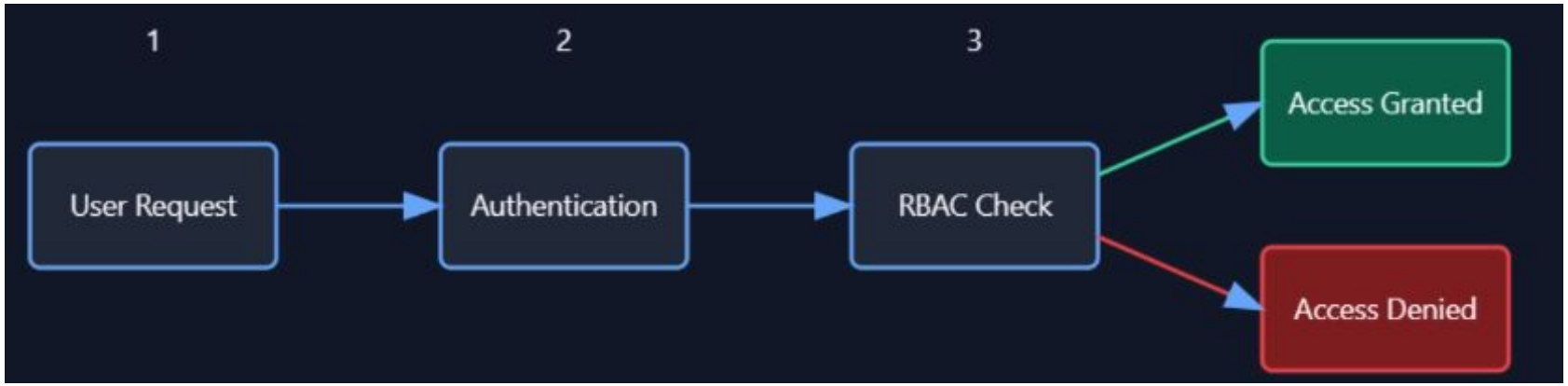
Authorization in Kubernetes



{OPEN}

Role-Based Access Control

RBAC allows us to control what actions users and service accounts can perform on resources with your cluster



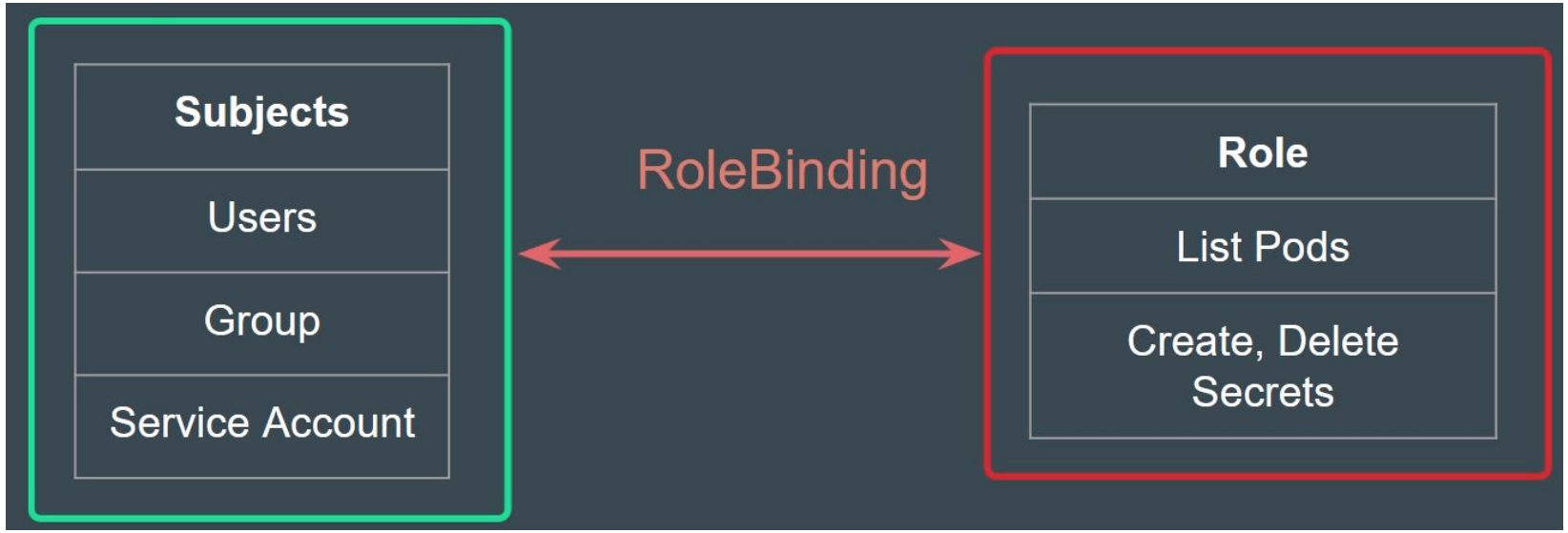
{OPEN}

Role-Based Access Control

Subjects can be users, groups, services accounts.

Roles defines a set of permissions.

RoleBindings tie the permission defined in the Role to Subjects (e.g., user, group, or service accounts).



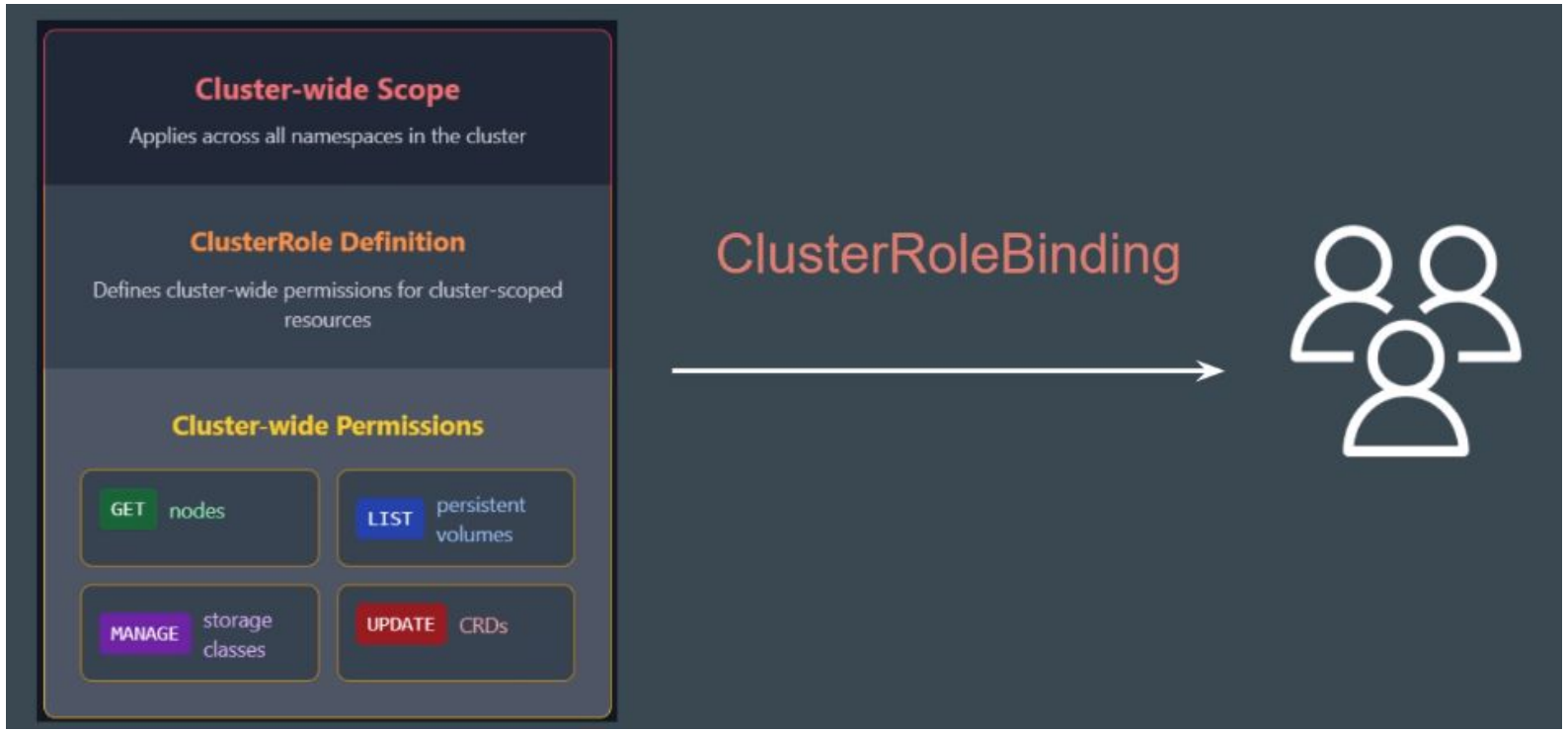
/!\ A Role always sets permissions **within a particular namespace.**

{OPEN}

ClusterRole and ClusterRoleBinding

Similar to Role and RoleBinding, but the main differences is that the permissions granted by a ClusterRole apply across all namespaces in the cluster.

ClusterRoleBinding connects ClusterRole to Subjects



{OPEN}

AppArmor profiles

Discretionary Access Control (DAC)

DAC allows restricting access to objects based on the identity of subjects and/or groups to which they belong.

```
agatheblaise@Agathes-MBP ~ % minikube ssh "ls -l /etc/kubernetes/"
total 44
drwxr-xr-x 2 root root 4096 Mar  7 13:04 addons
-rw------- 1 root root 5647 Mar  7 13:03 admin.conf
-rw------- 1 root root 5652 Mar  7 13:03 controller-manager.conf
-rw------- 1 root root 1971 Mar  7 13:03 kubelet.conf
drwxr-xr-x 1 root root 4096 Mar  7 13:03 manifests
-rw------- 1 root root 5604 Mar  7 13:03 scheduler.conf
-rw------- 1 root root 5671 Mar  7 13:03 super-admin.conf
```

Challenges: DAC allows programs to inherit the full permissions of the user running them. If a user can access sensitive files, any program they run (including malware) can access those same files.

Sample use case

You have a binary file that performs some basic operation on server like deleting old log files to cleanup resources.

Suddenly you have seen that binary file is connecting to the Internet and sending network traffic.

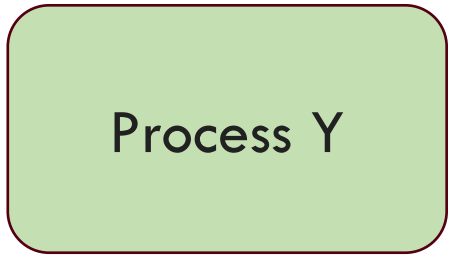
{OPEN}

Mandatory Access Control (MAC)

MAC is a security model in which access to resources is strictly regulated by a central authority based on predefined security policies.

Two important concepts: Confined (restricted) and Unconfined (not restricted)

Confined



Not confined

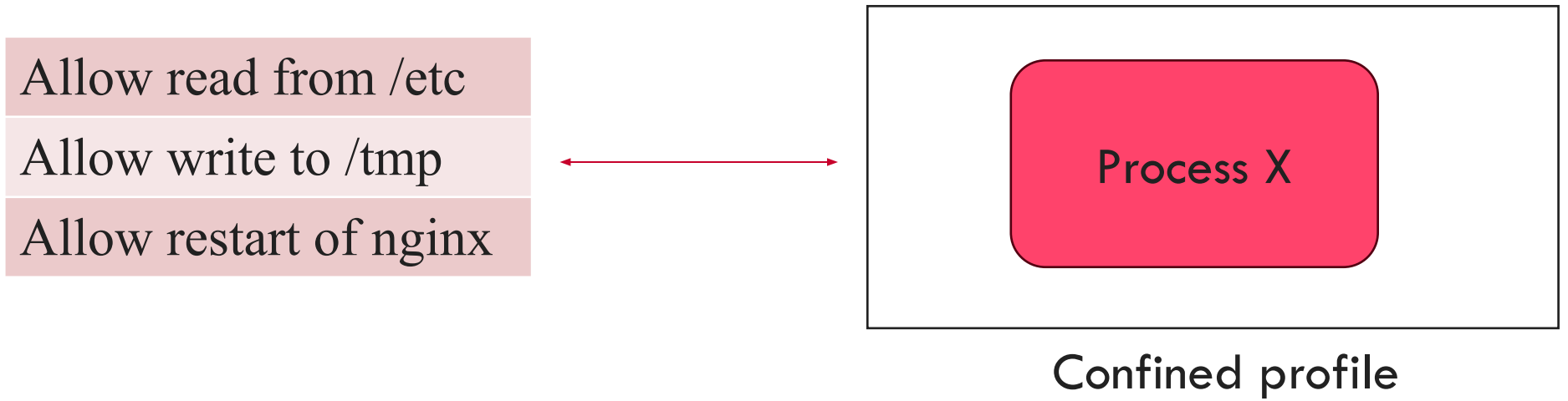
{OPEN}

Confined Process

Confined Processes are restricted.

Everything that process intends to do must be listed in a profile.

If the capability is not listed in the profile, the process will not be allowed to run that.



{OPEN}

AppArmor profiles (one MAC system)

Certain set of operations are allowed

```
docker@minikube:~$ cat /etc/apparmor.d/abstractions/ubuntu-helpers
profile sanitized_helper {
  include <abstractions/base>
  include <abstractions/X>

  # Allow all networking
  network inet,
  network inet6,

  # Allow all DBus communications
  include <abstractions/dbus-session-strict>
  include <abstractions/dbus-strict>
  dbus,

  # Needed for Google Chrome
  ptrace (trace) peer=**//sanitized_helper,

  # Allow exec of anything, but under this profile. Allow transition
  # to other profiles if they exist.
  /{usr/,usr/local/,}{bin,sbin}/* Pixr,

  # Allow exec of libexec applications in /usr/lib* and /usr/local/lib*
  /usr/{,local/}lib*/{,**/}* Pixr,
```

While some are forbidden

```
  # Full access
  / r,
  /** rwkl,
  /{,usr/,usr/local/}lib{,32,64}/{,**/}*.so{,.*} m,

  # Dangerous files
  audit deny owner /**/* m,              # compiled libraries
  audit deny owner /**/*.py* r,          # python imports
```


{OPEN}

Primary modes of AppArmor

Modes	Description
Enforce	Actively **enforces** the defined AppArmor security profile.
Complain	Violations are **logged**, but the application runs normally without restrictions.
Unconfined	The application runs **without** AppArmor restrictions.

{OPEN}

AppArmor (Kubernetes)

- Kubernetes allows you to apply AppArmor profiles to Pods and containers

```yaml
apiVersion: v1
kind: Pod
metatada:
  name: hello-apparmor
spec:
  securityContext:
    appArmorProfile:
      type: Localhost
      localhostProfile: k8s-apparmor-example-deny-write
  containers:
  - name: hello
    image: busybow
    command: [ "sh", "-c", "echo 'Hell AppArmor!' && sleep 1h" ]
```

AppArmor profile types available

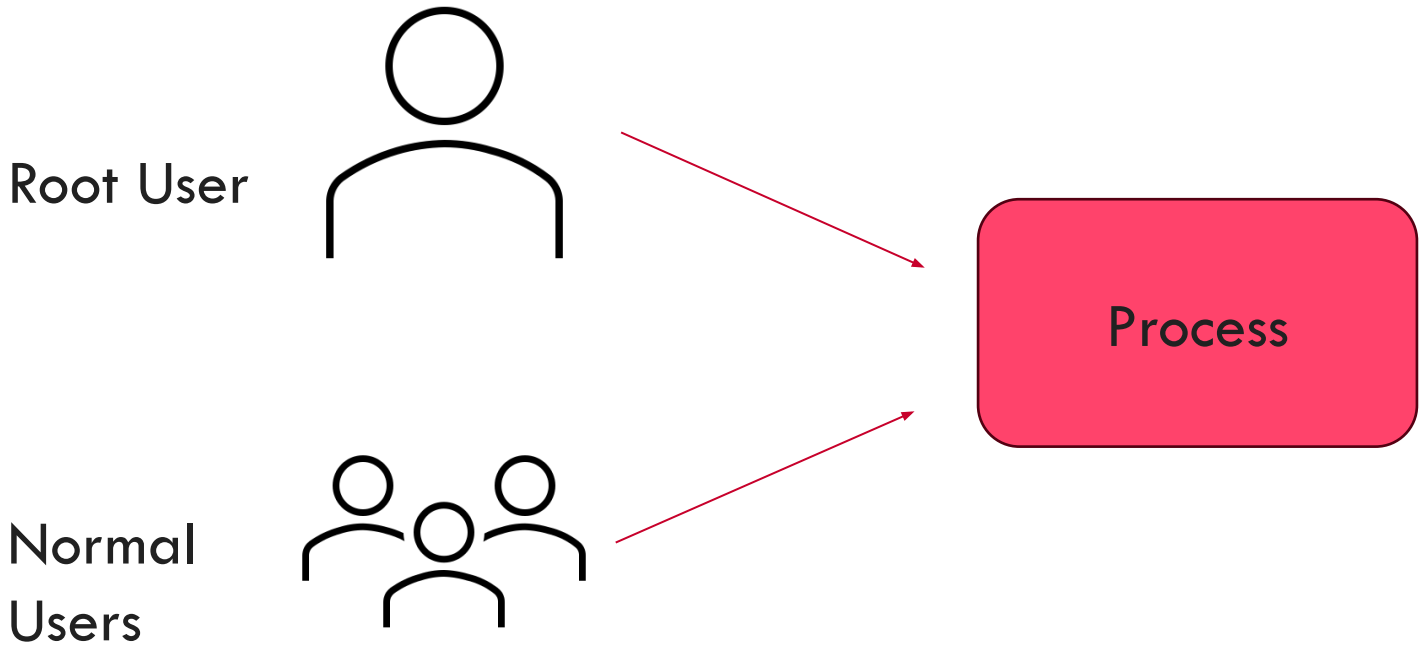
Profile Type	Description
RuntimeDefault	To use the runtime's **default** profile.
LocalHost	Uses a **custom** security profile stored on the node's filesystem.
Unconfined	To run **without** AppArmor

{OPEN}

Linux capabilities

Setting the base (system calls)

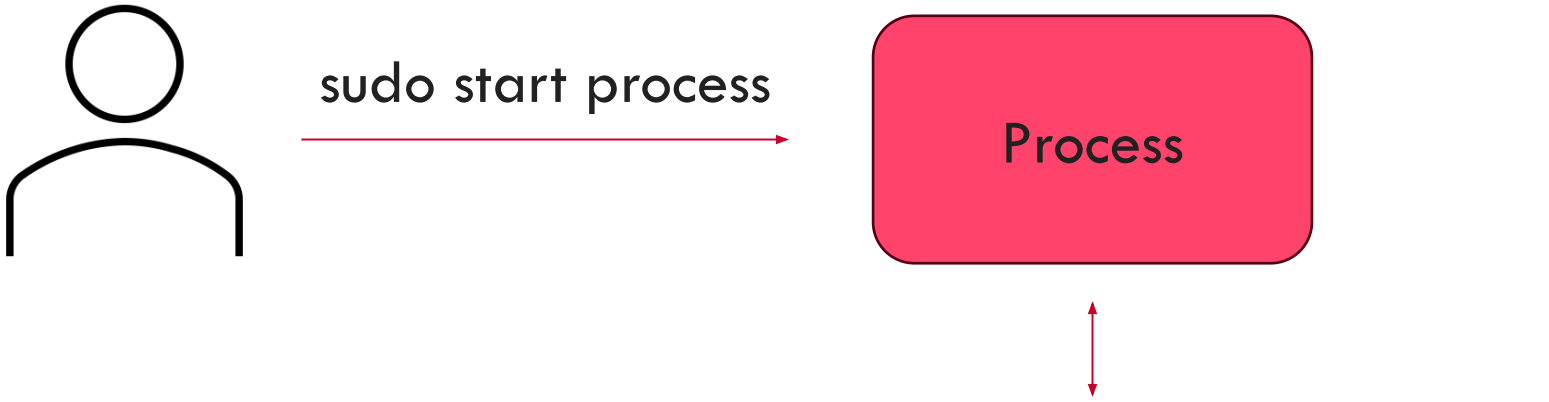
- Processes on a Unix-like system runs primarily with the permissions of either a user account, or with root permissions.



{OPEN}

Understanding the Challenge

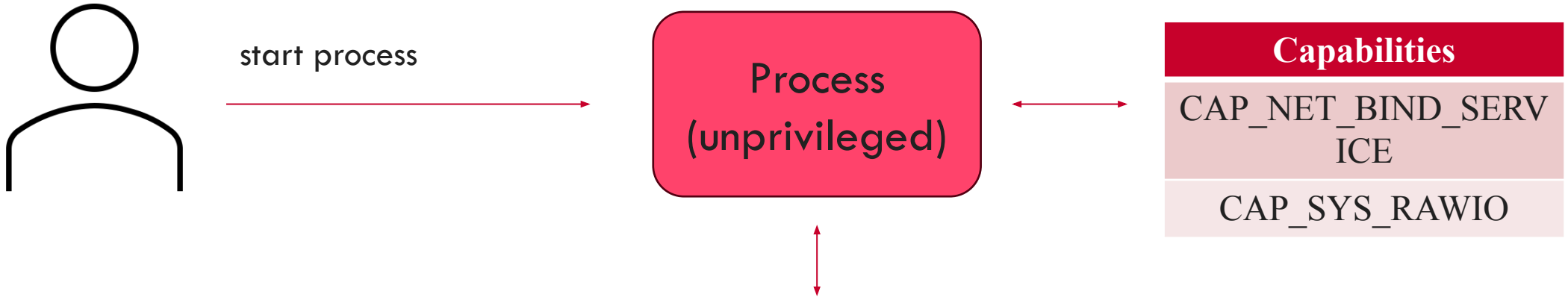
- This approach of root user having unlimited privilege and a normal user having limited privilege is not a good enough model.
- User John wants to start the process that requires certain privileged access. Most organizations will grant John full access using sudo.



{OPEN}

Introducing Linux capabilities

- Linux capabilities are used to allow binaries (executed by non-root users) to perform privileged operations without providing them all root permissions
- It also allows process started with root to have limited privileged



{OPEN}

Capabilities available

- There are wide range of capabilities available by default.
- 42 distinct capabilities as of Linux kernel 6.1

Capabilities	Description
CAP_NET_BIND_SERVICE	Bind to port <1024
CAP_SYS_RAWIO	Use raw sockets
CAP_SYS_TIME	Modify system clock
CAP_SYS_ADMIN	Perform various administrative tasks
CAP_DAC_OVERRIDE	Bypass file permissions

{OPEN}

Seccomp profiles

{OPEN}

System calls (syscalls)

- A syscall is a way for a user-space program (like a web browser or a database) to request a service from the **operating system's kernel.**

Since normal programs **cannot directly access hardware or critical system resources,** they must ask the kernel to do things like:

- Reading/writing files
- Allocating memory
- Creating network connections
- Executing other programs

{OPEN}

System calls (syscalls)

- **Example:**

```
with open("example.txt", "r") as file:
    content = file.read()
```

- System calls in action can be monitored using **strace** (Linux debugging tool)
- In this case, will show 2 system calls handling the file operation: *open*() and *read*()

{OPEN}

Seccomp (secure computing) profiles

- Restricts system calls a process can make, preventing it from executing unwanted or dangerous operations.
- **How it works:** you define a seccomp **profile** that allows or blocks specific system calls.
- **Example:**

```
{
  "defaultAction": "SCMP_ACT_ALLOW",
  "syscalls": [
    {
      "names": ["settimeofday"],
      "action": "SCMP_ACT_ERRNO"
    }
  ]
}
```

Summary

1. AppArmor Profiles: File and Resource Restrictions

AppArmor is a Mandatory Access Control system that controls what files, network resources, and capabilities a process can access.

- *How it works*: defining profiles that specify: 1) what files a process can read/write/execute, 2) what capabilities it is allowed to use, 3) whether it can access the network.

2. Linux Capabilities: Fine-Grained Privileges

Instead of giving full root access, capabilities allow a process to perform specific privileged actions (e.g., binding to ports <1024, modifying system time).

- *How it works*: a process is given only the capabilities it needs, reducing security risks.

3. Seccomp Profiles: System Call Restrictions

Seccomp (Secure Computing Mode) restricts system calls a process can make, preventing it from executing unwanted or dangerous operations.

- *How it works*: defining a seccomp profile that allows or blocks specific system calls.

Summary

Feature	What it controls	Example rule
AppArmor	What files, network, and capabilities a process can access	Allow access to `/var/www`, deny `/etc/shadow`
Capabilities	What privileged actions a process can perform	`CAP_SYS_TIME` allows modifying the system clock
Seccomp	What system calls a process can make	Block `settimeofday()` even if it has `CAP_SYS_TIME`