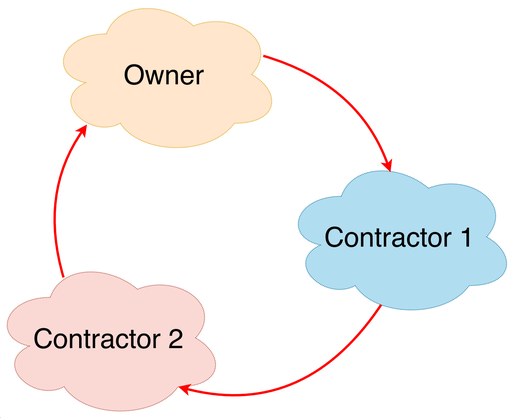
Securing Workflows using the Microservices Architecture

Loïc Miller, Pascal Mérindol, Antoine Gallais, Cristel Pelsser

Université de Strasbourg

ICUBE

Context



- We define a **workflow** as a sequence of tasks processed by a set of actors.
- The instigator of the workflow (the **owner of the data**) interacts with **contractors** to realize a task.
- Data leaks can occur in an unsecure environment, by **eavesdropping** or **intentional leaks** by the actors.
- As data can be hosted by third parties, it needs to be **encrypted**.

Objectives

How can we enforce a given workflow, which guarantees **data security at rest** and **in transport**, and **prevents data leaks**?

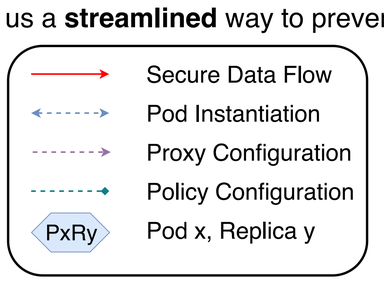
We propose and show how to use the **microservices architecture** to ensure those properties.
Our goal is to make an architecture that is **multi-tenant** and follows a **generic** design.
We also want the architecture to be **easily configurable**, **deployable** and **testable**, by using pre-existing blocks.

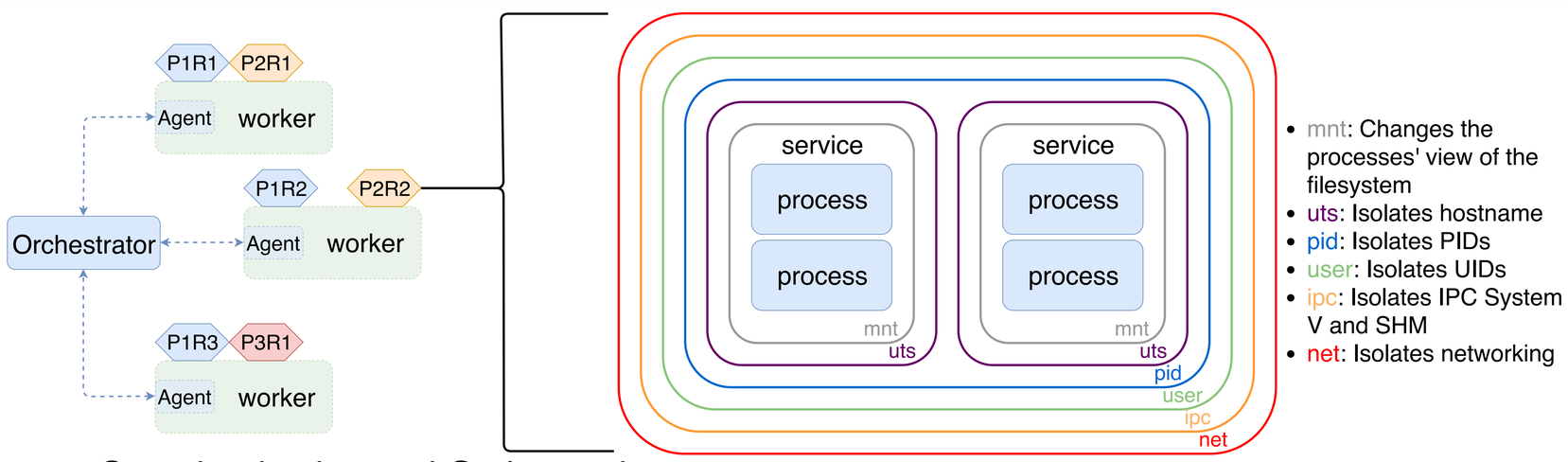
- Service Mesh
- Orchestration
- Containerization

Building Blocks

Container: Standard unit of software packaging application code (**service**) and its dependencies in an isolated environment.
Orchestrator: System to automate the management of containers and their lifecycles. The orchestrator does so by interacting with **agents** running on physical machines (**workers**), which control groups of containers (**pods**).

Using containers grants us portability and a **standardized environment**.
This aspect, along with the fact that container communications can be constrained and monitored gives us a **streamlined** way to prevent data leaks.



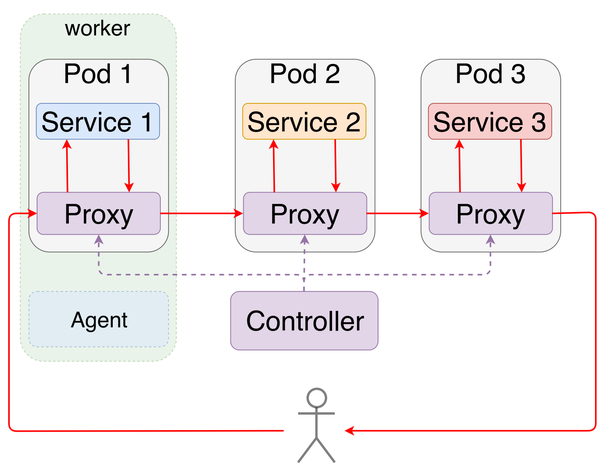


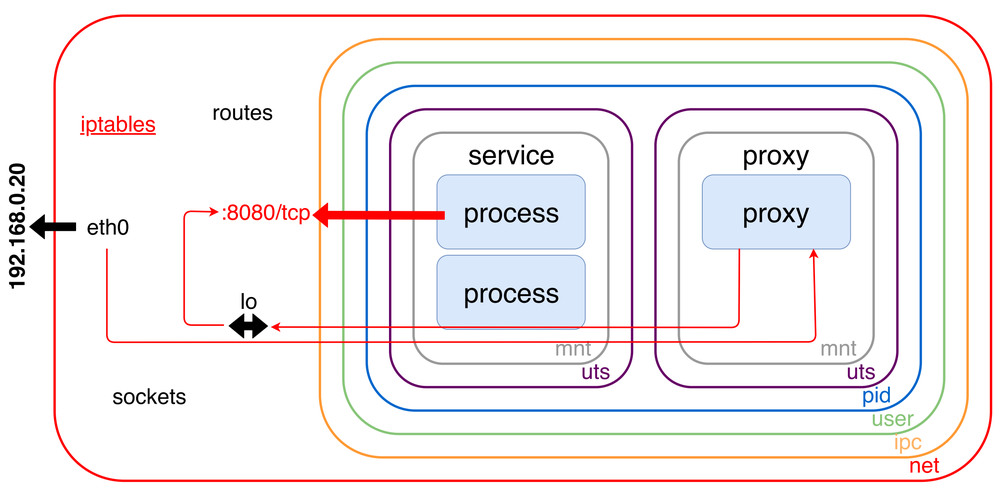
- mnt: Changes the processes' view of the filesystem
- uts: Isolates hostname
- pid: Isolates PIDs
- user: Isolates UIDs
- ipc: Isolates IPC System V and SHM
- net: Isolates networking

Containerization and Orchestration

Service Mesh: System to automate the communication, security and monitoring of containerized services. The **controller** configures the **proxies** (routing, security, monitoring, ...).

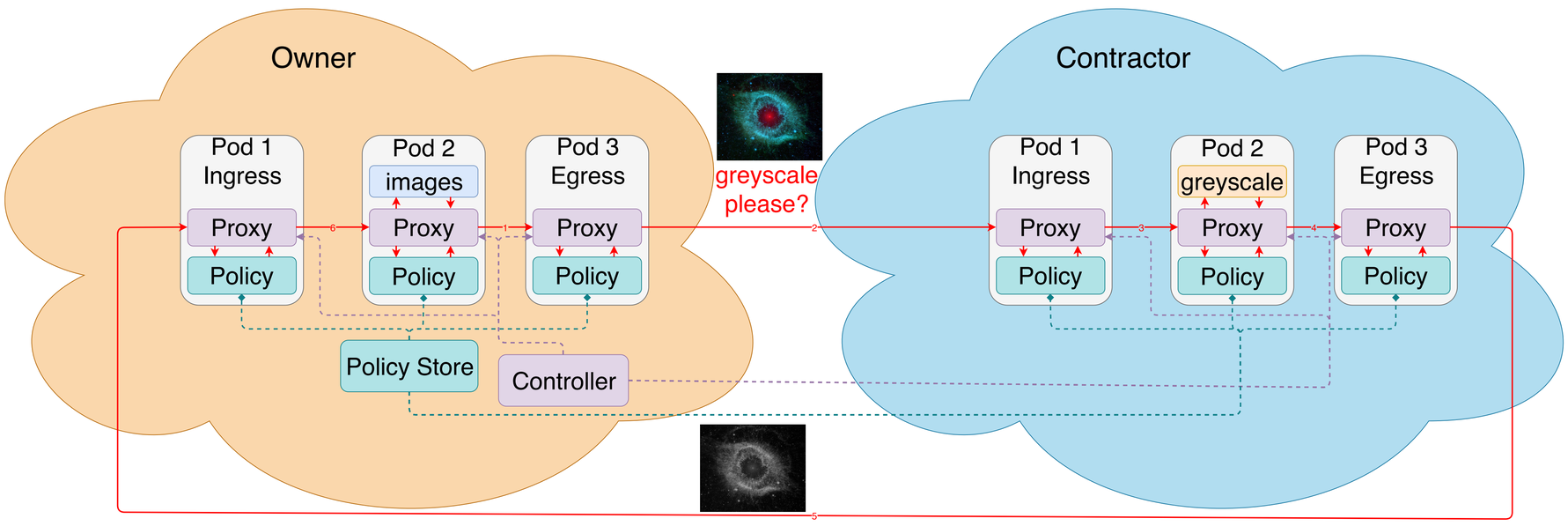
- Proxies intercept ingoing/outgoing traffic to their service (**iptables**).
- Controller has a Certificate Authority (CA).
- Proxies generate a **key pair** and associate their identity to a certificate via the CA.
- Key pair is used to communicate securely between services via **mTLS**.





Service Mesh

Putting It Together



- A Policy **sidecar** is used for authorization.
- Policies are periodically pulled from the Policy Store.
- Proxy checks Policy to **authorize requests**.
- Ingress and Egress pods are used as **gateways** to better control communications

Controller + Proxy = **Identity + Authentication + Encryption** ⟶ **Security at rest / in transport**

Policy Store + Policy = **Authorization** ⟶ **Leak prevention**

Policy Language: Rego

```
default allow = false
allow {
  required_roles[r]
  ...
}

required_roles[r] {
  perm := role_perms[r][_]
  perm.method = http_request.method
  perm.path = http_request.path
}

role_perms = {
  "images": [
    {"method": "PUT", "path": "/greyscale"},
  ],
  "greyscale": [
    {"method": "PUT", "path": "/images"},
  ]
}
```

Only allows communication from images to greyscale and greyscale to images using the PUT method.

Conclusion & Challenges

Using the microservices architecture can help us secure workflows by providing **Identity**, **Authentication**, **Authorization**, as well as means to operate in a **potentially insecure environment**.

Projects already exist to help us implement this infrastructure:

- Containers (Docker (Docker, 2019), containerd, ...)
- Orchestrators (Kubernetes (Kubernetes, 2019), Nomad, ...)
- Service Meshes (Istio (Istio, 2019) + Envoy/NGINX/HAProxy/...)
- General Purpose Policy Engines (Open Policy Agent)

In future works, we plan to investigate policy **placement** (Lipton and Moser, 2013; Ranathunga et al., 2016a), policy **migration** (Barrere, Badonnel and Festor, 2013) and policy **testing** (Ranathunga et al., 2016b; Schnepf et al., 2017) in the microservices environment.

References

Barrere, M., Badonnel, R., & Festor, O. (2013). **Vulnerability assessment in autonomic networks and services: a survey.** *IEEE communications surveys & tutorials*, *16*(2), 988-1004.

Docker. (2019). *Docker*. [online] Available at: https://www.docker.com/ [Accessed 11 Jun. 2019].

Istio. (2019). *Istio*. [online] Available at: https://istio.io/ [Accessed 11 Jun. 2019].

Kubernetes. (2019). *Kubernetes*. [online] Available at: https://kubernetes.io/ [Accessed 11 Jun. 2019].

Lipton, P. and Moser, S. (2013). ***Topology and Orchestration Specification for Cloud Applications Version 1.0.*** [online] docs.oasis-open.org. Available at: https://docs.oasis-open.org/tosca/TOSCA/v1.0/os/TOSCA-v1.0-os.pdf [Accessed 11 Jun. 2019].

Ranathunga, D., Roughan, M., Kernick, P., & Falkner, N. (2016a, July). **The Mathematical Foundations for Mapping Policies to Network Devices.** In *SECRYPT* (pp. 197-206).

Ranathunga, D., Roughan, M., Kernick, P., Falkner, N., Nguyen, H. X., Mihailescu, M., & McClintock, M. (2016b, July). **Verifiable Policy-defined Networking for Security Management.** In *SECRYPT* (pp. 344-351).

Schnepf, N., Badonnel, R., Lahmadi, A., & Merz, S. (2017, July). **Automated verification of security chains in software-defined networks with Synaptic.** In *2017 IEEE Conference on Network Softwarization (NetSoft)* (pp. 1-9). IEEE.

This work has been made possible in part by a grant from the Cisco University Research Program Fund, an advised fund of Silicon Valley Foundation.