ORSYS formation

Formation :

Containers, enjeux, usages et solutions

Formateur

Ahmed Hosni

Technology manager

Contact: ahmedhosni.contact@gmail.com

ORSYS formation

FORMATION & CONSULTING INFORMATION TECHNOLOGY

Containers, enjeux, usages et solutions

Un État de l'art des solutions d'orchestration de containers et de leur écosystème pour mettre en œuvre une plateforme de type CaaS (Container as a Service).

Il apporte des réponses sur le fonctionnement, la mise en place ou l'utilisation de containeurs dans une organisation et apporte des conseils pour leur usage.

Participants

Architectes, responsables des infrastructures IT, chefs de projet, administrateurs système et/ou réseau ou développeurs.

OBJECTIFS PEDAGOGIQUES

1. Détailler les différents aspects de la technologie de containerisation, son écosystème
2. Découvrir le fonctionnement de Kubernetes, ses composants internes et externes
3. Comprendre les interactions avec le Cloud privé/public et le legacy.
4. Appréhender les bénéfices et les limites des architectures micro-services en termes techniques et organisationnels

Les fondamentaux

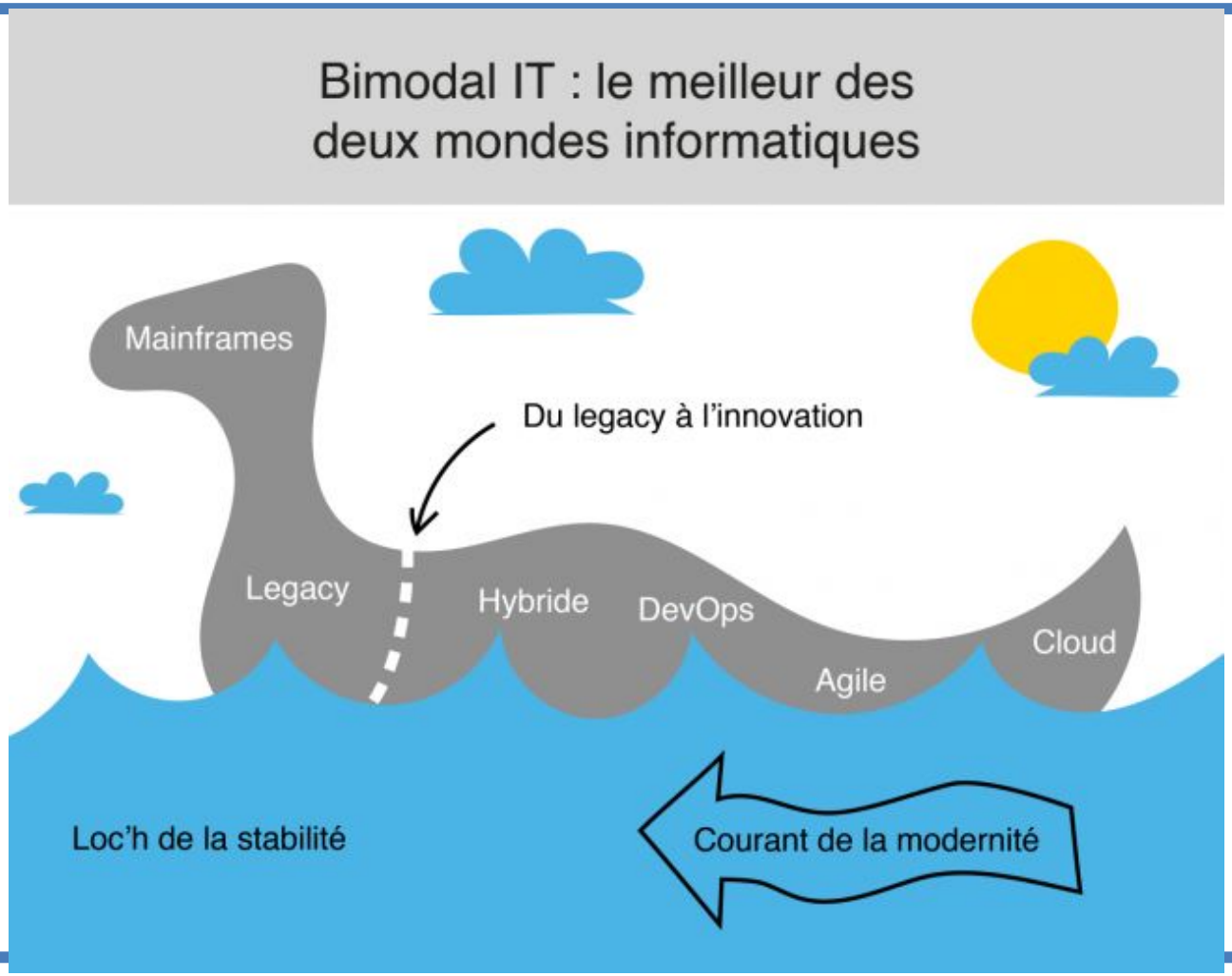
ORSYS formation

Agenda

1. IT Bimodale et DevOps.
2. Application en 12 facteurs, intégration, déploiement continu (CI/CD), applications Cloud-native.
3. SaaS, PaaS, IaaS, Stockage objet et bloc. Cloud privé, public, hybride : problématique du lock-in.
4. Architecture élastique, Cattle versus Pet, Infrastructure as Code.
5. Outils existants (Terraform, Ansible). Apport des containers versus Machines Virtuelles.

IT Bimodale et DevOps

ORSYS formation



IT Bimodale et DevOps

ORSYS

Bimodal IT = Marathon Runners + Sprinters

Think Marathon Runner

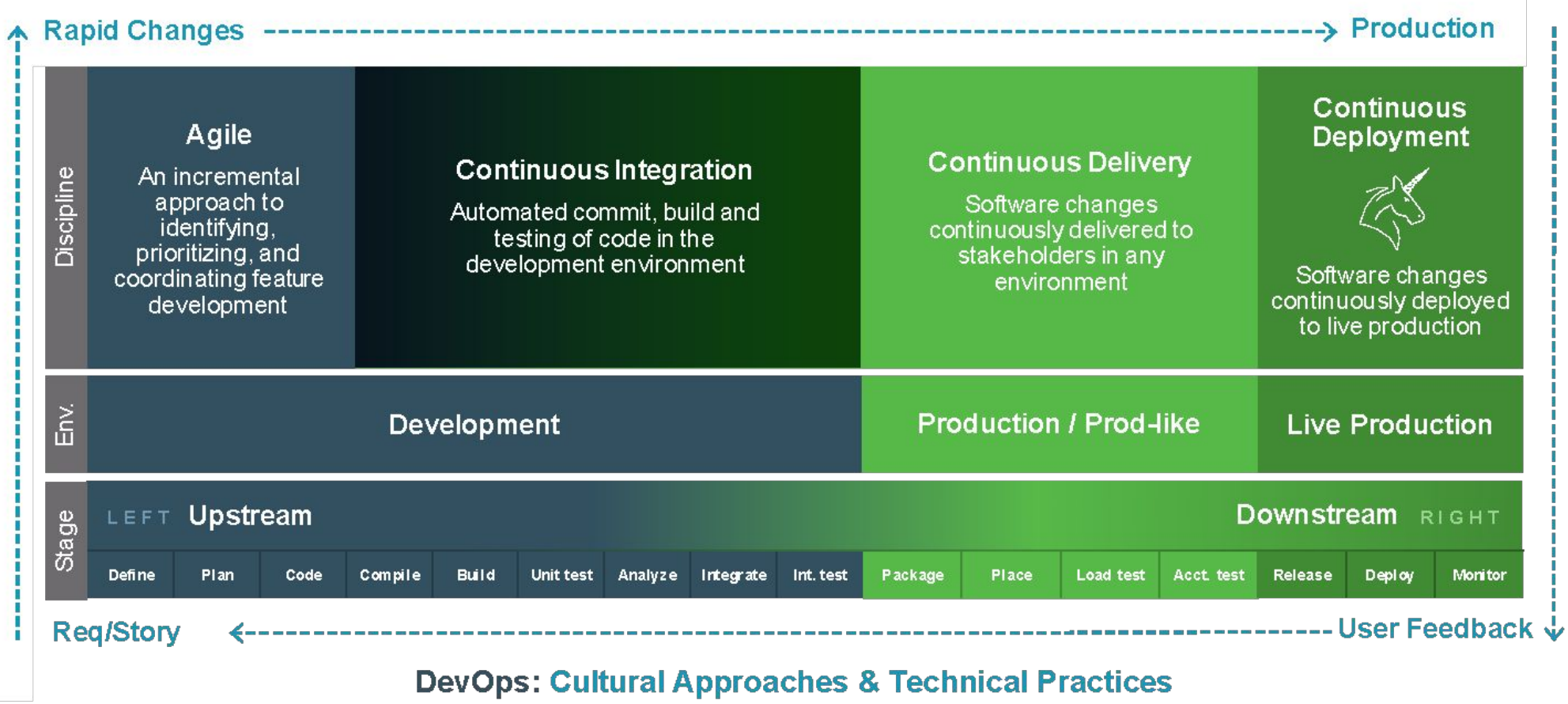
Mode 1		Mode 2
Reliability	**Goal**	Agility
Price for performance	**Value**	Revenue, brand, customer experience
Waterfall, V-Model, high-ceremony IID	**Approach**	Agile, Kanban, low-ceremony IID
Plan-driven, approval-based	**Governance**	Empirical, continuous, process-based
Enterprise suppliers, long-term deals	**Sourcing**	Small, new vendors, short-term deals
Good at conventional process, projects	**Talent**	Good at new and uncertain projects
IT-centric, removed from customer	**Culture**	Business-centric, close to customer
Long (months)	**Cycle times**	Short (days, weeks)

Think Sprinter

Gartner

Context DevOps

ORSYS formation



DevOps: Cultural Approaches & Technical Practices

Application en 12 facteurs

ORSYS formation

I. Codebase Une base de codes suivie dans l'SCM, de nombreux déploiements.

II. Dependencies Déclarer et isoler explicitement les dépendances

III. Config Stockage de la configuration dans l'environnement

IV. Backing services Traiter les backing services comme des ressources rattachées

V. Build, release, run Étapes de construction et d'exécution sont strictement séparées

VI. Processes Exécuter l'application comme un ou plusieurs stateless processus

Application en 12 facteurs

ORSYS formation

VII. Port binding Services d'exportation via le port binding

VIII. Concurrency Mise à l'échelle via le modèle de processus

IX. Disposability Maximisez la robustesse avec un démarrage rapide et un arrêt gracieux.

X. Dev/prod parity Maintenir le développement, la mise en scène et la production aussi semblables que possible

XI. Logs Traiter les journaux comme des flux d'événements

XII. Admin processes Exécuter des tâches administratives ou de gestion en tant que processus ponctuels

ORSYS formation

Qu'est ce que l'intégration Continue (Ci) ?

Continuous Integration is a software development practice where members of a team integrate their work frequently. Usually each person integrates at least daily - leading to multiple integrations per day.

— Martin Fowler - Continuous Integration

ORSYS formation

Continuous

- Each integration is verified by an automated build (including test)
- Integrate code often, at least daily, to make integration a non-event
- Continuously build and integrate
  - Trigger builds automatically based on commit and merge actions and success of upstream builds

Common pitfalls

ORSYS formation

Not putting everything into the code repository:
Everything that's needed to build and configure the application and the system should be in your repository.

Not automating the build process:
Manual steps create opportunities for mistakes and leave steps undocumented.

Not fixing broken builds right away:
A key goal of CI is having a stable build from which everyone can develop.
If the build can't be fixed in a few minutes, the offending change should be identified and reverted.

Not triggering quick tests on every change:
Full end-to-end tests are necessary, but quick tests (unit tests) are also important to enable fast feedback.

Not merging into trunk often enough:
Many organizations have automated tests and builds, but don't enforce a daily merge into trunk.
This leads to long-lived branches that are harder to integrate, and to long feedback loops for the developers.

ORSYS formation

Ways to measure CI

Factor to test	What to measure
Code commits trigger a build of the software	The percentage of code commits that result in a software build without manual intervention.
Code commits trigger a series of automated tests	The percentage of code commits that result in a suite of automated tests being run without manual intervention.
Automated builds and tests are executed successfully every day	The percentage of automated builds and the percentage of automated tests that are executed successfully every day.
Current builds are available to testers for exploratory testing	The availability of builds to testers, or its inverse, namely the unavailability of builds to testers.
Developers get feedback from the acceptance and performance tests every day	The availability of feedback to developers from acceptance and performance tests— that is, the percentage of tests that provide feedback that is available to developers within a day.
Broken builds are fixed immediately	The time it takes between the build breaking and having it fixed, either with a check-in that fixes the problem, or by reverting the breaking change.

ORSYS formation

Continuous Delivery (Cd)

How long would it take to your organization to deploy a change that involves just one single line of code?

Mary Poppendieck

Tom Poppendieck

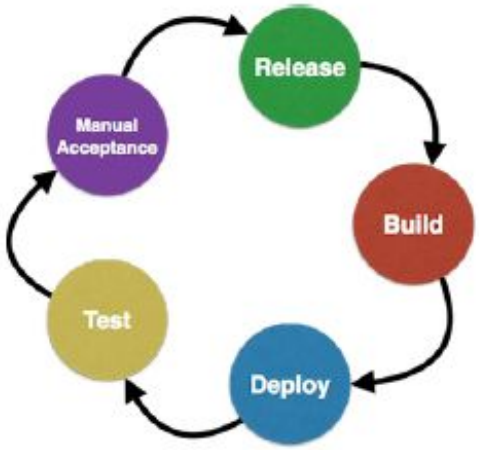
- Reduce deployment risks
- Allow more frequent user feedback
- Make progress observable for everyone

Why Use Continuous Delivery?

ORSYS formation

Continuous Delivery is the next step after Continuous Integration:

- Each change to the system can be released for production
- Delivery can be done at any time, on any environment
- CD Pipeline produces a validated deployable artifact



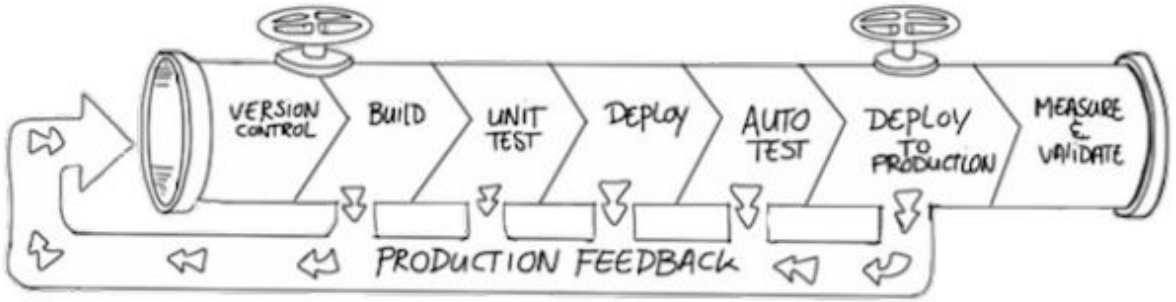
Your team prioritizes keeping the software deployable over working on new features

— Martin Fowler

How to Do Continuous Delivery?

ORSYS formation

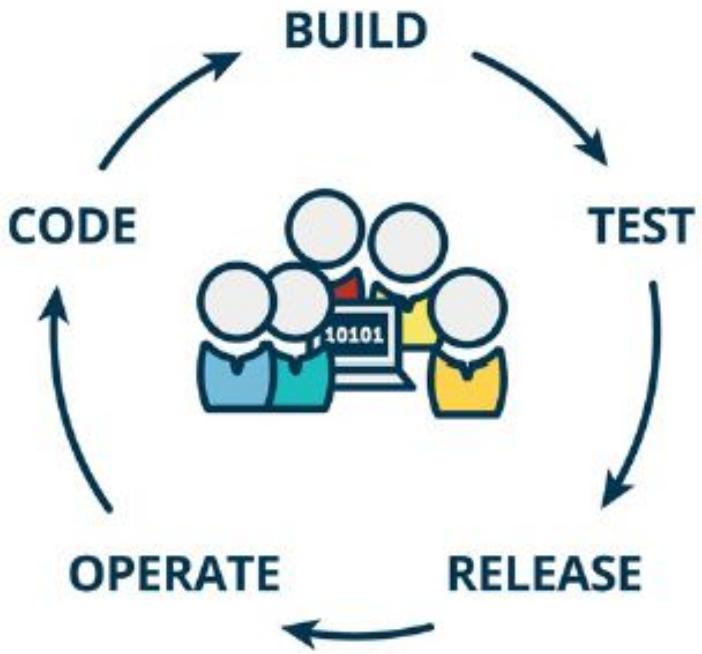
- Have a collaborative working relationship with everyone involved
  - DevOps is a popular model i.e. 2 Pizza Team
- Use Delivery Pipelines, which are automated implementations of your application's build lifecycle process. This is discussed more later in this course.



ORSYS formation

What is Continuous Deployment ?

- Continuous Deployment is the last step after Continuous Delivery.
  - Used by Amazon and Netflix
- Any change to the system is automatically pushed to production.
  - Not appropriate in all scenarios

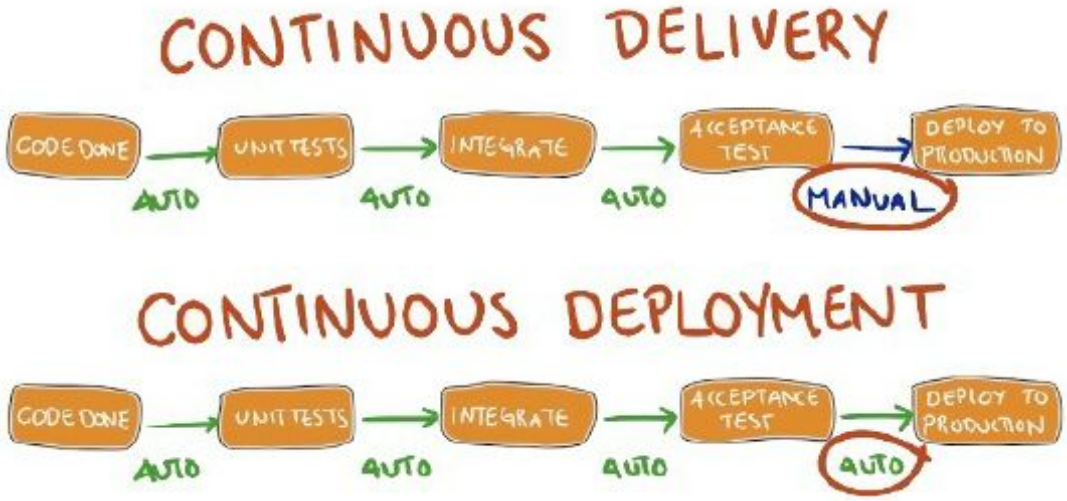


ORSYS formation

Continuous Delivery Versus Deployment

These two terms are easily confused:

- Continuous Delivery needs a manual step before it is deployed to production.
- Continuous Deployment deploys the modified and tested software to production automatically.
  - Full automated testing and reporting is essential to trusting Continuous Deployment



ORSYS formation

WHAT IS A CI/CD PIPELINE?

- Model of the value stream of your software production business
- Defines and models build, testing, and/or deployment processes as continuous workflows
- Fundamental to achieving Continuous Delivery
  - Continuous Delivery is necessary to achieve Continuous Deployment

ORSYS formation

WHY USE A CI/CD PIPELINE?

- Using a "CD Pipeline" gives same outcome as a value or supply chain in other industries
  - This model is easier to analyze and manage
- It is driven by the fail-fast concept
  - "Fast is cheap": Pipeline detects failures as early as possible

ORSYS formation

- Composed of *stages*, mapped to atomic steps
  - Scoped to cacheable result
  - You should never have to rerun a successful step, just retrieve the cached result
- Transition between 2 stages is a *gate*
  - Manual or Automated
  - A transition that confirms a requirement is met is a Validation Gate
  - *gates* trigger next stage
- Stages can be run sequentially and/or in parallel
  - Thus *gates* can branch and even be conditional
- Driven by atomic change: generally SCM single commit

ORSYS formation

- Deployment is the set of actions that makes a software system ready for usage
- Deploying software is done in environments
- Environments are isolated runtimes with specific properties:
  - Production environments are where software lives for end users
  - Testing environments are locations where tests are run against software
  - Disaster Recovery Environment is an emergency location from which software can be run after a disaster occurs on a production environment

ORSYS formation

Automated Deployment

Using *automation* for a deployment stage:

- Make deployment repeatable and less error-prone
- Bring ROI when deploying on new environment
- Make frequent releases possible by bringing *confidence*
- Still can require a manual (read: *human*) trigger to have validation gate

ORSYS formation

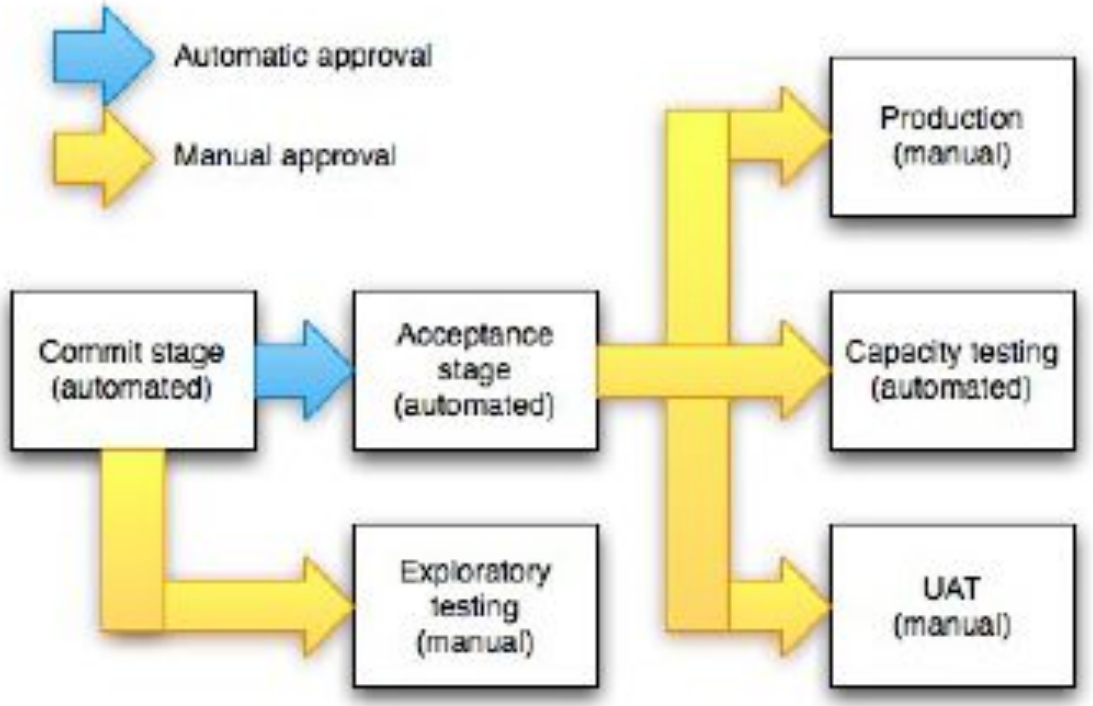
Distributing

- The *build* stage:
  - Fully automated stage.
  - Automated Gates to trigger: event from the SCM (commit, merge, pull-request)
- Easy to distribute to be efficient:
  - Parallelize builds of different target architectures or OSes (ARM, Windows, Android...)
  - Parallelize builds on independents modules of your code before assembling (Map & Reduce)
  - Unit and Integration tests are tied to the code: run them in parallel !

- The *deployment* stage:
  - Should be automated, but can be manual
  - Gates can be manual or automated
- Easy to distribute the automated ones
- You can also distribute manual gate-based if they are on different environments

ORSYS formation

An Optimized Pipeline Example



Always Build Upon Good Foundations

ORSYS formation

- Introducing Binary reuse: *"Only Build Your Binaries Once"*
  - Compiling again a given binary violates the efficiency rule, (shorter build time)
  - Ensure same version of codebase is used across the pipeline
  - If the binary does not comply with "deployable" rule, then it must fail a test
  - Focus on failing-fast

ORSYS formation

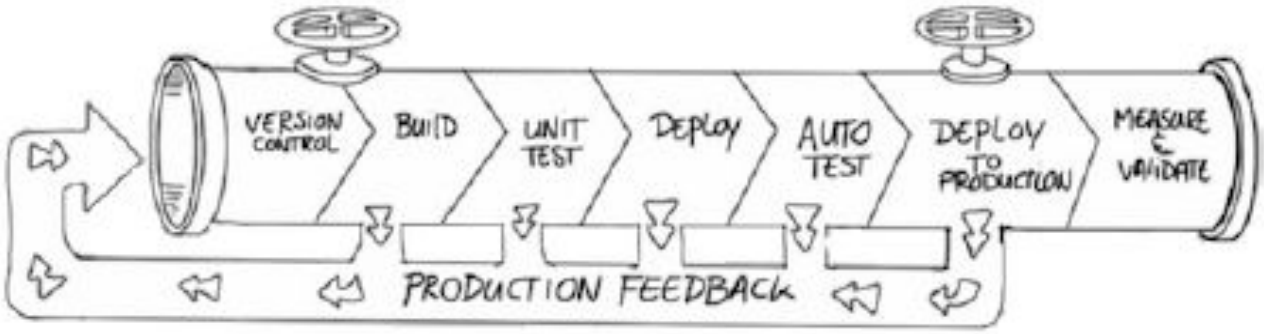
Why Have a Feedback Loop?

- *Context:* Continuous Delivery implemented with CD Pipeline
- *Target:* Focusing on fast feedback to react quickly
- Need systems that give quick feedback to pipeline's actors
  - Key to success on the *efficiency* rule
  - Give confidence to the actor: they know when when it breaks

ORSYS formation

La Feedback Loop?

- The feedback loop is the set of tools that:
  - Provides status of each step of the pipeline
  - Gives the right message to the right actor of the pipeline
  - Is also a great source of measurements



ORSYS formation

SaaS, PaaS, IaaS

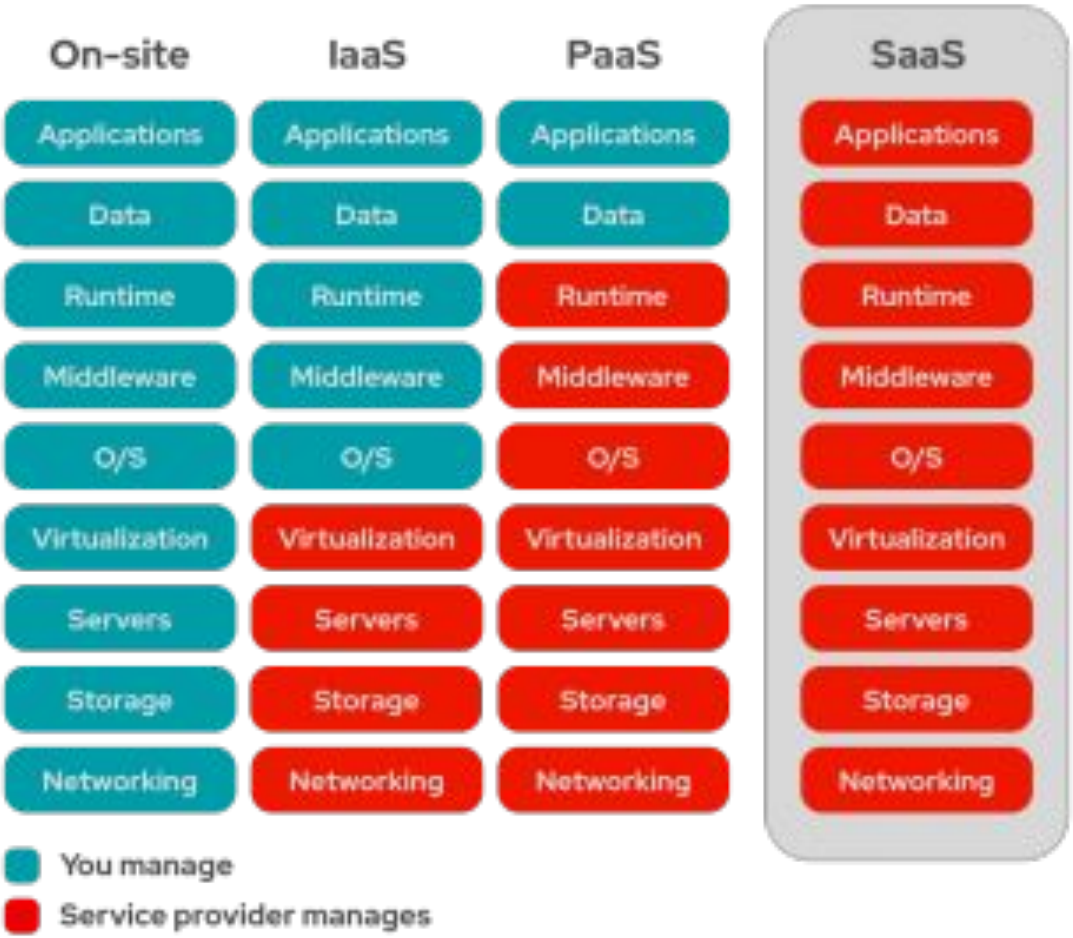
SaaS

ORSYS formation

Le SaaS est un service qui fournit une application logicielle - que le fournisseur de services cloud gère - à ses utilisateurs.

En générale, les applications SaaS sont des applications Web ou des applications mobiles auxquelles les utilisateurs peuvent accéder via un navigateur Web.

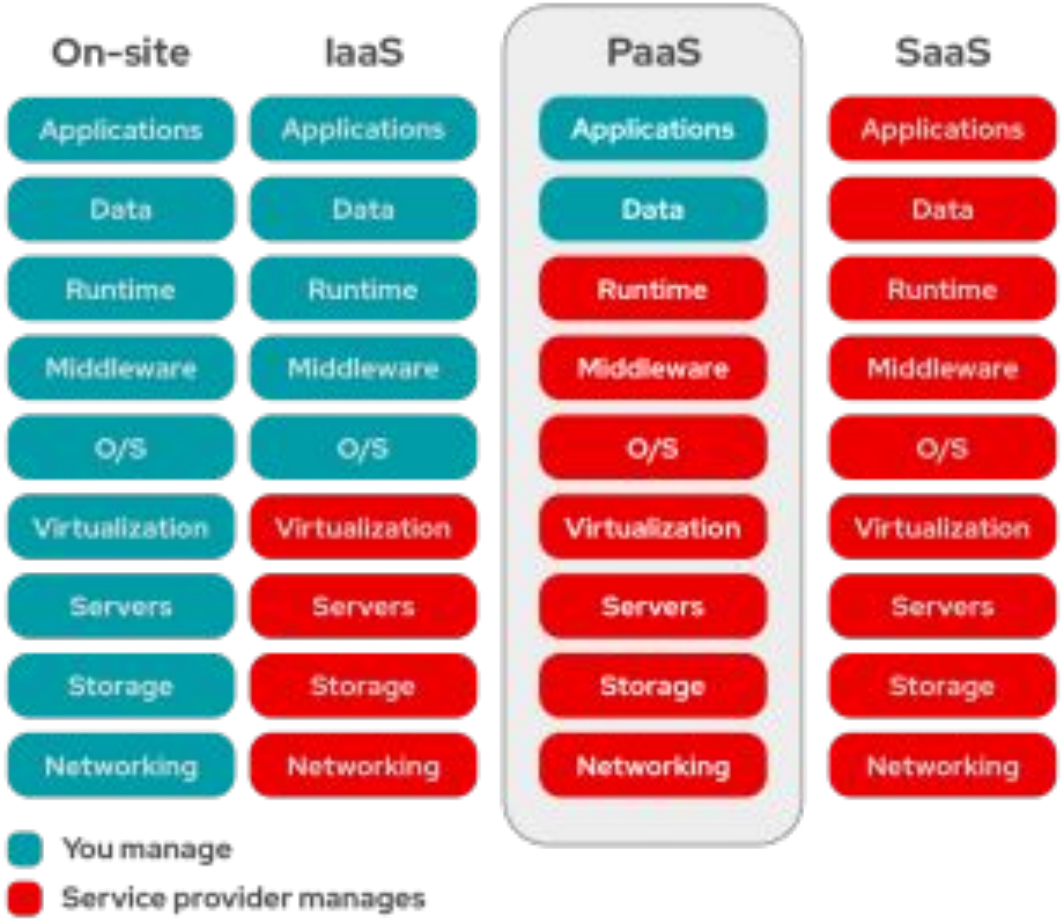
Les mises à jour logicielles, les corrections et toute maintenance logicielle générale sont prises en charge pour l'utilisateur et se connectent aux applications cloud via un tableau de bord ou une API.



PaaS

ORSYS formation

PaaS signifie que le matériel et une plate-forme logicielle d'application sont fournis et gérés par un fournisseur de services cloud externe, mais que l'utilisateur gère les applications exécutées au-dessus de la plate-forme et les données sur lesquelles l'application s'appuie.

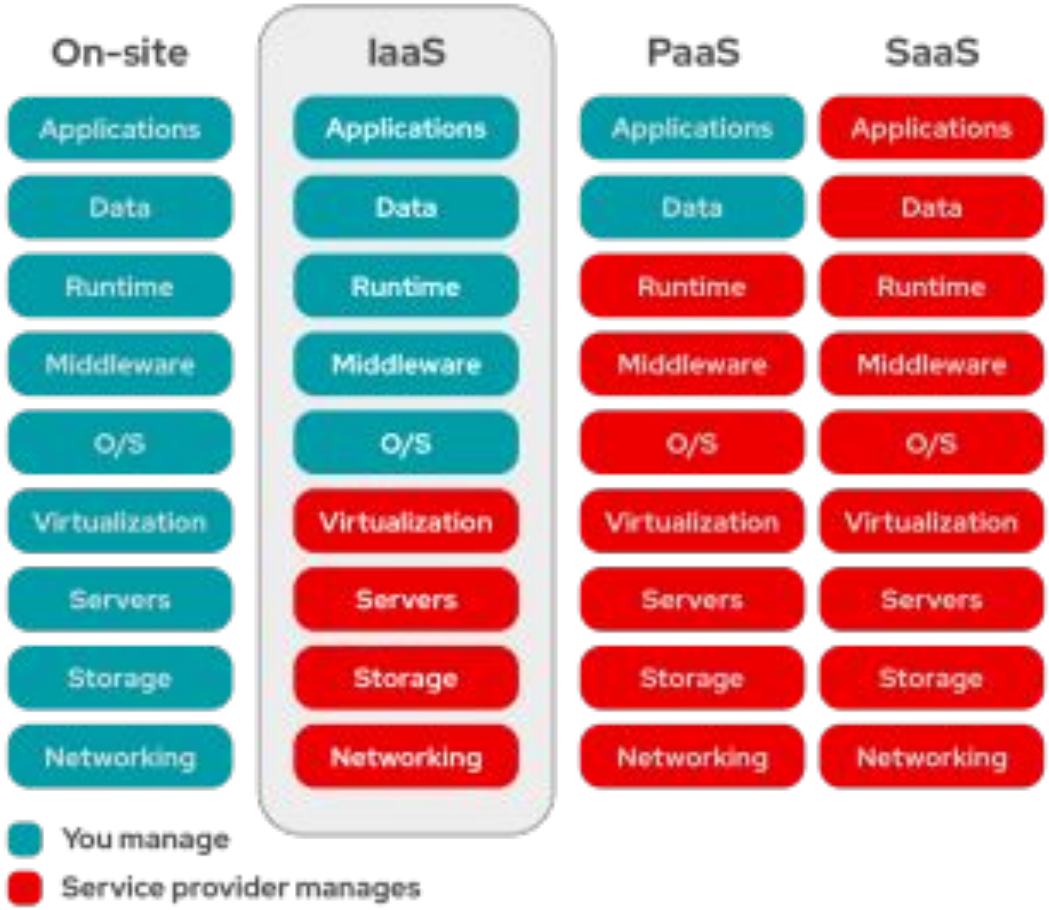


IaaS

ORSYS formation

- IaaS signifie qu'un fournisseur de services cloud gère l'infrastructure pour vous (les serveurs, le réseau, la virtualisation et le stockage de données) via une connexion Internet.

- L'utilisateur a accès via une API ou un tableau de bord, et loue essentiellement l'infrastructure.



ORSYS formation

Cloud privé, public, hybride :

problématique du lock-in.

Cloud privé

ORSYS formation

- Les clouds privés sont vaguement définis comme des environnements cloud uniquement dédiés à un seul utilisateur final ou à un seul groupe, où l'environnement s'exécute généralement derrière le pare-feu de cet utilisateur ou groupe.

- Tous les clouds deviennent des clouds privés lorsque l'infrastructure informatique sous-jacente est dédiée à un seul client avec un accès complètement isolé.

- Mais les clouds privés ne doivent plus provenir d'une infrastructure informatique sur site.
- Les organisations construisent maintenant des clouds privés sur des centres de données loués et appartenant aux fournisseurs situés hors site, ce qui rend obsolètes les règles de localisation et de propriété.

Cloud public

ORSYS formation

- Les clouds publics sont des environnements cloud généralement créés à partir d' une infrastructure informatique n'appartenant pas à l'utilisateur final.

- Certains des plus grands fournisseurs de cloud public incluent Alibaba Cloud, Amazon Web Services (AWS), Google Cloud, IBM Cloud et Microsoft Azure.

- Les clouds privés ne doivent plus provenir d'une infrastructure informatique sur site. Les organisations construisent maintenant des clouds privés sur des centres de données loués et appartenant aux fournisseurs situés hors site, ce qui rend obsolètes les règles de localisation et de propriété.

Cloud hybride

ORSYS formation

- Un cloud hybride est un seul environnement informatique transparent créé à partir de plusieurs environnements connectés via des réseaux locaux (LAN), des réseaux étendus (WAN), des réseaux privés virtuels (VPN) et / ou des API.

- Les caractéristiques des clouds hybrides sont complexes et les exigences peuvent différer selon la personne à qui vous le demandez. Par exemple, un cloud hybride peut inclure:

✔ Au moins 1 cloud privé et au moins 1 cloud public

✔ 2 nuages privés ou plus

✔ 2 clouds publics ou plus

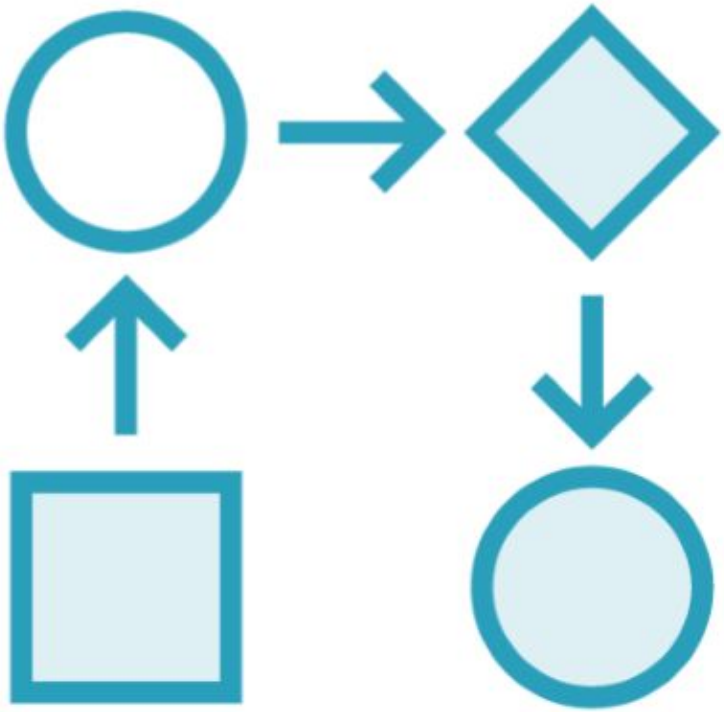
✔ Un environnement bare-metal ou virtuel connecté à au moins 1 cloud public ou privé

ORSYS formation

Infrastructure as Code

ORSYS formation

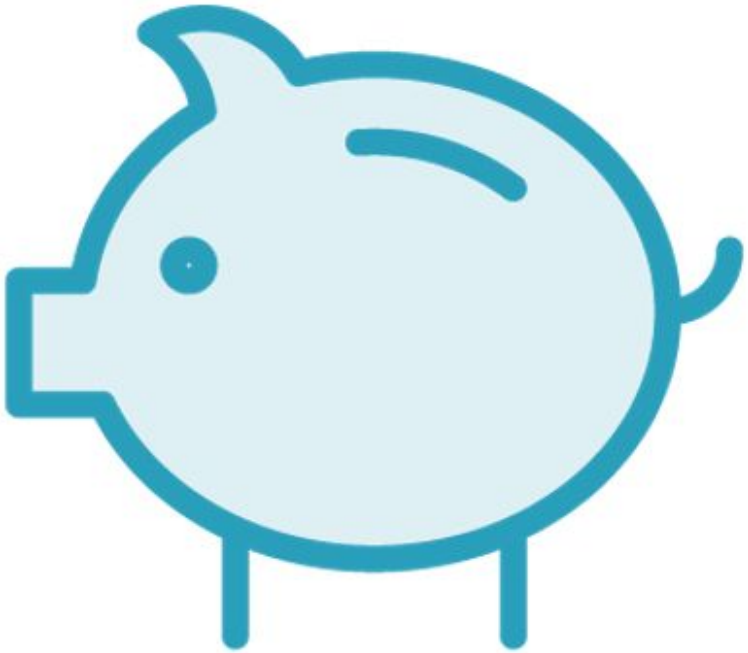
IaC: le Pipeline



The workflow of moving a project through stages of work

Common names: Release, Build, Deployment

Le gain pour le business et les clients

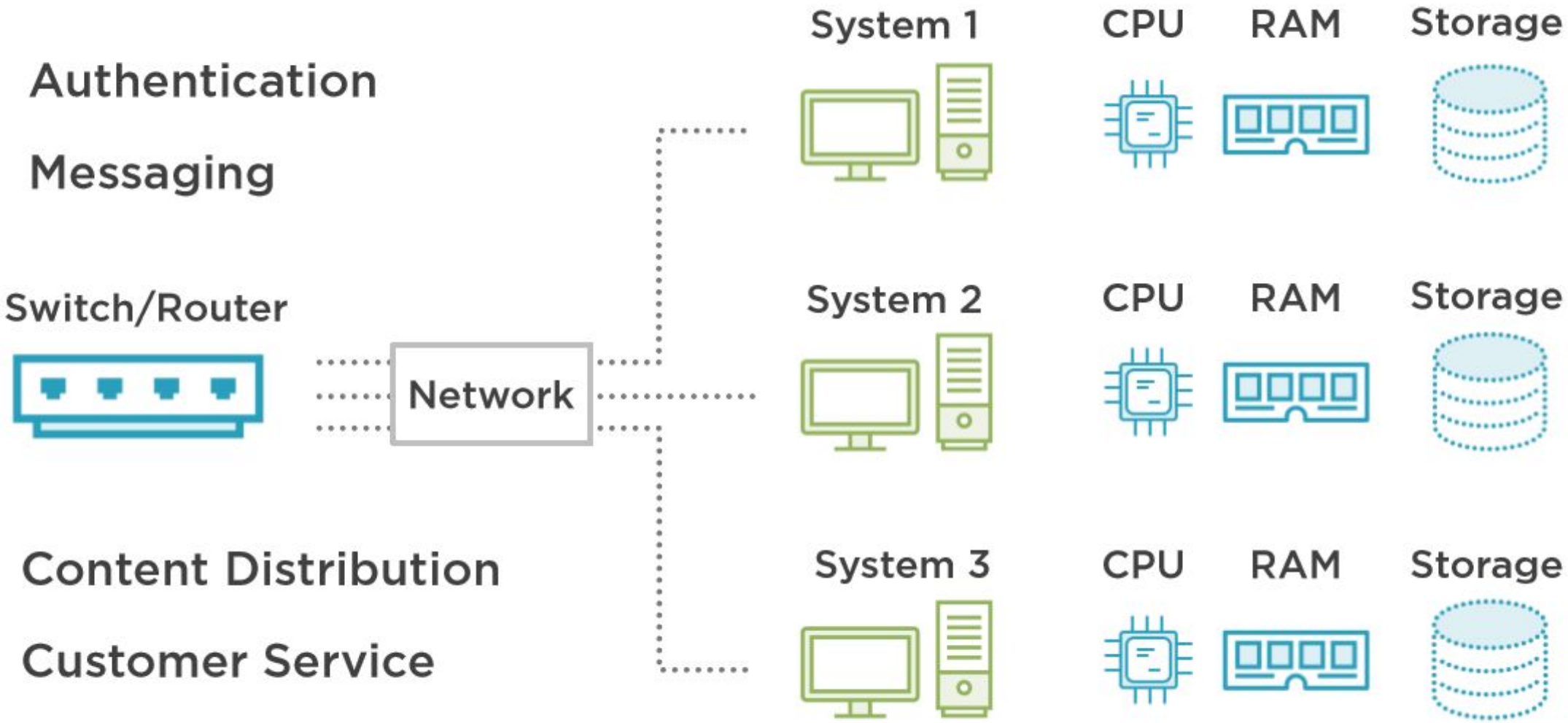


Change and release management

Improved security

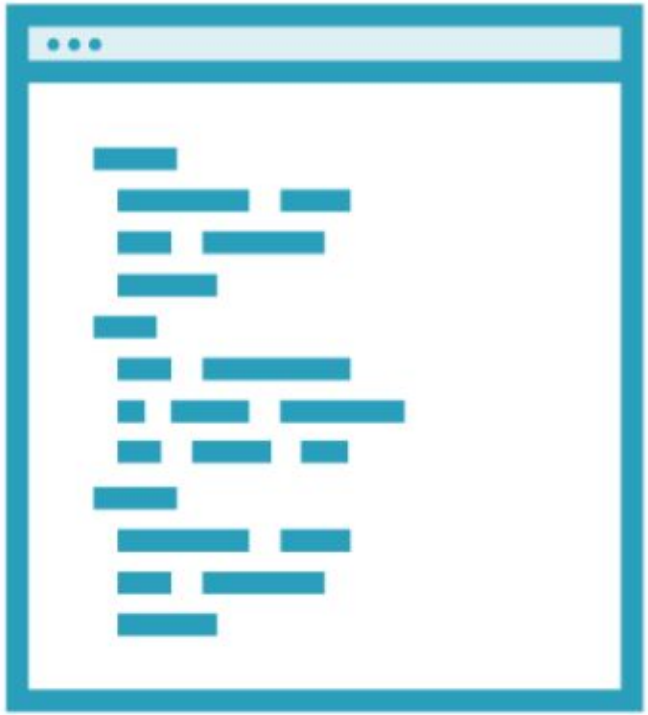
Reduced recovery time

Le gain pour le business et les clients



ORSYS formation

La configuration



Read by an automation process

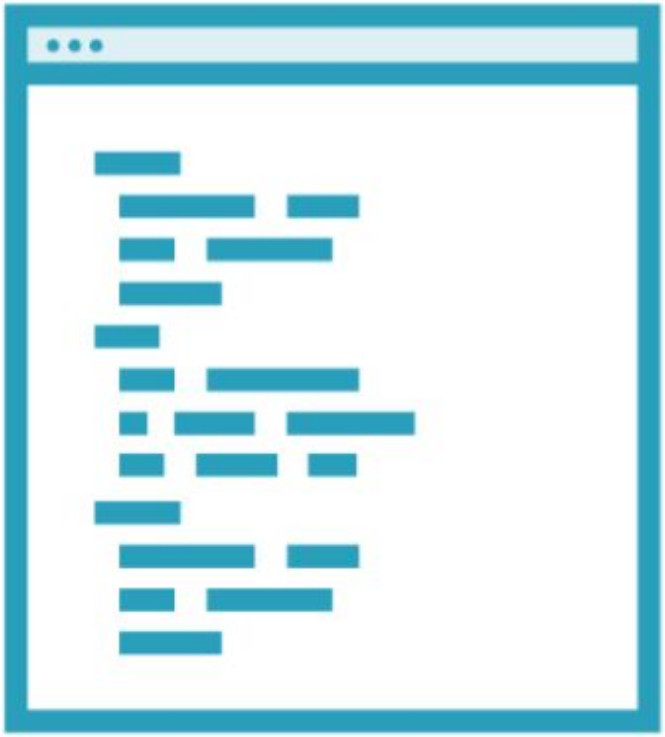
Contains configuration properties

Imperative – How to do

Declarative – What you want

ORSYS formation

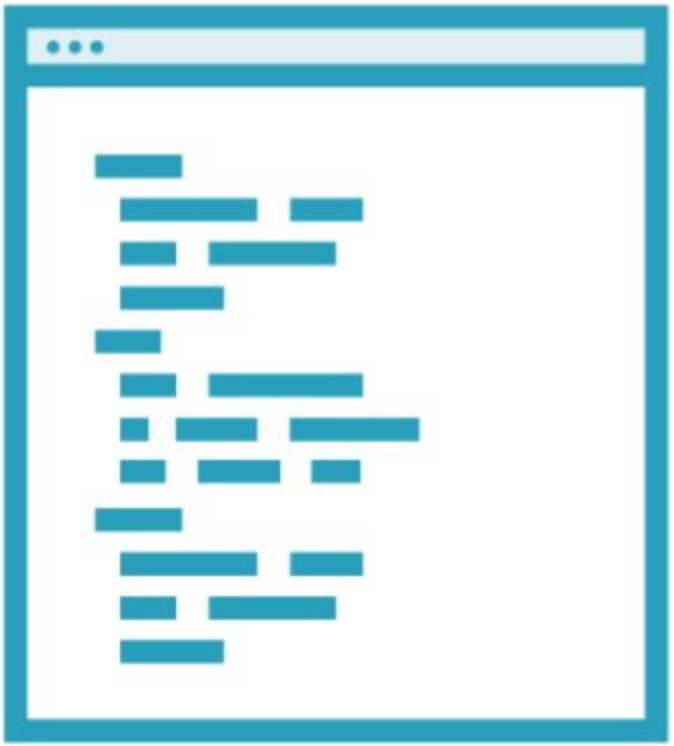
Impératif – Démarrer une voiture



1. Open driver side car door
2. Get in car
3. Locate ignition switch
4. Insert key into ignition switch
5. Turn ignition
6. Slightly press on the gas pedal

ORSYS formation

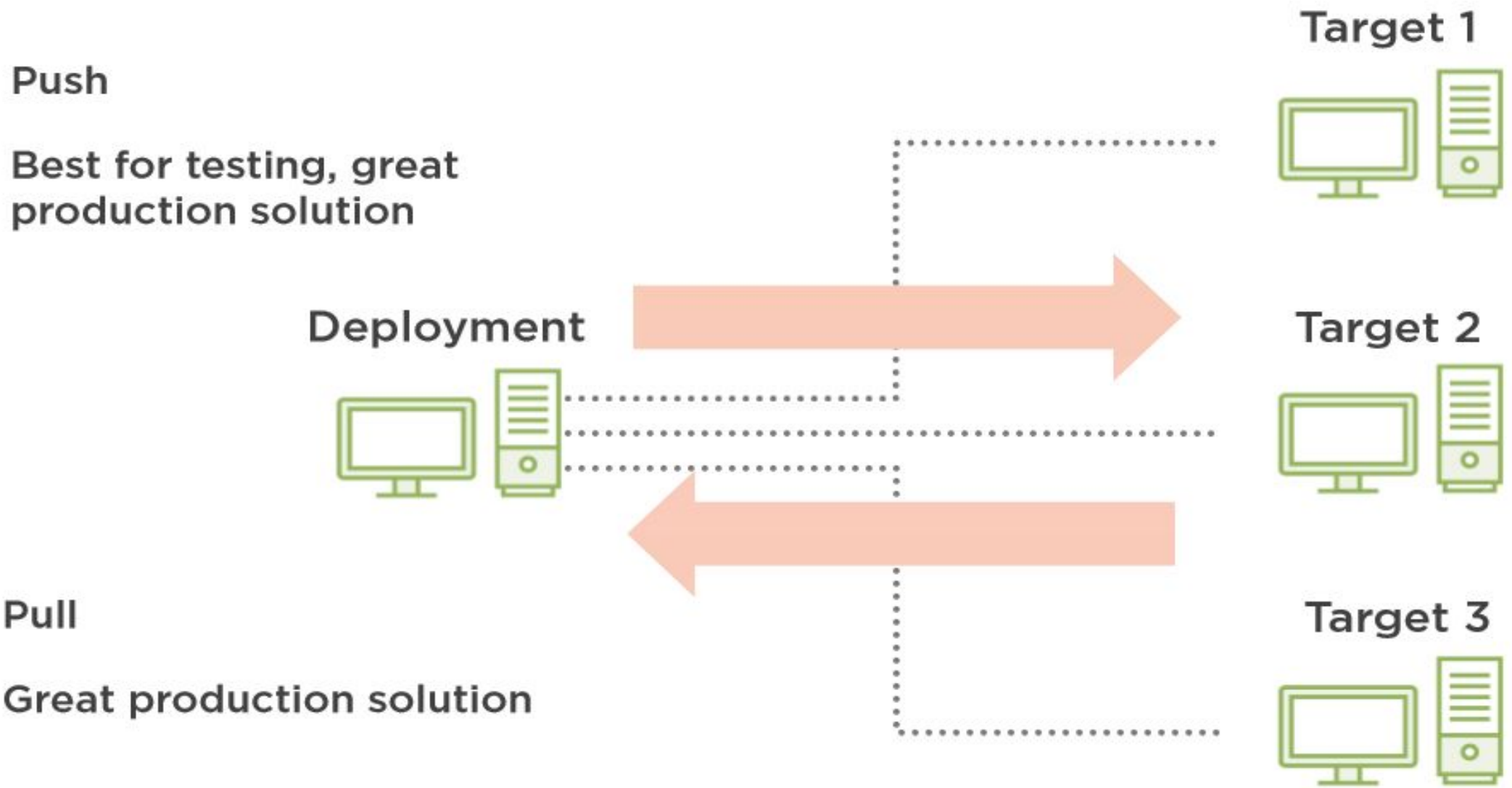
Déclaratif – Démarrer une voiture



1. Start the car

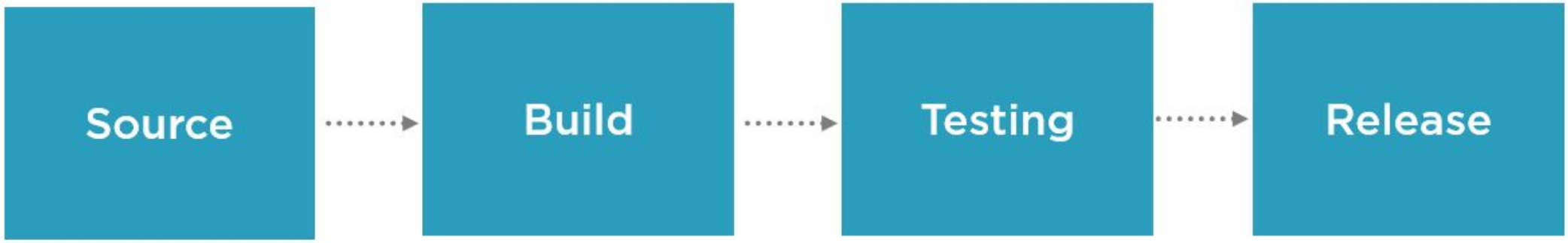
ORSYS formation

Cibles de livraison Push/Pull



ORSYS formation

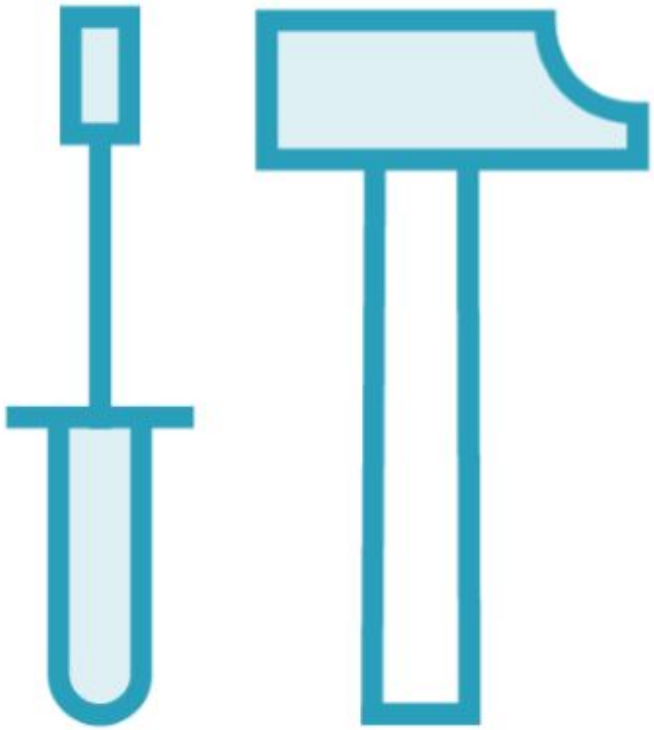
Cibles de livraison Push/Pull



Automation and orchestration – Monitoring and validating

ORSYS formation

Outils CCA - Continuous Configuration Automation



Referred to as the DevOps Toolchain

Tools that automate and orchestrate

May be specific to a platform

Need to meet your security and compliance needs

Needs to scale

ORSYS formation

Une nouvelle vision de “l’échec”

Due to inconsistence procedures, lack of automation, lack of change control – IT Ops managers are slow to make change because of failure

Infrastructure From Code provides a reliable, repeatable process that can learn.

Rush to failure!

ORSYS formation

What Does Release Mean?

After the process successfully completes

Configuration change is in Source

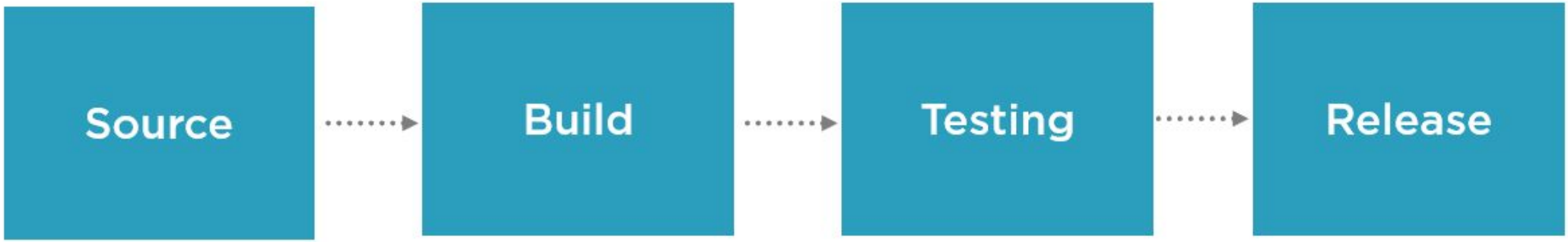
Build process completes

Testing has passed

A build artifact is ready to be deployed

ORSYS formation

Le pipeline de Release



GitHub
Apache Subversion
Microsoft Visual Studio
Team Foundation server

Team City
AppVeyor
Jenkins
Microsoft Visual Studio

ServerSpec, Rspec
Chef InSpec
Pester
Test-Kitchen
Vagrant

Environment and situation dependent

ORSYS formation

Continuous Delivery (CD)

Also known as Continuous Deployment with slight changes

Best with immutable servers – every server matches exact

Server can be re-deployed entirely if failure occurs

Example: Web servers in a farm

ORSYS formation

Incremental Deployment

For servers that must be maintained over a long period of time

For servers that cannot be re-deployed

Configuration changes are delivered as updates

Example: Database, Domain Controllers

ORSYS formation

Continuous Integration

Bringing the process all together with a workflow (orchestration)

Add quality gates (testing) and approval

Can suspend new changes until failures are resolved

ORSYS formation Démarrer avec l'Infrastructure as Code

Start small

Start with configuration management

Add source control

Add testing

Build the Pipeline

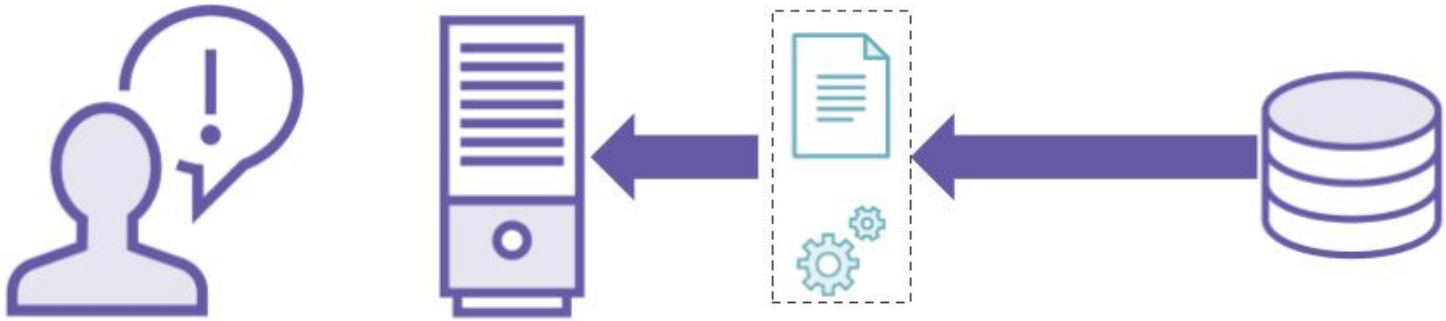
ORSYS formation

Infrastructure Immuable

Immutable

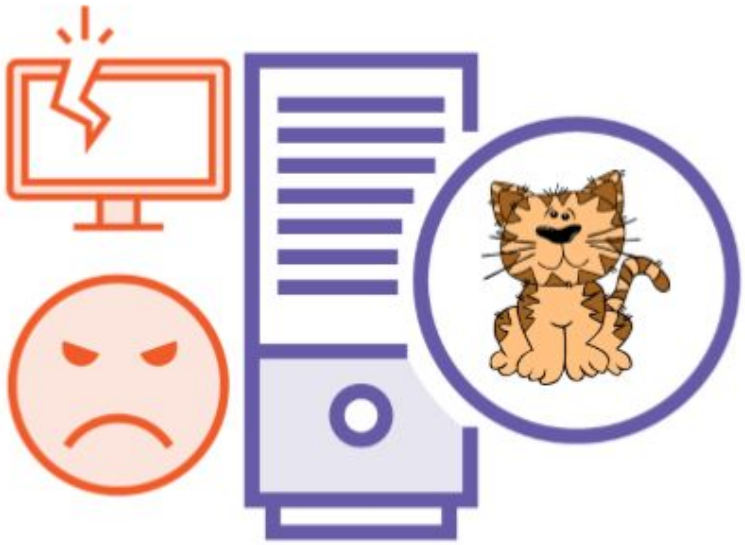
Adjective:

Unchanging over time, or unable to be changed



ORSYS formation

Pets vs Cattle



Mittens!

XB306-1

XB306-2

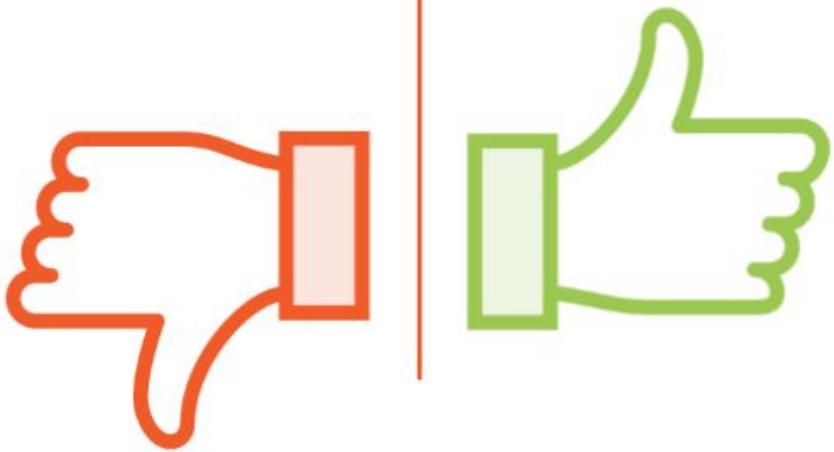
XB306-3

...

ORSYS formation

Pets vs Cattle

Cats	Cattle
Kept alive at all costs	Expendable
Need manual intervention	Work “out of the box”
Difficult to Scale	Easy to Scale
High-stress	Low-stress



ORSYS formation

Autres avantages

Testable Infrastructure

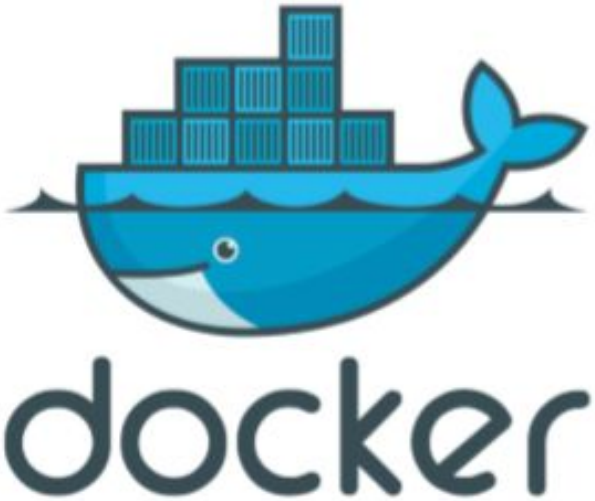
Reproduce Production

Unit of Deployment

Confident Changes

ORSYS formation

Docker?



~~No Windows support~~

Application re-architecture

Orchestration

Sometimes worth it, sometimes not

ORSYS formation

Packer!

Utilize native tools

Integrate with configuration management

Easy to convert to Docker

ORSYS formation

Un Template Packer

```
{
    “builders”: [
    ],
    “provisioners”: [
    ],
    “post-processors”: [
    ]
}
```

ORSYS formation

Un Template Packer

```
{
    "type": "amazon-ebs",
    "name": "MyCowServer-AWS",
    "region": "us-west-2"
}
```

Builders

Generate your image

Provider-specific

Multiple builders = cross-provider builds!

ORSYS formation

Un Template Packer

```
{
    "type": "shell",
    "script": "makeMyCowAwesome.sh",
    "only": ["MyCowServer-AWS"]
}
```

Provisioners

Customize your image

Scripts or configuration management

Can be builder-specific

ORSYS formation

Un Template Packer

```
{
    "type": "compress",
    "output": "MyCowServer-VirtualBox.tar.gz",
    "only": ["MyCowServer-VirtualBox"]
}
```

Post-Processors

Put on the finishing touches

Integration with other services

Key to Docker conversion

ORSYS formation

Provisioners

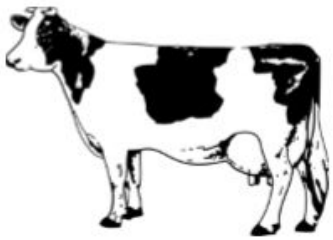
Provisioners

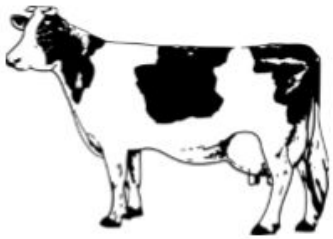
- Reuse your Existing Work
- Fun with Scripts!
- Move My Files!

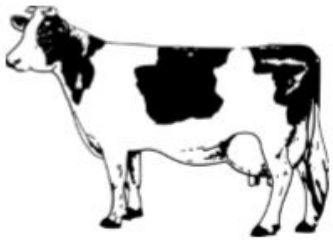
User Variables

Bash and Ansible

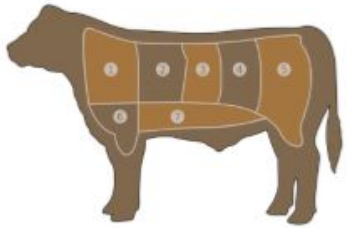
Provisioners







Provisioners



ORSYS formation

Provisioners

```
{
    "type": "shell",
    "script": "makeMyCowAwesome.sh",
    "only": ["MyCowServer-AWS"]
}
```

Provisioner Types

Configuration Management

File

Script

ORSYS formation

Provisioners de type Configuration Management

ORSYS formation

Provisioners de type Script

Local Shell

Remote Shell

Powershell

Windows Shell

Windows Restart

ORSYS formation

Variables

User Variables

```
{
    "variables": {
        "aws_region": ""
    },
    "builders": [{
        "type": "amazon-ebs",
        "region": "{{user `aws_region`}}"
    }]
}
```

ORSYS formation

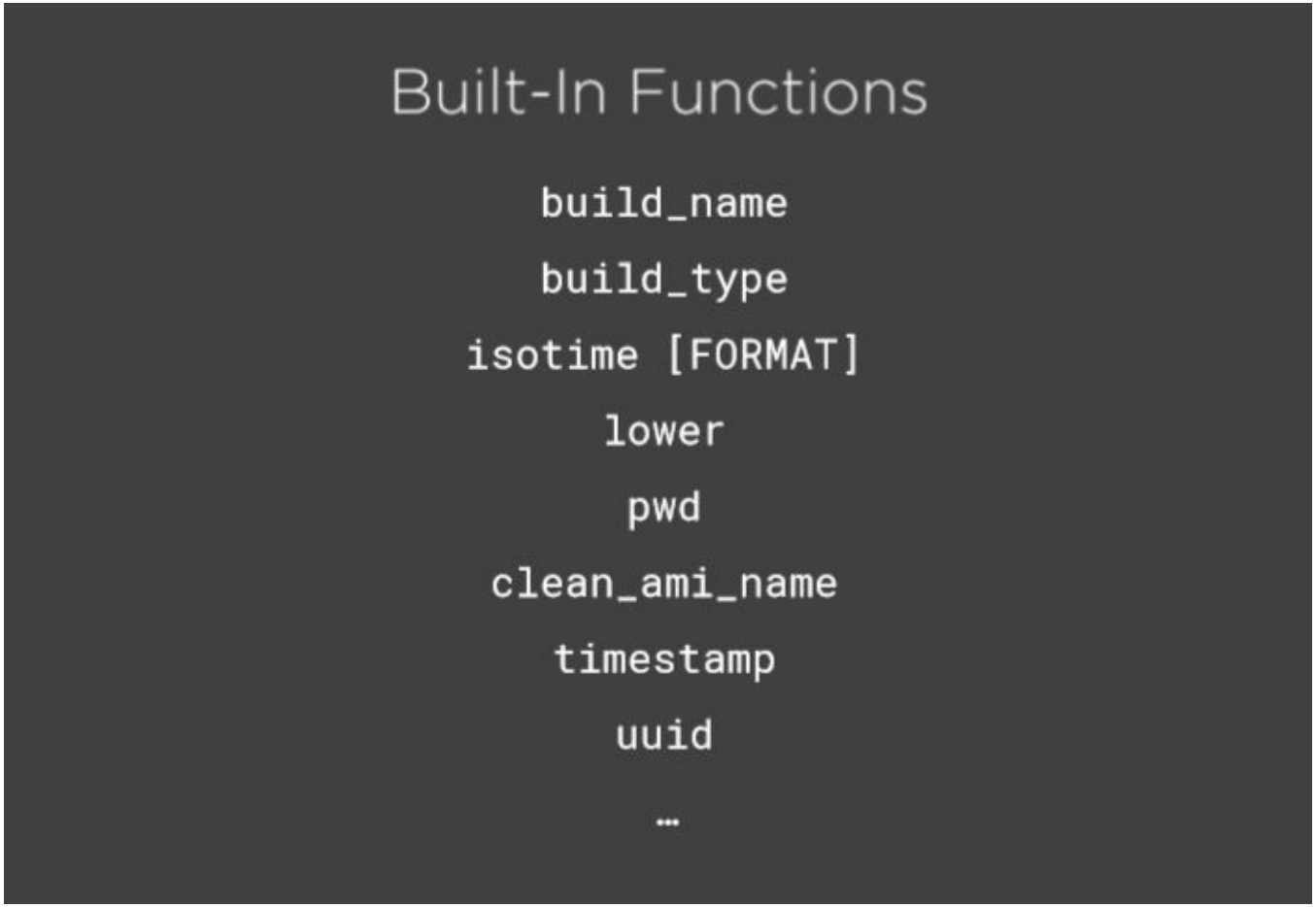
Variables

User Variables – Environment Variables

```
{
    "variables": {
        "aws_region": "{{env `AWS_REGION`}}"
    },
    "builders": [{
        "type": "amazon-ebs",
        "region": "{{user `aws_region`}}"
    }]
}
```

ORSYS formation

Built-In Functions



ORSYS formation

clean-ami-name

Amazon Only

Remove unwanted characters

Apply with pipe

- `{{ "10:02:36" | clean_ami_name }}`
- `Output: 10-02-36`

Commonly used with isotime

ORSYS formation

Timestamp

Replace with Unix Timestamp

Keep Image Names from Colliding

ORSYS formation

Configuration des variables utilisateur

```
> packer build –var ‘aws_region=us-west-1’ template.json

> packer build –var-file=variables.json template.json

{
    “aws_region”: “us-west-1”
}
```

La transition de Packer vers Docker

ORSYS formation

Packer vs Docker, Fight!

- Differences
- Why Switch? When?

Packer and Docker – Playing Nice

- Transition to Docker
- Hybrid Environments

ORSYS formation

Packer – Qu'est ce qu'on obtient?

Images for cloud and on-premise
- Near-identical

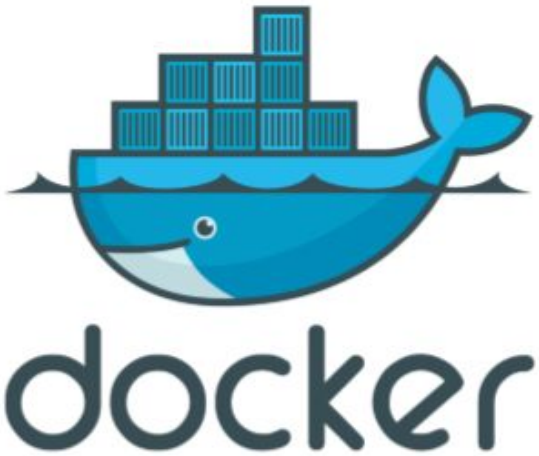
Source controlled

Easy updates, easy rebuilds

Minimal code duplication
- Integration with existing systems

ORSYS formation

Docker - Qu'est ce qu'on obtient de plus?



Binpacking

- Deployment flexibility
- Maximized server utilization

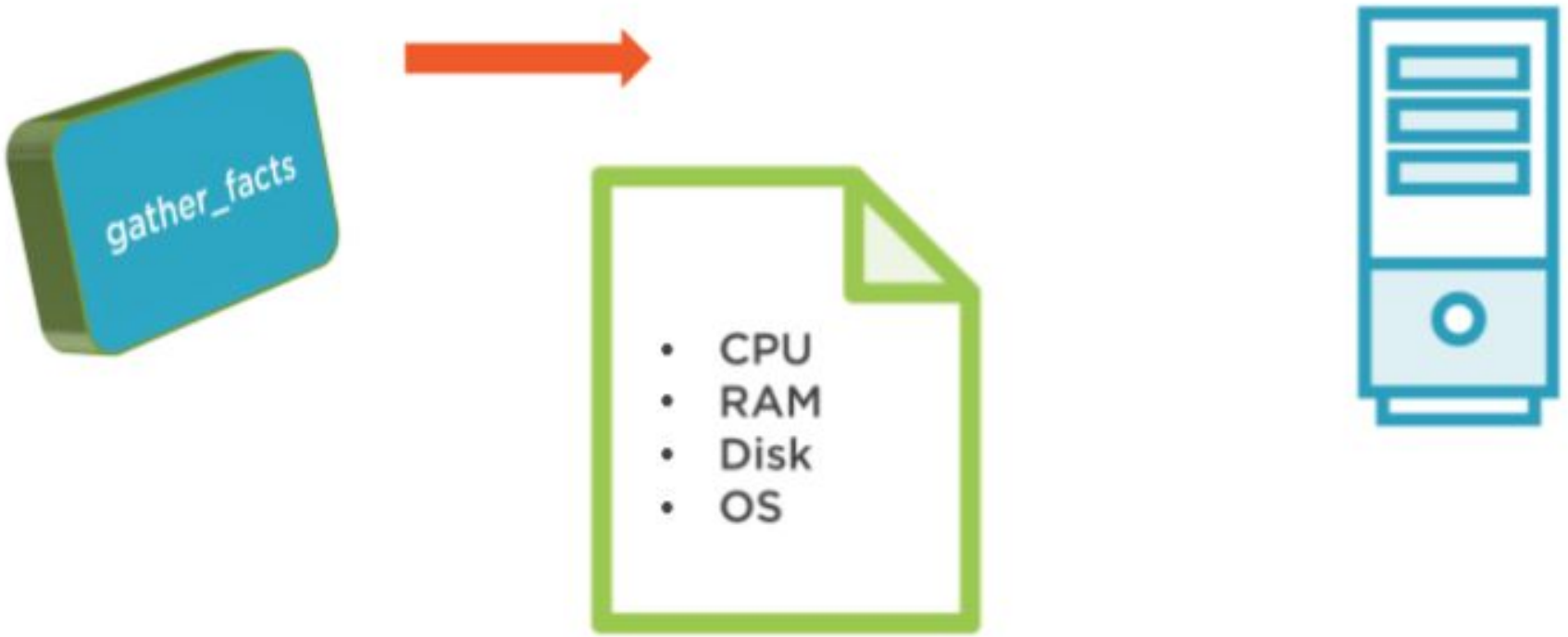
Speed

Built-In Versioning

Containers – Identical Across Environments

Les modules sont idempotents

Le module gather_facts



ORSYS formation

Découvrons les Playbooks

Playbooks are groups of tasks that usually call modules

Playbooks are written in YAML

May contain variables, lists of hosts, groups, and tasks

ORSYS formation

Les playbooks sont sauvegardés dans un SCM

ORSYS formation

Une play

playbook.yaml

```yaml
---
- hosts: linux_servers
  vars:
    foo: "bar"
  tasks:

  - name: Copy a file
    copy:
      src: "/tmp/config.json"
      dest: "/etc/config.json"
```

This is one play

ORSYS formation

Hosts

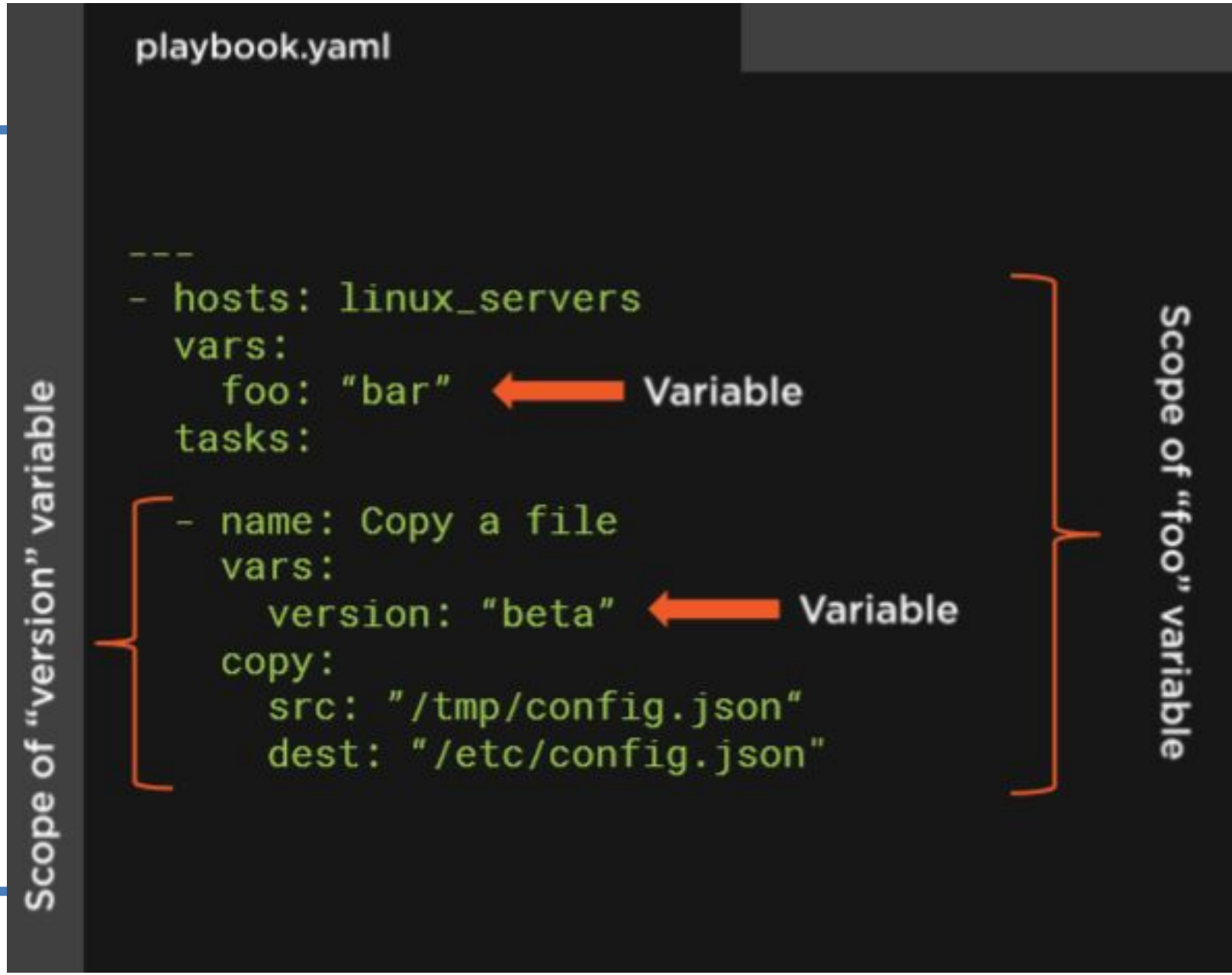
playbook.yaml

```
---
- hosts: linux_servers
  vars:
    foo: "bar"
  tasks:

  - name: Copy a file
    copy:
      src: "/tmp/config.json"
      dest: "/etc/config.json"
```

ORSYS formation

Variables



ORSYS formation

Tasks

playbook.yaml

```
---
- hosts: linux_servers
  vars:
    foo: "bar"
  tasks:

#1  - name: Copy a file          <-- Task
      copy:                     <-- Module
        src: "/tmp/config.json"
        dest: "/etc/config.json"   } Arguments

#2  - name: Start Apache         <-- Task
      service:                  <-- Module
        name: httpd
        state: started             } Arguments
```

ORSYS formation

Tasks

playbook.yaml

```
---
- hosts: linux_servers
  vars:
    foo: "bar"
  tasks:

  - name: Copy a file
    copy:
      src: "/tmp/config.json"
      dest: "/etc/config.json"
```

Module

Arguments

ORSYS formation

Ecrire des playbooks Ansible

- Playbooks are essentially sets of instructions (plays) that you send to run on a single target or groups of targets (hosts).
- For this example, the target host will be a RHEL/CentOS 7 base install.
- There will be a web server installed (NGINX) and then an index.html file will be created in the default webroot.
- After the install and file tasks are completed the service will be started.

Ecrire des playbooks Ansible

Playbooks start with the YAML three dashes (---) followed by:

- **Name**: good for keeping the Playbooks readable
- **Hosts**: identifies the target for Ansible to run against
- **Become statements**: true statement is included here to ensure nginx installs without a problem (it's not always required)

```
1 ---
2 - name: Install nginx
3   hosts: host.name.ip
4   become: true
```

Ecrire des playbooks Ansible

The state:

- Present statement lets Ansible check the state on the target first before performing any further action.
- In both cases if the repo or package is already present, Ansible knows it doesn't have to do any more for this task and continues.

```
tasks:
  - name: Add epel-release repo
    yum:
      name: epel-release
      state: present

  - name: Install nginx
    yum:
      name: nginx
      state: present
```

ORSYS formation

Ecrire des playbooks Ansible

- Insert the index page in the default NgInx location

```
1   - name: Insert Index Page
2     template:
3       src: index.html
4       dest: /usr/share/nginx/html/index.html
```

- The last thing the Playbook will do is make sure that the nginx service has been started (and if not, start it).

```
1 - name: Start NGiNX
2     service:
3       name: nginx
4       state: started
```

ORSYS formation

Ecrire des playbooks Ansible

```yaml
---
- name: Install nginx
  hosts: host.name.ip
  become: true

  tasks:
  - name: Add epel-release repo
    yum:
      name: epel-release
      state: present

  - name: Install nginx
    yum:
      name: nginx
      state: present

  - name: Insert Index Page
    template:
      src: index.html
      dest: /usr/share/nginx/html/index.html

  - name: Start NGiNX
    service:
      name: nginx
      state: started
```

ORSYS formation

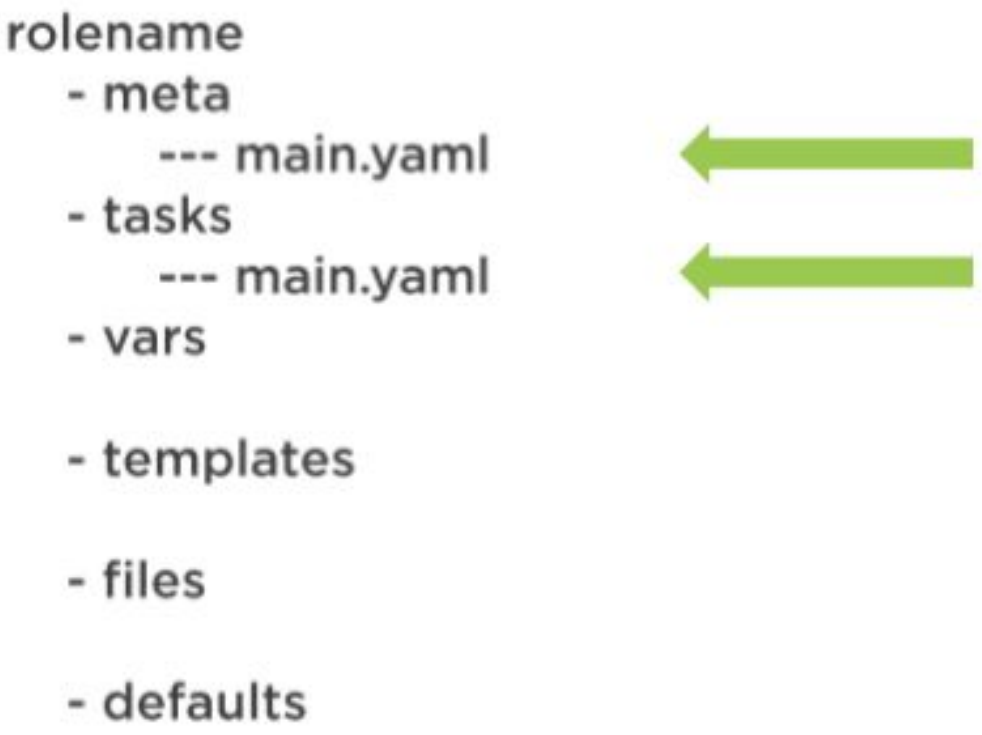
Ecrire des playbooks Ansible

Roles help organize all the files and folders of your code

Pre-defined conventions of folders and files

No special project files, just folder conventions

Structure d'un Role Ansible



ORSYS formation

Le dossier Meta

Descriptions of the role, dependencies, Ansible Galaxy information

Information about what roles this one depends on

Do we need to run a role before this one

Le dossier Task

ORSYS formation

Primary entry for the actions to start happening

Same format as typical tasks in a playbook

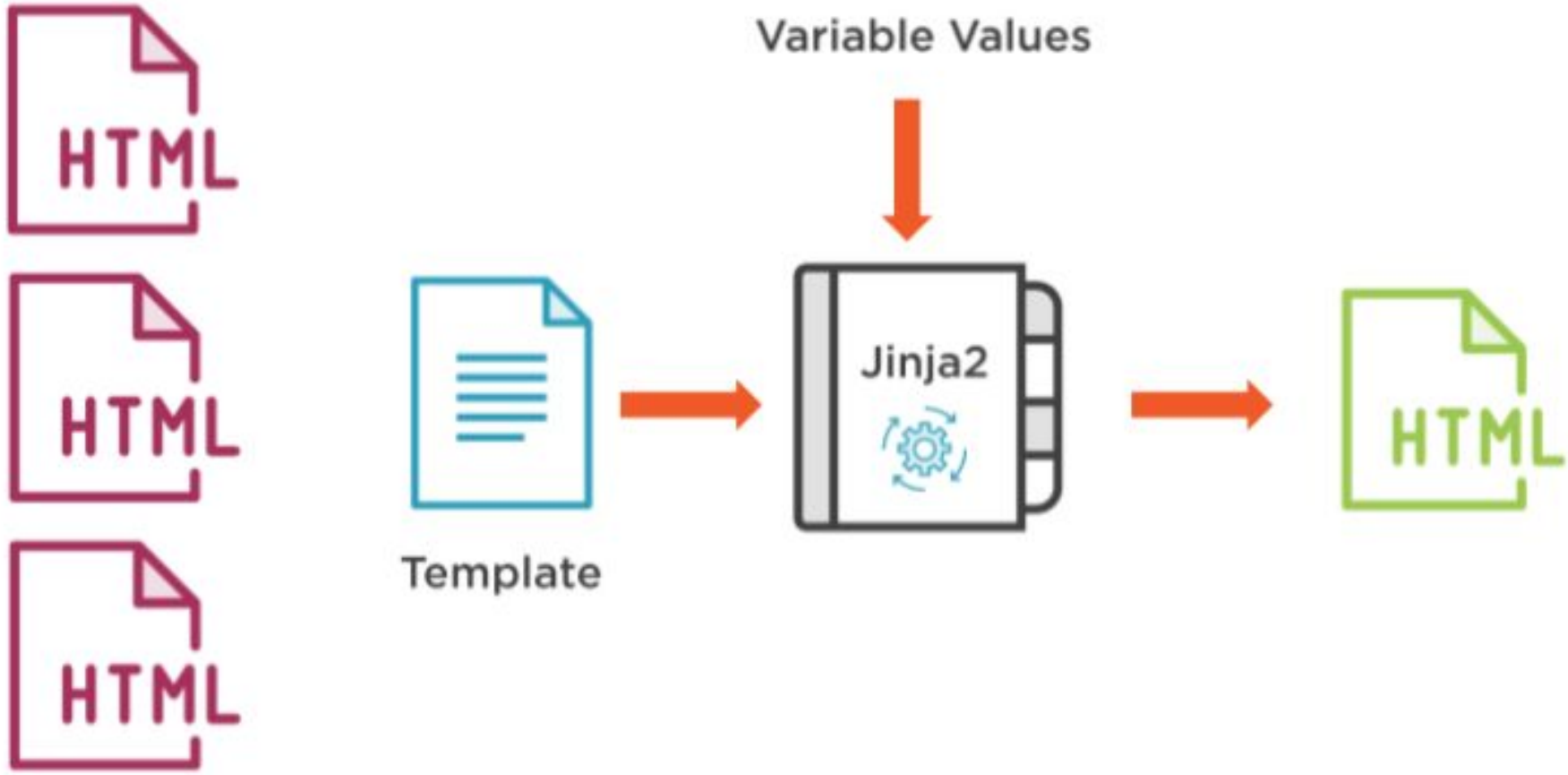
Can have multiple task files included

ORSYS formation

D’autres dossiers Ansible Role

rolename
- meta
- tasks
- vars
- templates → Templated configuration recipes
- files → Hold files that need to be copied to hosts
- defaults → Good for variable defaults with overrides

Templates Ansible avec Jinja2



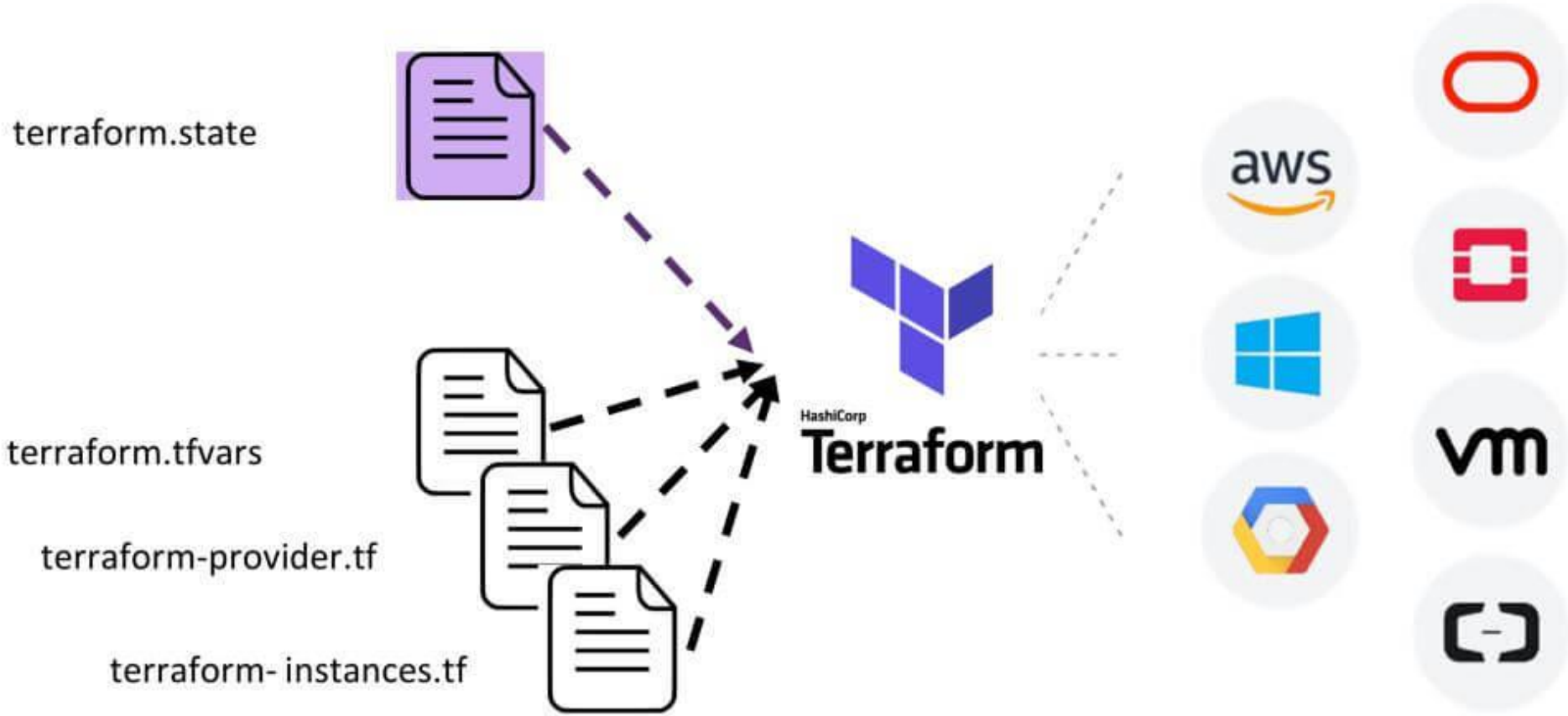
Terraform

ORSYS formation

- Terraform est un outil permettant de créer, modifier et versionner une infrastructure de manière sûre et efficace.
- Terraform peut gérer différents fournisseurs d'infrastructure, allant des Cloud Providers (AWS, Azure, GcP, Alibaba Cloud, etc ...)
- Ainsi que les solutions internes personnalisées (VmWare, Kubernetes, etc ...).

Terraform

ORSYS formation



ORSYS formation

Démonstration avec Terraform et Aws

ORSYS formation

Containers avec Docker

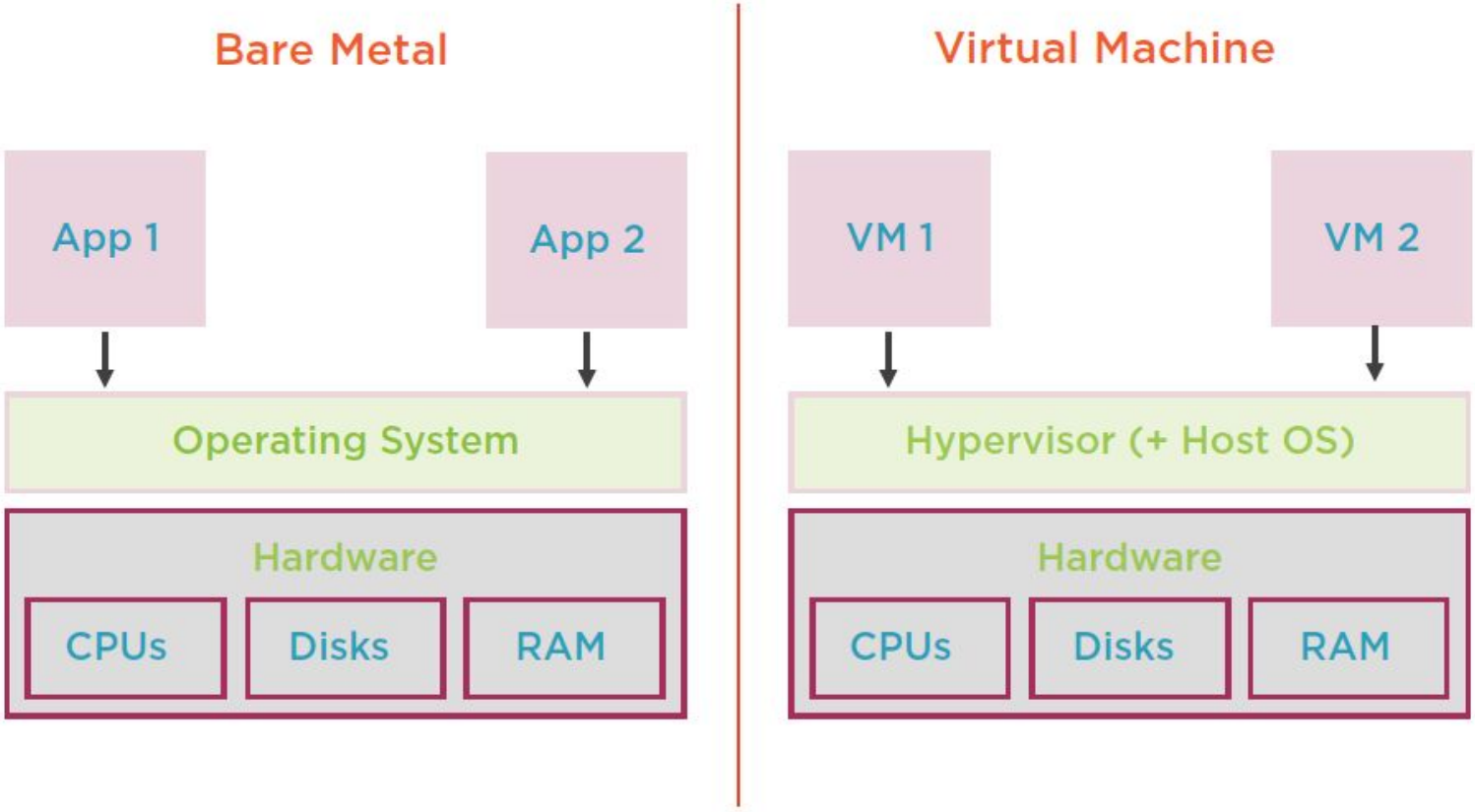
ORSYS formation

Agenda

1. Concepts de base : immutabilité, image, layers, registry, problématique réseau et stockage.
2. Automatisation avec Dockerfile/docker-compose, intégration avec Github, Jenkins, DockerHub.
3. Bénéfices attendus : reproductibilité, manageabilité.
4. Apports en termes d'élasticité, Agilité, évolutivité.
5. Impacts sur les équipes de développement et d'infrastructure.

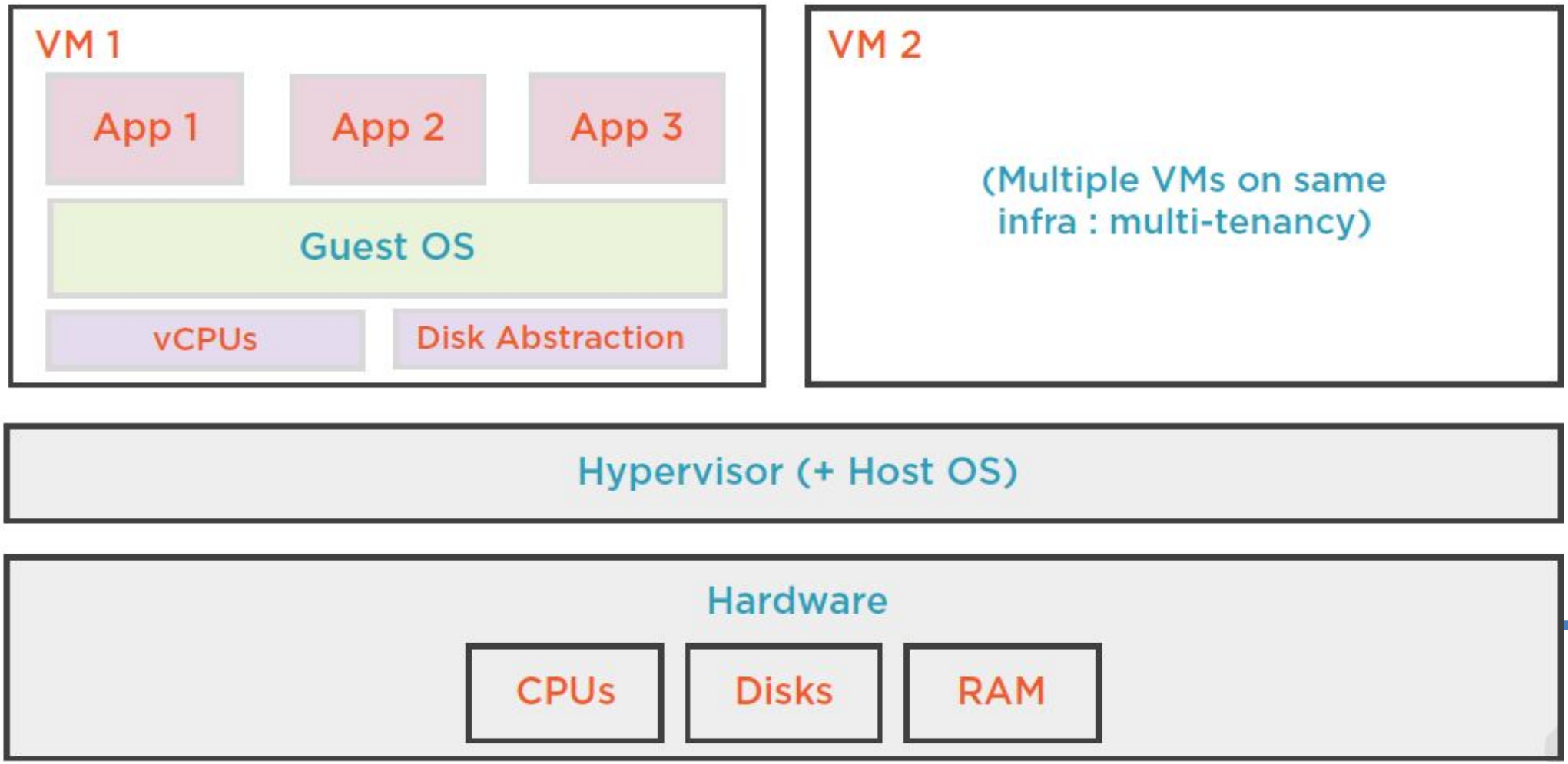
ORSYS formation

Bare Metal and Virtual Machines



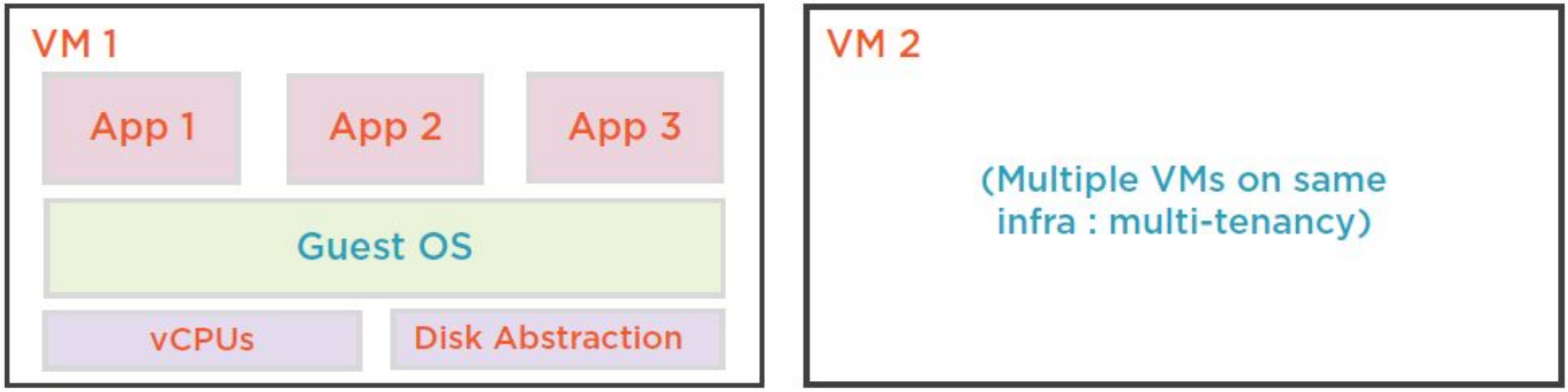
ORSYS formation

Apps on Virtual Machines



ORSYS formation

Apps on Virtual Machines

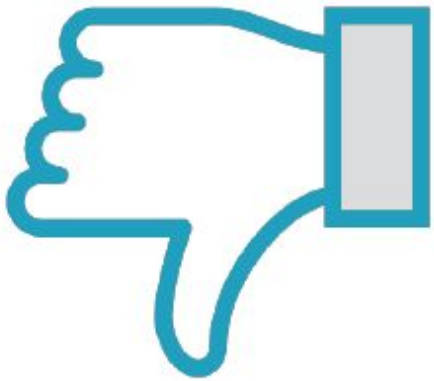


ORSYS formation

VMs are far more lightweight than bare metal environments and are often used to run applications

ORSYS formation

Drawbacks of VMs



Contain guest OS

- Introduces platform dependency
- Bloats image size to GB (apps far smaller)

Heavyweight

- Slow to boot up

Not trivial to migrate

- VM migration tools needed

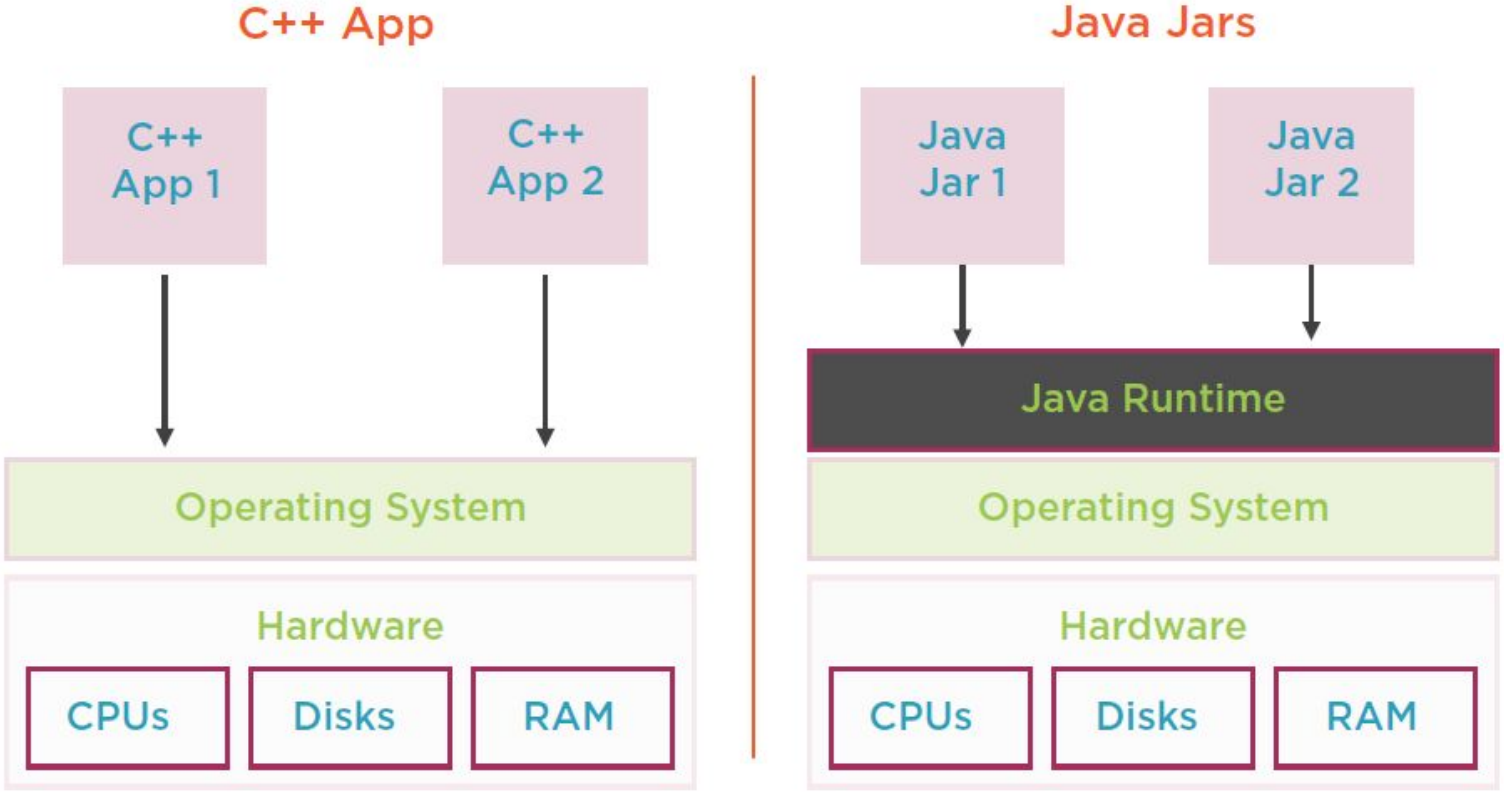
ORSYS formation

Container

A container image is a lightweight, stand-alone, executable package of a piece of software that includes everything needed to run it: code, runtime, system tools, system libraries, settings

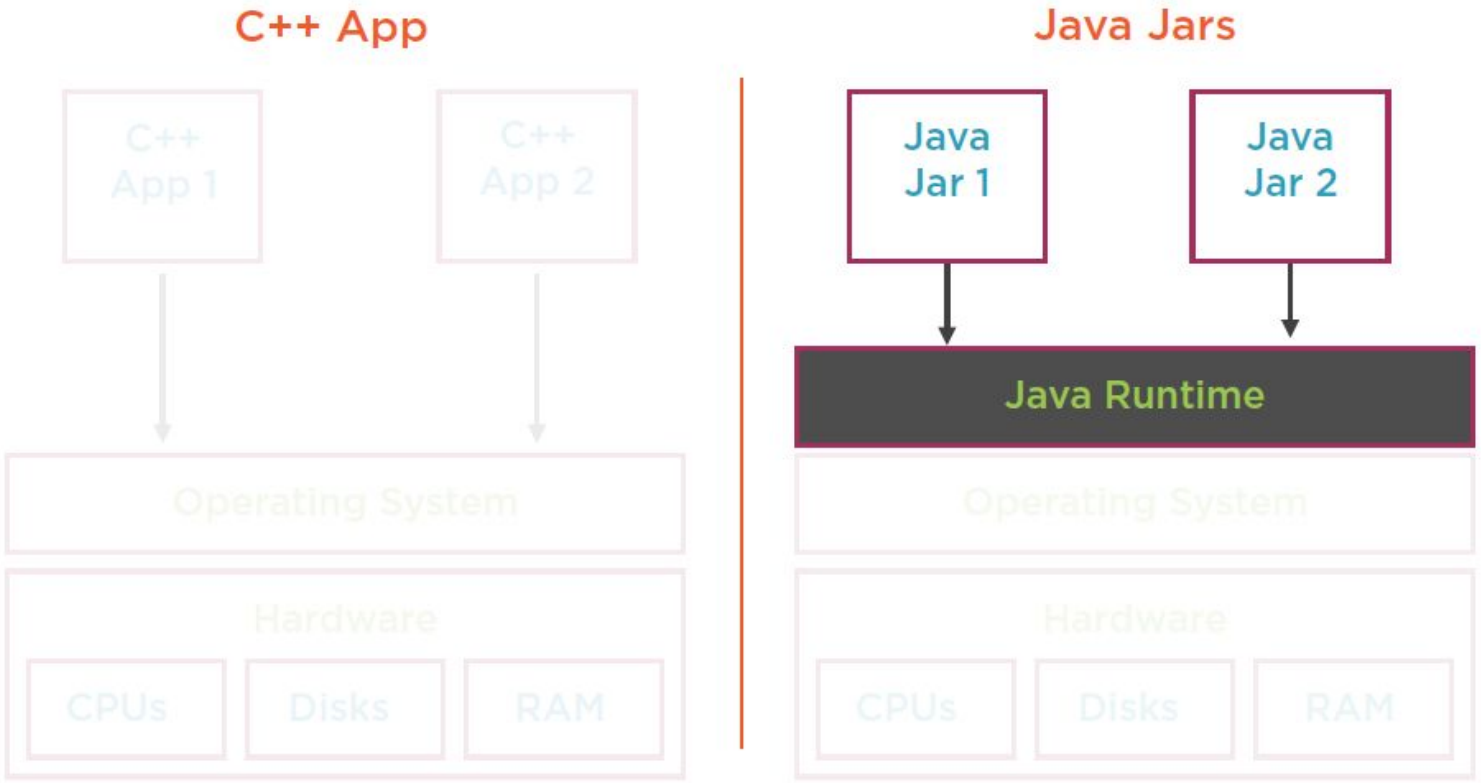
ORSYS formation

Containers Are 'Like' Jars



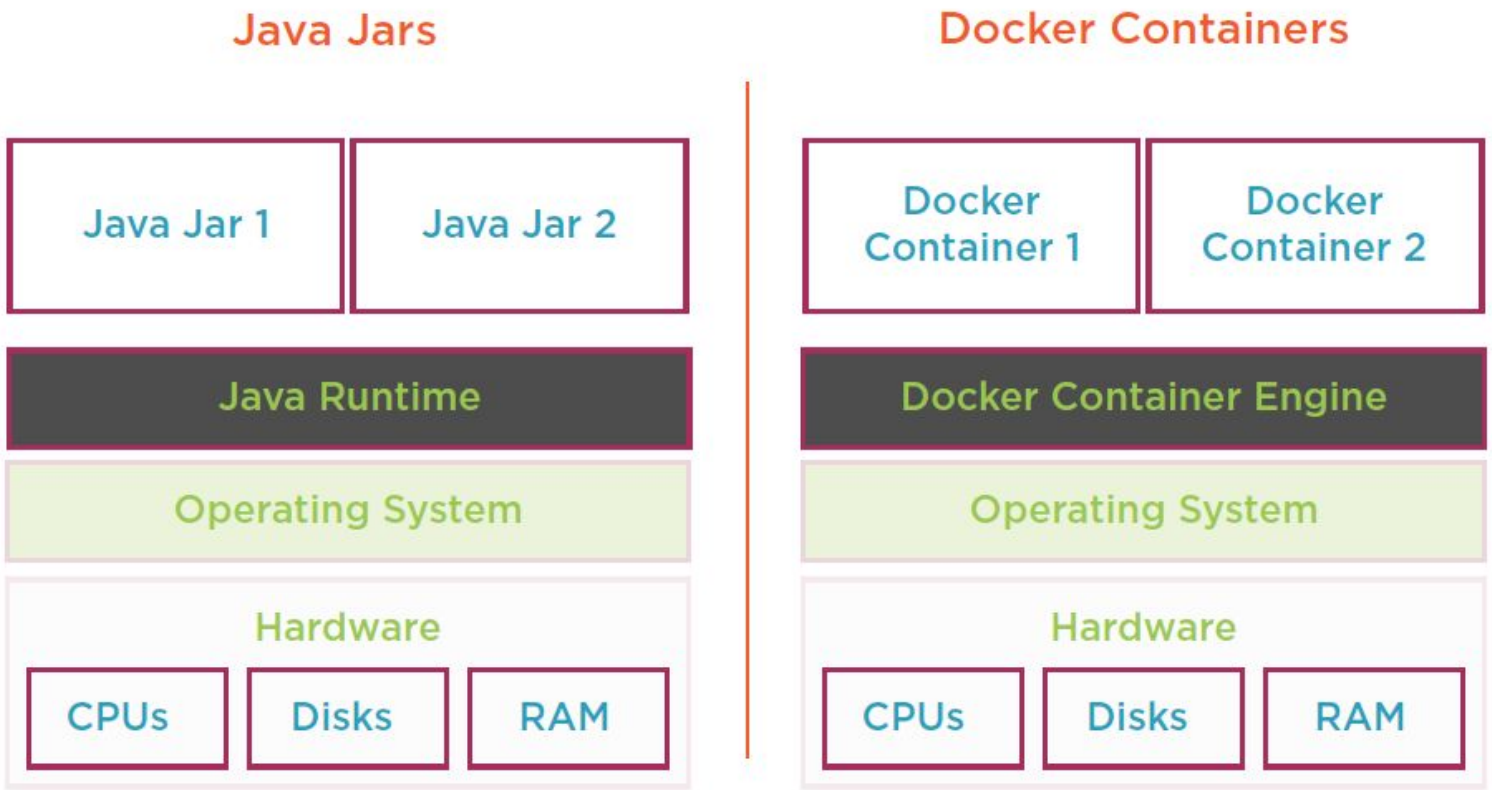
ORSYS formation

Containers Are 'Like' Jars



ORSYS formation

Containers Are 'Like' Jars



ORSYS formation

Java Jars and Containers

Java Jar Files	Containers
Files containing apps	Files containing apps
Platform independent	Platform independent
Run on layer of abstraction	Run on layer of abstraction
Java Runtime	Docker Runtime

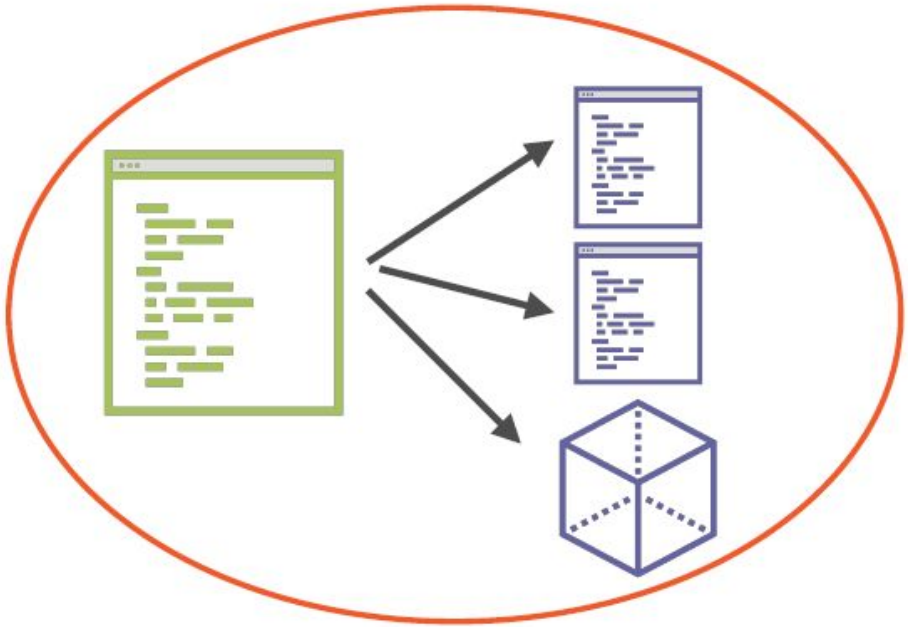
ORSYS formation

Container

A container image is a lightweight, stand-alone, executable package of a piece of software that includes everything needed to run it: code, runtime, system tools, system libraries, settings

ORSYS formation

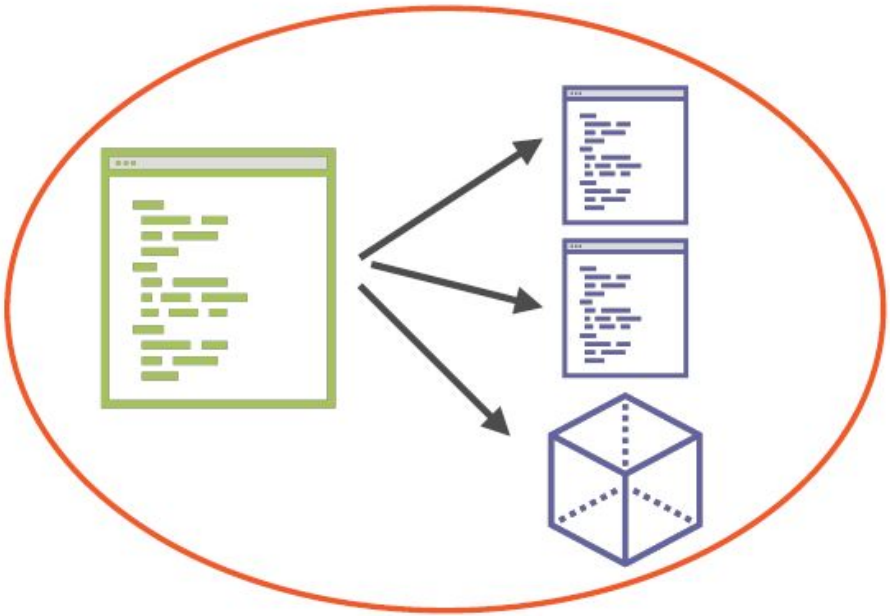
Docker Containers



A container packages code and its dependencies into an image

ORSYS formation

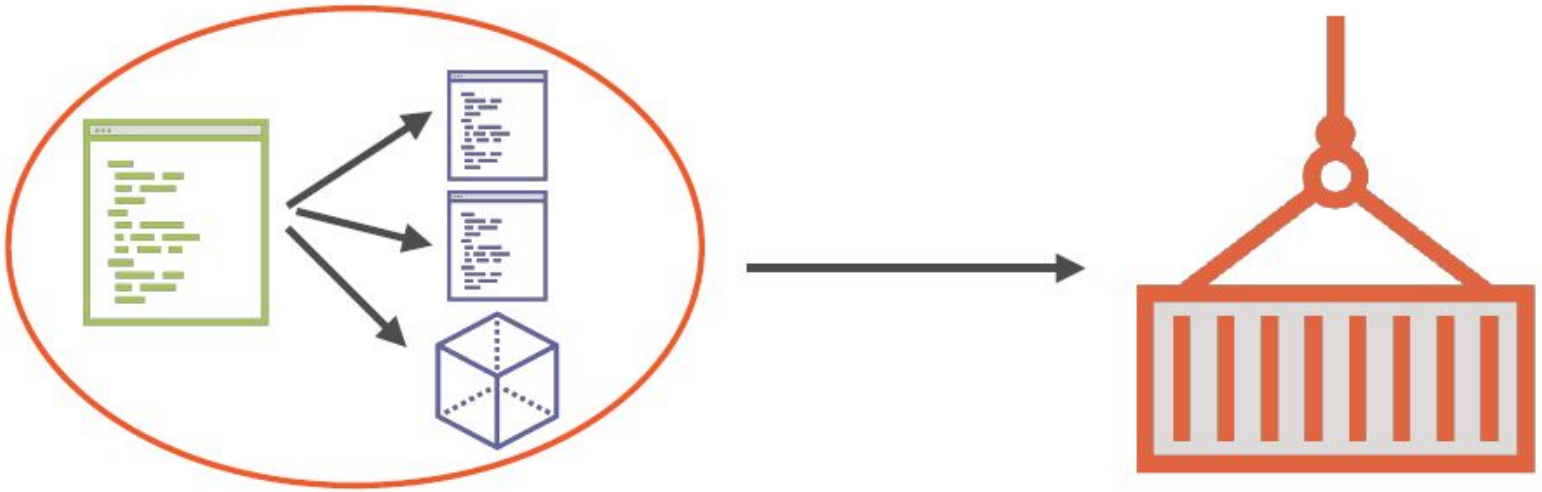
Docker Containers



A fully self-contained environment

ORSYS formation

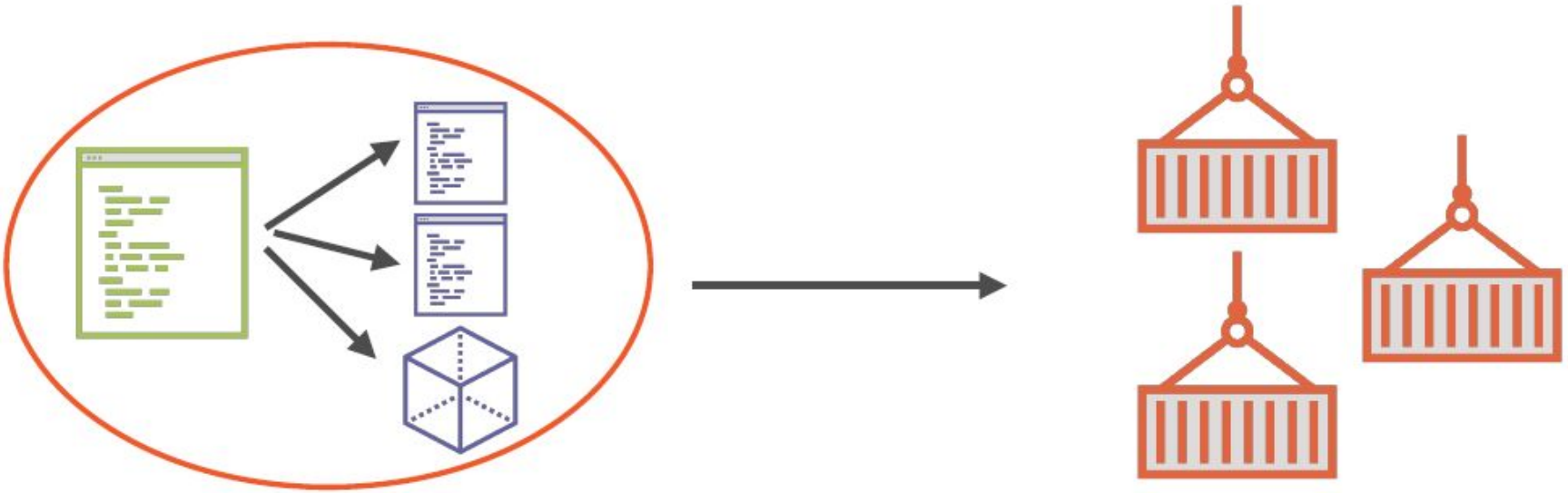
Docker Containers



Code is abstracted away from the machine

ORSYS formation

Docker Containers



Create as many containers as you want
from the same image

ORSYS formation

Démonstration:

intégration avec Github

ORSYS formation

Lab:

Qu'est-ce qu'un conteneur?

Qu'est-ce qu'un conteneur?

ORSYS formation

Processus

Les conteneurs ne sont que des processus Linux normaux avec une configuration supplémentaire appliquée. Lancez le conteneur Redis suivant afin que nous puissions voir ce qui se passe sous les couvertures.

```
> docker run -d --name=db redis:alpine
```

```
$ docker run -d --name = db redis: alpin
Impossible de trouver l'image 'redis: alpine' localement
alpin: Extraire de la bibliothèque / redis
801bfaa63ef2: Tirage complet
9a8d0188e481: Tirage complet
8a3f5c4e0176: Tirage complet
3f7cb00af226: Tirage complet
e421f2f8acb5: Tirage terminé
f41cc3c7c3e4: Pull terminé
Digest: sha256: 2cd821f730b90a197816252972c2472e3d1fad3c42f052580bc958d3ad641f96
Statut: Image plus récente téléchargée pour redis: alpin
3a1def9d32b79cf98868f85465c38acc52cb5be1b8a5e731e0f442464333bc97
```

Le conteneur Docker lance un processus appelé redis-server. Depuis l'hôte, nous pouvons voir tous les processus en cours d'exécution, y compris ceux démarrés par Docker.

```
> ps aux | grep redis-server
```

```
$ ps aux | grep redis-serveur
999 1769 0,2 1,2 28968 12892? Ssl 21:56 0:00 serveur redis
racine 1996 0,0 0,0 14220 1008 pts / 0 S + 21:58 0:00 grep --couleur = auto serveur redis
```

Qu'est-ce qu'un conteneur?

ORSYS formation

- Docker peut nous aider à identifier les informations sur le processus, y compris le PID (ID de processus) et PPID (ID de processus parent) via `>Docker top db`
- Qu'est ce que le PPID? Utilisez **`ps aux | grep <ppid>`** pour trouver le processus parent. Probablement Containerd.
- La commande **`pstree`** listera tous les sous-processus. Voir l'arborescence des processus Docker à l'aide de **`pstree -c -p -A $(pgrep dockerd)`**

```
root@user-HP-ProBook-440-G5:/home/user# pstree -c -p -A $(pgrep dockerd)
dockerd(17504)-+-{dockerd}(17505)
               |-{dockerd}(17506)
               |-{dockerd}(17507)
               |-{dockerd}(17508)
               |-{dockerd}(17509)
               |-{dockerd}(17510)
               |-{dockerd}(17511)
               |-{dockerd}(17512)
               |-{dockerd}(17513)
               |-{dockerd}(17514)
               |-{dockerd}(17515)
               |-{dockerd}(17516)
               |-{dockerd}(17517)
               |-{dockerd}(17518)
               |-{dockerd}(17519)
               `-{dockerd}(17639)
```

docker top db

ORSYS formation

Répertoire de processus

- Linux n'est qu'une série de fichiers et de contenus magiques, ce qui rend amusant l'exploration et la navigation pour voir ce qui se passe sous les couvertures, et dans certains cas, modifiez le contenu pour voir les résultats.

- La configuration de chaque processus est définie dans le `/proc` répertoire. Si vous connaissez l'ID de processus, vous pouvez identifier le répertoire de configuration.

- La commande ci-dessous répertorie tout le contenu de `/proc` et stocke le PID Redis pour une utilisation future **`DBPID=$(pgrep redis-server) echo Redis is $DBPID ls /proc`**

```
root@user-HP-ProBook-440-G5:/home/user# DBPID=$(pgrep redis-server)
root@user-HP-ProBook-440-G5:/home/user# echo Redis is $DBPID
Redis is 23692
```

- Chaque processus a sa propre configuration et ses propres paramètres de sécurité définis dans différents fichiers.
- Par exemple, vous pouvez afficher et mettre à jour les variables d'environnement définies pour ce processus. **`cat /proc/$DBPID/environ`**

`> docker exec -it db env`

ORSYS formation

Lab :

Lancement de conteneurs à l'aide de Podman et Libpod

ORSYS formation

Installer podman sur Ubuntu 18

- Installer les packages podman:

```
> sudo apt-get install software-properties-common -y
> sudo add-apt-repository -y ppa:projectatomic/ppa
> sudo apt-get install podman -y
> podman info
```

- Travailler avec Podman:

Avant de commencer, vous devrez définir les registres docker.io dans le fichier **/etc/containers/registries.conf**.

```
> echo -e "[registries.search]\nregistries = ['docker.io']" | sudo tee /etc/containers/registries.conf
```

Ensuite, téléchargez et exécutez l'image hello-word avec la commande suivante :

```
> podman run hello-world
```

ORSYS formation

Installer podman sur Ubuntu 18

```
root@user-HP-ProBook-440-G5:/etc/containers# podman run hello-world
Trying to pull docker.io/library/hello-world...
Getting image source signatures
Copying blob 0e03bdcc26d7 done
Copying config bf756fb1ae done
Writing manifest to image destination
Storing signatures

Hello from Docker!
This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:
 1. The Docker client contacted the Docker daemon.
 2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
    (amd64)
 3. The Docker daemon created a new container from that image which runs the
    executable that produces the output you are currently reading.
 4. The Docker daemon streamed that output to the Docker client, which sent it
    to your terminal.

To try something more ambitious, you can run an Ubuntu container with:
 $ docker run -it ubuntu bash

Share images, automate workflows, and more with a free Docker ID:
 https://hub.docker.com/

For more examples and ideas, visit:
 https://docs.docker.com/get-started/
```

Lancement de conteneurs à l'aide de Podma et Libpod

ORSYS formation

Exercice:

libpod fournit une bibliothèque pour les applications qui cherchent à utiliser le concept Container Pod popularisé par Kubernetes. libpod contient également un outil appelé podman pour gérer les pods , les conteneurs et les images de conteneurs.

> ***podman --help***

L'interface de ligne de commande pour Podman est compatible avec Docker, ce qui signifie qu'elle devrait être familière. La commande ci-dessous lancera un nouveau conteneur.

> ***podman login docker.io***

```
root@user-HP-ProBook-440-G5:/home/user# podman login docker.io
Username: hosniah
Password:
Login Succeeded!
```

> ***podman run -d --name http-noports hosniah/docker-http-server:1.0***

```
root@user-HP-ProBook-440-G5:/home/user# podman run -d --name http-noports hosniah/docker-http-server:1.0
Trying to pull docker.io/hosniah/docker-http-server:1.0...
Getting image source signatures
Copying blob f139eb4721ae skipped: already exists
Copying config 0d6ee549ae done
Writing manifest to image destination
Storing signatures
56c92b0f01de9b68c0bcc5419538a1a0939018712abf9bdc5498c5fd4967c0e6
```

Lancement de conteneurs à l'aide de Podman et Libpod

ORSYS formation

```
> podman ps
```

```
root@user-HP-ProBook-440-G5:/home/user# podman ps
CONTAINER ID  IMAGE                                    COMMAND  CREATED         STATUS             PORTS  NAMES
56c92b0f01de  docker.io/hosniah/docker-http-server:1.0  /app     5 minutes ago  Up 5 minutes ago          http-noports
```

Étant donné que ces conteneurs ne s'exécutent pas via Docker, ils ne seront pas visibles via l'interface de ligne de commande Docker.

```
> docker ps
```

Les conteneurs Libpod ont les mêmes principes en termes de configuration et de sécurité.
Si nous voulions que les ports soient accessibles, nous devons les lier explicitement.

```
> podman run -d --name http -p 80:80 hosniah/docker-http-server:1.0
```

```
> curl localhost
```

ORSYS formation

CLI compatible Docker

La configuration d'un conteneur peut être sortie via inspect. La sortie est compatible avec l'API Docker.

```
> podman inspect http
```

Comme nous exécutons un service compatible, il est possible de définir simplement un alias pour remplacer la CLI Docker et avoir la même expérience.

```
> alias docker=podman
```

Désormais, lorsque nous exécuterons Docker ps, il sera basé sur les conteneurs exécutés par Podman.

```
> docker ps
```

CLI compatible Docker

ORSYS formation

```
root@user-HP-ProBook-440-G5:/home/user# podman inspect http
[
    {
        "Id": "28866519f23bc5185d78c0a3bf36580ab6fcfe63a7595778da2f4e24b25b1790",
        "Created": "2021-01-24T20:52:05.513358189+01:00",
        "Path": "/app",
        "Args": [
            "/app"
        ],
        "State": {
            "OciVersion": "1.0.1-dev",
            "Status": "configured",
            "Running": false,
            "Paused": false,
            "Restarting": false,
            "OOMKilled": false,
            "Dead": false,
            "Pid": 0,
            "ExitCode": 0,
            "Error": "",
            "StartedAt": "0001-01-01T00:00:00Z",
            "FinishedAt": "0001-01-01T00:00:00Z",
            "Healthcheck": {
                "Status": "",
                "FailingStreak": 0,
                "Log": null
            }
        },
        "Image": "0d6ee549ae1314ff0c0b8fea18042f3891250ba6b920fc714563be624e3c62a3",
        "ImageName": "docker.io/hosniah/docker-http-server:1.0",
        "Rootfs": "",
```

Sécurité des conteneurs Podman

ORSYS formation

Les conteneurs peuvent également être exécutés au premier plan, par exemple en lançant un shell basé sur Alpine.

```
> podman run -it alpine sh
```

Une fois que vous avez un shell, la sécurité du conteneur peut être testée en utilisant "Am I Contained" par Jessfraz .

```
> wget -O amicontained
https://github.com/jessfraz/amicontained/releases/download/v0.3.0/am
icontained-linux-amd64; chmod +x amicontained; ./amicontaine
```

Sécurité des conteneurs Podman

ORSYS formation

```
root@user-HP-ProBook-440-G5:/home/user# podman run -it alpine sh
Trying to pull docker.io/library/alpine...
Getting image source signatures
Copying blob 596ba82af5aa done
Copying config 7731472c3f done
Writing manifest to image destination
Storing signatures
/ # wget -O amicontained https://github.com/jessfraz/amicontained/releases/download/v0.3.0/amicontained-linux-amd64; chmod +x amicontained; ./
amicontained
Connecting to github.com (140.82.121.4:443)
Connecting to github.com (140.82.121.3:443)
Connecting to github-production-release-asset-2e65be.s3.amazonaws.com (52.217.85.92:443)
saving to 'amicontained'
amicontained         100% |********************************************************************************************| 1764k  0:00:00 ETA
'amicontained' saved
Container Runtime: not-found
Has Namespaces:
        pid: true
        user: false
AppArmor Profile: unconfined
Capabilities:
        BOUNDING -> chown dac_override fowner fsetid kill setgid setuid setpcap net_bind_service net_raw sys_chroot mknod audit_write setfcap
        AMBIENT -> chown dac_override fowner fsetid kill setgid setuid setpcap net_bind_service net_raw sys_chroot mknod audit_write setfcap
Chroot (not pivot_root): false
Seccomp: filtering
```

ORSYS formation

Kubernetes
orchestrateur de containers

ORSYS formation

Agenda

1. Nœuds Master/Workers, concepts de Pods, service, différents types d'Ingress Controller.
2. Stockage : stateful, stateless, shared (NFS, GlusterFS, CEPH, rook).
3. Gestion de configuration. Usage des Jobs et DaemonSets.
4. Composant interne (etcd, kubelet, kube-dns, kube-proxy, apiserver), complémentaire (Helm/Tiller, envoy, side-car proxy).
5. Service Discovery/Mesh (Istio), calico, cilium.

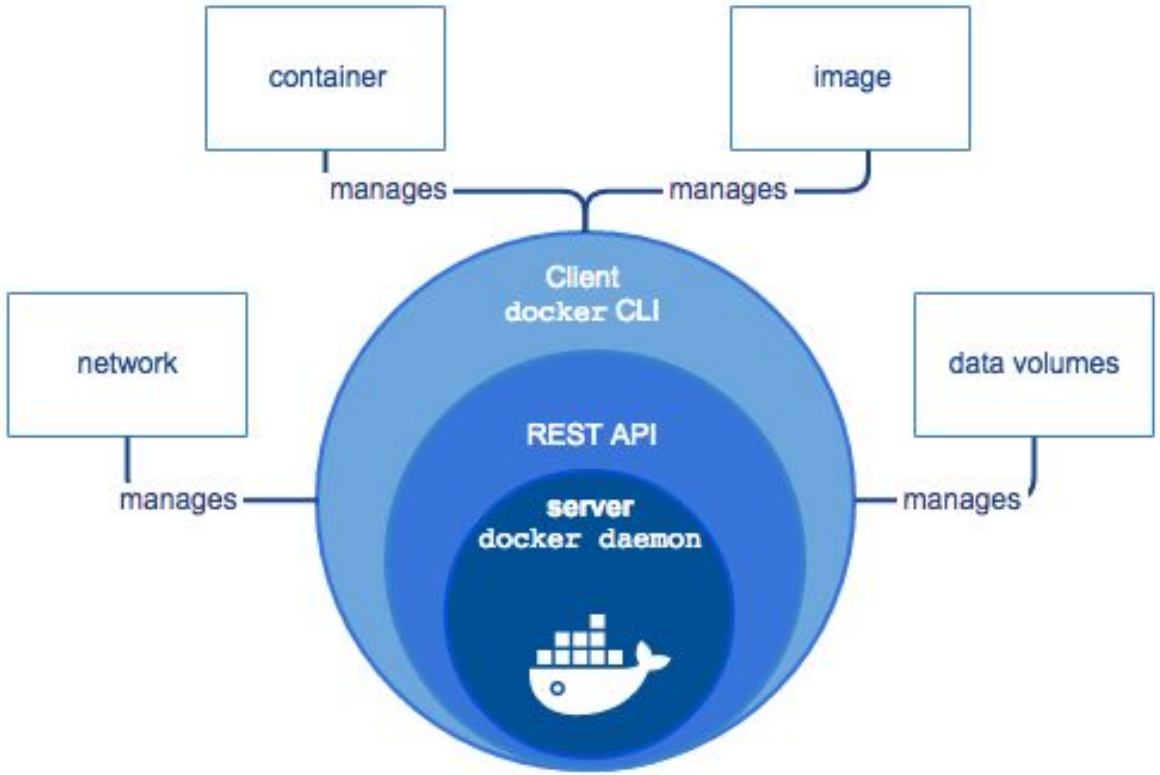
ORSYS formation

Containers?

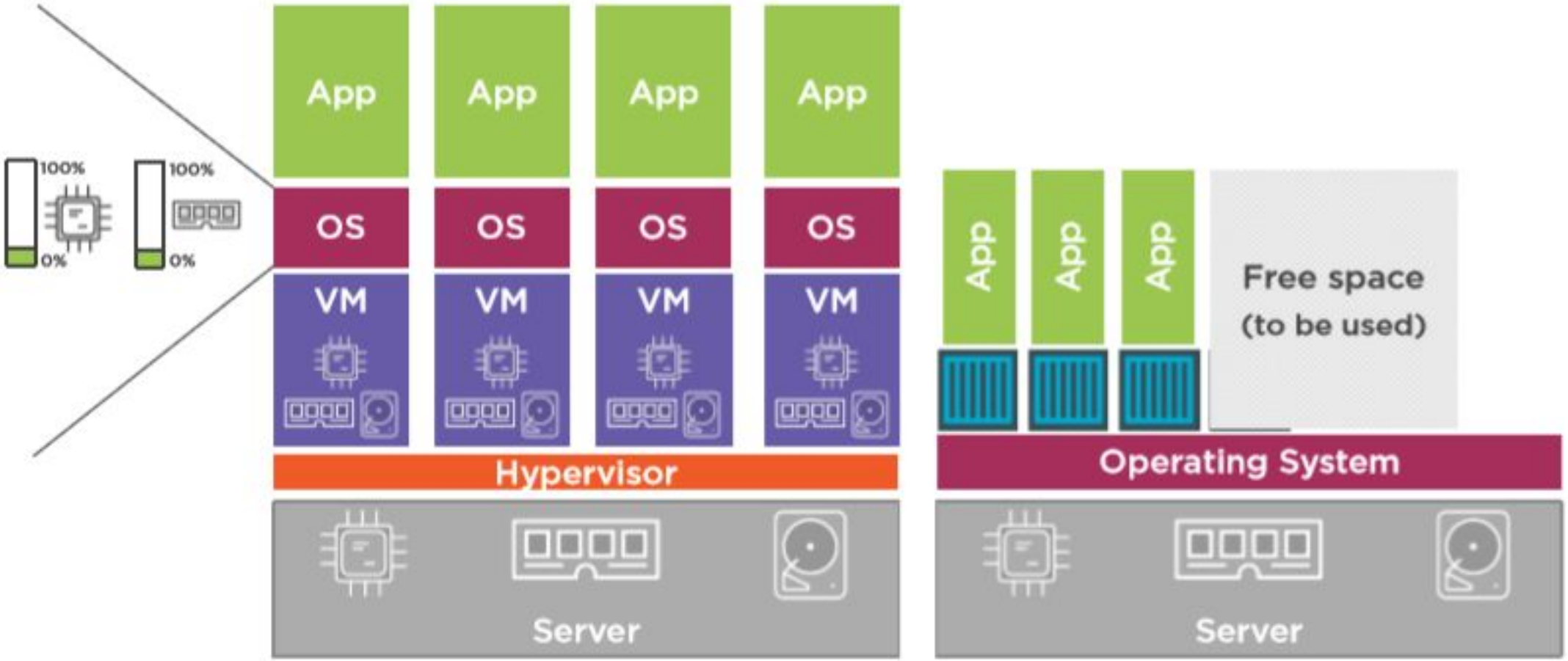
- Namespace?
- Cgroups?
- Layered filesystems (Rootfs)?

ORSYS formation

Docker



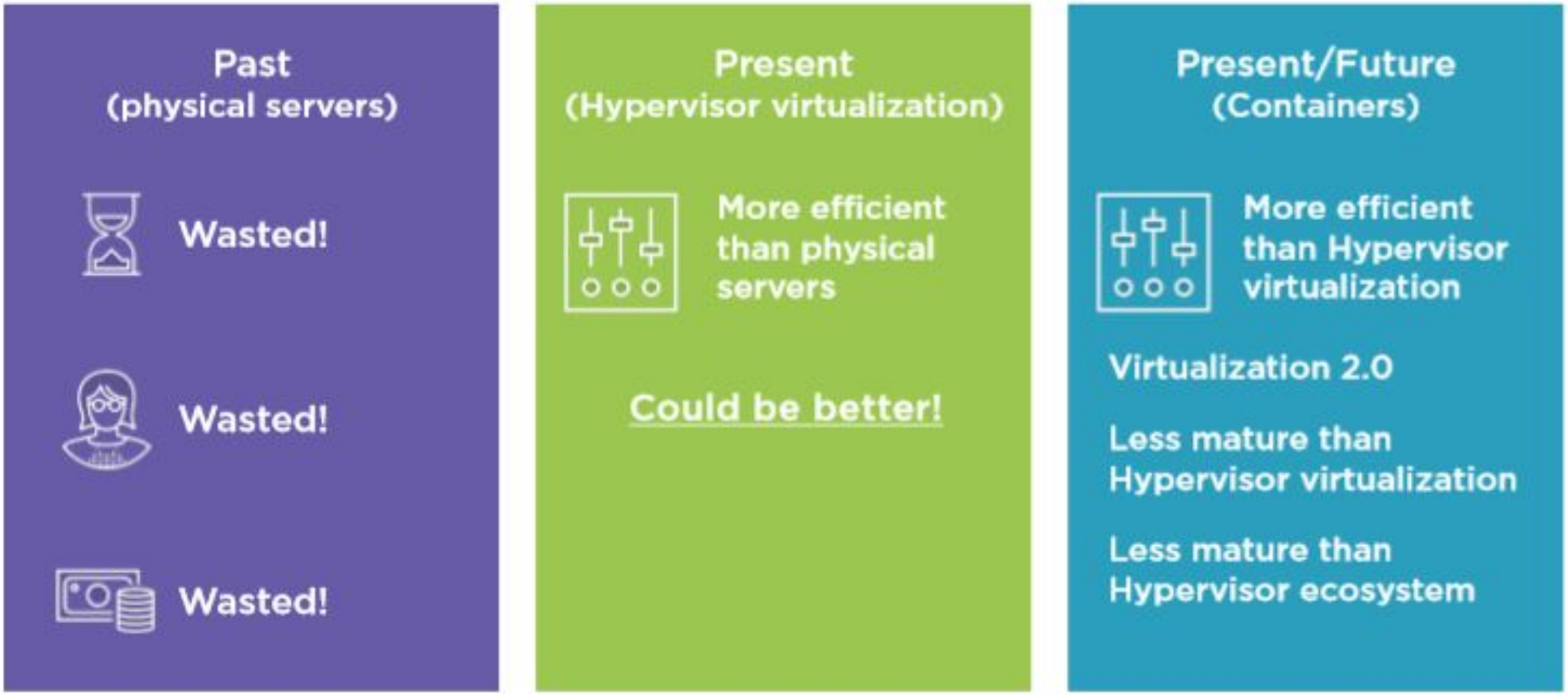
Architecture de base de Kubernetes



Hypervisor Architecture

Container Architecture

Architecture de base de Kubernetes

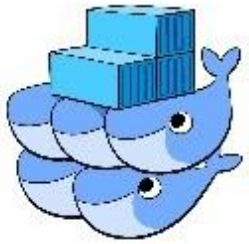


ORSYS formation

Orchestration de containers?

Container Orchestration Tools

Marathon (Mesosphere)



Docker Swarm

Nomad (HashiCorp)

Kubernetes

ORSYS formation

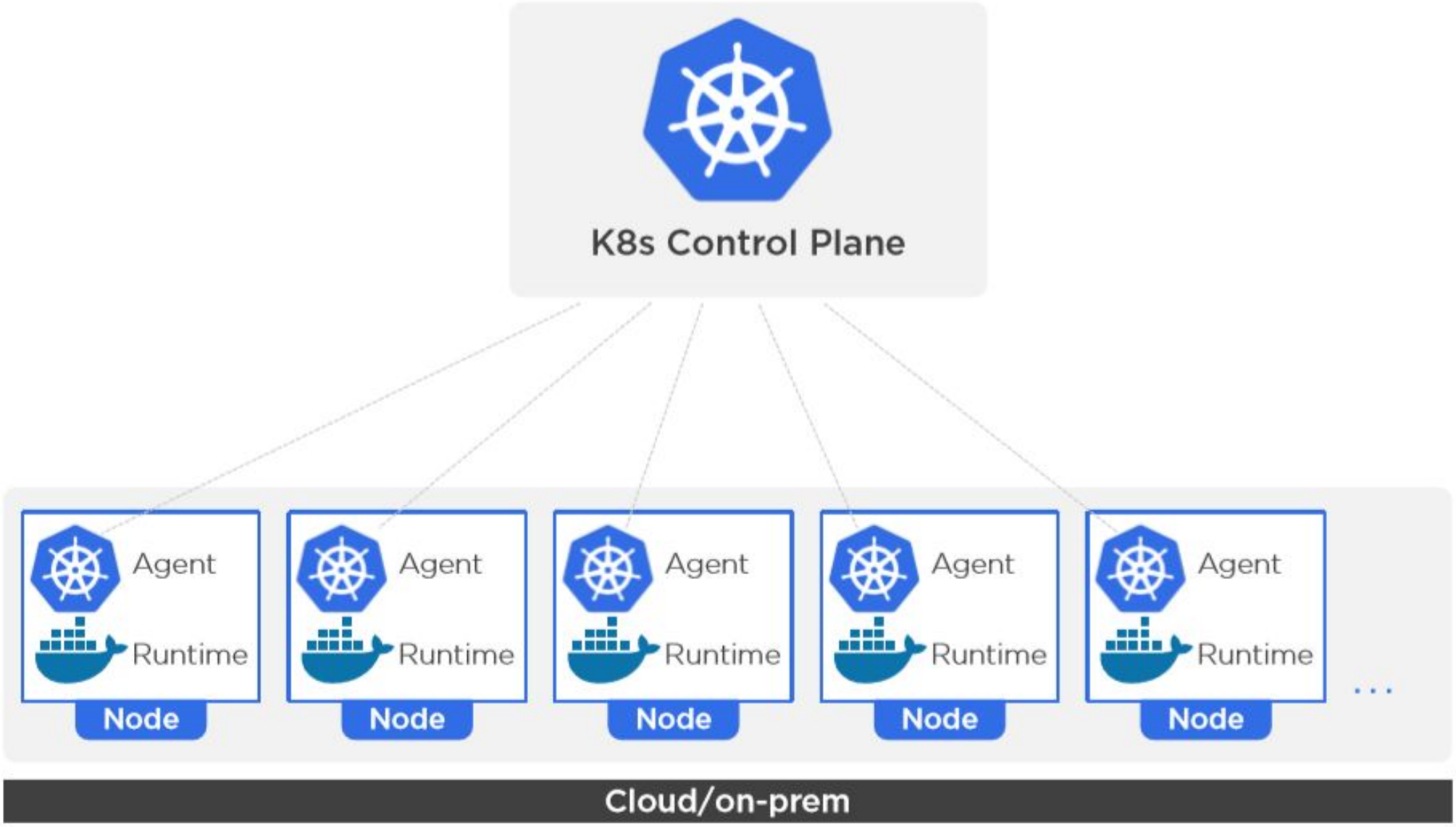
Kubernetes?

Scheduling, scaling, healing, updating...

start | stop | delete ...

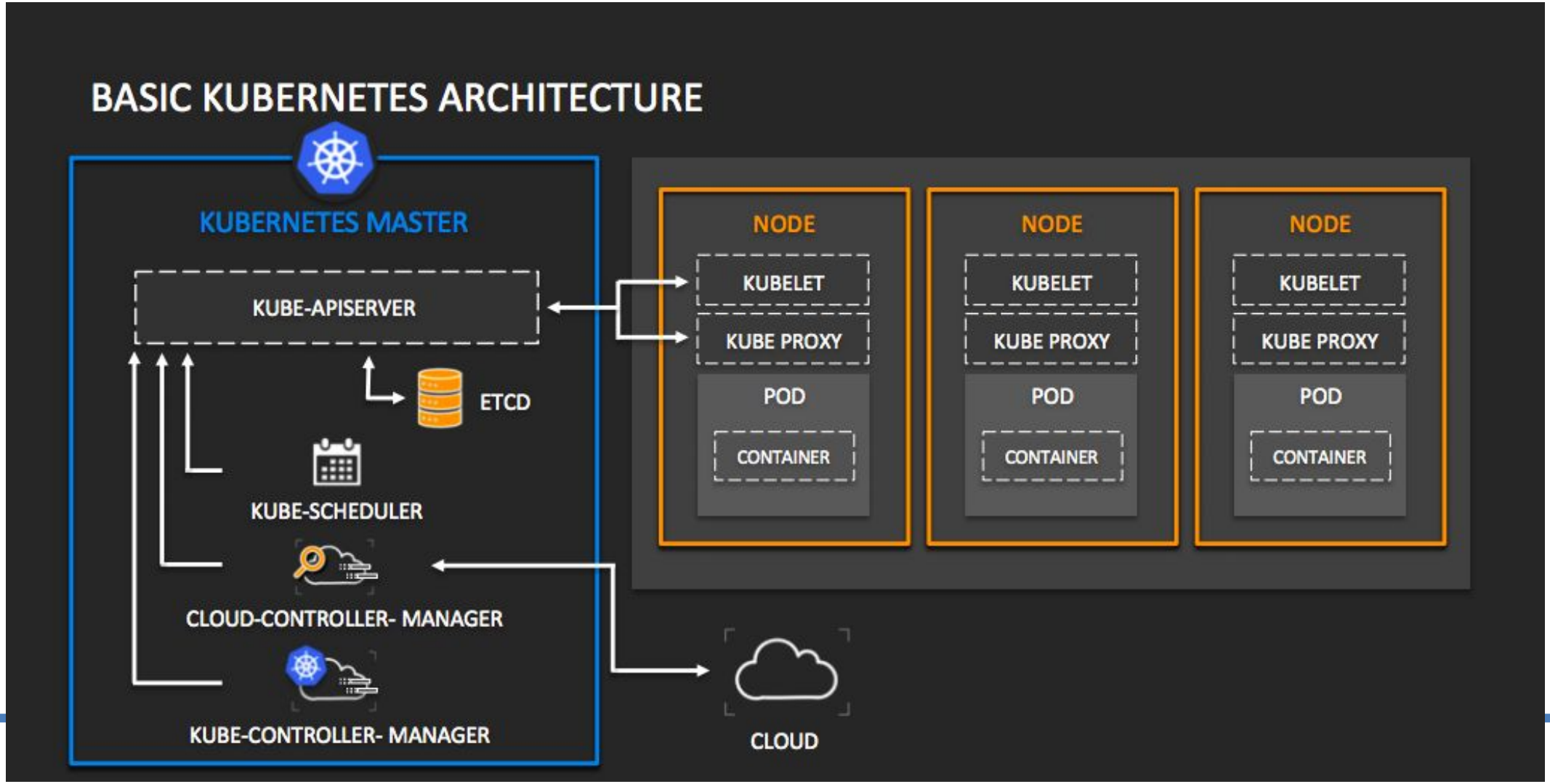
ORSYS formation

Kubernetes?



ORSYS formation

Architecture de base de Kubernetes

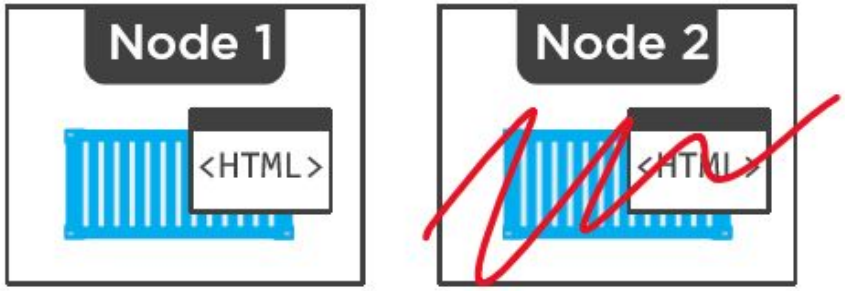


ORSYS formation

Stateless vs Stateful

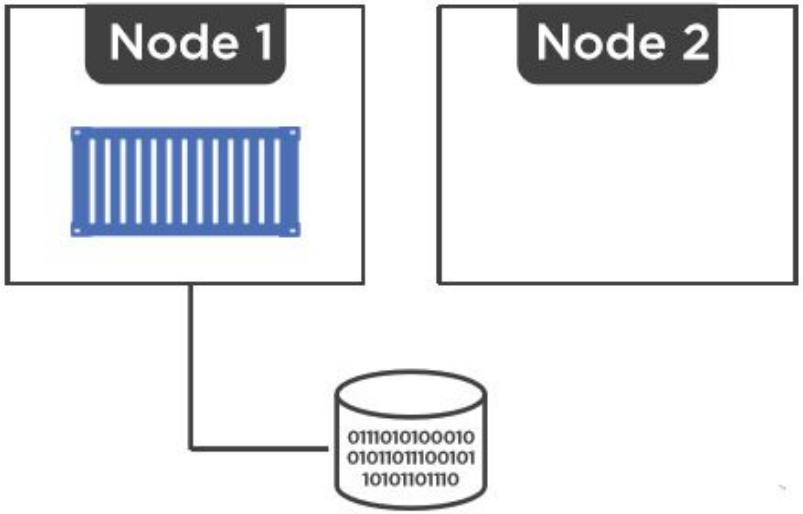
✓ Stateless

- Doesn't remember stuff



Stateful ✓

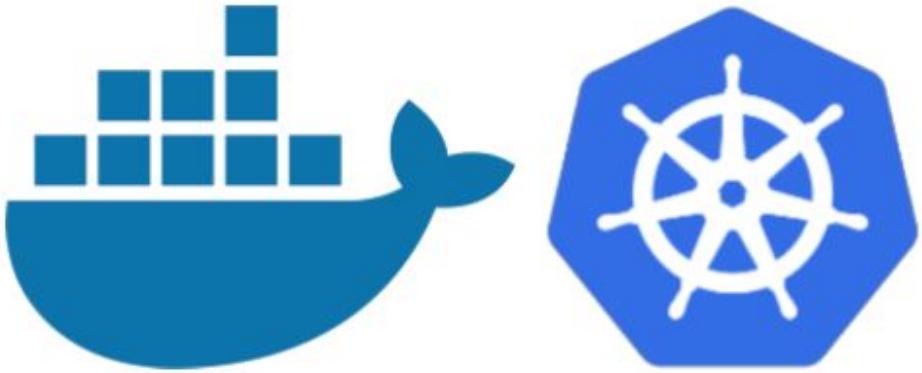
- Has to remember stuff



ORSYS formation

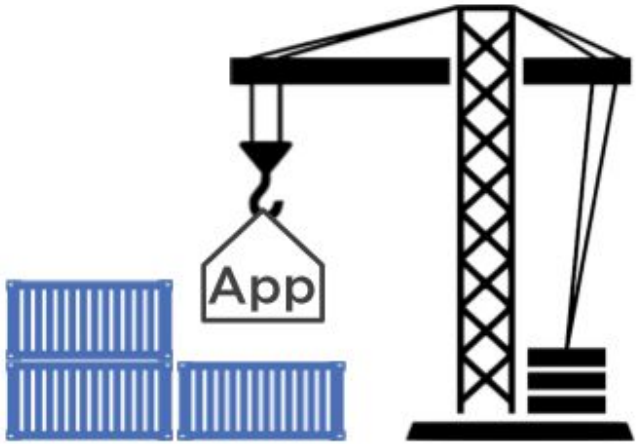
Migrating d'applications legacy

Stateful



Added features for stateful apps

Legacy



Some apps can be migrated directly to containers

ORSYS formation

Helm!

ORSYS formation

Attractions of Containers

No guest OS

- Platform independent
- Considerably smaller than VM images

Lightweight

- Small and fast
- Quick to start
- Speeds up autoscaling

Hybrid, multi-cloud

- Hybrid: Work on-premise and on cloud
- Multi-cloud: Not tied to any specific cloud platform

Standalone Container Limitations

No autohealing

- Crashed containers won't restart automatically
- Need higher level orchestration

No scaling or autoscaling

- Overloaded containers don't spawn more automatically
- Need higher level orchestration

No load balancing

- Containers can't share load automatically
- Need higher level orchestration

No isolation

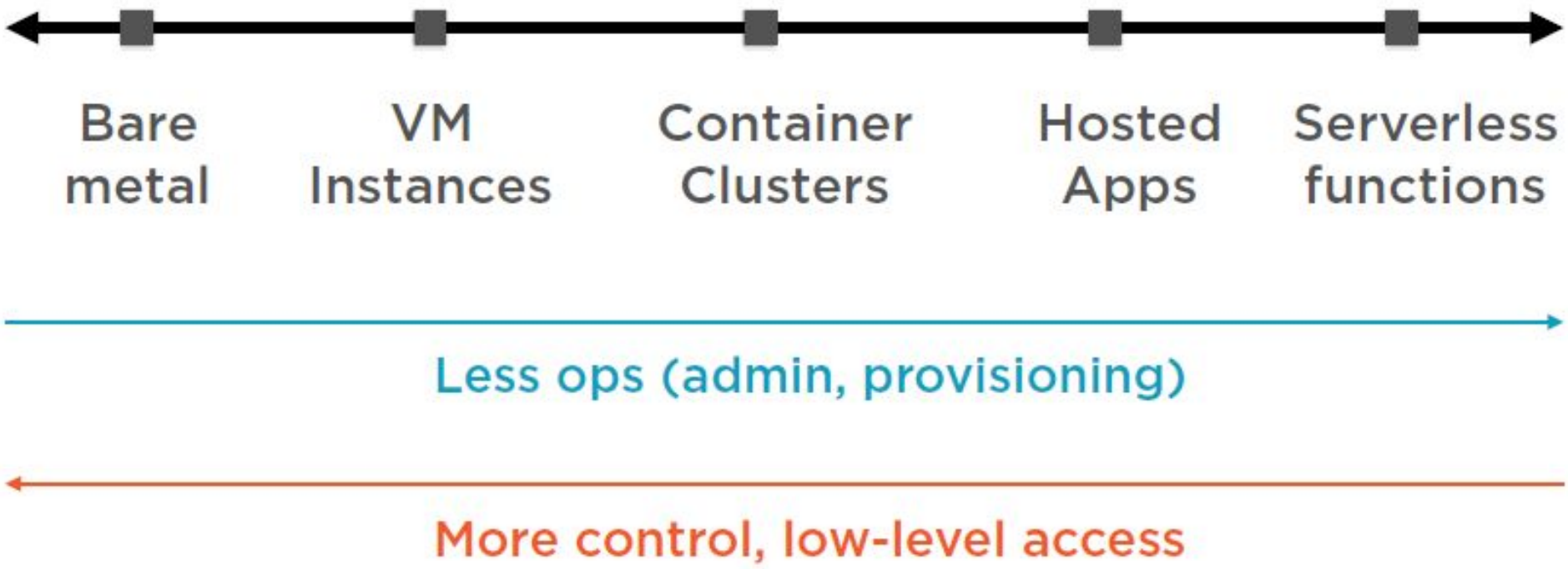
- Crashing containers can take each other down
- Need sandbox to separate them

ORSYS formation

Kubernetes

Orchestration technology for containers - convert isolated containers running on different hardware into a cluster

Compute Choices



Kubernetes is fast emerging as middle-ground between IaaS and PaaS in a hybrid, multi-cloud world

Hybrid, Multi-cloud

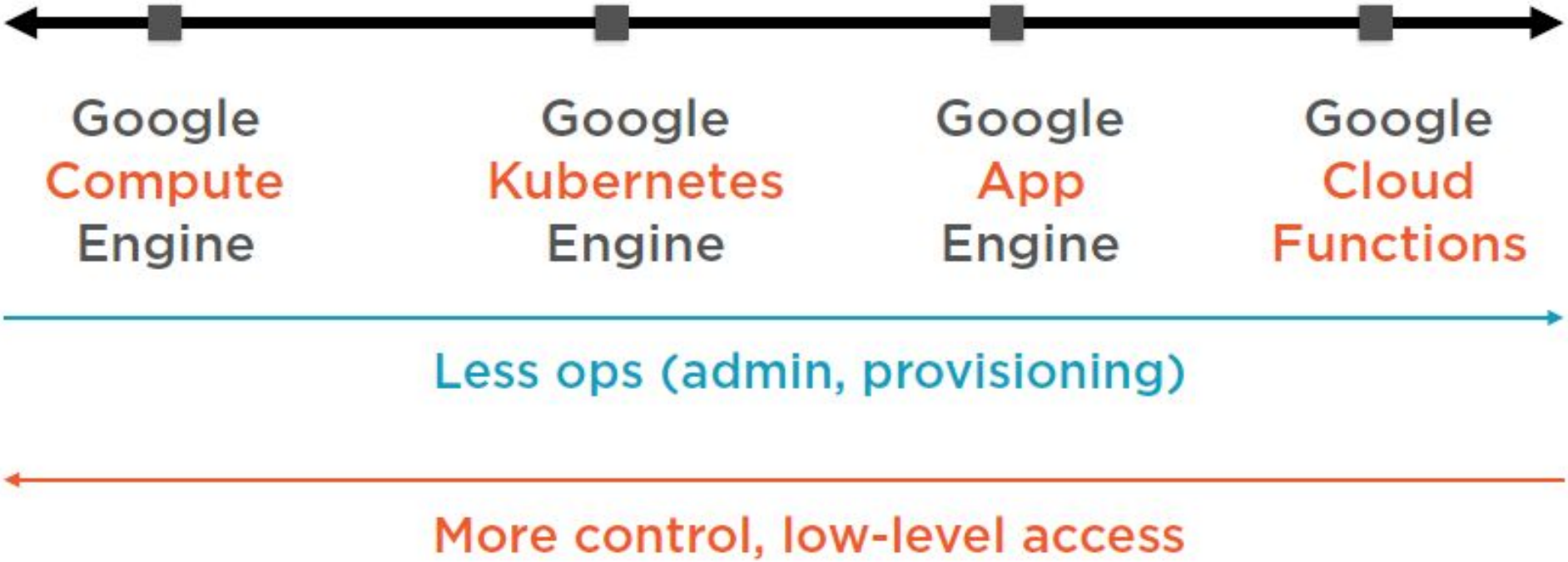
Hybrid: Runs on-premise and on cloud

- Provides smooth migration path

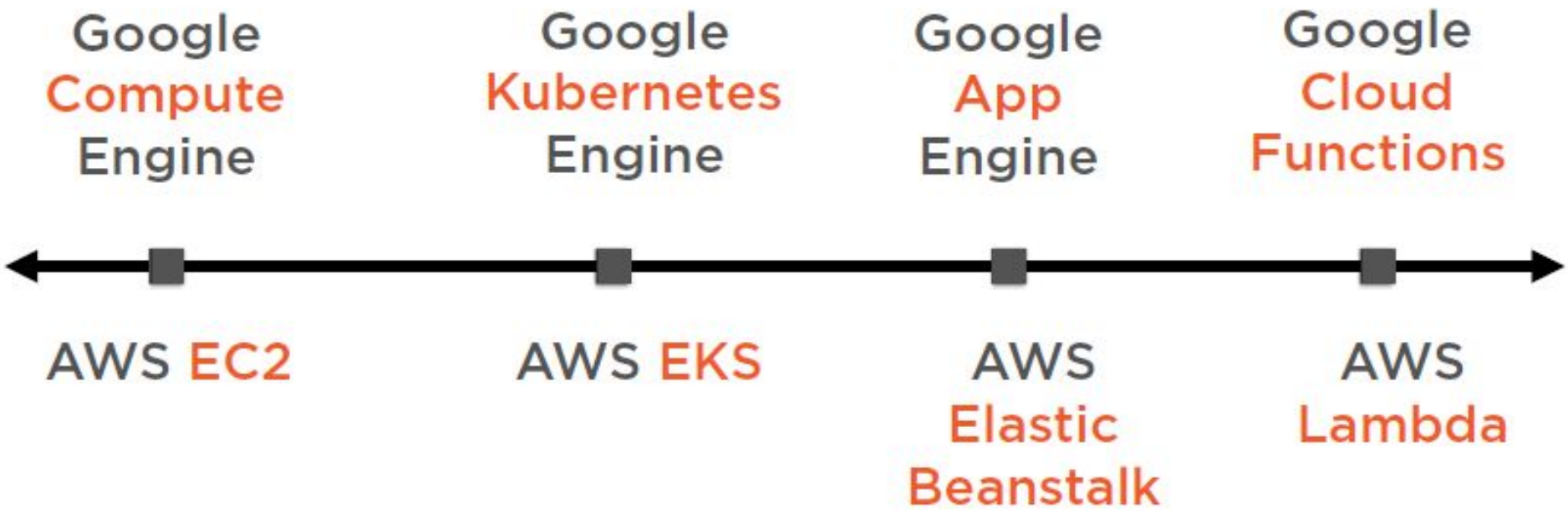
Multi-cloud: Supported by all big cloud platforms

- Important for strategic independence
- Amazon-Whole Foods merger

GCP Compute Choices



GCP Compute Choices

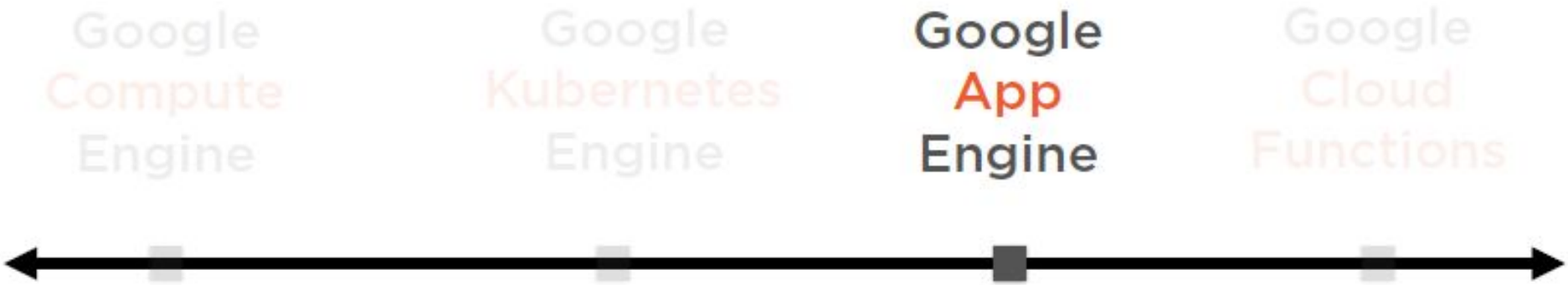


Every major cloud platform supports the same range of compute choices

GCP Compute Choices

"Infrastructure-as-a-Service" (IaaS)

GCP Compute Choices



"Platform-as-a-Service" (PaaS)

IaaS vs. PaaS

Infrastructure-as-a-Service

Heavy operational burden

Migration is hard

Platform-as-a-Service

Provider lock-in

Migration is very hard

Compute Choices

Containers

hybrid, multi-cloud

Container Clusters

Kubernetes

Kubernetes as Orchestrator

Fault-tolerance

Autohealing

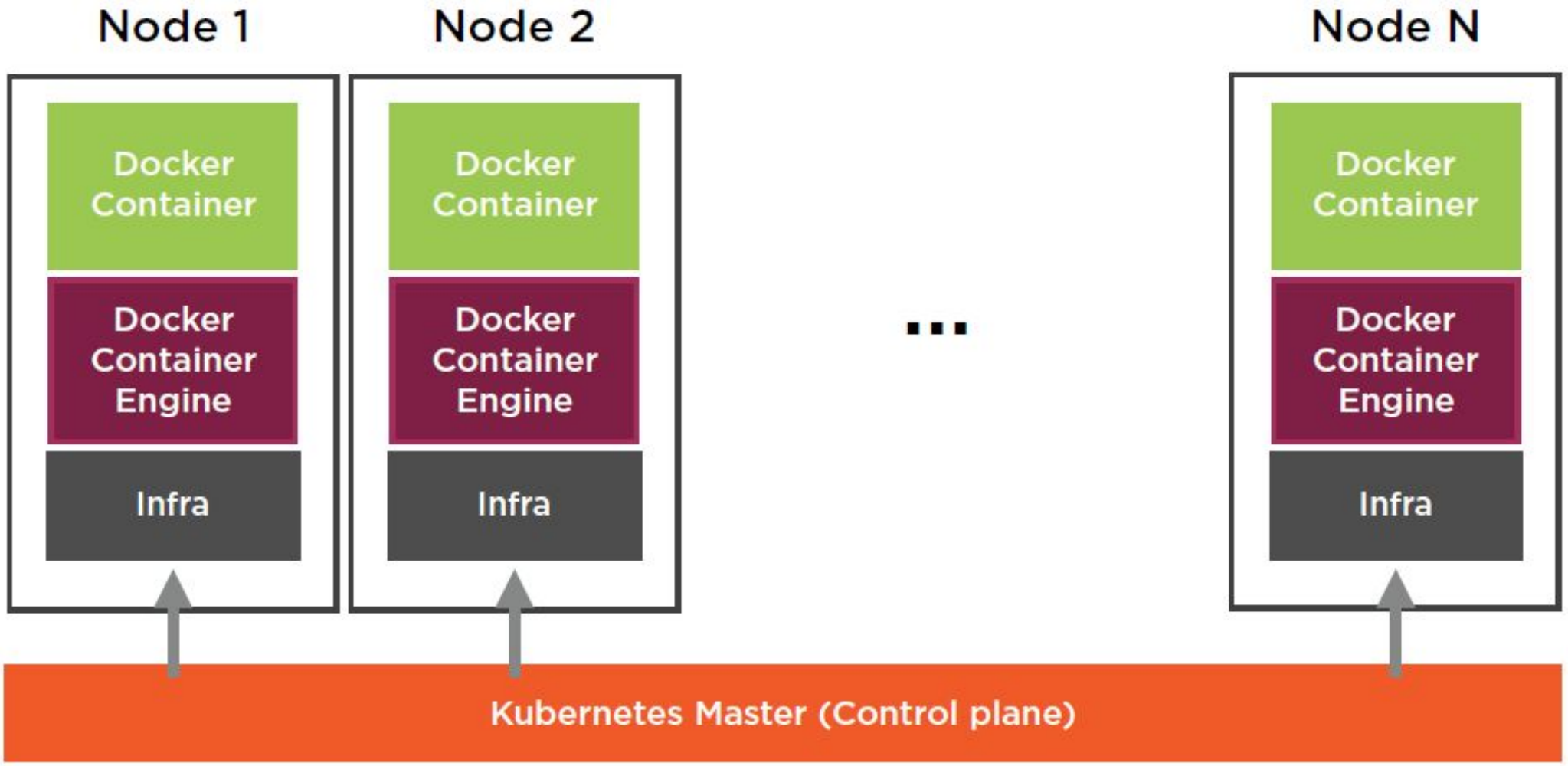
Isolation

Scaling

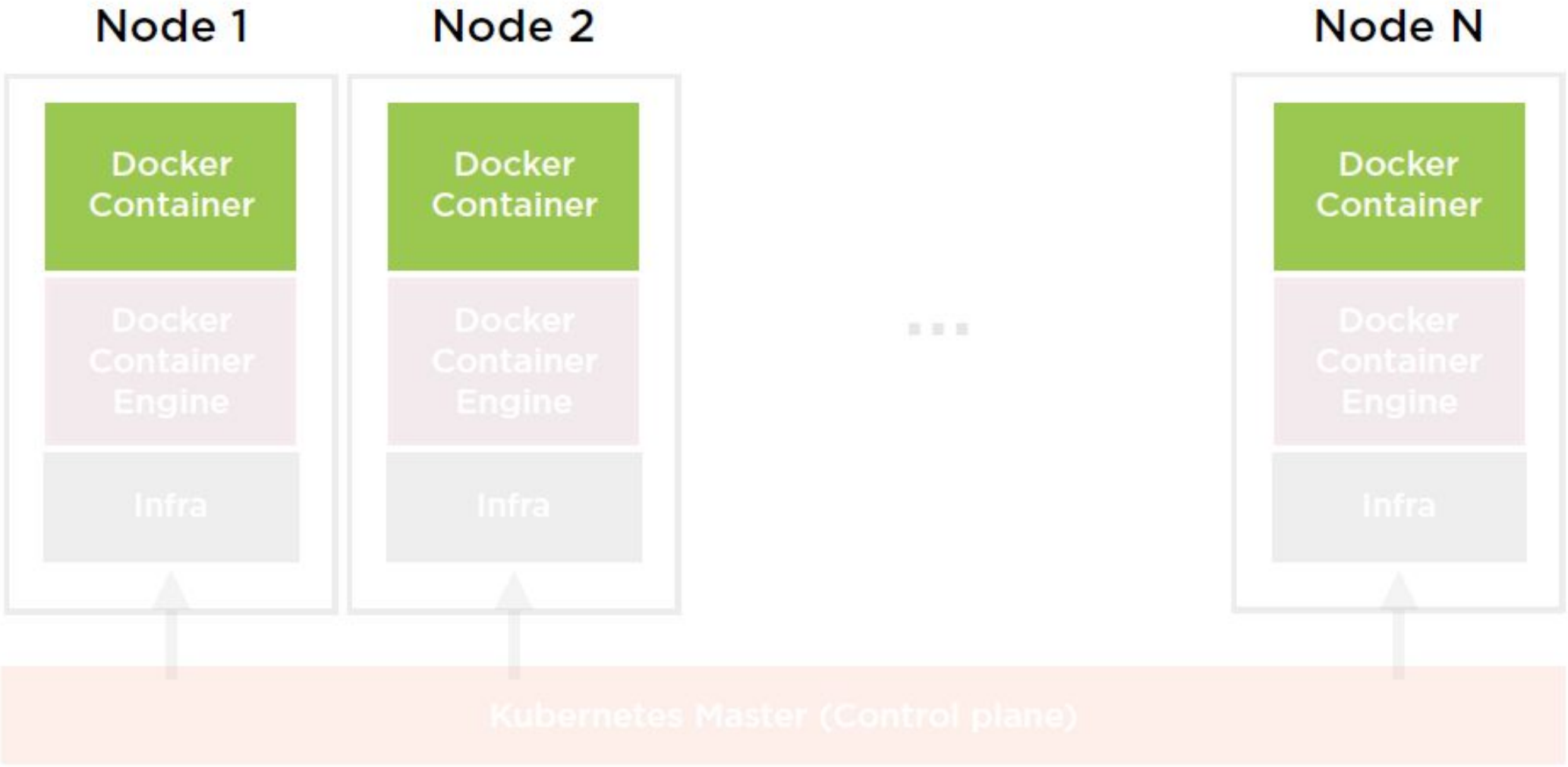
Autoscaling

Load balancing

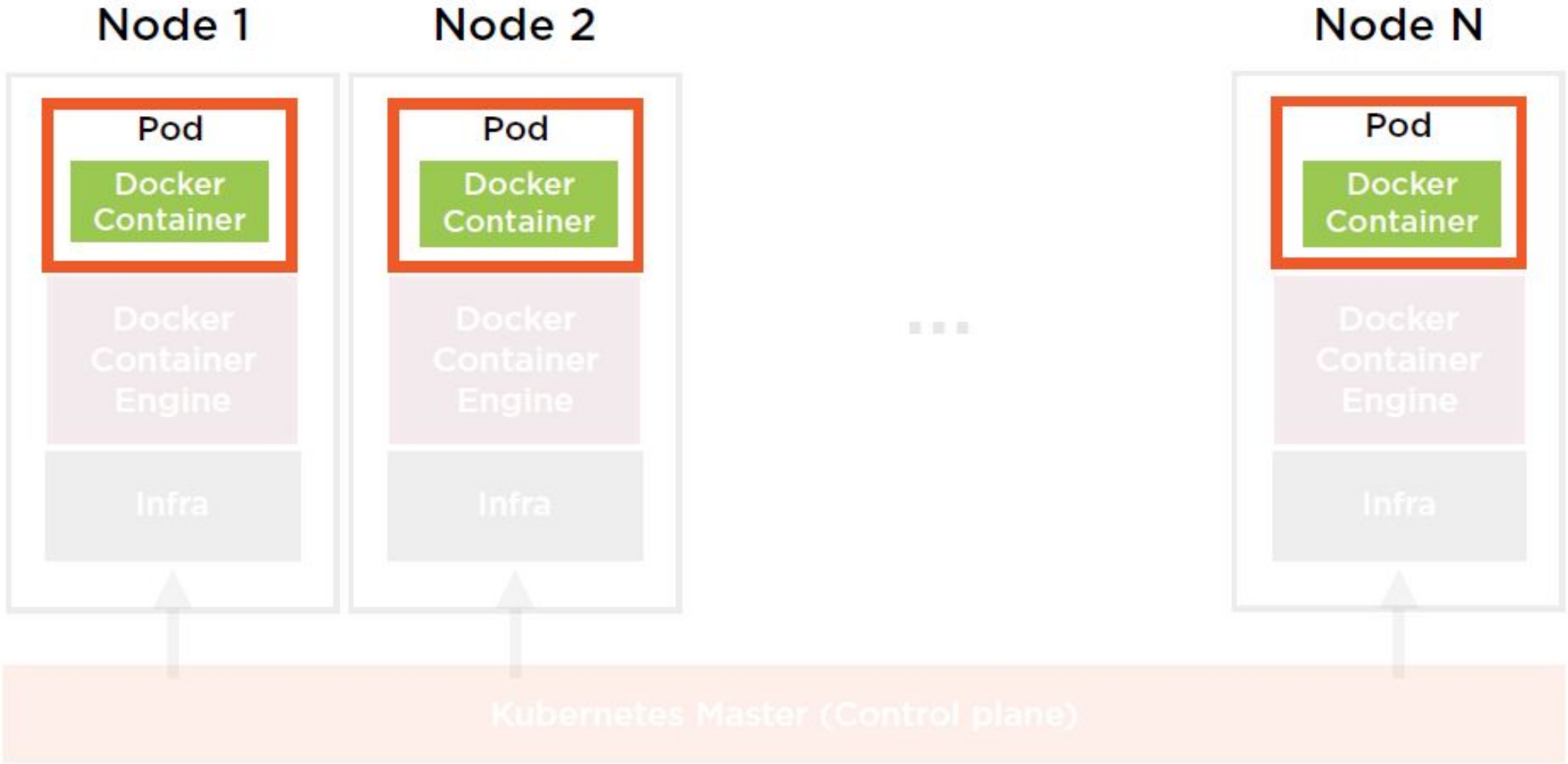
Kubernetes: Cluster Orchestration



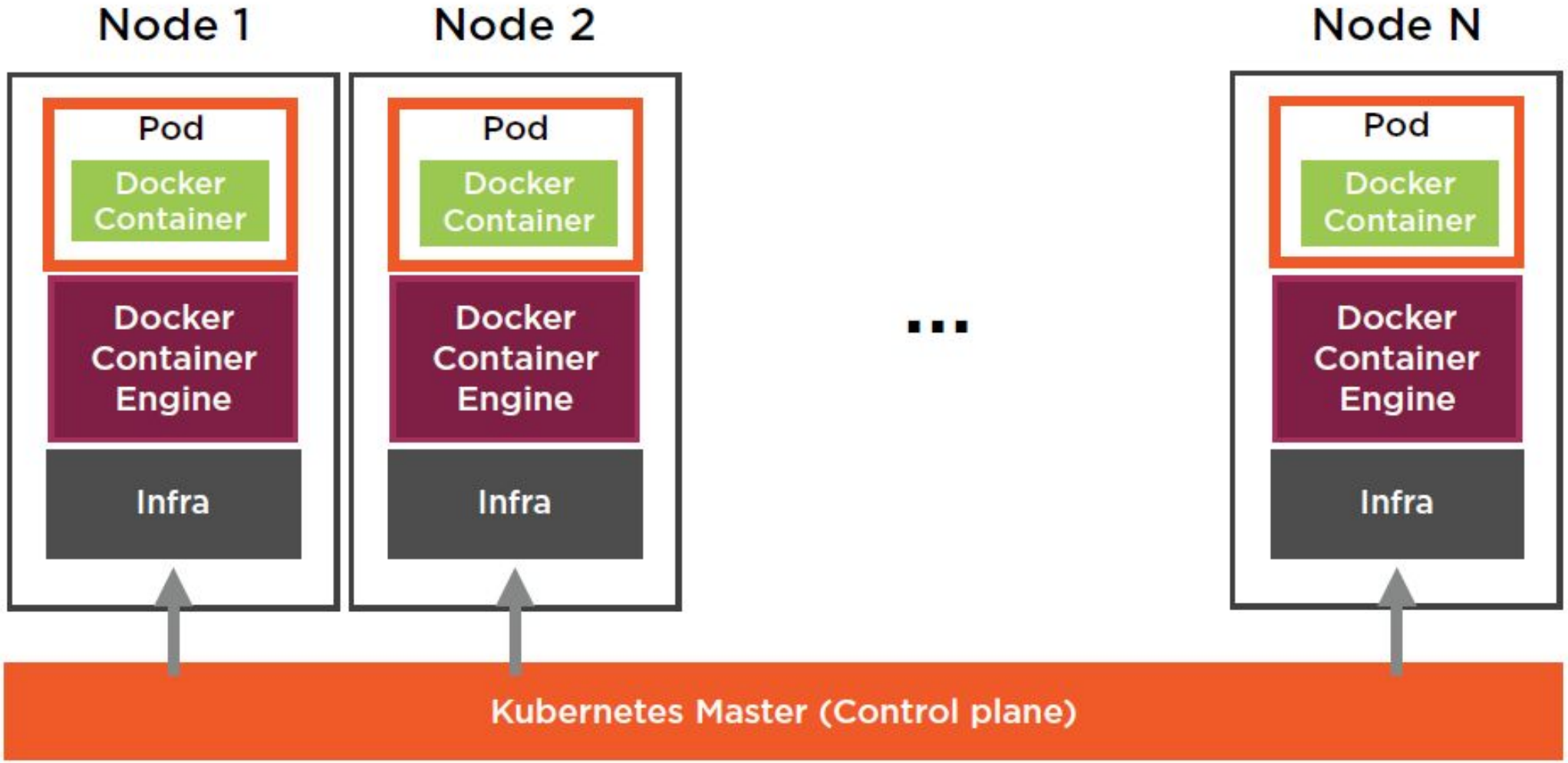
Kubernetes: Cluster Orchestration



Kubernetes: Containers Run Within Pods



Kubernetes: Cluster Orchestration



Pods as Atomic Units

Container deployment

All containers in pod are deployed, or none are

Node association

Entire pod is hosted on the same node

Pod is atomic unit of deployment in Kubernetes

The ReplicaSet Object

Multiple identical pods which are replicas of each other

ReplicaSet

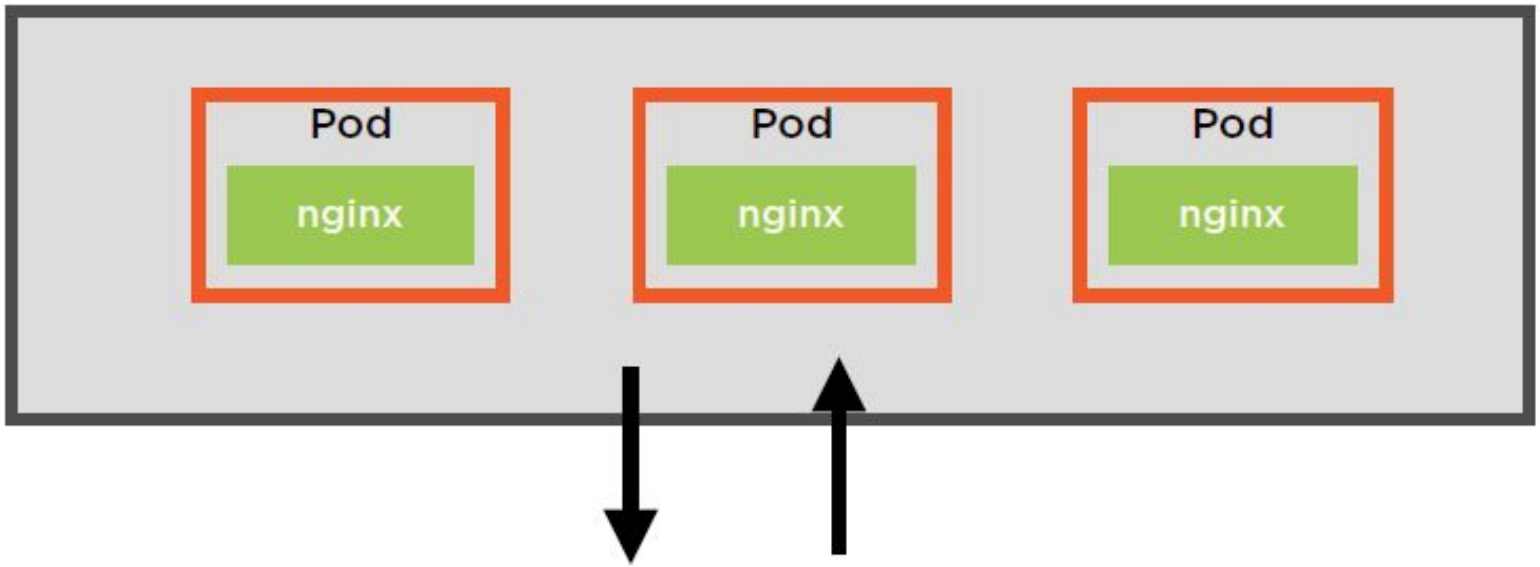
Self healing and autoscaling for our pods

The Deployment Object

Adds on deployment and rollback functionality

ORSYS formation

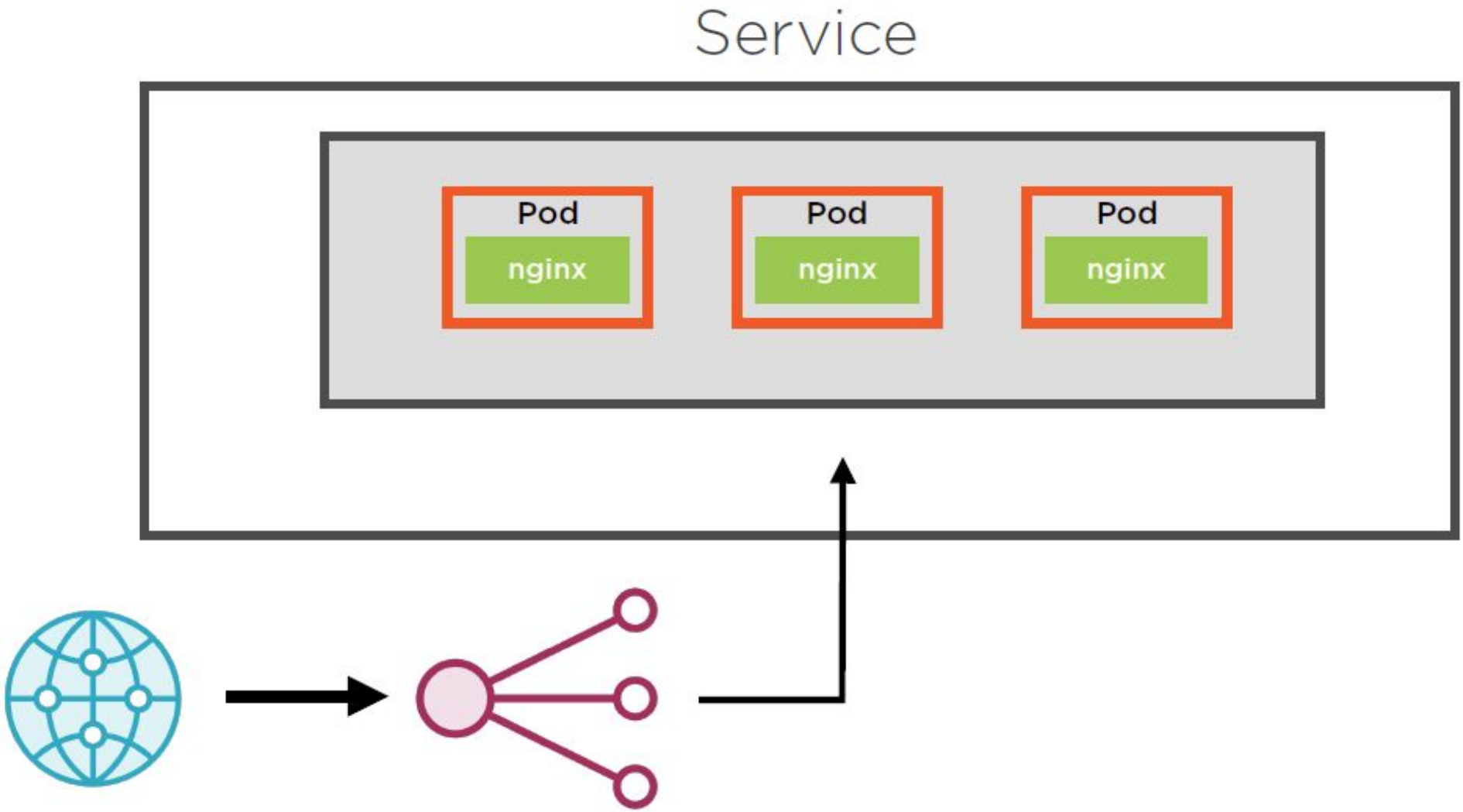
Deployment



Support for versions, and production-level operations such as rollbacks

ORSYS formation

Services provide stable IP addresses for external connections and load balancing


Service
Pod
nginx
Pod
nginx
Pod
nginx

ORSYS formation

Lab:
Les fondamentaux de K8S

ORSYS formation

CGroups and Namespaces

ORSYS formation

CGroups and Namespaces

Dans cet exercice, vous apprendrez les bases de CGroups (Control Groups) et Namespces pour appliquer des restrictions de sécurité aux conteneurs. Voici quelques exemples des types de groupes de contrôle et d'espaces de noms qui existent.

Exemples de Cgroups:

```
--cpu-shares
--cpuset-cpus
--memory-reservation
--kernel-memory
–-blkio-weight (block IO)
--device-read-iops
--device-write-iops
```

CGroups and Namespaces

Exemples de Namespaces:

Cgroup	CLONE_NEWCGROUP	Cgroup root directory
IPC	CLONE_NEWIPC	System V IPC, POSIX message queues
Network	CLONE_NEWNET	Network devices,stacks,ports, etc.
Mount	CLONE_NEWNS	Mount points
PID	CLONE_NEWPID	Process IDs
User	CLONE_NEWUSER	User and group IDs
UTS	CLONE_NEWUTS	Hostname and NIS domain name

1- Définir les limites de mémoire

Les CGroups contrôlent la quantité de ressources qu'un processus peut utiliser. En ajoutant des restrictions, vous pouvez offrir une qualité de service garantie aux applications en vous assurant qu'elles disposent de suffisamment d'espace disponible.

Il est également possible de protéger le système contre les utilisateurs ou applications potentiellement malveillants visant à exécuter des applications de déni de service (DoS) via l'épuisement des ressources.

Cela peut également aider à limiter les applications des fuites de mémoire ou d'autres bogues de programmation en définissant des limites supérieures.

ORSYS formation

1- Définir les limites de mémoire

- Exemple:

> docker run -d --name mb100 --memory 100m alpine top

```
[root @ host01 ~] # docker run -d --name mb100 --memory 100m sommet alpin
7849578514a5da111b1ba6d868d36a8daf2e82d6a0e1cfa6e10ff292820110b4
```

L'utilisation de la mémoire et les limites des conteneurs peuvent être identifiées via la commande *docker stats* .

> docker stats --no-stream

```
[root @ host01 ~] # stats docker --no-stream
CONTAINER ID NAME CPU% MEM USAGE / LIMIT MEM% NET I / O BLOCK I / O PI
DS
7849578514a5 mb100 0,00% 436KiB / 100MiB 0,43% 5,33kB / 90B 1,06MB / 0B 1
```

ORSYS formation

2- Définir les partages CPU

- Alors que les limites de mémoire définissent un maximum défini, les limites de processeur sont basées sur les partages.

- Ces parts représentent un poids entre le temps de traitement d'un processus par rapport à un autre.

- Si un processeur est inactif, le processus utilisera toutes les ressources disponibles. Si un deuxième processus nécessite la CPU, le temps CPU disponible sera partagé en fonction de la pondération.

2- Définir les partages CPU

- Prenons un exemple de démarrage d'un conteneur avec différents partages. Le paramètre **--cpu-shares** définit un partage entre 0 et 768.

- Si un conteneur définit une part de 768, tandis qu'un autre définit une part de 256, le premier conteneur aura une part de 75%, l'autre ayant 25% du total de part disponible.

- Ces chiffres sont dus à l'approche de pondération pour le partage du processeur au lieu d'une capacité fixe.

- Lançons le premier conteneur qui sera autorisé à avoir 75% de la part et Le deuxième conteneur qui sera limité à 25%.

2- Définir les partages CPU

> docker run -d --name c768 --cpuset-cpus 0 --cpu-shares 768 benhall/stress

```
[root@host01 ~]# docker run -d --name c256 --cpuset-cpus 0 --cpu-shares 256 benhall/stress
0f77bb0f7950fcc3e6982bd76202caae414e3919ba61dc90c1ab80dddce932a0
[root@host01 ~]# sleep 5
[root@host01 ~]# docker stats --no-stream
CONTAINER ID        NAME                CPU %               MEM USAGE / LIMIT   MEM %               NET I/O             BLOCK I/O           PI
DS
0f77bb0f7950        c256                25.30%              764KiB / 737.5MiB   0.10%               3.64kB / 90B        0B / 0B             3
d937be5b4b3b        c768                74.51%              712KiB / 737.5MiB   0.09%               4.77kB / 90B        0B / 0B             3
[root@host01 ~]# docker rm -f c768 c256
c768
c256
```

> docker run -d --name c256 --cpuset-cpus 0 --cpu-shares 256 benhall/stress

> sleep 5

> docker stats --no-stream

> docker rm -f c768 c256


```
[root@host01 ~]# docker run -d --name c256 --cpuset-cpus 0 --cpu-shares 256 benhall/stress
0f77bb0f7950fcc3e6982bd76202caae414e3919ba61dc90c1ab80dddce932a0
[root@host01 ~]# sleep 5
[root@host01 ~]# docker stats --no-stream
CONTAINER ID        NAME                CPU %               MEM USAGE / LIMIT   MEM %               NET I/O             BLOCK I/O           PI
DS
0f77bb0f7950        c256                25.30%              764KiB / 737.5MiB   0.10%               3.64kB / 90B        0B / 0B             3
d937be5b4b3b        c768                74.51%              712KiB / 737.5MiB   0.09%               4.77kB / 90B        0B / 0B             3
[root@host01 ~]# docker rm -f c768 c256
c768
c256
```


- Il est important de noter qu'un processus peut avoir 100% du partage, quel que soit le poids défini, si aucun autre processus n'est en cours d'exécution.

3- Utiliser les « Network Namespace »

- Alors que les groupes de contrôle contrôlent la quantité de ressources qu'un processus peut utiliser, les namespaces contrôlent ce qu'un processus et voit et accéder.

> docker run -it alpine ip addr show

```
[root@host01 ~]# docker run -it alpine ip addr show
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
8: eth0@if9: <BROADCAST,MULTICAST,UP,LOWER_UP,M-DOWN> mtu 1500 qdisc noqueue state UP
    link/ether 02:42:ac:12:00:02 brd ff:ff:ff:ff:ff:ff
    inet 172.18.0.2/24 brd 172.18.0.255 scope global eth0
       valid_lft forever preferred_lft forever
```

- En changeant l'espace de noms en host , au lieu que le réseau du conteneur soit isolé avec son interface, le processus aura accès à l'interface réseau des machines hôtes.

- Si le processus écoute sur les ports, ils seront écoutés sur l'interface hôte et mappés sur le conteneur.

ORSYS formation

3- Utiliser les « Network Namespace »

> docker run -it --net=host alpine ip addr show

```
[root@host01 ~]# docker run -it --net=host alpine ip addr show
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: ens3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP qlen 1000
    link/ether 02:42:ac:11:00:2b brd ff:ff:ff:ff:ff:ff
    inet 172.17.0.43/16 brd 172.17.255.255 scope global ens3
       valid_lft forever preferred_lft forever
    inet6 fe80::bb30:716c:4fa4:9465/64 scope link
       valid_lft forever preferred_lft forever
3: docker0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue state DOWN
    link/ether 02:42:1a:78:f1:bd brd ff:ff:ff:ff:ff:ff
    inet 172.18.0.1/24 brd 172.18.0.255 scope global docker0
       valid_lft forever preferred_lft forever
```

4- Utiliser le Namespace Pid

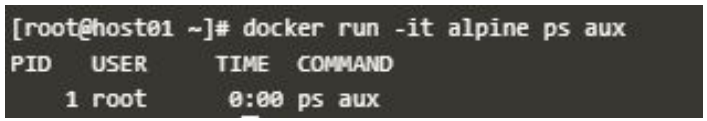
- Comme pour les réseaux, les processus qu'un conteneur peut voir dépendent également de l'espace de noms auquel il appartient.

- En modifiant l'espace de noms Pid, un conteneur peut interagir avec des processus au-delà de sa portée normale.

- **Exemple:**

Le premier conteneur s'exécutera dans son espace de noms de processus. En tant que tels, les seuls processus auxquels il peut accéder sont ceux lancés dans le conteneur.

> docker run -it alpine ps aux



4- Utiliser le Namespace Pid

En remplaçant le namespace par l'hôte, le conteneur peut également voir tous les autres processus en cours d'exécution sur le système.

> docker run -it --pid=host alpine ps aux

```
296 root      0:00 -bash
299 root      0:02 docker-containerd --config /var/run/docker/containerd/cont
026 root      0:00 [kworker/u2:0]
497 root      0:00 [kworker/0:0]
884 root      0:00 [kworker/0:1]
942 root      0:00 [kworker/0:2]
183 root      0:00 docker run -it --pid=host alpine ps aux
198 root      0:00 docker-containerd-shim -namespace moby -workdir /var/lib/d
227 root      0:00 ps aux
270 root      0:00 [kworker/0:3]
289 root      0:00 {dhcpcd-run-hook} /bin/sh /usr/lib/dhcpcd/dhcpcd-run-hooks
299 root      0:00 docker-runc --root /var/run/docker/runtime-runc/moby --log
302 root      0:00 [dhcpcd-run-hook]
```

5- Partage de Namespaces

- Fournir aux conteneurs l'accès à l'espace de noms de l'hôte est parfois nécessaire, comme pour les outils de débogage, mais est considéré comme une mauvaise pratique. En effet, vous sortez du modèle de sécurité du conteneur, ce qui peut introduire des vulnérabilités

- Au lieu de cela, si nécessaire, utilisez un espace de noms partagé pour fournir un accès uniquement aux espaces de noms dont le conteneur a besoin.

5- Partage de Namespaces

Exemple:

- Le premier conteneur démarre un serveur Nginx. Cela définira un nouvel espace de noms de réseau et de processus.

- Le serveur Nginx se liera au port 80 de l'interface réseau nouvellement définie.

> docker run -d --name http nginx:alpine

```
[root@host01 ~]# docker run -d --name http nginx:alpine
Unable to find image 'nginx:alpine' locally
alpine: Pulling from library/nginx
801bfaa63ef2: Pull complete
b1242e25d284: Pull complete
7453d3e6b909: Pull complete
07ce7418c4f8: Pull complete
e295e0624aa3: Pull complete
Digest: sha256:c2ce58e024275728b00a554ac25628af25c54782865b3487b11c21cafb7fabda
Status: Downloaded newer image for nginx:alpine
b588fb6334b3293231c850114b2af4704df914b2fcfa6b0e43d74cf95fc6a2d5
```

5- Partage de Namespaces

- Le premier conteneur démarre un serveur Nginx. Cela définira un nouvel espace de noms de réseau et de processus. Le serveur Nginx se liera au port 80 de l'interface réseau nouvellement définie.

> docker run -d --name http nginx:alpine

```
[root @ host01 ~] # docker run -d --name http nginx: alpin
865f7f6c19422e34ed55dfd940d2e44bb877def13761927c509fd21327c4d495
```

- D'autres conteneurs peuvent désormais réutiliser cet espace de noms à l'aide de la syntaxe **container: <nom>**.

- Ci-dessous, la commande curl peut accéder au serveur HTTP s'exécutant sur localhost car ils partagent la même interface réseau.

ORSYS formation

5- Partage de Namespaces

> docker run --net=container:http benhall/curl curl -s localhostdocker run --net=container:http benhall/curl curl -s localhost

```
[root@host01 ~]# docker run --net=container:http benhall/curl curl -s localhost
Unable to find image 'benhall/curl:latest' locally
latest: Pulling from benhall/curl
12b41071e6ce: Pull complete
fb1cef6edba2: Pull complete
a3ed95caeb02: Pull complete
Digest: sha256:637b3e063550593071b172916c896ea3590d4ecd67ae2399e54672b5770e114e
Status: Downloaded newer image for benhall/curl:latest
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
    body {
        width: 35em;
        margin: 0 auto;
        font-family: Tahoma, Verdana, Arial, sans-serif;
    }
</style>
</head>
<body>
<h1>Welcome to nginx!</h1>
<p>If you see this page, the nginx web server is successfully installed and
working. Further configuration is required.</p>

<p>For online documentation and support please refer to
<a href="http://nginx.org/">nginx.org</a>.<br/>
Commercial support is available at
<a href="http://nginx.com/">nginx.com</a>.</p>

<p><em>Thank you for using nginx.</em></p>
</body>
</html>
```

5- Partage de Namespaces

> docker run --pid=container:http alpine ps aux

- Il peut également voir et s'interfacer avec les processus dans le conteneur partagé.

```
[root@host01 ~]# docker run --pid=container:http alpine ps aux
PID   USER     TIME   COMMAND
    1 root       0:00 nginx: master process nginx -g daemon off;
   29 101        0:00 nginx: worker process
   30 root       0:00 ps aux
```

- Ceci est utile pour les outils de débogage, tels que **strace**.

- Cela vous permet d'accorder plus d'autorisations à des conteneurs spécifiques sans modifier ni redémarrer l'application.

ORSYS formation

Conclusion

Objectifs de la formation

- Comprendre l'état de l'art des solutions d'orchestration de containers et de leur écosystème ainsi que la mise en œuvre une plateforme de type CaaS (Container as a Service).
- Comprendre le fonctionnement, la mise en place et l'utilisation de containeurs dans une organisation.

- Ce qu'on a couvert:

1) Les fondamentaux

2) Docker

3) Kubernetes, orchestrateur de containers

4) Container as a Service (CaaS)

5) Sécurité du CaaS/Kubernetes/Docker

A bientôt ☺