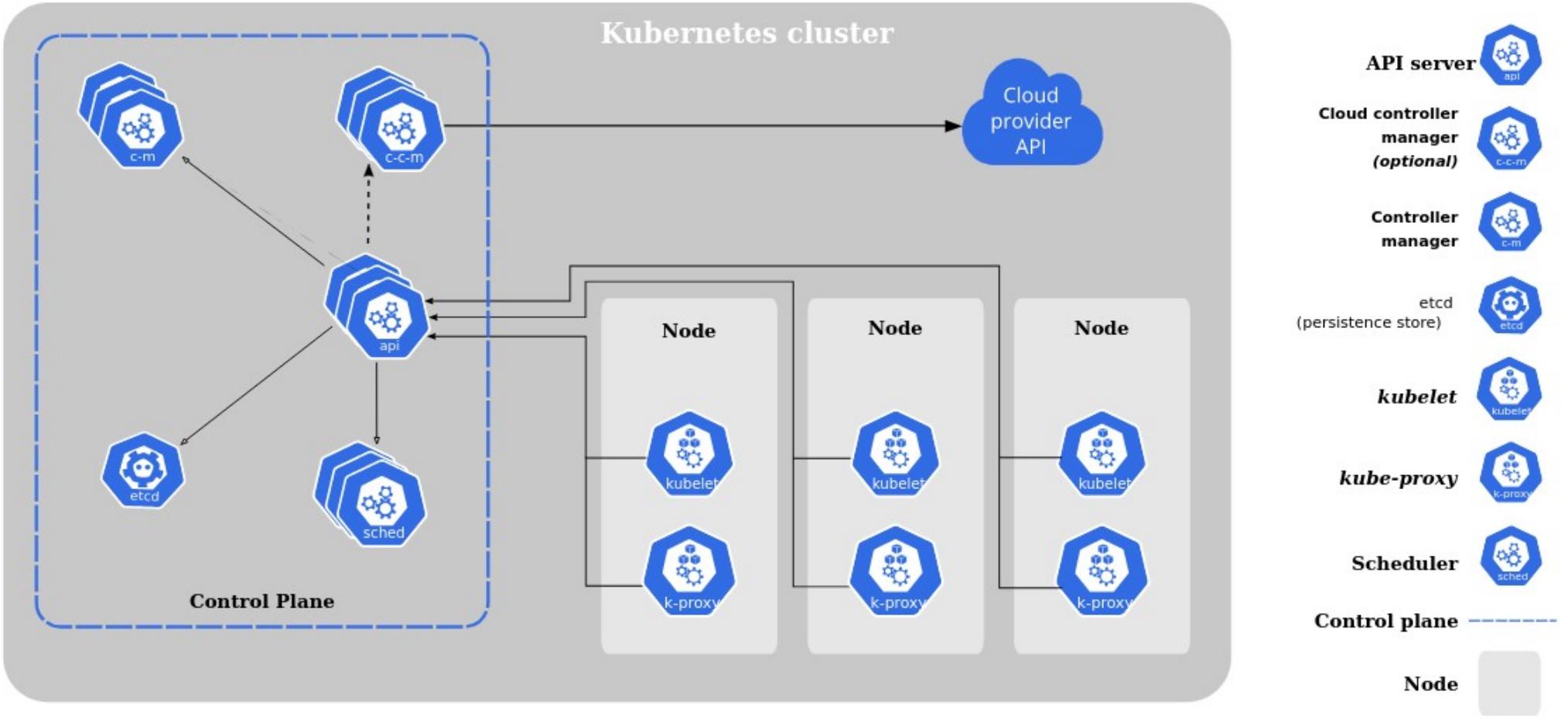
KUBERNETES

CONTROL PLANE



CONTROL PLANE COMPONENTS

The control plane's components make global decisions about the cluster (for example, scheduling), as well as detecting and responding to cluster events.

- **kube-apiserver** is running as a Docker container on your master node, it is the front end for the Kubernetes control plane.
- **Etcd:** Consistent and highly-available key value store used as Kubernetes' backing store for all cluster data.
- **Kube-scheduler:** Control plane component that watches for newly created Pods with no assigned node, and selects a node for them to run on.
  Factors taken into account for scheduling decisions include: individual and collective resource requirements, hardware/software/policy constraints, affinity and anti-affinity specifications, data locality, inter-workload interference, and deadlines.
- **Cloud Controller Manager**: it embeds cloud-specific control logic. The cloud controller manager lets you link your cluster into your cloud provider's API, and separates out the components that interact with that cloud platform from components that only interact with your cluster. Is the key for interoperability between cloud providers

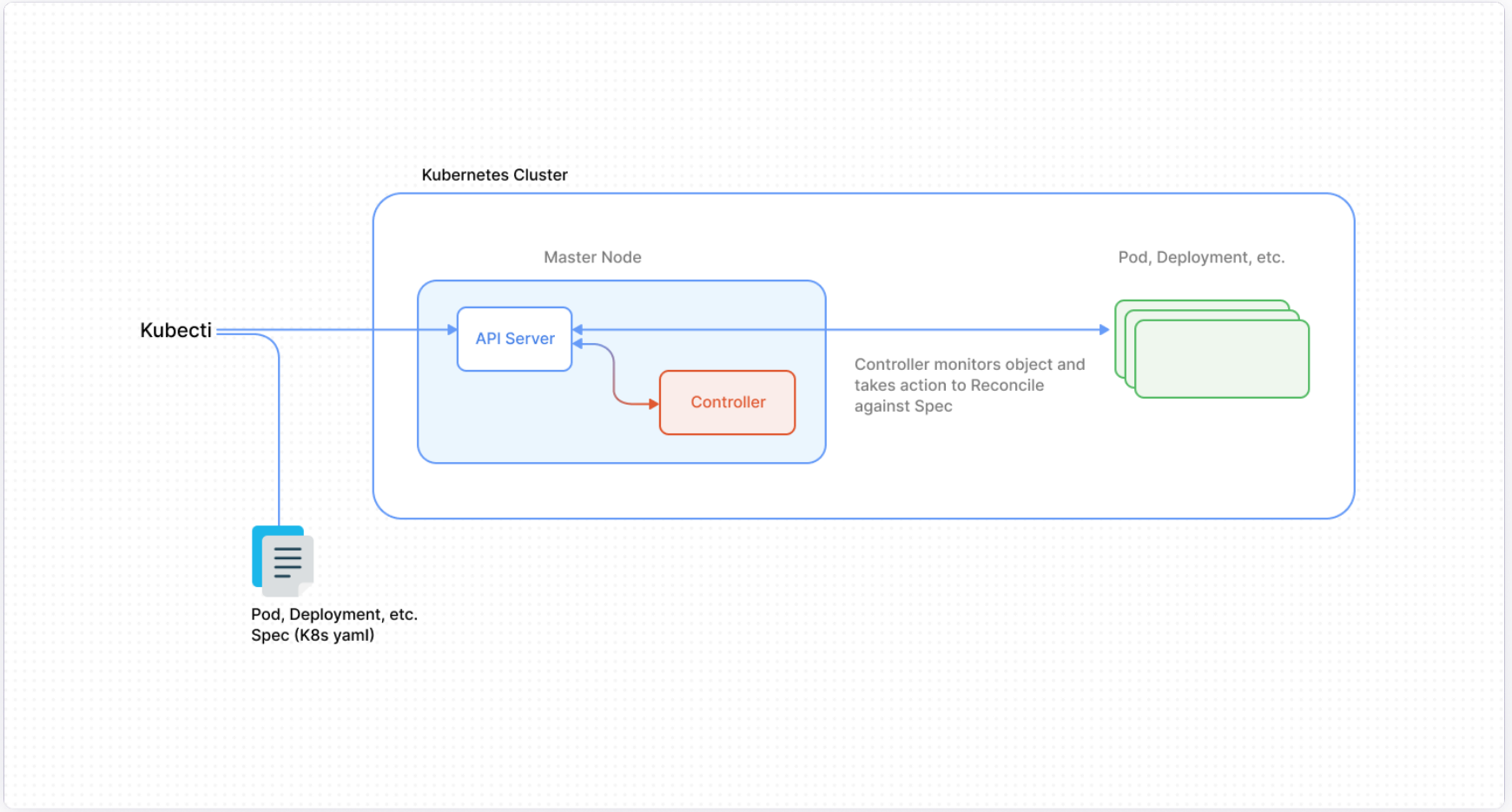
• CONTROL PLANE COMPONENTS -

Kube controller manager

- **Kube controller manager**: Control plane component that runs controller processes. Logically, each controller is a separate process, but to reduce complexity, they are all compiled into a single binary and run in a single process. This component is responsible for running different types of controllers that maintain the overall desired state of the cluster. All the controllers are packaged and shipped in the kube-controller-manager, which is a single daemon.
  Resources (or objects) in Kubernetes are defined by manifest files that specify the desired state. When resources are deployed, the appropriate controllers ensure that the cluster's existing state is updated to match the desired state. Each controller is a control-loop that watches the shared state of the cluster via the API server. Based on information about the deployed resources, the controllers will make changes to move the current cluster state toward the desired state.

• CONTROL PLANE COMPONENTS -

Kube controller manager



• CONTROL PLANE COMPONENTS -

Kube controller manager

There are different types of controllers that enable you to configure behavior on your Kubernetes cluster. Below is a list of the main controllers:

- Replicaset
- Deployment
- Daemonset
- Statefulset
- Job
- Cronjob

NODE COMPONENTS

Node components run on every node, maintaining running pods and providing the Kubernetes runtime environment. Kubernetes runs your workload by placing containers into Pods to run on Nodes. A node may be a virtual or physical machine, depending on the cluster. Each node is managed by the control plane and contains the services necessary to run Pods. Typically you have several nodes in a cluster; in a learning or resource-limited environment, you might have only one node.

- The components on a node include the **kubelet**, a **container runtime**, and the **kube-proxy**.
- **Kubelet**: is the primary "node agent" that runs on each node. It can register the node with the apiserver.
- **container runtime:** containerd, CRI-O, Docker Engine
- **kube-proxy** runs on each node of a Kubernetes cluster. Is a key part of routing mechanism that rely on iptables

KUBEADMIN

Kubeadm est un outil conçu afin que les commandes kubeadm init et kubeadm join soient la meilleure façon de créer rapidement des clusters Kubernetes.

Kubeadm effectue les actions nécessaires afin d'obtenir un cluster minimal mais fonctionnel et en état de marche. Son objectif est d'assembler le cluster sans pour autant provisioner les machines qui le composent. De la même façon, kubeadm ne supporte pas l'installation des extensions habituelles comme le Dashboard (tableau de bord), les solutions de surveillance ou bien encore les extensions spécifiques aux fournisseurs cloud.

https://medium.com/@mrdevsecops/set-up-a-kubernetes-cluster-with-kubeadm-508db74028ce

TERMINOLOGY

MANAGED vs UNMANAGED: With the unmanaged Kubernetes installation, users need to update control plane components and the nodes. For instance with the managed EKS service, the control plane will be the responsibility of AWS.

Infrastructure as a Service: The capability of provisioning storage, networks, processing as well as other resources where the user can deploy and run software.

Platform as a Service: A cloud-computing model that allows user to develop, run, manage, and deploy applications within a cloud infrastructure, using supported technologies, without building and maintaining said infrastructure.

Software as a Service: Software as a service (SaaS) is a software distribution model in which a third-party provider hosts applications and makes them available to customers over the Internet.

On Premises: software ("on-prem") is installed and runs on computers on the premises of the organisation using the software, rather than at a remote facility.

MANAGE A K8s CLUSTER

https://kubernetes.io/fr/docs/tasks/tools/install-kubectl/
https://docs.digitalocean.com/reference/doctl/how-to/install/

doctl kubernetes cluster kubeconfig save 84cecf75-e055-45b3-a60c-b33d135dd39e

Cat ~/.kube/config

Kubectl get ns

K8S OBJECTS

DEPLOYMENT : it provides declarative updates for Pods

STATEFULSET: Like a Deployment, a StatefulSet manages Pods that are based on an identical container spec. Unlike a Deployment, a StatefulSet maintains a sticky identity for each of their Pods. These pods are created from the same spec, but are not interchangeable: each has a persistent identifier that it maintains across any rescheduling.

DAEMONSET: it ensures that all (or some) Nodes run a copy of a Pod. As nodes are added to the cluster, Pods are added to them. As nodes are removed from the cluster, those Pods are garbage collected.

JOBS: it creates one or more Pods and will continue to retry execution of the Pods until a specified number of them successfully terminate. As pods successfully complete, the Job tracks the successful completions.

K8S OBJECTS

POD: Pods are the smallest deployable units of computing that you can create and manage in Kubernetes. A Pod (as in a pod of whales or pea pod) is a group of one or more containers, with shared storage and network resources, and a specification for how to run the containers.

CONFIGMAP: A ConfigMap is an API object used to store non-confidential data in key-value pairs. Pods can consume ConfigMaps as environment variables, command-line arguments, or as configuration files in a volume. There are four different ways that you can use a ConfigMap to configure a container inside a Pod:

- Inside a container command and args
- Environment variables for a container
- Add a file in read-only volume, for the application to read
- Write code to run inside the Pod that uses the Kubernetes API to read a ConfigMap

SECRET : A Secret is an object that contains a small amount of sensitive data such as a password, a token, or a key. Such information might otherwise be put in a Pod specification or in a container image. Using a Secret means that you don't need to include confidential data in your application code.
Secrets are similar to ConfigMaps but are specifically intended to hold confidential data.

SEALED SECRET: to store secrets in public repository can be useful to encrypt it via kubernetes cluster certificates.

K8S OBJECTS - DEPLOYMENTS

IMAGE PULL POLICY: If imagePullPolicy is set to **Always**, Kubernetes will always pull the image from the Repository. With **IfNotPresent**, Kubernetes will only pull the image when it does not already exist on the node.

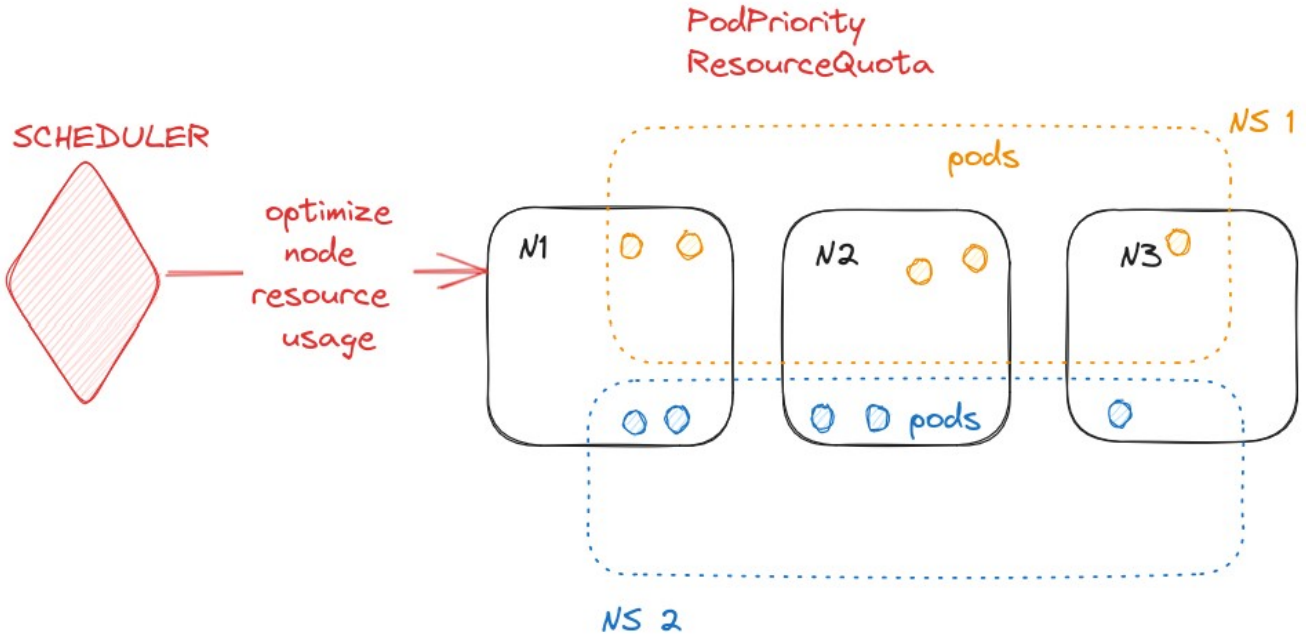
ENVIRONMENTS: **env** allows you to set environment variables for a container, specifying a value directly for each variable that you name.
envFrom allows you to set environment variables for a container by referencing either a ConfigMap or a Secret.

PROBES: The kubelet uses **liveness** probes to know when to restart a container. For example, liveness probes could catch a deadlock, where an application is running, but unable to make progress. Restarting a container in such a state can help to make the application more available despite bugs.
The kubelet uses **readiness** probes to know when a container is ready to start accepting traffic. A Pod is considered ready when all of its containers are ready. One use of this signal is to control which Pods are used as backends for Services. When a Pod is not ready, it is removed from Service load balancers.
The kubelet uses **startup** probes to know when a container application has started. If such a probe is configured, liveness and readiness probes do not start until it succeeds, making sure those probes don't interfere with the application startup

K8S OBJECTS

NAMESPACE: In Kubernetes, namespaces provides a mechanism for isolating groups of resources within a single cluster. Names of resources need to be unique within a namespace, but not across namespaces. Namespace-based scoping is applicable only for namespaced objects (e.g. Deployments, Services, etc) and not for cluster-wide objects (e.g. StorageClass, Nodes, PersistentVolumes, etc).

ResourceQuota: A quota restricts the number of objects, of a particular type, that can be created in a namespace.



K8s LOCAL DEV

- Install MINIKUBE https://minikube.sigs.k8s.io/docs/start/
- Install HELM https://helm.sh/docs/intro/install/
- Start the helloworld
  https://github.com/ynov-campus-sophia/conteneurisation/tree/master/k8s
- Follow the tutorial: https://helm.sh/fr/docs/intro/using_helm/