Ministero dell'Università e della Ricerca | Italiadomani | PNC Piano nazionale per gli investimenti complementari al PNRR | DARE DIGITAL LIFELONG PREVENTION

Kubernetes Installation and Administration Course

Architecture, installation and management

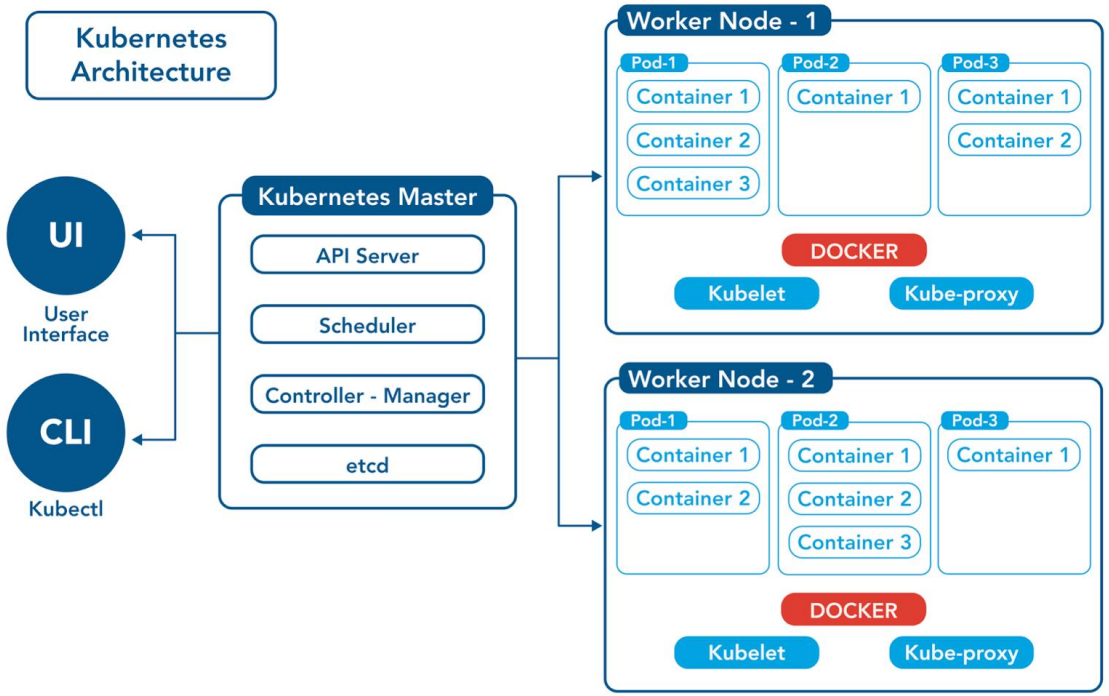
Francesco Sinisi (francesco.sinisi@cnaf.infn.it)

INFN

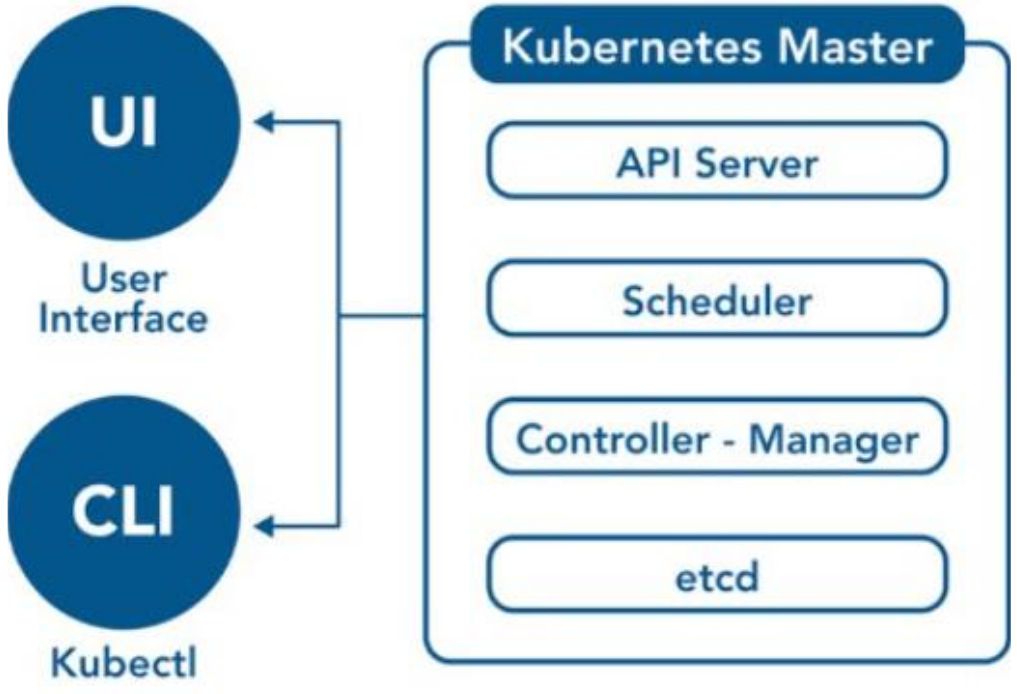
K8s cluster architecture

Node types and main components

Kubernetes Architecture



Node types and components

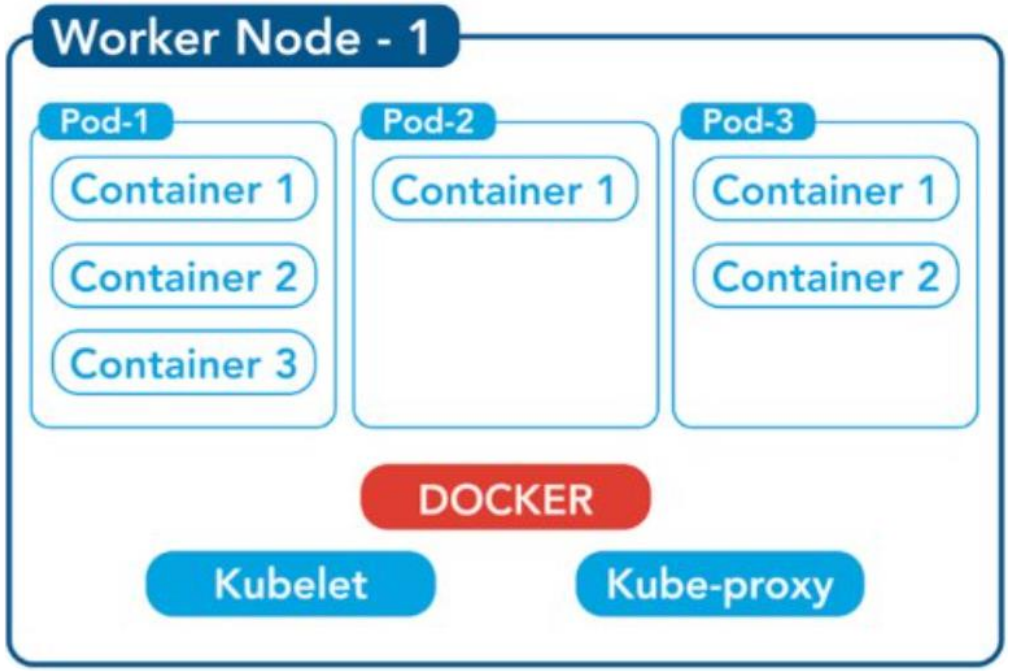


There are 2 types of nodes: **control Plane** or **master**, which contain all the main components, and **worker**, the node that makes its resources available to applications.

In the control Plane we find:

- **ControllerManager**: an infinite loop that checks the cluster state and makes sure it matches the administrator's desired state. It does this by continuously contacting the APIs and taking corrective action if it doesn't match.
- **API Server**: the cluster entry point, an interface for interacting with the cluster both by itself (ControllerManager) and by users (administrator or developer).
- **ETCD**: reliable key-value database where the cluster state is saved. Every operation is written to the ETCD or consulted when we request information.
- **Scheduler**: takes care of distributing Pods on nodes (often workers). It finds the best node that respects the constraints set by the administrator, users, space required by the application, space available on the node, etc.

Node types and main components



There are 2 types of nodes: **control Plane** or **master**, which contain all the main components, and **worker**, the node that makes its resources available to applications.

In the worker we find:

- **Kubelet**: daemon running inside the nodes and takes care of communication to API Server, implementation of commands, checks the health status of the Pods.
- **Kube-proxy**: It takes care of networking such as exposing ports, routing traffic and any networking rules.
- **Container Runtime**: allows containers to run.

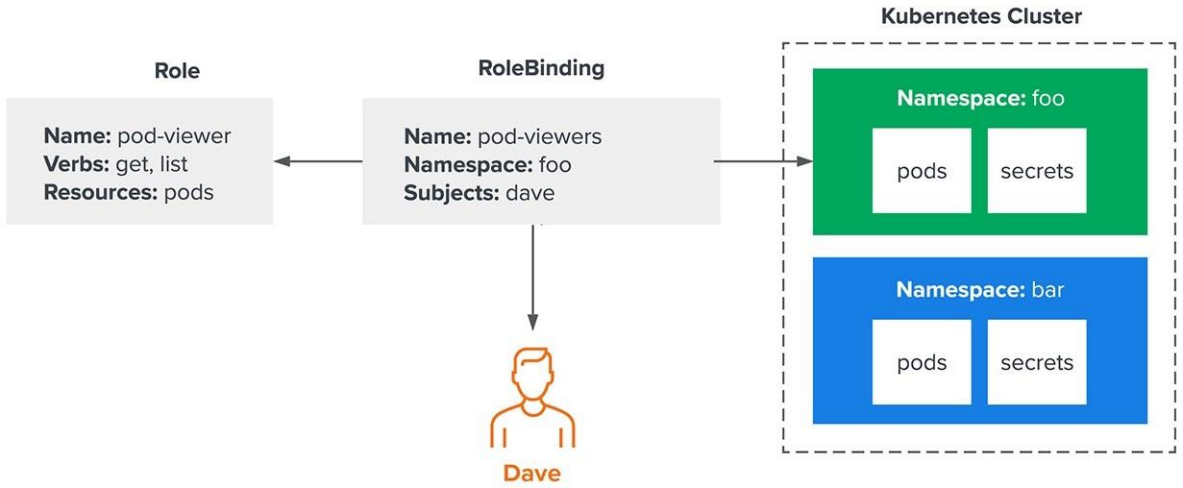
API Server

How to interact with the cluster via APIs and RBAC

Contact the cluster

The elements to contact a cluster are 2:

- The executable **kubectl** (downloadable from this guide);
- A file (**kubeconfig**) containing the coordinates of the cluster and the identity card of the user who wishes to contact it.



RBAC

After contacting the cluster, the request must pass the RBAC (*who can do what*) rules, formed by the pair:

- **Role**: list of actions (GET, CREATE, DELETE, etc.) that can be performed on some K8s components (Pod, Service, Deployment, etc.);
- **RoleBinding**: binds the user to the role.

Installation tools

Short list of tools for installing a K8s cluster

Manual installation

Preliminary steps:

- Compatible Linux host;
- At least 2 GB of RAM and 2 CPUs per machine;
- Full network connectivity between all machines in the cluster;
- univocity for every node (hostname, MAC address, and product_uuid);
- Some ports need to be opened on the machines;
- Swap disabled on nodes.

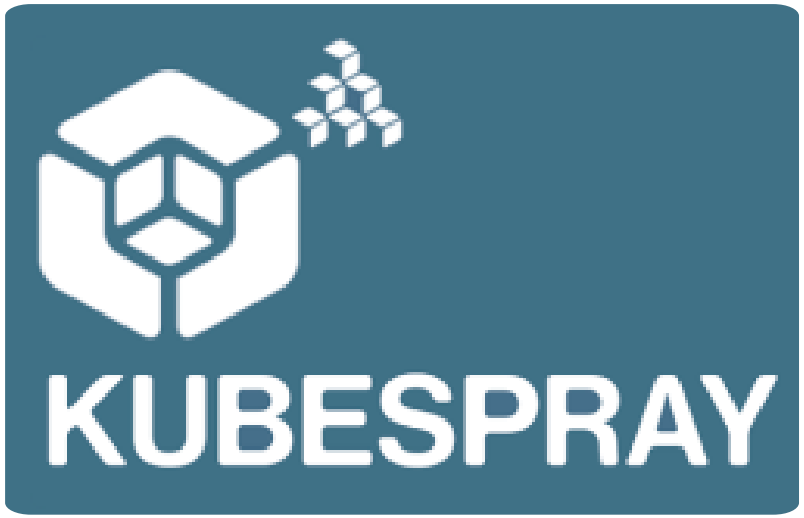
Manual installation

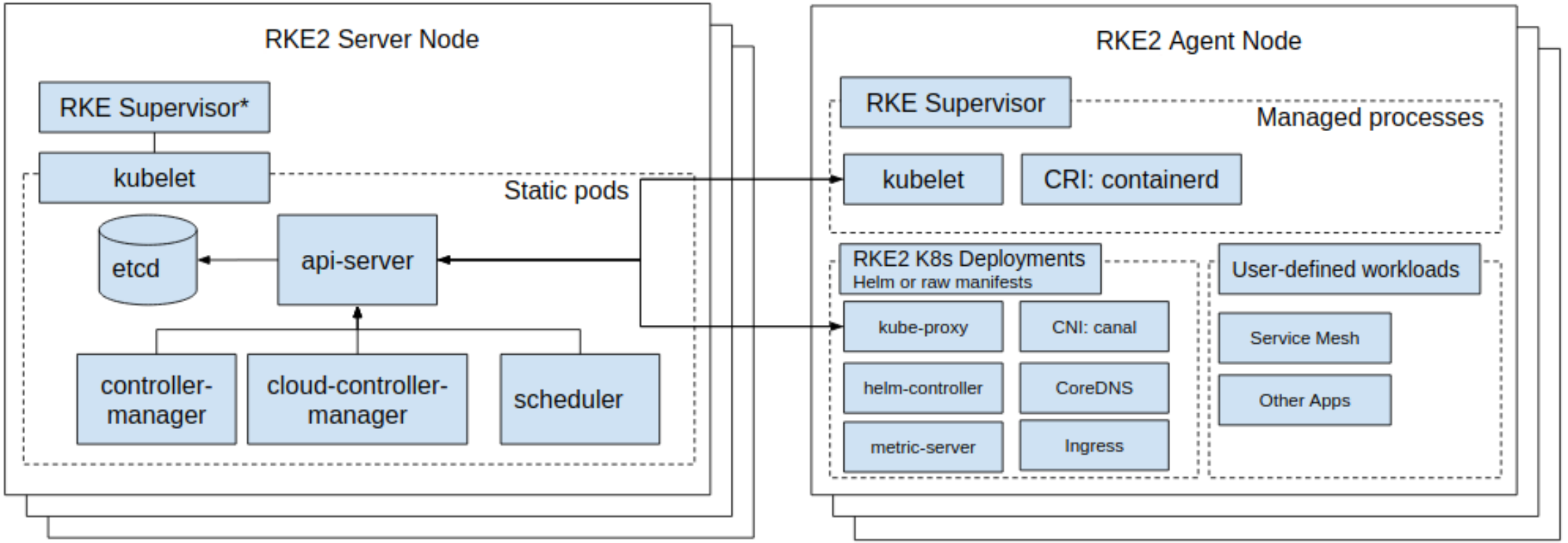
Installation:

- Installing CRI (containerd, CRIO-O);
- Installing kubeadm, kubelet and kubectl;
- Initializing your control-plane node (`kubeadm init`);
- Installing a Pod network add-on (calico, flannel, canal, cilium);
- Adding more control plane nodes (`kubeadm join`);
- Adding worker nodes (`kubeadm join`).

Kubespray

- Kubespray is a composition of Ansible playbooks, inventory, provisioning tools, and domain knowledge for generic OS/Kubernetes clusters configuration management tasks.
- The tool allows you to manage the cluster from an external machine, which will act as an Ansible server.





RKE2

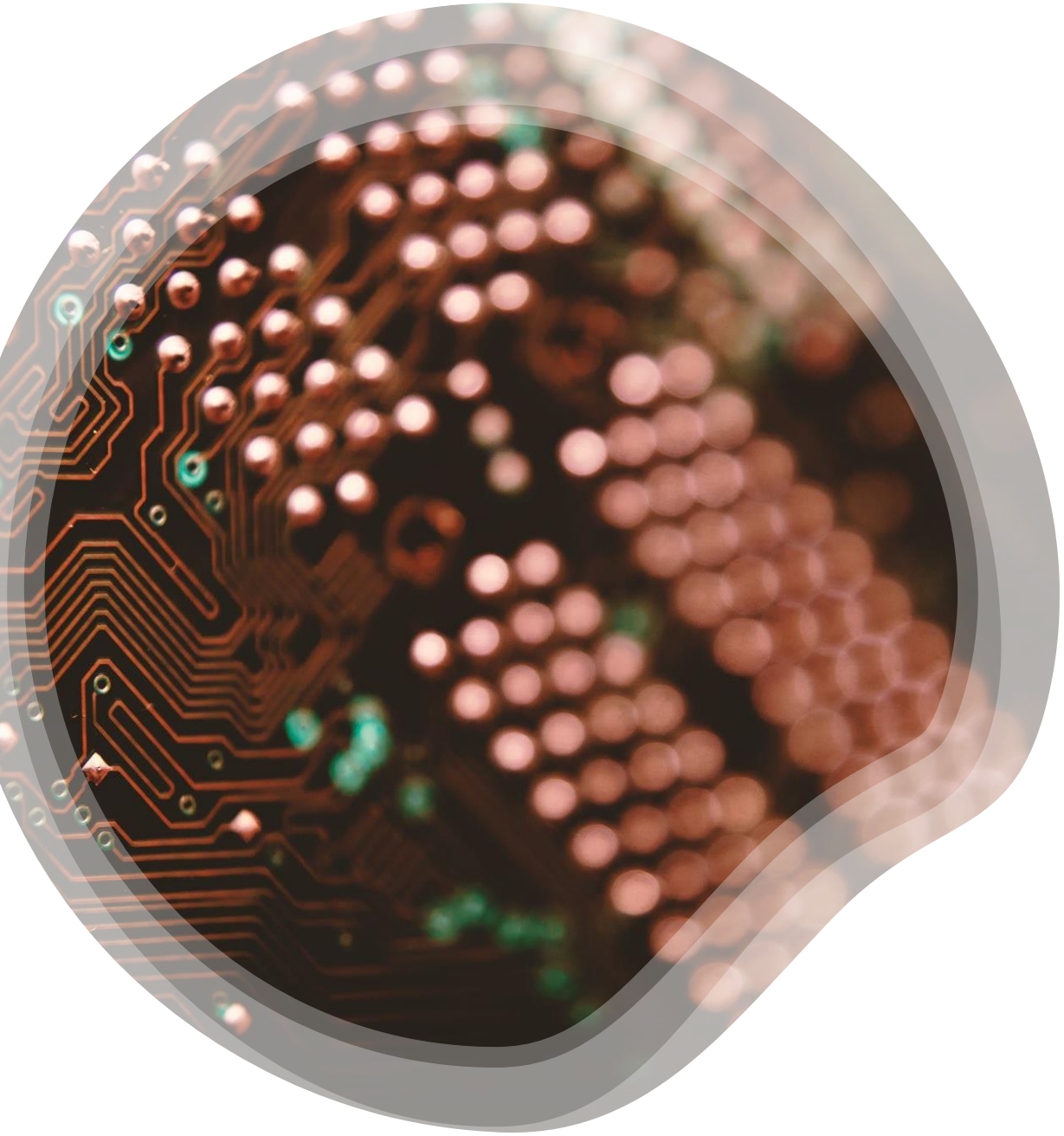
- RKE2 is a tool to **automate** the creation of a K8s cluster. It can boast a large community behind, quick software updates, periodic scanning for CVEs, good documentations.
- It **installs software** like CRI, CNI, helm, ingress Nginx, metric-server, kube-proxy, etc.
- Launches a **daemon** on servers and agents that supervises the main components of the cluster (kubelet, etcd, controller, scheduler, helm), created as static Pods.

RKE2 Installation

Install and configure a K8s cluster with RKE2

RKE2 Prerequisites

- Unique hostname value
- OS versions that have been validated with RKE2 (RKE2 Support Matrix)
- Hardware
  - RAM: 4GB Minimum (recommend at least 8GB)
  - CPU: 2 Minimum (recommend at least 4CPU)
- Networking (communication between nodes and open ports)



RKE2 First Server Node

- RKE2 provides an installation script, that is a convenient way to install it as a service on systemd based systems

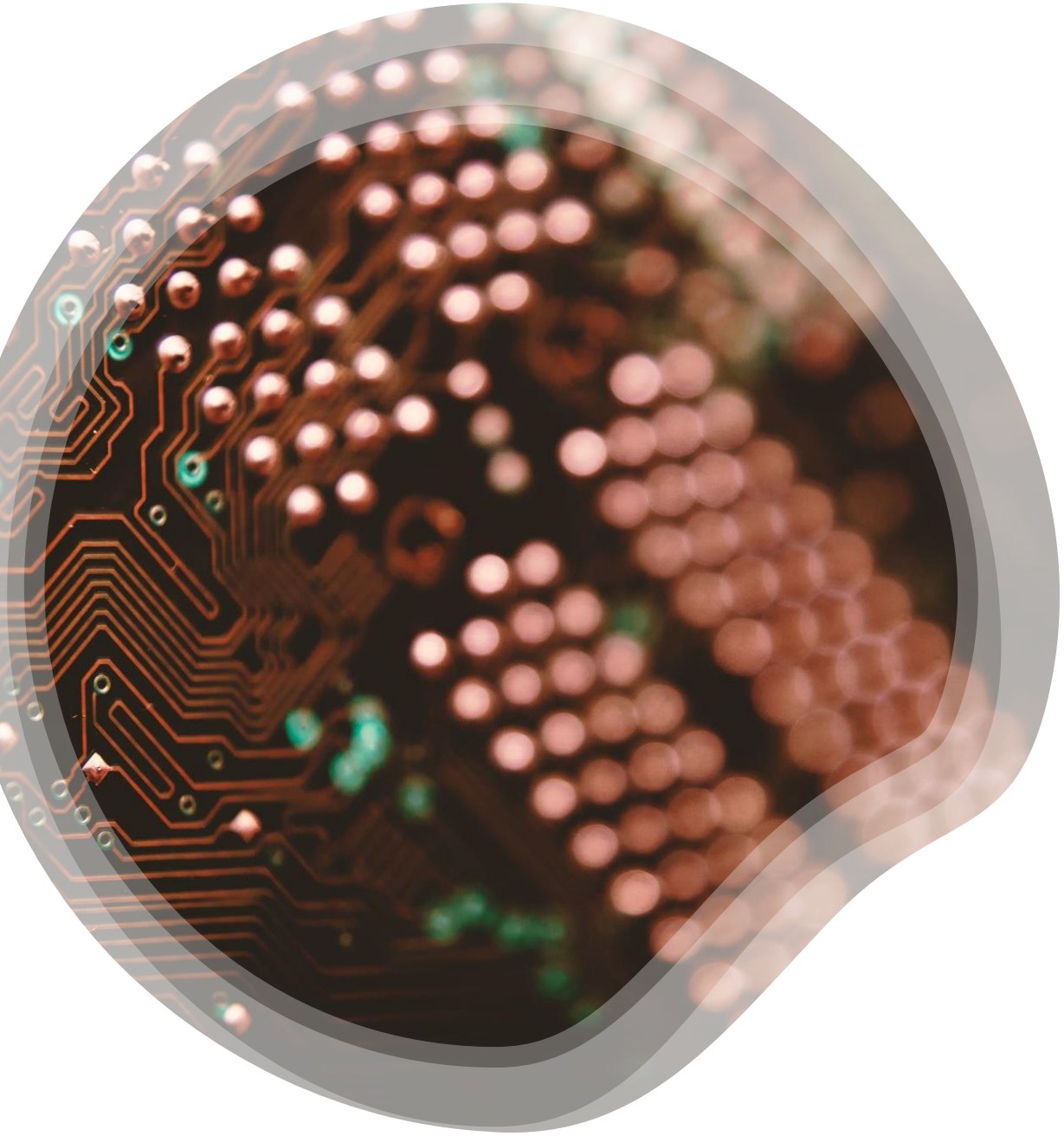
```
$ curl -sfL https://get.rke2.io | sh -
```

- If you want add some customizations in the file `/etc/rancher/rke2/config.yaml`
- Enable and start the service

```
$ systemctl enable rke2-server.service
$ systemctl start rke2-server.service
```

- Follow the logs, if you like

```
$ journalctl -u rke2-server -f
```

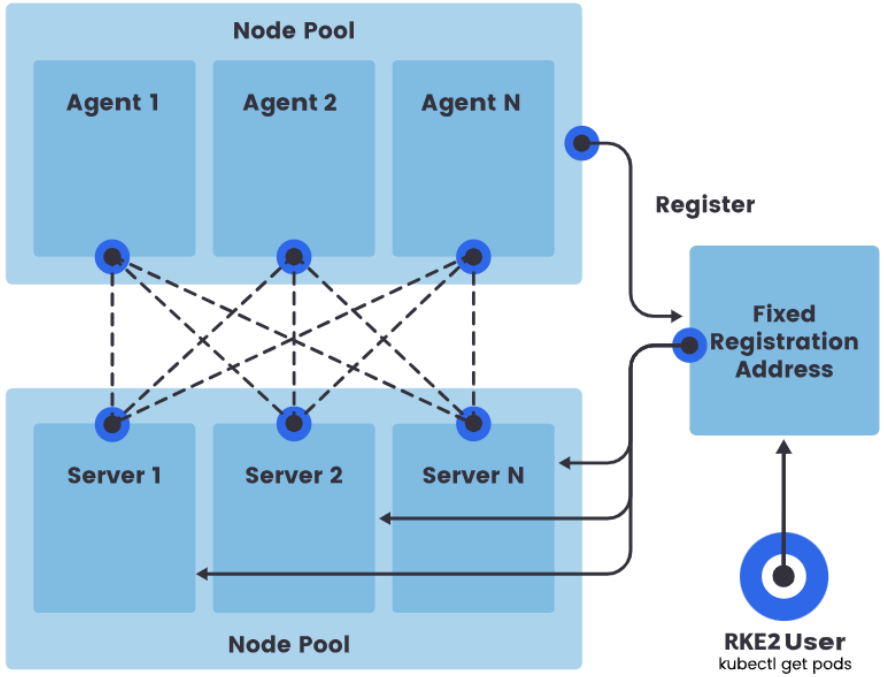


RKE2 First Server Node

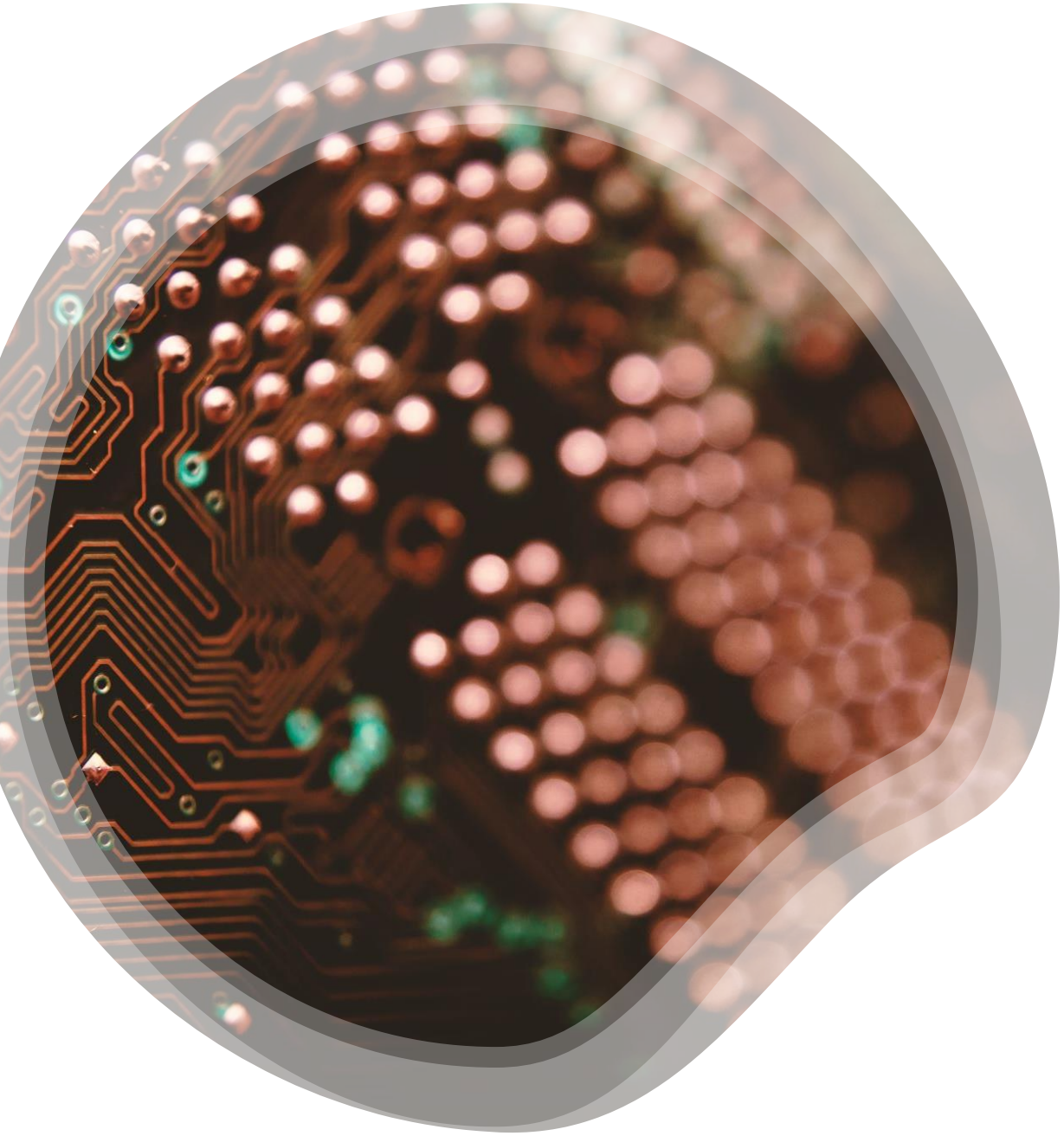
After running this installation:

- The rke2-server service will be installed. The rke2-server service will be configured to automatically restart after node reboots or if the process crashes or is killed.
- Additional utilities will be installed at `/var/lib/rancher/rke2/bin/`. They include: `kubectl`, `crictl`, and `ctr`. Note that these are not on your `$PATH` by default.
- Two cleanup scripts, `rke2-killall.sh` and `rke2-uninstall.sh`, will be installed to the `$PATH`.
- A kubeconfig file will be written to `/etc/rancher/rke2/rke2.yaml`.
- A token that can be used to register other server or agent nodes will be created at `/var/lib/rancher/rke2/server/node-token`.

Launch additional server nodes (optional)



- Additional server nodes are launched much like the first, except that you must specify the **server** and **token** parameters so that they can successfully connect to the initial server node.
- These 2 parameters must be inserted in the `config.yaml` file, before starting the service.



RKE2 Agent Node

- As before, launch the script, passing it the agent parameter

```
$ curl -sfL https://get.rke2.io |
INSTALL_RKE2_TYPE="agent" sh –
```

- Enable the service

```
$ systemctl enable rke2-agent.service
```

- Configure the node by inserting the token generated by the first server and its endpoint into the config.yaml file

```
$ mkdir -p /etc/rancher/rke2/
$ vim /etc/rancher/rke2/config.yaml
```

- Start the service

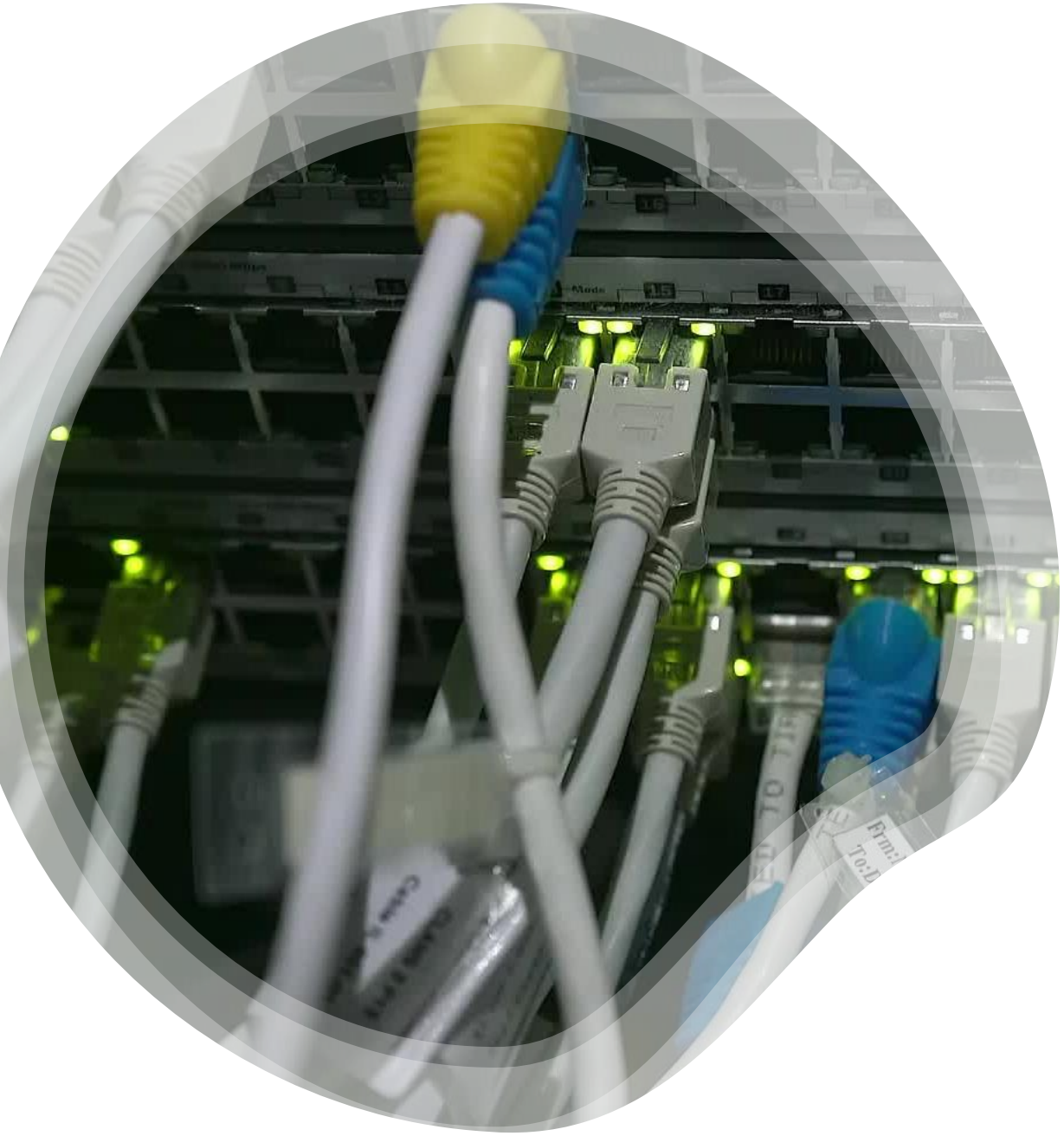
```
$ systemctl start rke2-agent.service
```

- Follow the logs, if you like

```
$ journalctl -u rke2-agent -f
```

RKE2 Management

How to Upgrade, Customize, and Secure Your Cluster

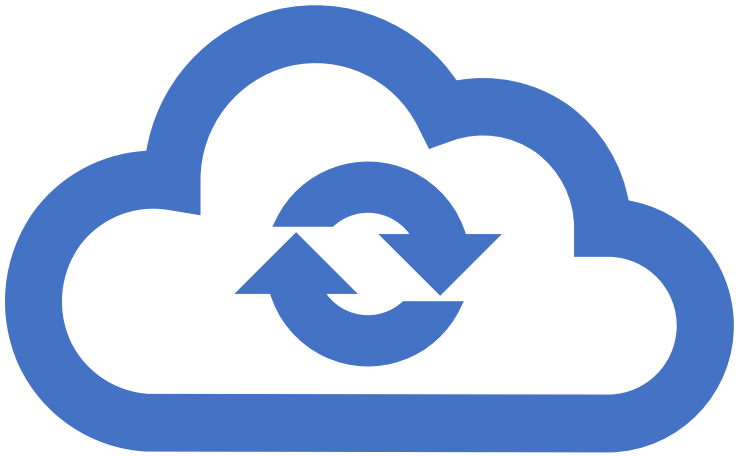


Advanced Configuration Options

- The primary way to configure RKE2 is through its `config.yaml` file, located in `/etc/rancher/rke2/` by default. Command line arguments and environment variables are also available, but since RKE2 is installed as a systemd service, they are not recommended.

- On the official guide, we find the complete list of configurations that can be used on servers (link) and agents (link). Some options must be set to the same value on all servers in the cluster.

Manual Upgrades



RKE2 can been updated through the same script used in the installation phase. Upgrade the server nodes first, one at a time. Once all servers have been upgraded, you may then upgrade agent nodes.

- To upgrade rke2 to a **specific version**

```
$ curl -sfL https://get.rke2.io |
INSTALL_RKE2_VERSION=<version> sh –
```

- If upgrading **agent nodes**, you should specify the INSTALL_RKE2_TYPE environment variable

```
$ curl -sfL https://get.rke2.io |
INSTALL_RKE2_VERSION=<version>
INSTALL_RKE2_TYPE=agent sh –
```

- Remember to **restart** the rke2 process after installing

```
# Server nodes:
$ systemctl restart rke2-server
# Agent nodes:
$ systemctl restart rke2-agent
```

Helm Integration

- Helm (installed by default in RKE2) is the package manager for Kubernetes
- Any Kubernetes manifests found in `/var/lib/rancher/rke2/server/manifests` will automatically be deployed to RKE2 in a manner similar to `kubectl apply`

```
apiVersion: helm.cattle.io/v1
kind: HelmChart
metadata:
  name: prom
  namespace: kube-system
spec:
  chart: kube-prometheus-stack
  repo: https://prometheus-community.github.io/helm-charts
  version: 66.3.0
  targetNamespace: monitoring
  createNamespace: true
```

Etcd Backup and Restore

- Snapshots are **enabled by default** and stored on each etcd node
- you can choose the snapshot **frequency** (12 h), the **path** where to save them (`/var/lib/rancher/rke2/server/db/snapshots`) and the **retention** number (5)
- There is always the option to **manually** take or delete a snapshot while RKE2 is running

Etcd Backup and Restore

Restoring a Snapshot to Existing Nodes:

- stop RKE2 service on all server nodes

```
$ systemctl stop rke2-server
```

- initiate the restore from snapshot on the first server node

```
$ rke2 server --cluster-reset --cluster-reset-restore-path=<PATH-TO-SNAPSHOT>
```

- Once the restore process is complete, start the rke2-server service on the first server node

```
$ systemctl start rke2-server
```

- Remove the rke2 db directory on the other server nodes

```
$ rm -rf /var/lib/rancher/rke2/server/db
```

- Start the rke2-server service on other server nodes

```
$ systemctl start rke2-server
```

Containerd Registry Configuration

Containerd can be configured to connect to **private registries** (i.e. Harbor) and use them to pull private images on each node. RKE2 will check to see if a `registries.yaml` file exists at `/etc/rancher/rke2/` and instruct containerd to use any registries defined in the file.

A private registry **caches** downloaded images on cluster nodes. If the image doesn't exist, it is downloaded from the official repositories, cached in the registry, and then **proxied** to the cluster.

In this way, in the rare event of a complete cluster downtime, we avoid multiple requests to official image registries when recreating the cluster, as these often impose **download limits** within a specific time frame.

Security

One of the advantages of using RKE2 is the ability to easily create a hardened K8s cluster with a simple setup. It outlines the configurations and controls required to address Kubernetes benchmark controls from the **Center for Internet Security** (**CIS**).

RKE2 is designed to be "hardened by default" and pass most of Kubernetes CIS controls without modification. Other security checks are satisfied by using the CIS profile. There are, instead, a few notable exceptions to this, left to the discretion of the administrator, that require manual intervention to fully pass the CIS Benchmark.

Useful links

- Guides and repo:
  - Repository associated with the course
  - Official Kubernetes Guide
  - Official RKE2 Guide
  - Kubespray Repository
- YouTube Channels:
  - TechWorld with Nana
  - That DevOps Guy
  - Just me and Opensource