CCN-CERT

BP/27

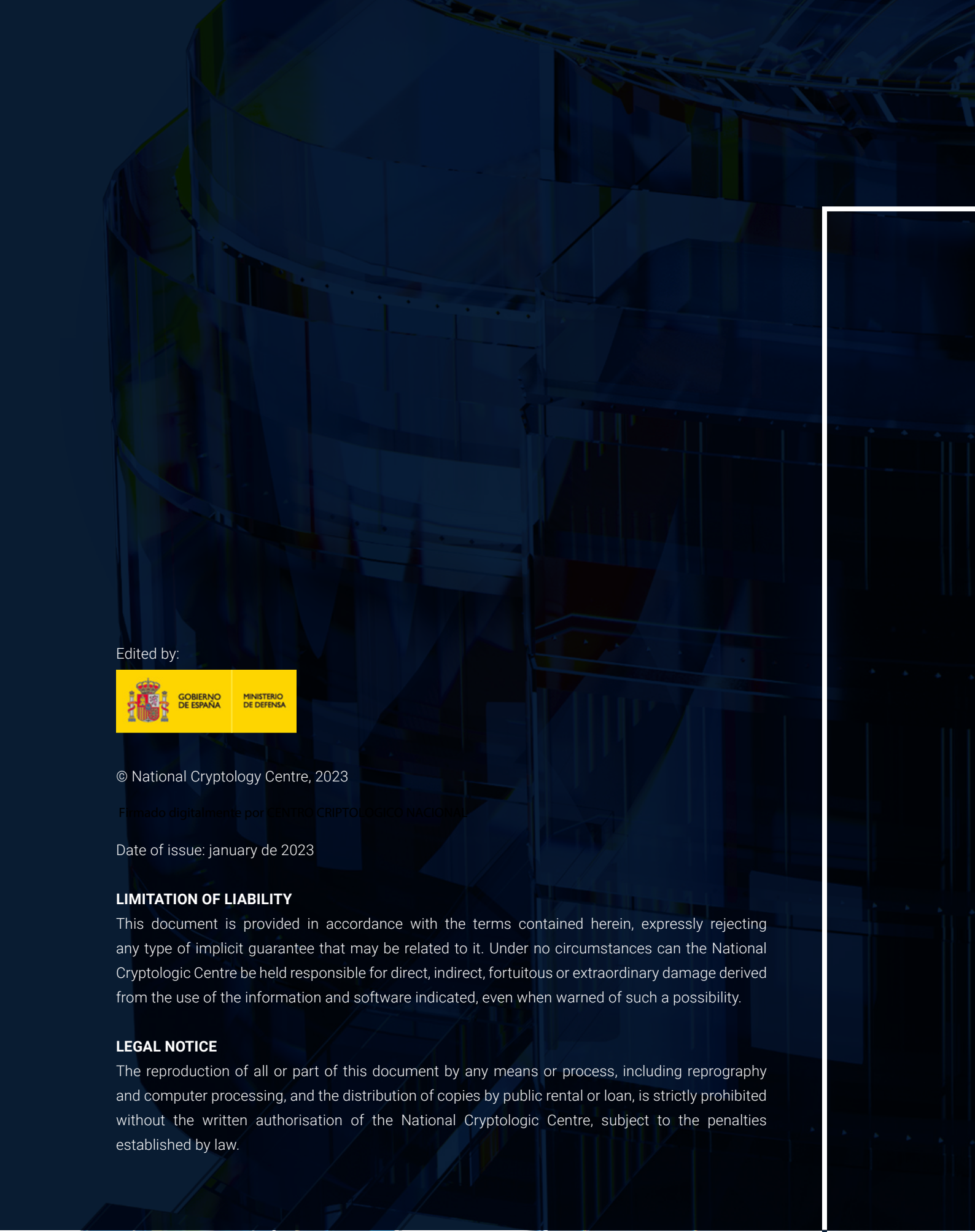
Kubernetes security recommendations

BEST PRACTICES REPORT

SEPTEMBER 2022

ccn-cert centro criptológico nacional

CCN centro criptológico nacional



Edited by:

GOBIERNO DE ESPAÑA MINISTERIO DE DEFENSA

Date of issue: january de 2023

LIMITATION OF LIABILITY

This document is provided in accordance with the terms contained herein, expressly rejecting any type of implicit guarantee that may be related to it. Under no circumstances can the National Cryptologic Centre be held responsible for direct, indirect, fortuitous or extraordinary damage derived from the use of the information and software indicated, even when warned of such a possibility.

LEGAL NOTICE

Index

1. Introduction

With the arrival of containerisation technology (Docker), there was a way to quickly build, deploy and scale applications and services or perform software deployment.

Obtaining the support of large companies such as Google, RedHat or Microsoft among others, it has become a recurring option when implementing solutions for companies, either in a local "on-premise" way or with hosting in the cloud. It is a technology with a continuous evolution, achieving the need to generate more and more critical microservices for the proper functioning of a company or entity, reaching high availability structures with redundancy of resources, and the need to protect the data handled by the hosted services.

In the face of this progress, and as increasingly complex structures are forged, the need arises to generate an orchestration of all the sets of containers, giving rise to the figure of Kubernetes.

Kubernetes, a word originating from the Greek word meaning "helmsman" or "pilot", is responsible for managing the coordination of deployments, the supervision of services and the scaling of resources. In short, the management of all the services or microservices generated by a distributed architecture of the systems.

Kubernetes is a platform for orchestrating containers, handling the coordination, monitoring, and scaling of resources in a distributed architecture. Containerization technology (Docker) is recurrent in companies.

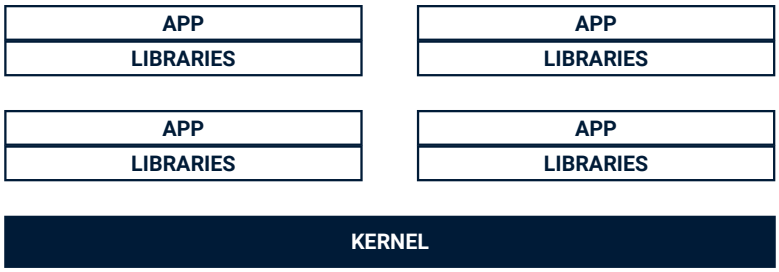


Illustration 1 - Container structure

2. Kubernetes

Kubernetes is an open-source platform designed for managing clusters of services and applications deployed in containers, which can be hosted on-premise or in the cloud. Its main functionality is the centralised management and automation of workloads.

Developed by Google and the engineers Craig McLuckie and Brendan Burns, and released in 2014, the project is managed by the Cloud Native Computing Foundation (CNCF), a section of the Linux Foundation.

Kubernetes is often referred to as “k8s”, which is an abbreviation for the eight letters in Kubernetes between the initial “k” and the final “s”.

The use of this technology, as it can be implemented in a virtualised form, provides several options for flexibility in its configuration and different layers of security.

Kubernetes is a platform for managing clusters of applications and contained services, it offers flexibility and security when used virtualized.

2.1 Versions

Kubernetes offers a major release cadence of three (3) minor updates in annual cycles, supported by security patches for approximately one year after release.

Kubernetes versions are expressed with a numerology divided into three blocks and separated by numbers.

- **First block.** This is the major or major version of the kubernetes release.
- **Second block.** Shown as the minor or secondary version.
- **Third block.** Indicates the patch version number applied to the kubernetes application.

NOTE: Additional information on the product life cycle can be found at the following link:

https://kubernetes.io/releases/release/

2.2 Components

A Kubernetes cluster can be defined as a set of different nodes, which can be different virtual or physical machines. These nodes are classified into:

- Control Plane.
- Worker nodes.

2.2.1 Kubernetes control plane

It takes care of the management of the Kubernetes ecosystem, and makes the global decisions for the cluster, reacting to all events triggered by the different nodes, such as the response to the creation of a task replica.

The components of the control plane are as follows.

- **Kube-apiserver:** Handles interaction with the various users of the system through a system control interface, managing requests to the Kubernetes cluster. The Kubernetes API server determines which requests are valid and how to process them.
- **ETCD:** This is a non-volatile data storage class, where all information relevant to the Kubernetes cluster configuration is stored.
- **Kube-scheduler:** This is where cluster management priorities are ordered and planned using APIs from the command line.
- **Kube-controller-manager:** In charge of reviewing the independent processes of the components of the Kubernetes ecosystem. These components are:
  - **Replication control.**
  - **Endpoint controller (services and pods).**
  - **Token and service account controller (namespaces).**
  - **Node controller**
- **Cloud-controller-manager:** This is in charge of running controllers that interact with the cloud. It can interact with load balancers, mediate updates or deletions from different worker nodes, or configure network paths.

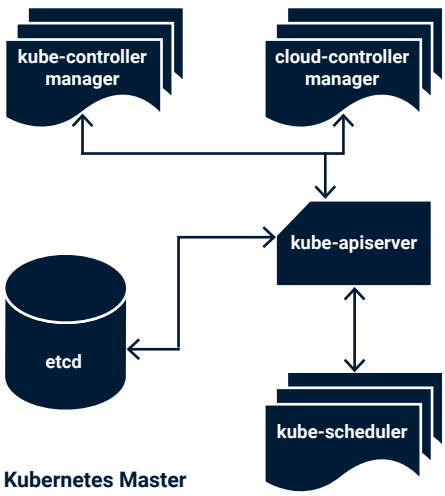


Illustration 2 - Master node structure.

2.2.2 Worker nodes

In the early days of Kubernetes development, the worker node was called a "minion", and could consist of one or more machines, either virtual or physical, depending on the type of cluster.

Each worker node is controlled by a "master node", which contains all the tools necessary for the management of one or more pods.

A pod, or pods, is a logical grouping used by Kubernetes to perform a more uniform management of a set of one or more containers that host services or applications, which are offered by Kubernetes as the final product solution.

Containers in the pod share system resources in terms of disk space location and the network names (namespaces) given to them for identification.

If several pods share a functionality, for efficient and homogeneous management, Kubernetes groups all these pods into a logical unit called "services", thus facilitating, for example, the allocation of external access IPs, or facilitating high-availability balancing.

The components of a worker node are listed below:

- **Kubelet:** Cada nodo de un clúster de Kubernetes posee un agente, el cual tiene la competencia de verificar la ejecución de los pods, comunicándolo al plano de control.
- **Kube-proxy:** Agente encargado de las reglas de red, así como de los reenvíos de conexiones con el anfitrión que aloja el clúster.
- **Entorno de ejecución de contenedores:** Es la ejecución de los contenedores en sí en el entorno de Kubernetes, soportando diferentes tecnologías de contenerización como Docker, containerd, cri-o o rktlet. Básicamente ejecuta cualquier implementación de contenerización capaz de soportar una interfaz de runtime de Kubernetes (Kubernetes CRI).

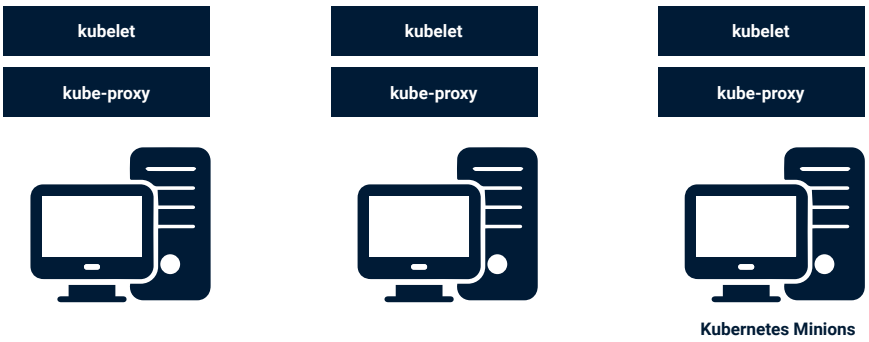


Illustration 3 - Node worker structure.

2.3 Architecture

The architecture used by Kubernetes is a “master-slave” architecture between master nodes and worker nodes.

The servers or equipment in a cluster usually comprise one or more ‘nodes’, which in turn may be hosted on one or more physical or virtual on-premise servers or in the cloud.

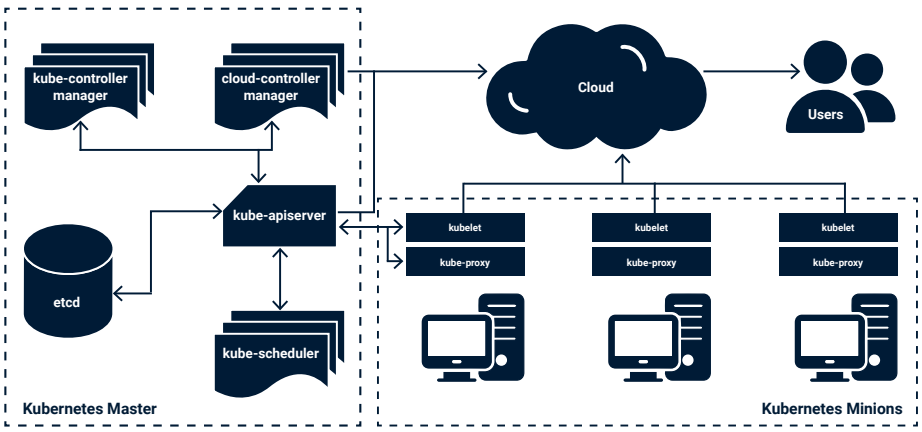


Figure 4 - Kubernetes cluster architecture

2. Kubernetes

Kubernetes clusters, when hosted in the cloud, must be placed with Kubernetes-certified service providers, which will manage part of the infrastructure of the Kubernetes environment. It should be noted that most of the default configurations used by these service providers are not usually accompanied by ironclad security policies, but rather prioritise functionality and access to the service.

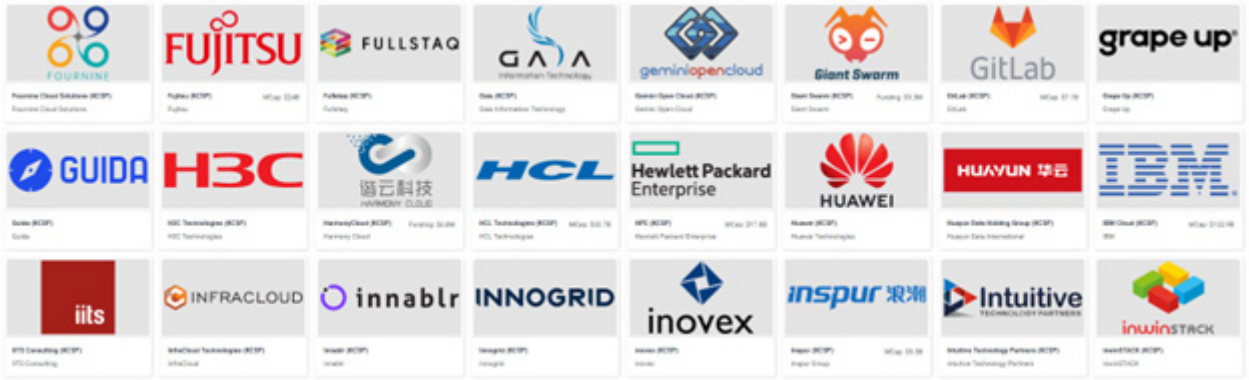


Illustration 5 - Example of certified service providers.

NOTE: For a complete list of CSPs (Certified Service Providers), please visit for more information.

www.l **https://Kubernetes.io/partners/#kcsp**

An organisation that needs to have additional security features and control over them may choose to establish Kubernetes services via local on-premises servers.

3. Kubernetes security

An organisation may have all its data hosted on a Kubernetes cluster, information that can be coveted by cybercriminals.

Likewise, the computing power of the computers that host the different Kubernetes processes and services can be usurped to make use of techniques for the extraction of cryptocurrencies by mining methods.

Another method of aggression by a cybercriminal is the denial of public services offered by the different Kubernetes solutions of an organisation, causing downtime and even financial losses.

Kubernetes security can be divided into three points:

Security in Kubernetes is divided into three points: data protection, preventing misuse of resources and preventing denial of service attacks.

Elements that make up the system and its infrastructure.

Kubernetes component configurations.

Incidents in execution times.

3.1 Infrastructure security

3.1.1 Virtualisation

The use of virtualisation technology to run a container system can provide extra layers of security for the operation of different solutions and services. Such a system is given a “fraction” of the total resources of the physical server, so it cannot access the entire pool of resources of the host where it is hosted, avoiding a total denial of the system. Container isolation is provided.

The virtualisation of systems provides the functionality to perform “restore points”, in case of the need to revert to a point in time prior to a configuration applied to the system.

3.1.2 Kernel-based protection

The use of kernel tools, such as “seccomp”, can be used to limit the calls made by a container on the system, thus reducing exposure to a potential attacker.

Another tool for kernel protection of the system is SElinux. The use of the security context provided by SElinux provides the container with protection against unwanted changes and a higher level of security auditing.

3.1.3 Network policies and segmentation

Containers within a Pod share IP address and port and these IP addresses must be declared in the "localhost" configuration files in order to make connections.

Bearing in mind that the information exchanges between pods and services are carried out through network communications, network segmentation is a point to be taken into account, in order to make connections between the different elements in a secure way, isolating the containers and services that are necessary.

A correct network policy must be implemented, delimiting it with the use of firewalls and encryption of allowed traffic for the protection of data traffic between the different points of the system.

Encryption of communications must be carried out using TLS certificates, in their versions TLS 1.2 or TLS 1.3, versions of protocols, which are secure.

The following table shows the main configuration parameters for certificate-encrypted communications.

File	Parameter	Value	Remarks
kube-apiserver.yaml	etcd-certfile	[route]	Encrypted connection between APIserver and etcd
kube-apiserver.yaml	etcd-keyfile	[route]	Encrypted connection between APIserver and etcd
kube-apiserver.yaml	tls-cert-file	[route]	APIserver use of encrypted traffic
kube-apiserver.yaml	tls-private-key-file	[route]	APIserver use of encrypted traffic
etcd.yaml	auto-tls	false	Avoid using self-signed certificates
etcd.yaml	peer-client-file	[route]	TLS configuration for etcd
etcd.yaml	peer-key-file	[route]	TLS configuration for etcd
etcd.yaml	peer-client-cert-auth	true	Secure authentication etcd
etcd.yaml	peer-auto-tls	false	Avoid using self-signed certificates

File	Parameter	Value	Remarks
etcd.yaml	trusted-ca-file	[route]	Use of certification authority
10-kubeadm.conf	tls-cert-file	[route]	Use of encrypted traffic for Worker Node
10-kubeadm.conf	tls-private-key-file	[route]	Use of encrypted traffic for Worker Node
10-kubeadm.conf	RotateKubeletServerCertificate	true	Certificate rotation for kubelet server

Table 1 - TLS configurations.

For the creation of a network policy, a Container Interface Plug-in (CNI), which supports NetworkPolicy APIs, is required.

Each pod in the kubernetes cluster has its own private IP in the cluster and can be treated just like a real computer where a port is assigned to the IP address (socket). These sockets can host resources and access to applications of an organisation, so there must be an element that provides stability to the network elements between the kubernetes pods, since, in a process, for example, of changing or updating pods, these IP addresses can be modified resulting in a loss of connection or loss of access to a resource.

Kubernetes addresses these potential connection losses due to IP switching by unifying IP addresses into logical sets of Pods through services that are tagged.

The way to access these services from an external source to Kubernetes is usually done through “NodePorts”, randomly granting a port to an IP, which can be statically configured.

For proper network segmentation, Kubernetes CNI add-ons can use nodeports or firewalls, as well as load balancers, for this purpose. In this way, segmenting the network limits the attack surface for a cybercriminal.

3.1.4 Container namespaces

"Namespaces" are a way of delimiting Kubernetes resources by assigning a label to a scope of cluster resources, thus forming a unit on which a set of RBAC (role-based access control) authorisations and network policies can be applied.

There are three default namespaces:

- **Kube-system. Used for kubernetes components.**
- **Kube-public. Used to tag public resources.**
- **Default. These are user resources.**

3.1.5 Resource policies

A good management practice is to limit the use of physical hardware resources in order to avoid an overflow of Kubernetes server resources. Restrictions can be done on a per-container or per-namespace basis.

Restrictions are made by creating policy files with the extension "yaml" and applying them using the "kubectl" command.

As an example, a creation of resource limitation policies and how to apply them in a Kubernetes cluster is shown.

Passage	Description
1.	Create a namespace so that the resources are isolated from the rest of your cluster. `$ kubectl create namespace nombre-quota-mem-cpu`

2.	Create the policy configuration file with the extension “yaml” [policy.yaml]:
	`apiVersion: v1`
	`kind: ResourceQuota`
	`metadata:`
	`  name: mem-cpu-demo`
	`spec:`
	`  hard:`
	`  requests.cpu: “1”`
	`  requests.memory: 1Gi`
	`  limits.cpu: “2”`
	`  limits.memory: 2Gi`

3.	Apply the resource limitation policy with the following command. `$ kubectl apply -f [política.yaml] --namespace=nombre-quota-mem-cpu`
4.	You can check the policy enforcement with the following command. `$ kubectl get quota --namespace=nombre-quota-mem-cpu`

3.1.6 Fortification of the control plane

It is the core of Kubernetes, managing the containers, the various maintenance tasks, as well as the management of the cluster's sensitive information, making it a sensitive point in fortifying the entire Kubernetes ecosystem.

Communications must be encrypted using TLS protocols (versions 1.2 and 1.3), have role-based access control (RBAC) and strong authentication methods with sufficient password complexity.

Another measure is the network access control of the communications carried out by the Kubernetes API, limiting by means of firewalls the types of connections and their addressing, in order to establish a correct use of the API with the greatest possible reliability.

Below is a table with the main ports used by the Kubernetes API, as well as their protocols.

Component Node	Traffic guidance	Protocol	Ports
Kube-controller-manager	Starter	TCP	10257
Kube-scheduler (task scheduler)	Starter	TCP	10259
API Kubernetes	Starter	TCP	6443
API etcd	Starter	TCP	2379-2380
API kubelet	Starter	TCP	10250

Table 2 - Kubernetes API network traffic.

The local configuration files that manage the control node components must have permission and user access management.

File	Permits	Owner
kube-apiserver.yaml	644 or more restrictive	root:root
kube-controller-manager.yaml	644 or more restrictive	root:root
kube-scheduler.yaml	644 or more restrictive	root:root
etcd.yaml	644 or more restrictive	root:root
CNI configurations	644 or more restrictive	root:root
BBDD_etcd	644 or more restrictive	root:root
admin.conf	644 or more restrictive	root:root
scheduler.conf	644 or more restrictive	root:root
controller-manager.conf	644 or more restrictive	root:root
pki certificates (crt extension)	640	root:root
pki certified keys (extension key)	400	root:root

Table 3 - Control node membership and file permissions.

NOTE: The default location for yaml files is /etc/kubernetes/manifests/. The default location for conf files is "/etc/kubernetes/". For more information, see:

https://kubernetes.io/docs

3.1.7 Fortification of worker nodes

As with a master node, network access control must be in place for each worker node in the Kubernetes cluster. By identifying the direction of each node's network traffic, you can limit access to the nodes using a firewall by configuring them through their sockets.

The ports and services used by the Kubernetes worker nodes are shown below.

Component Node	Traffic guidance	Protocol	Ports
Node Services	Starter	TCP	30000-32767
API kubelet	Starter	TCP	10250

Table 4 - Worker node network traffic.

The local configuration files that manage the control node components must have permission and user access management.

File	Permits	Owner
10-kubeadm.conf	644 or more restrictive	root:root
Archivo “kubeconfig” de kube-proxy	644 or more restrictive	root:root
kubelet.conf	644 or more restrictive	root:root

Table 5 - Worker node membership and file permissions.

NOTE: For more information, please consult the following links:

https://Kubernetes.io/docs/admin/kube-proxy/

https://Kubernetes.io/docs/admin/kubelet/

https://Kubernetes.io/docs/tasks/administer-cluster/kubelet-config-file/

3.1.8 Sensitive information

Sensitive information in kubernetes (secrets), such as passwords or certificates, are placed for use in configurations in "yaml" files, container images or environment variables, being safer to use their content as a file, and not as a variable, since access can be regulated by permissions and users in a file.

This sensitive data is stored by Kubernetes unencrypted, it encodes it in base64 and can be encrypted by configuring its data encryption in the server API

.

The activation of the API server encryption settings is shown below as an example.

File	Parameter to configure	Value
kube-apiserver.yaml	--encryption-provider-config	/[ubicación fichero]/secrets.yaml

Table 6 - Apiserver configuration.

NOTE: For more information, please consult the following link.

https://Kubernetes.io/docs/tasks/administer-cluster/encrypt-data/.

3.1.9 Updates

Over time, software and applications in the Kubernetes environment are susceptible to security flaws as a result of the continuous evolution of attacks and the emergence of vulnerabilities and security exposures. For this reason, they may need to be updated and/or patched, regardless of the media on which they are deployed, which in turn must apply the updates and/or security patches published by their respective vendors.

3. Kubernetes security

Kubernetes offers a major release cadence of three (3) minor updates in annual cycles, supported by security patches for approximately one year after release.

Kubernetes versions are expressed with a numerology divided into three blocks and separated by numbers.

- **First block.** This is the major version of the kubernetes release.
- **Second block.** Shown as the minor or secondary version.
- **Third block.** Indicates the patch version number applied to the kubernetes application.

When an application undergoes an update, and especially when it is a major version update, it is usually necessary to restart the cluster in order to correctly apply the changes, which will cause an outage in the services deployed by the Kubernetes solution. To avoid these service outages, it is sometimes necessary to install a high availability system in order to obtain redundancy in the services.

It should be noted that, in a production environment, the application of major changes to applications as a result of an update may cause the updated application to be incompatible with its configuration prior to the update. Updates should be applied beforehand in a pre-production or test environment, checking the correct operation of the deployed applications, and once their correct operation has been confirmed, apply the updates and configuration changes to the production environment.

3.2 Container and pod security

3.2.1 Pod access control

The access to a Kubernetes cluster is done through user authentication. To prevent impersonation attacks and to be able to make unauthorised calls to the Kubernetes API, the user authentication supports different access mechanisms that are not enabled by default.

There is a differentiation to be made in the users of a Kubernetes system, with two kinds of accounts:

- **Service accounts.**
- **Nominal accounts for users.**

The pods each make requests and calls to the Kubernetes API. This request is made with a service account attached to the pod, allowing the "admission controller" call of "ServiceAccount" to be made. Each namespace must have a service account associated with it, otherwise the admission handler will attach a default account to the namespace.

Service accounts must be created individually and given specific permissions in order to delimit their functions.

Default service accounts should not be used, as they are assigned a set of permissions by default. Likewise, when a default account is created, it is given a token to be able to access the API in an uncontrolled way.

To prevent the creation of tokens to default service accounts and to prevent the assignment of permissions, kubernetes has a control mechanism for this purpose.

File	Parameter to configure	Value
/var/run/secrets/Kubernetes.io/serviceaccount/token	automountServiceAccountToken	false

Table 7 - Pod access configuration.

For named users or administrators, a specific authentication method must be applied. The methods for authentication in the Kubernetes system are as follows:

- **X509 Certificates**
- **Bootstrap tokens**
- **OpenID tokens**

You must have a password policy and authentication methods with sufficiently high complexity to prevent unintended access, avoiding anonymous requests or forms of access. In kubernetes versions 1.6 and later, anonymous requests are enabled by default in the system.

Anonymous access denial must be configured in the API server.

Fichero	Parámetro a configurar	Valor
/etc/Kubernetes/manifests/kube-apiserver.yaml	anonymous-auth	false

Table 8 - Apiserver access configuration.

3.2.2 Role-based access control

Role-based access control (RBAC) is used to manage access to cluster resources by users.

You must have the RBAC enabled in the API in order to properly control and manage the cluster resources. To start the role-based access control, run the following command.

```
$ kubectl get quota --namespace=nombre-quota-mem-cpu
```

In a poor access role security policy, "AlwaysAllow" policies will appear. Such configuration parameters should be reviewed and the correct access role policy margins set at the level of user operability within the cluster, always imposing a least privilege policy on the access to the system.

Two types of permissions can be established:

Roles. Sets permissions for a particular "space names".

Cluster Roles. Sets permissions on all resources in the cluster, regardless of "namespaces".

Roles are used to add permissions, they do not have denial rules. The kubernetes API server will deny permissions that are not explicitly allowed.

Once the roles or role clusters have been created, the "RoleBinding" shall be used to associate them to a user or group.

RoleBinding" or "ClusterRoleBinding" is used when you want to manage a set of similar permissions for different namespaces.

Privileges assigned to users, groups and service accounts should follow the principle of least privilege, giving permissions only to perform a task, as well as roles.

3.2.3 Immutable containers

In the event that one of the containers is compromised and an attacker has taken control of it, he could delete files, create scripts or even modify the configuration parameters of a component of the container.

All of the above scenarios can be avoided by locking the container file system by kubernetes, putting the system in a read-only state. However, this mode can lead to anomalous behaviour in the operation of the container, so if it is implemented, a prior study must be carried out in order to obtain an optimal operating result.

File	Parameter to configure	Value
/etc/Kubernetes/manifests/kube-apiserver.yaml	readOnlyRootFilesystem	true

Table 9 - Apiserver access configuration.

3.2.4 Creation of containers

A container image can be created from scratch, or it is usually downloaded from a repository and modifications can be made on the basis of this image.

Whenever images are downloaded from repositories, these repositories must be from trusted sources.

You can configure the Kubernetes environment to restrict deployment to the cluster of container images that do not have a digital signature from a trusted source.

When inserting an image into the Kubernetes ecosystem, it must be checked for deprecated libraries, known vulnerabilities, as well as configurations that violate port security or insecure permissions.

3.2.5 Security of pods

Kubernetes has two mechanisms for securing cluster pods. These mechanisms are:

- **Pod Security Admission.**
- **Pod Security Policies (PSP).**

The "Pod Security Admission", a security policy implemented by Kubernetes, is a default security policy from version 1.23 onwards. Pods are classified (basic, restricted or privileged) providing a simpler implementation than in PSP.

With the "Pod Security Admission" feature enabled, namespaces can be configured to define a set of admission control modes for security usage in the pod and per namespace.

Kubernetes defines a set of tags to define standardised levels of pod security.

Mode	Description
Enforce	Any policy that is in breach will be rejected by the pod.
Audit	Policy violations will be enforced but will be recorded by the system's audit logs.
Warn	Policy violations will be notified to the user but changes in the system will be enforced.

Table 10 - Security policy labels.

In the now deprecated Pod Security Policies (PSP), obsolete since version 1.21, it validates the creation of pods and updates the requests to the system's PAI using a set of rules for their management.

3.2.6 Security in Kubernetes environments

Just as important as the security of the kubernetes cluster itself is the security of the environment that hosts the solution. The command-line cluster management tool is “kubectl”, which can be deployed on Windows, linux and macOS operating systems.

NOTE: Additional information on operating system installation and support can be found at the following link:

https://Kubernetes.io/es/docs/tasks/tools/_print/

If hypervisor-hosted containerisation is used, virtualisation limits the use of hardware, rather than the operating system on which kubernetes would be deployed.

A hypervisor isolation is more secure than traditional container isolation, as it can further segregate different nodes and limit system resources to those allocated by the hypervisor.

In a Linux kernel-based solution, the seccomp tool can be used to limit container system calls, reducing the attack surface on the kernel. Similar tools in functionality can be Selinux or APParmor.

Some container engine solutions offer an isolation between container application and host kernel called “SandBox”, confining resources in an isolated virtual space.

3.3 Incidents in execution times

3.3.1 Non-root containers

By default, there are services in the containers that are executed with the “root” user, a user with access to all system resources. In many of these tasks, it is not necessary to use this administrator user, but any other user with a correct configuration of access to the cluster resources can do it.

An archive can be configured so that certain tasks are not performed by the “root” user and this non-default setting is usually set at the time the archive image is created.

An alternative in Kubernetes is to use a pod with the “SecurityContext” parameter set to “runAsUser”, specifying that the user identifier is not “0”, since this identifier belongs to the “root” user.

3.3.2 Token security for service accounts

By default, kubernetes grants a “secret” token to a service account when creating a pod. This token is granted within the pod at the runtime of the service it performs.

Not all applications within containers require direct access to service accounts. If an application is compromised by an attacker, integrations and access to tokens can be compromised and used to access the system’s API.

A study should be made of the need to automatically grant a token to a service account. To avoid creating tokens to default service accounts and to avoid assigning permissions, kubernetes has a control mechanism for this purpose.

File	Parameter to configure	Value
/var/run/secrets/Kubernetes.io/serviceaccount/token	automountServiceAccountToken	false

Table 11 - Access configuration.

If by necessity this security measure cannot be implemented, e.g. in the provisioning of authentications to external services, control by role-based access control policies should be enforced, minimising the privileges of the pod within the cluster.

3.3.3 Audit logs

Kubernetes audit logs provide a way to track security-relevant information on your system. Log entries are generated to monitor as much information as possible about events occurring on the system. This information is crucial in mission-critical environments to determine who is violating security policies and what actions have been taken. Auditing does not provide additional security to your system, rather it can be used to discover violations of the security policies used on the system.

These violations can be prevented by additional security measures that, with the study of events collected in the system logs, help in the elaboration, such as role-based access control policies.

Audit logs monitor the activity of the cluster. Chronologically, logs are kept on security issues and on the activities performed by system users and other components, thus enabling the facts of a particular event to be verified.

System administrators must implement a method of logging and monitoring of resources and access to them by any user.

Some of the most significant points for monitoring Kubernetes are:

- **API calls**
- **Performance metrics**
- **Resource consumption**
- **Network traffic**
- **Image and container modification**
- **Changes of privileges**
- **Creation and modification of the task scheduler.**
- **Modification of the file hierarchy.**

3.3.4 System logs

Event log captures in security audits should be performed on a regular basis, taking into account previously collected log histories and cross-checking significant changes between these files, as well as performing an analysis of why changes have been made. If excessive consumption of system resources is detected that is a significant change from past audits, for example, this may indicate unauthorised access by an attacker to the system's computing power.

If the organisation has a centralised event collector (syslog), Security Information and Event Management (SIEM), the transmission of these security logs can help the detection of cluster anomalies as close as possible to runtime.

Whenever information or log records are transferred to a centralised system, the connection must be made securely using certificates, such as TLS versions 1.2 and 1.3, in order to guarantee the integrity of the information transferred from one point to another.

When using a log server external to Kubernetes, you must configure the log forwarder with append-only access to external storage. This protects externally stored logs from being deleted or overwritten from within the cluster.

3.3.5 Kubernetes audit logs

In the Kubernetes cluster, whenever a request is made by the front-end kubi-apiserver, an event log is produced, either by a request from a user, an application or by the control plane. When an event is logged, the kube-apiserver looks for the first match in its audit policies to perform the event classification, not by default.

Logging policy files are stored in “yaml” format and are used to set the rules and specify what level of audit should be logged for an event that matches a previously created policy.

```yaml
apiVersion: audit.k8s.io/v1 # Esto es obligatorio.
kind: Policy
# No generar eventos de auditoría para las peticiones en la etapa RequestReceived.
omitStages:
  - "RequestReceived"
rules:
  # Registrar los cambios del pod al nivel RequestResponse
  - level: RequestResponse
    resources:
    - group: ""
      # Los recursos "pods" no hacen coincidir las peticiones a cualquier sub-recurso de pods,
      # lo que es consistente con la regla RBAC.
      resources: ["pods"]
  # Registrar "pods/log", "pods/status" al nivel Metadata
  - level: Metadata
    resources:
    - group: ""
      resources: ["pods/log", "pods/status"]

  # No registrar peticiones al configmap denominado "controller-leader"
  - level: None
    resources:
    - group: ""
      resources: ["configmaps"]
      resourceNames: ["controller-leader"]
```

Figure 6 - Kubernetes registry policy snippet.

NOTE: For further information on this point, please visit the following link.

https://Kubernetes.io/es/docs/tasks/debug-application-cluster/audit/

For a policy or rule to be considered valid, it must have one of the audit levels managed by the Kubernetes cluster.

- **NONE.** Events that execute the logging rule are not logged.
- **METADATA.** The metadata of the request, such as the timestamp, is recorded, but not the request itself, nor the response to the request.
- **RESQUEST.** The metadata and the request that triggers the execution of the event are logged, but the response is not logged. This does not apply to non-resource requests.
- **RESQUESTRESPONSE.** The metadata, the request and the response are recorded. This does not apply to non-recourse requests.

The maximum level of audit logs is "RESQUESTRESPONSE", providing the maximum amount of information available in case of an event that needs to be monitored.

Logs with the maximum level of "RESQUESTRESPONSE" may fall into the capture of "secret" objects where sensitive information, such as passwords, may be stored in base64 encryption. In addition, it generates a considerable amount of records especially in clusters that are in production, which can lead to space shortage incidents.

A study should be established, for later implementation, of policies in the organisation's Kubernetes cluster audit logs, oscillating the policies with the different levels of auditing and with special monitoring of the processes or calls that are critical for the correct functioning of the application.

Kube-apiserver, has a series of functionalities and configuration parameters for the handling, storage and rotation of logs or audit logs generated by kubernetes security policy events. "Webhook is an http backend and endpoint, which kubernetes will invoke at the time of an event execution, configured to send to an external http API the desired logs or information.

NOTE: For further information on this point, please visit the following link.

https://Kubernetes.io/docs/tasks/debug/debug-cluster/audit/

File	Parameter	Value	Remarks
kube-apiserver.yaml	audit-log-path	[route]	Location of audit logs
kube-apiserver.yaml	audit-log-maxage	Value in days] [Value in days	Rotation of audit logs
kube-apiserver.yaml	audit-log-maxbackup	Value in days] [Value in days	Retention of audit logs
kube-apiserver.yaml	audit-log-maxsize	Value in MegaBytes] [Value in MegaBytes	Maximum size of audit logs

Table 12 - Log configuration.

3.3.6 Registration of containers and worker nodes

Kubelet performs audit logging on each node individually within each container, managing the logs, storing and rotating them according to policy settings.

Kubelet, is a command console tool managed by the kubectl command.

Command	Parameter	POD	Remarks
kubectl	--namespace [nombre]	logs [POD]	Display container records

Table 13 - Command kubectl.

3.3.7 Audit by SECCOMP

All the types of logs mentioned above can be completed by performing audit logs of the kernel calls that occur. For this purpose, the "seccomp" application can be used to audit container calls in the Kubernetes ecosystem.

Seccomp is disabled by default, by enabling and configuring it, it can limit system calls by containers, reducing the attack surface of the system. Another functionality is to record or log all calls made to the kernel.

File	Parameter	Value	Remarks
Kubelet-config.yaml	SeccompDefault	true	Activation seccomp

Table 14 - Command kubectl.

The seccomp configurations are made by profiles, allowing, denying or logging calls. You will need to configure seccomp profiles, with the parameters that best suit the needs of your organisation.

File	Parameter	Remarks
/var/lib/kubelet/seccomp/[profile]	seccomp configuration for a pod] [seccomp configuration for a pod	Seccomp activation and configuration

Table 15 - Seccomp. configuration

With the help of seccomp, it is possible to identify which system calls are necessary for standardised cluster operations, and thus use patterns of pod operations to identify any anomalous behaviour patterns and identify malicious activities.

3.3.8 Threat detection

The use of log collectors, such as journald, syslog and rsyslog, speed up the collection of system information and help to find patterns for the detection of system threats.

Continuous monitoring of logs and system resources at runtime is a very powerful tool for understanding the operation of Kubernetes and detecting anomalies in the processes, and thus being able to determine whether malicious agents have attacked or are attacking the system. For such tasks, the use of external tools such as a SIEM can be of great help to an organisation, not only in terms of security, but also in terms of system operability, being able to visualise in real time the management of resources by Kubernetes.

4. Checklist

Criticality	Description
High	The host computer hosting the Kubernetes application must have its system updated, all security patches published by the product manufacturer applied, and its lifecycle in force.
High	Kubernetes must have the latest updates and security patches released by the manufacturer installed.
High	Connections between nodes and to nodes are encrypted by certificates.
High	Communications are encrypted using TLS protocols in versions 1.2 or 1.3.
High	"Control plane" configuration files may only be modified by administrator users.
High	Certificates and their key generators in Kubernetes are owned by the "root" user and the "root" group.
High	Only administrator users have access to the certificates and their kubernetes key generators.
High	The "worker node" configuration files may only be modified by administrator users.
High	Login to the Kubernetes environment is only established by authenticated users.
High	Audit logs are configured in Kubernetes.
High	Containers have role-based access control, and container namespaces are defined.

Criticality	Description
Medium	The “control plane” configuration files belong to the “root” user and the “root” group.
Medium	The configuration files of the “worker nodes” shall belong to the user “root” and the group “root”.
Medium	Sensitive information called “secrets” is encrypted.
Medium	The pods’ service accounts (ServiceAccount) are configured with specific permissions to perform their tasks.
Medium	The pop security policies (PSP) are configured and active.
Medium	Service accounts are configured so that they are not automatically issued an authentication token.
Medium	Event logs are configured in Kubernetes according to audit levels.
Medium	An audit log storage policy has been configured.
Medium	Kernel call logs are configured in Kubernetes.
Low	Audit logs are stored on an external medium.
Low	Kubernetes cluster resources are configured with quota-based segmentation.
Low	In case the containers do not need modification, they are found as “immutable”.

5. Decalogue of recommendations

The following are ten security recommendations for using Kubernetes.

Decalogue of recommendations for Kubernetes

1. It is recommended to **always use the latest stable version** with the latest updates recommended by the manufacturer, thus eliminating attack vectors with remediations applied by the manufacturer.

2. The **use of secure cryptographic algorithms** is recommended. The use of a robust and reliable encryption algorithm eliminates the possibility of interception of system messages.

3. It is recommended to **use official and/or trusted sources** for deploying container images. The kubernetes environment manages containers that perform certain functions for an organisation. Such containers can be created by the organisation or downloaded from official and trusted sources for deployment on the system.

4. It is recommended to **always use secure protocols (TLS),** to secure end-to-end communications, in order to avoid interception of information.

5. The **use of strong and complex password policies** is recommended, with a minimum length of characters, in addition to the use of upper-case letters, lower case, symbols and numbers, together with a period of validity of the password.

6. The **use of minimal and up-to-date images** is recommended. By using minimal images, less resource consumption and a smaller attack vector is achieved.

7. It is recommended to **evaluate the privileges used by the containers.** The principle of minimum functionality and minimum privilege should be followed.

8. **Physical segregation of the network** is recommended and the optimisation of resources and the segmentation of network elements, limiting access to information and preventing the spread of security incidents.

9. It is recommended **to segregate roles and permissions for users.** The execution needs within Kubernetes and the need for access to the information being managed are defined for each user.

10. It is recommended to **document the Kubernetes platform.** Kubernetes infrastructure can be very large, hosting several control nodes and workers. Keeping the system documented helps to quickly identify idle or under-utilised containers, enabling resource optimisation and better system management. It is advisable to update this documentation with each relevant change made to the system.

6. Glossary

This section contains a description of the terms most used in this document for identification and understanding during the course of the document.

Term	Description
K8s	Abbreviation for kubernetes.
Pod	A set of one or more containers, and a specification of how to run those containers.
Cloud controller manager (CSP)	Optional component used for cloud-based deployments. The cloud controller interacts with the cloud service provider (CSP) to manage load balancers and virtual networks for the cluster.
ETCD	The persistent backup store where all information about the state of the cluster is stored.
SELinux	Security module for the Linux kernel.
AppArmor	Security module for the Linux kernel.
RBAC	Role-based access control.
Seccomp	A facility in the Linux kernel that allows you to limit the number of system calls a process can make.
API	Application Programming Interface.
CSP	Certified service provider.
CA	Certification Body.
TLS	Cryptographic protocol used in networks. Transport Layer Security.
HTTPS	Secure Hypertext Transfer Protocol.
Docker or container	Open source project that automates the deployment of applications.
on-premise	Installation of software or hardware locally.
Control plan	A set of components that run on a single node in the Kubernetes cluster controlling various aspects of the cluster.

6. Glossary

Term	Description
Worker nodes	Set of machines performing the requested tasks assigned by the control plane.
Kube-apiserver	It oversees the interaction with the different users of the Kubernetes system.
Kube-sheduler	It orders and plans priorities in cluster management.
Kube-controller-manager	Review the kubernetes processes.
Pod	A set of one or more containers deployed on a single node. It is the smallest and simplest object in Kubernetes.
Kubectl	Command line interface where you can manage your Kubernetes cluster.
Kubelet	Application of each node, which communicates with the control plane.
Kube-proxy	Kubernetes Network Proxy.
Yaml	Human-readable data serialisation format inspired by different programming languages.
NodePort	Access through a port to the nodes.
Namespace	Virtual clusters that are backed up by the same physical cluster
Cluster	Distributed computing systems linked together, typically by a high-speed network, and behaving as if they were a single server.
Secrets	Secret objects in Kubernetes to store and manage sensitive information such as passwords, OAuth tokens and ssh keys.
hipervisor	Virtual Machine Monitor
log	The sequential recording in a file or database of all events affecting a particular process.
Front-end	User interface.
Back-end	Server or service engine.

centro criptológico nacional

centro criptológico nacional

www.ccn.cni.es

www.ccn-cert.cni.es

oc.ccn.cni.es