Here's What's Actually Going On with Kubernetes Security

The complete Kubernetes security guide in plain language - Free sample (Chapter 1)



Luis Felipe Ariza Vesga

Here's What's Actually Going On with Kubernetes Security

Here's What's Actually Going On with Kubernetes Security

The complete Kubernetes security guide in plain language
- Free sample (Chapter 1)

Luis Felipe Ariza Vesga

Here's What's Actually Going On with Kubernetes Security
The complete Kubernetes security guide in plain language - Free sample (*Chapter 1*)

Copyright © Luis Felipe Ariza Vesga.
All rights reserved.

The information in this book is distributed on an "as is" basis, without warranty. While every precaution has been taken in the preparation of this work, neither the author nor the publisher shall have any liability to any person or entity with respect to any loss or damage caused or alleged to be caused directly or indirectly by the information contained in this book.

ISBN: 9798163541126

For the warrah, the Falkland Islands wolf (Dusicyon australis), the only land mammal native to those islands. It did not fear people because it had never had a predator, and Charles Darwin, who saw it in 1833, wrote that the gauchos killed it in the evening by holding out a piece of meat in one hand and a knife in the other. It was hunted for its fur, poisoned by settlers who saw it as a threat to their sheep, and by 1876 none were left; the IUCN lists it as extinct. Darwin saw it coming: within a few years, he wrote, this fox would be classed with the dodo. And for everyone who looks after a cluster: trust is earned with evidence, never handed out at the gate.

Contents

Conventions Used in This Book **1**
Typography and References 1
The Four Callout Boxes 2
Third-party credits and licenses 3

Preface **5**
What This Book Is About 7
What This Book Is Not 7
How This Book Is Organized 8
About the Commands and Code Examples 9
How to Contact Us 10
Acknowledgments 11

I Cluster Setup (15%) **13**

1 The Rule Book for Who Gets Into Which Tower **15**
1.1 The Complex With No Visitor Policy 15
1.2 Two Rule Books, Not One 17
1.3 The Condition That Decides Whether the Rules Hold 22
1.4 The Inspection Before the Shift 24
1.5 Practice in the lab 25
1.6 Summary 26
Bibliography 27

Conventions Used in This Book

This book's small visual system, demonstrated live, not just described.

Before we start, half a page on how this book is written. It takes two minutes to read and saves you confusion later.

Every chapter opens with an everyday scene. That scene is not decoration: it **is** the technical concept told another way, and it comes back at the end of the chapter, in the summary, explained with the right words. If a technical explanation does not land, go back to the opening scene: the answer is usually there in the everyday analogy. And the scenes are not loose pieces. They tell one story, that of the Los Nogales residential complex, which opens its doors in chapter 1 with no rules, no cameras and a guard who lets everyone in, and which learns, chapter by chapter, to look after itself.

The analogy is there to translate technical language into situations you already know, not to make you laugh. It never replaces rigor: after it comes the official definition, with a link to its source, and every command printed in these pages was run on a real cluster first.

Typography and References

Monospaced type is used for everything you would type or read on a screen: object names such as `NetworkPolicy` and `Secret`, YAML fields such as `seccompProfile`, commands such as `trivy image` and `cosign verify`, and paths such as `/var/lib/kubelet/seccomp/profiles`.

Bold marks the first appearance of an important term, and also the words that tend to show up verbatim in the wording of exam tasks.

Angle brackets mark what you must replace: in `trivy image <image-name>`,

you type the real image name and leave out the brackets.

Examples always follow the same order: one line saying what you are about to see, the YAML file when one is needed, a block with the command exactly as it was typed, and a block with the output the cluster returned on that run. When an output depends on timing or on a version, the book says so.

Quotations from the official documentation are woven into the text and carry a link to the source the first time they are used in a chapter, so you can check any claim with one click.

Every chapter closes with a **bibliography** of at least three official sources, and **figures** are cited in the text that explains them.

The Four Callout Boxes

Throughout the book you will find four kinds of box. Each one means something different: a command to verify, a shortcut for the exam terminal, a behavior seen on a real cluster or a production risk. None of them warns you about trick questions, because the CKS has none: it is a hands-on exam that gives you tasks and grades the final state of the cluster. It pays to recognize them at a glance, because they are not all equally urgent.

> **Note**
> Extra information that extends what you just read, usually with a way to check it yourself. You can skip it without losing the thread. Example: kube-bench does not have to be installed on the node. The project publishes a Job manifest that runs it inside the cluster and mounts, read-only, the node folders it inspects, such as `/etc/kubernetes` and `/var/lib/kubelet`.

> **Tip**
> A shortcut for the exam terminal, a rule to remember or a faster way to do something. Example: a tag can move, a digest cannot. Anyone with write access to the registry can point `v1.2` at another image, while the `sha256:...` digest always identifies the same content.

Warning
A common mistake or a behavior that surprises people on a real cluster, the kind that leaves a task half done. If you only read one kind of box before the exam, read these. Example: RBAC has no deny rules. Permissions are purely additive, so someone with two Roles can do everything both allow, and the only way to take something away is to remove the binding that grants it.

Caution
Something that can do real damage on a production cluster: data loss, an outage or a security hole. There are few of them and they mean it. Example: an audit policy at the `RequestResponse` level for Secrets copies the content of every Secret that is read into the audit file. That file becomes as sensitive as etcd and needs the same protection.

Third-party credits and licenses

This book reproduces and translates excerpts from the official Kubernetes documentation, published by The Kubernetes Authors under the Creative Commons Attribution 4.0 International (CC BY 4.0) license. It also cites the official Falco documentation, published under the same CC BY 4.0 license. Changes relative to the original are of two types: the translation into Spanish and the everyday narrative accompanying each concept; the official definition always appears separately, with a link to the source.

If an official page changes after the book is published, the current source is the one to trust.

Kubernetes is a registered trademark of The Linux Foundation. The names of the certifications mentioned belong to the Linux Foundation and the Cloud Native Computing Foundation. This book is independent study material and is not affiliated with, endorsed by or sponsored by those organizations or any of the projects or companies mentioned. The names of the projects that appear in the book, such as Falco, Trivy, kube-bench, Cosign, Cilium, Istio, gVisor or kind, belong to their respective owners and are used only to identify the software you practice with.

Preface

How this book is organized, the voice it explains the CKS exam in, and why trusting by default is dangerous.

This book comes from the same question that started the whole "Here's What's Actually Going On" series: what if a technical exam were explained in the voice people use to talk about life in their neighborhood? The Kubernetes security exam, the CKS (Certified Kubernetes Security Specialist), is usually described as a list of loose tools: RBAC, NetworkPolicy, seccomp, AppArmor, Falco, Cosign. But under that vocabulary, securing a cluster looks a lot like looking after a residential complex: you did not build the towers and you do not choose who lives in them, but you do decide who gets through the front gate, which badge each person carries, which doors each key opens, which cameras actually work and what happens the night someone got in without permission. That is the thread of the fifteen chapters.

This is the fourth book of the series, and each one builds on the one before. Book 1, "Here's What's Actually Going On with Kubernetes", covers the basics. Book 2, "Here's What's Actually Going On with Kubernetes Administration", teaches you to install and run the cluster. Book 3, "Here's What's Actually Going On with Kubernetes Application Development", takes applications through their whole lifecycle. Book 4 is grateful for that whole road and stands on the other side: the side of whoever must stop someone from abusing what the other three built. When a topic was already explained, such as RBAC in book 2 or NetworkPolicies in books 2 and 3, it is recalled here in a few lines and looked at through an attacker's eyes.

Because that is the stance this book asks for: be paranoid, on purpose. What if this credential leaks? What if this image carries something nobody checked? What if whoever claims to come from the kube-apiserver does not? Nobody gets in because they say who they are: they get in because they proved it, and each person or appli-

cation may do what it was hired to do, and not one comma more. The badge is never lent, and the plan for the night of the emergency must already be written. Security gets neglected because nothing happens almost all the time, until the day it does: that day a company can lose the data of every customer, with their addresses and their documents, and that trust does not come back easily.

It is also worth being honest from the first page. No mix of encryption, permissions and policies makes a cluster invulnerable: a new vulnerability can always appear, or someone can steal an administrator's credentials. What those layers do is make every step of the attacker cost more, show up sooner and leave a record. And that cost matters more every year. In November 2025, Anthropic documented an espionage campaign in which AI performed 80 to 90 percent of the work against roughly thirty targets, and called it the first documented case of a large-scale cyberattack executed without substantial human intervention. In February 2026, CrowdStrike's annual report said that operations by AI-enabled adversaries grew 89 percent in one year, by that company's measure. When the attacker works at machine speed, security stops being something for later.

The official CKS curriculum, published by the CNCF (Cloud Native Computing Foundation), describes what a person who secures a cluster can do:

- Restrict access inside the cluster with network policies.
- Review component configuration against the CIS (Center for Internet Security) Benchmark.
- Set up Ingress with TLS (Transport Layer Security), protect node metadata and endpoints, and verify platform binaries before deploying them.
- Reduce what RBAC (Role-Based Access Control) and ServiceAccounts expose, restrict access to the API and upgrade Kubernetes to close vulnerabilities.
- Shrink the node's operating system surface, apply least privilege and use kernel hardening tools such as AppArmor and seccomp.
- Apply the Pod Security Standards, manage Secrets, isolate workloads with sandboxed containers and encrypt Pod-to-Pod traffic with Cilium or Istio.
- Minimize the base image, understand the supply chain with an SBOM (Software Bill of Materials), sign and validate artifacts, and analyze manifests and images with tools such as kubesec and KubeLinter.
- Detect malicious behavior, investigate the phases of an attack, keep containers

immutable at runtime and read the audit logs.

What This Book Is About

This is a study book for the CKS certification exam from the CNCF and the Linux Foundation. It follows the official v1.34 curriculum, organized in six parts that follow the exam's six domains: Cluster Setup (15%), Cluster Hardening (15%), System Hardening (10%), Minimize Microservice Vulnerabilities (20%), Supply Chain Security (20%), and Monitoring, Logging and Runtime Security (20%). There are fifteen chapters, each one opened with a scene from the Los Nogales complex that turns into the technical explanation of a real security mechanism, and closed with a summary that picks that scene up again. Each part has its own lab on a real kind cluster.

A note on versions. At the time of this edition, the curriculum published by the CNCF is v1.34, and the official certification page says the exam is based on Kubernetes v1.35. That is why the examples and labs in this book run on Kubernetes v1.35. Before you book your exam, check that page and the github.com/cncf/curriculum repository: if the curriculum changed, that page wins over this book.

What This Book Is Not

This book is not a reference manual for every field, flag or tool it mentions; each project's official documentation does that job, and every chapter cites it. Nor is it a Kubernetes course from scratch: the official CKS page requires that you have taken and passed the CKA before attempting this exam, so the book assumes you already run a cluster, already know what a Deployment and a Service are, and have already solved tasks in a terminal against the clock. No cloud provider gets a chapter of its own: the book stays with what the exam assesses on a generic cluster. And it does not reproduce questions from real exams or from simulators such as killer.sh: that would break the ethics and the confidentiality agreement of the exam, and it would not help you either, because the exam task arrives with another namespace, another image and another policy.

If any of the basics feels too big, the earlier books of the series explain them calmly: Kubernetes concepts in book 1, running the cluster in book 2 and the application lifecycle in book 3.

How This Book Is Organized

The book follows the order of the exam's six domains, grouped in six parts, and tells one story: that of Los Nogales, from the night it opens with no rules to the morning it can rebuild, minute by minute, who came in and what they did. In the first part (cluster setup), the complex writes the rules for who gets into each tower with NetworkPolicy and checks itself against the CIS Benchmark with kube-bench (chapter 1), then installs a front gate that checks IDs, with TLS on the Ingress and binaries verified before they are deployed (chapter 2). In the second part (cluster hardening), everyone gets a badge that opens only their own doors, with RBAC and least-privilege ServiceAccounts (chapter 3), and the electric gate of the management office, the cluster API, is closed to strangers and kept current with every release (chapter 4). In the third part (system hardening), the rooms nobody uses are emptied out of the node's operating system (chapter 5) and the back exits are closed with AppArmor and seccomp (chapter 6). The fourth part (microservice vulnerabilities) sets the rules of coexistence with the Pod Security Standards (chapter 7), checks the safe that holds the Secrets and the etcd encryption (chapter 8), and builds walls of their own with sandboxes and an encrypted intercom between Pods (chapter 9). The fifth part (supply chain) walks through the complex's shop: the shop window with only what it sells, that is, the minimal base image (chapter 10), the receipt for where every box came from with an SBOM (chapter 11), and the authorized supplier with its seal of approval, signatures and static analysis (chapter 12). The sixth part (monitoring and runtime) installs cameras that learn what normal looks like with Falco (chapter 13), replays the footage of who came in with the audit logs (chapter 14) and puts on the lock that does not open mid-shift, with immutable containers (chapter 15).

So that the story does not get tangled, the base analogy is fixed for the whole book:

Kubernetes	At the Los Nogales complex
Cluster	The whole residential complex
Namespace	A tower of the complex
Pod	An apartment, with the people who live in it

Kubernetes	At the Los Nogales complex
Node	The ground and structure the apartments stand on
kube-apiserver	The management office, reached through the electric gate
User and ServiceAccount	A resident with a badge, and a service robot with its own
NetworkPolicy	The rules for who gets into each tower
Admission and Pod Security	The rules of coexistence checked at the front gate

The complex's security is in the hands of Justo Segura, the guard you already know if you read book 2. Within each part, chapters move from the general to the specific, and each one starts by recalling what the earlier ones left behind. The glossary at the end gathers the key terms with this same analogy, and each part's lab assumes you have read the chapters of that part.

About the Commands and Code Examples

Every YAML, command and tool configuration example in this book was run on a kind cluster with Kubernetes v1.35, and the output printed is the output that run returned. Each tool was checked against its official documentation: kubernetes.io for Kubernetes, falco.org for Falco, docs.sigstore.dev for Cosign, and each project's own documentation for Trivy, kube-bench, Cilium and Istio. The book teaches you to use those tools, but it does not copy their installation steps: each chapter gives you the link to the official page that explains how to install them, because those steps change faster than a printed edition. When a behavior depends on the version, the book says so and shows you how to check it.

All the lab code is published at https://github.com/lfarizav/lo-que-pasa-es-que-asegurar-kubernetes. The repository has two folders: `es/` for the Spanish edition and `en/` for the English edition. Each holds one lab per part, from `lab-part-0`

`1` to `lab-part-06`, with its commands, its checks and the versions it was tested with, ready to clone and run in a Linux terminal:

```
git clone https://github.com/lfarizav/lo-que-pasa-\
es-que-asegurar-kubernetes.git
cd lo-que-pasa-es-que-asegurar-kubernetes/en
```

Each lab brings up its own isolated kind cluster and writes its kubeconfig to a private file, without touching your usual kubeconfig or any other cluster you have. Never run one against a production cluster: several steps deliberately change the control plane configuration or load profiles into your machine's kernel. Each step gives you the goal and the deliverable, and the command sequence is kept apart so you can reveal it after trying on your own. Grading queries the live cluster and does not read your command history. They resemble what you will be asked to do, because the CKS is not multiple choice: according to its official page, it is a two-hour, hands-on exam on the command line. Even so, they are not a simulator and they do not reproduce exam questions.

Every chapter closes, right before its summary, with a **Practice in the lab** section that tells you which lab belongs to it and which topics you will work on there; the steps, commands and checks live only in the repository.

How to Contact Us

This book is part of Luis Felipe Ariza Vesga's catalog. If you find an example that does not behave the way the book says, a field renamed in a recent Kubernetes release, a Falco rule that no longer matches the published version, a Cosign command that changed, or a claim that does not match the current official documentation, that feedback is welcome and taken seriously. In security, an outdated claim is not a small detail: it is a documented vulnerability. Kubernetes and the tools around it ship new releases every few months, so no edition is the last word, and a correction made in time is worth more than a promise that none will be needed. This book would rather be corrected than pretty, just as a good security team would rather hear an alert on time, even a false one, than enjoy a quiet silence that only means nobody was watching.

Acknowledgments

This book invented nothing from scratch, and that is exactly what it wants to thank. Kubernetes and its security exist because thousands of people chose to build the same tools together, in the open, instead of each keeping their own.

To the Cloud Native Computing Foundation (CNCF) and the Linux Foundation, for giving Kubernetes and the projects around it a neutral home, for publishing the CKS curriculum in the open and for running the certification this book prepares you for.

To the communities and companies that donated whole projects to the CNCF. Falco, Cilium, Istio, Sigstore and Kubernetes itself reached every administrator in the world that way, without asking anyone for permission. Every chapter of this book rests on one of those donations.

To the maintainers of Kubernetes, etcd, Falco, Trivy, kube-bench, kubesec, KubeLinter, Sigstore, Cilium, Istio, gVisor and kind. Answering security reports at midnight, shipping a patch before a vulnerability makes the news and saying no to a badly thought-out feature is work almost nobody sees and everybody depends on.

To every contributor: to those who write code, but also to those who write and translate the documentation, report bugs, test beta releases and answer questions in the forums. The documentation this book cites in every chapter exists thanks to them, open and free for anyone.

To the local open source communities: to the members of Cloud Native Colombia, KCD Colombia (Kubernetes Community Days) and the Colombia OpenInfra User Group. They organize meetups, workshops and events to encourage the use of open source without charging for it, and they often pay out of their own pockets for the snacks, the transport or the venue so that each meeting can happen. Much of what this book tells was learned at those meetings.

And thanks, finally, to the warrah of the Falklands, whose story lends the cover its hardest lesson: an animal that never had a reason to be wary had no time to learn it on the day it finally needed to.

PART I

Cluster Setup (15%)

1

The Rule Book for Who Gets Into Which Tower

Why a cluster with no NetworkPolicy lets every tower talk to every other one, why it takes two policies and not one, and what the CIS Benchmark checks before you trust the result.

1.1 The Complex With No Visitor Policy

The Los Nogales complex just finished construction. Eight towers, a central park, a pool that still smells of fresh chlorine and a front gate with a single guard on the morning shift: Justo Segura, the same man who in book 2 looked after the estate and decided which traffic could pass from one house to another, now starting a new job at a residential complex. On the gate's counter, next to the still-unused visitor logbook, Justo set down from day one a small wooden figure: a skinny little animal like a small dog, short-legged, with a tail that ends in a white tip. Nobody asked him where he got that figure; one day, though, he said in a serious voice: remember that trust is not the same as security.

The trouble is that the owners' assembly has not met yet, no rules have been signed and nobody handed Justo a list of who lives in which tower or who may visit whom. So he does the only thing he can do without instructions: he lets everyone in. A visitor says he is going to tower 3 and walks in unchecked. Another says tower 7, and walks

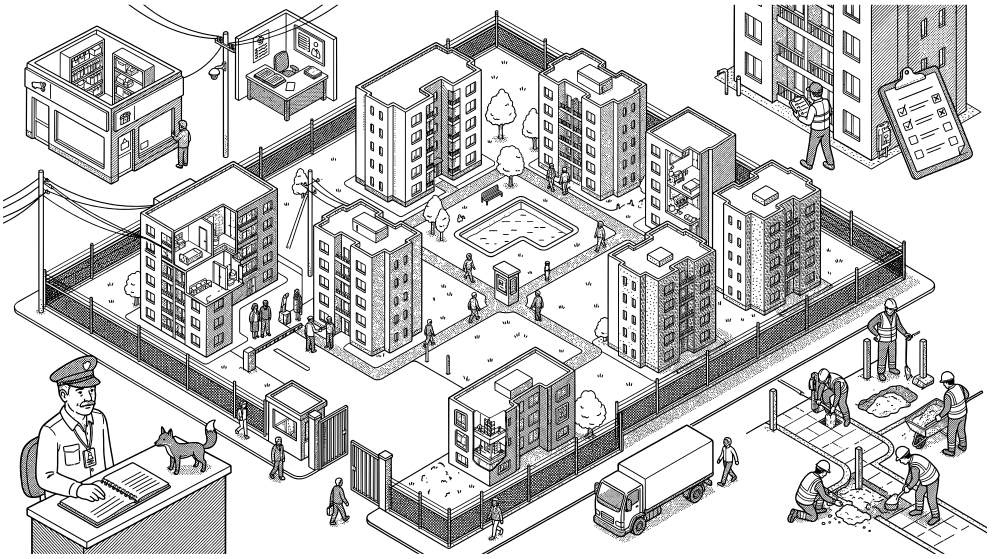


Figure 1.1. The whole Los Nogales complex, the map of this book: the front gate with Justo Segura and his figurine, the eight towers around the park, the electric fence, the corner shop, the management office and, at the bottom right, the workers setting the posts along the paths. Each chapter goes down into one part of this map.

in too. The gas truck drives straight into tower 1's parking lot, even though nobody in that tower ordered gas that day. The guard is not careless: without rules, he has no grounds to turn anyone away, and saying yes to everyone is, literally, the only thing left for him to do, apart from searching people on the way in and on the way out. Figure 1.1 shows the whole complex this book walks through chapter by chapter.

Here is what is actually going on: a freshly created Kubernetes cluster behaves the same way, and it is frightening once you compare it with the complex. The official page on Network Policies puts it plainly: by default, a pod is non-isolated for egress and for ingress, and all connections are allowed. A Pod in the billing namespace can talk to another team's database, to the admin dashboard or to any other Pod in the cluster, simply because nobody has written the rules yet. That is why the CKS curriculum opens its first domain, Cluster Setup, with this task: use network security policies to restrict cluster level access.

Put yourself for a moment in the shoes of someone who wants in. That person does

not need the key to the gate: controlling a single Pod is enough, for example by exploiting a vulnerability in an exposed web application. From there, in a cluster with no rules, they can walk the complex tower by tower, try which services answer and reach what they actually want, which is almost never the Pod they came in through. A network policy does not stop someone from entering that first apartment, but it does decide whether they can walk from there to the rest of the complex. In book 3, in the chapter about who a stall may talk to, you saw how a NetworkPolicy is written from the application's side; here you look at it from the side of whoever has to enforce it.

1.2 Two Rule Books, Not One

The rule book is written with an object Kubernetes calls a NetworkPolicy. It lives inside a namespace, just as one tower's rules only apply to that tower. With `podSelector` it picks which Pods it governs, and with `policyTypes` it declares whether it regulates incoming traffic (`Ingress`), outgoing traffic (`Egress`) or both. A policy only has to select a Pod for that Pod to become isolated in the directions the policy declares: from then on, only what some policy explicitly allows gets through. Figure 1.2 shows the gate before the rules, and Figure 1.3 the two policies between towers: the barrier for everyone and the lane for what is authorized.

Seeing the difference takes a test complex. The `torres.yaml` file creates two namespaces, `torre-3` and `torre-7`; tower 3 is home to `residente`, a web server with its Service, and tower 7 holds two visitors: `visita-autorizada`, labeled `rol: visita-autorizada`, and `desconocido`, with no label. Before any rules are written, the stranger walks right in:

```
kubectl -n torre-7 exec desconocido -- wget -qO- -T 3
 ↪  http://residente.torre-3 | grep -o '<title>.*</title>'
```

```
<title>Welcome to nginx!</title>
```

Any Pod from another tower read the resident's page and nobody asked it anything. That is the freshly created cluster the documentation talks about.

Here comes the most important point of the chapter: securing a tower takes **two** Net-



Figure 1.2. Justo Segura at the front gate, with the unused visitor logbook and the figurine of the thin animal with the white-tipped tail. With no rules, he lets everyone in: that is how a cluster with no NetworkPolicy behaves.

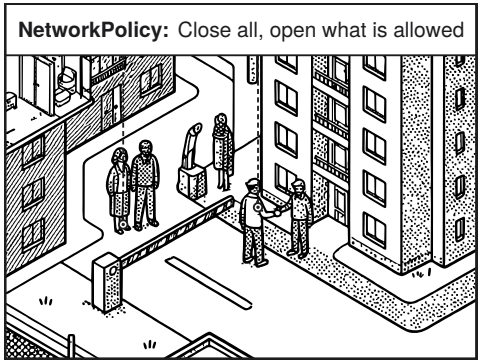


Figure 1.3. Between two towers, a barrier closed to everyone (the policy that denies everything) and, next to it, a guard letting through a visitor with an authorized pass (the policy that allows only legitimate traffic).

workPolicies, not one. The first one closes everything. It is the rule the documentation calls "default deny": an empty `podSelector`, which selects every Pod in the namespace, with `Ingress` and `Egress` in `policyTypes` and no rule that allows anything.

```yaml
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: negar-todo
  namespace: torre-3
spec:
  podSelector: {}
  policyTypes:
  - Ingress
  - Egress
```

The empty braces of `podSelector: {}` mean "every Pod in this namespace", and

the absence of `ingress` and `egress` blocks means "nothing allowed". Once it is applied, the tower is closed in both directions:

```
kubectl apply -f 01-negar-todo.yaml
kubectl -n torre-7 exec visita-autorizada -- wget -qO- -T
 ↪  3 http://residente.torre-3
kubectl -n torre-3 exec residente -- nslookup -timeout=3
 ↪  kubernetes.default.svc.cluster.local
```

```
networkpolicy.networking.k8s.io/negar-todo created
wget: download timed out
command terminated with exit code 1
;; connection timed out; no servers could be reached
command terminated with exit code 1
```

Nobody gets in anymore, not even the authorized visitor. And there is an effect that surprises people the first time: the resident itself lost DNS (Domain Name System), because asking CoreDNS for a name is outgoing traffic too. The documentation warns about it in these words: "A default deny-all egress policy also blocks DNS traffic. If your workloads need DNS resolution, you must add a separate NetworkPolicy that allows egress to your cluster's DNS service."

That is why the second policy exists. On top of the closed baseline, it opens only what is needed: egress to the CoreDNS Pods, which carry the label `k8s-app: kube-dns` in the `kube-system` namespace, on port 53 over UDP (User Datagram Protocol) and TCP (Transmission Control Protocol), and ingress on port 80 only for Pods labeled `rol: visita-autorizada` coming from tower 7.

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: permitir-lo-justo
  namespace: torre-3
spec:
  podSelector:
    matchLabels:
```

```
      app: residente
  policyTypes:
  - Ingress
  - Egress
  ingress:
  - from:
    - namespaceSelector:
        matchLabels:
          kubernetes.io/metadata.name: torre-7
      podSelector:
        matchLabels:
          rol: visita-autorizada
    ports:
    - protocol: TCP
      port: 80
  egress:
  - to:
    - namespaceSelector:
        matchLabels:
          kubernetes.io/metadata.name: kube-system
      podSelector:
        matchLabels:
          k8s-app: kube-dns
    ports:
    - protocol: UDP
      port: 53
    - protocol: TCP
      port: 53
```

Notice a detail of form: in `from` and in `to`, the `namespaceSelector` and the `podSelector` sit inside the same list element, with no dash between them. That means the source has to meet both conditions at once: be in tower 7 **and** carry the authorized-visitor label. With the second policy applied, each visitor gets the answer it deserves:

```
kubectl apply -f 02-permitir-lo-justo.yaml
kubectl -n torre-3 get networkpolicy
```

```
kubectl -n torre-7 exec visita-autorizada -- wget -qO- -T
 ↪  3 http://residente.torre-3 | grep -o
 ↪  '<title>.*</title>'
kubectl -n torre-7 exec desconocido -- wget -qO- -T 3
 ↪  http://residente.torre-3
kubectl -n torre-3 exec residente -- nslookup
 ↪  kubernetes.default.svc.cluster.local
```

```
networkpolicy.networking.k8s.io/permitir-lo-justo created
NAME                POD-SELECTOR    AGE
negar-todo          <none>          9s
permitir-lo-justo   app=residente   0s
<title>Welcome to nginx!</title>
wget: download timed out
command terminated with exit code 1
Server:     10.96.0.10
Address:    10.96.0.10:53
Name:   kubernetes.default.svc.cluster.local
Address: 10.96.0.1
```

The authorized visitor gets in, the stranger stays at the door and the resident resolves names again through the cluster DNS service, which answers from 10.96.0.10. The two policies live together without fighting, and the documentation gives the reason: "Network policies do not conflict; they are additive. If any policy or policies apply to a given pod for a given direction, the connections allowed in that direction from that pod is the union of what the applicable policies allow." Every policy that selects the Pod adds permissions and none takes them away; the order in which they were created does not change the result. That is why the pattern is always the same: one policy that denies everything and, on top of it, the ones that open just what is needed.

Warning
One extra dash changes what the policy means. If `namespaceSelector` and `podSelector` sit in the same `from` element, the source must meet both conditions. If each one starts with its own dash, they are two elements and meeting either one is enough: any Pod from tower 7 would get in, or any Pod with the label in the tower itself. The documentation recommends checking with `kubectl describe networkpolicy` how Kubernetes interpreted the policy before trusting it.

Caution
To select a namespace, use the `kubernetes.io/metadata.name` label. According to the documentation, the control plane sets it on every namespace, with the name as its value, and it cannot be changed. A made-up label, such as `torre: siete`, can be copied by anyone allowed to label namespaces, and their namespace would then meet the ingress rule. It is the Kubernetes version of putting on someone else's uniform to get waved through.

1.3 The Condition That Decides Whether the Rules Hold

There is one condition neither policy mentions, and it is the one that decides whether the rules hold or stay pinned to the notice board with nobody reading them: the cluster needs someone to enforce them. Kubernetes stores the NetworkPolicy, but it does not apply it by itself. The documentation leaves no doubt: "Network policies are implemented by the network plugin. To use network policies, you must be using a networking solution which supports NetworkPolicy. Creating a NetworkPolicy resource without a controller that implements it will have no effect." Figure 1.4 shows it in the complex: without posts, the barrier is only a painted line.

At the complex, that is like printing the rules, signing them at the assembly and pinning them on the gate's notice board while the guard on duty cannot read. The paper exists, `kubectl get networkpolicy` shows it with its name and age, and still no packet is blocked. That is why the examples above count as proof: the stranger's `download timed out` shows that this cluster does have a guard. This book's cluster runs on kind, whose network plugin, kindnet, enforces NetworkPolicy since kind 0.24.0, according to its release notes. On a production cluster the CNI (Con-

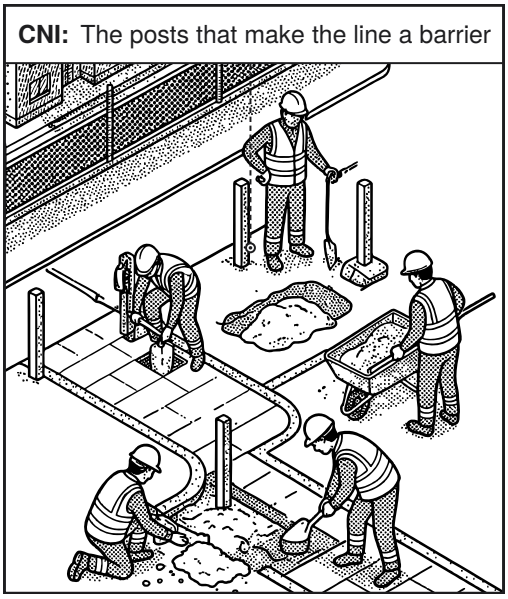


Figure 1.4. Workers setting posts along the paths. Without them, the barrier would be a painted line: the network plugin (CNI) is what enforces NetworkPolicies.

tainer Network Interface) may be Calico, Cilium or another one, and the question is answered the same way: with a test, not with faith that the applied object is working.

That test is also the first habit of a good security administrator, who never assumes something works because someone said it does. The attacker does not assume it either: the first thing they will do from the Pod they control is try to connect to other services, and all that matters is what the cluster answers.

> **Tip**
> After applying the two policies, test them before trusting them. From a Pod that should not get in, `wget -qO- -T 3 http://<service>.<namespace>` must end in `download timed out`. If the page shows up instead, that cluster's CNI is not enforcing NetworkPolicy and the rules exist only on paper. kube-bench control 5.3.1 asks the same question and leaves it as manual, because it has no way to answer it for you.

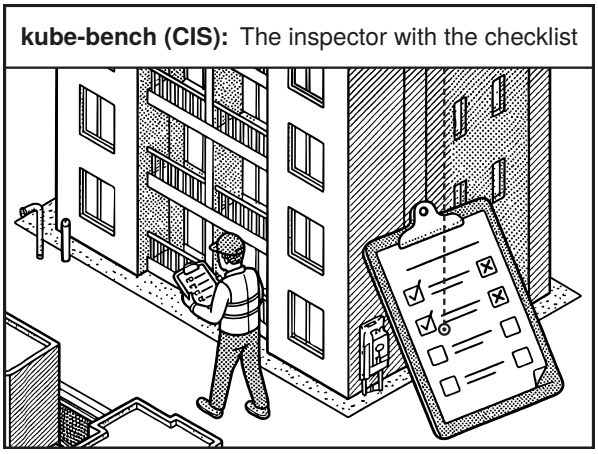


Figure 1.5. An inspector walks around a tower with a checklist and marks each item: the kube-bench review against the CIS Kubernetes Benchmark.

1.4 The Inspection Before the Shift

Even with rules and a guard, one practical problem remains: how does the management office know every tower has its rules posted, and not just tower 3 while the other seven stay open? A new namespace is born with no NetworkPolicy at all, just as a new tower is handed over with no rules until someone writes them. The first inspection is as simple as `kubectl get networkpolicy --all-namespaces`, compared against the list of namespaces that should be isolated, and it is repeated before every shift, because namespaces get created every day. Figure 1.5 shows that round.

The second inspection uses an outside standard: the CIS Kubernetes Benchmark, published by the Center for Internet Security, which is the list of secure configurations a cluster gets audited against. One tool that applies it is kube-bench, and its installation page explains the ways to get it running. The most convenient one is described in its running guide: a Kubernetes Job that mounts, read-only, the node folders it inspects. The `kube-bench-politicas.yaml` file is the project's `job.yaml`, version v0.16.0, with a single change: the arguments `run --targets policies`, so it only checks the policies section.

```
kubectl apply -f kube-bench-politicas.yaml
kubectl wait --for=condition=complete
 ↪  job/kube-bench-politicas --timeout=120s
kubectl logs job/kube-bench-politicas | grep -E
 ↪  '^\[(PASS|FAIL|WARN)\] 5\.3'
```

```
job.batch/kube-bench-politicas created
job.batch/kube-bench-politicas condition met
[WARN] 5.3.1 Ensure that the CNI in use supports
 ↪  NetworkPolicies (Manual)
[WARN] 5.3.2 Ensure that all Namespaces have
 ↪  NetworkPolicies defined (Manual)
```

The two controls in section 5.3 ask for exactly what this chapter did: 5.3.1, that the CNI enforce NetworkPolicy, and 5.3.2, that every namespace have its policies. And both come out as warnings, marked manual. kube-bench can read the components' files and arguments, but it cannot check for you whether a packet gets blocked or whether each tower's rules say anything sensible. An automated inspector that only counts papers taped to the door does not replace the guard who reads the rules line by line, and much of the next chapter is about that difference.

> **Note**
> In the same report, kube-bench closed the policies section with 0 checks passed, 0 failed and 34 warnings. That does not mean the cluster is right or wrong on those points: every control in that section is manual. A report with no red in that part proves nothing; it only tells you what you have to check yourself.

1.5 Practice in the lab

Topics from lab `lab-part-01` (Part I) for this chapter:

- Measure that, with no policies, a Pod without permission reaches the tower's service.
- Write the policy that denies all incoming and outgoing traffic in the namespace.

- Open only egress to the cluster DNS and ingress from the authorized neighbor, by label.
- Test the rules from the authorized neighbor and from the stranger, by name and by IP.
- Run kube-bench as a Job, tell the FAIL lines from the WARN lines, fix one control in the static manifest and run it again.

1.6 Summary

Back to Los Nogales on the day the rules finally arrive. Justo Segura, who used to let anyone in because he had no grounds for a no, now has clear instructions, and they apply in this order, the same path as Figure 1.6:

1. By default, nobody gets into or out of the tower: a NetworkPolicy with `podSelector: {}` and `Ingress` and `Egress` in `policyTypes`, with no rules.
2. On top of that closed baseline, a second policy opens just what is needed: egress to the cluster DNS on port 53 and ingress only for authorized visitors, by namespace and by label.
3. Policies add up and never subtract, so the order in which they are applied does not change the result.
4. All of that depends on the CNI enforcing NetworkPolicy, and the only way to know is to test it from a Pod that should not get in.
5. kube-bench checks the cluster against the CIS Benchmark, but it marks precisely the two network controls as manual: the last word belongs to a test you run yourself.

With traffic between towers settled, another question remains, the one asked by the visitor who arrives from the street and not from another tower: does the front gate ask for an ID before letting them in, or is saying a name enough? That is what the next chapter is about.

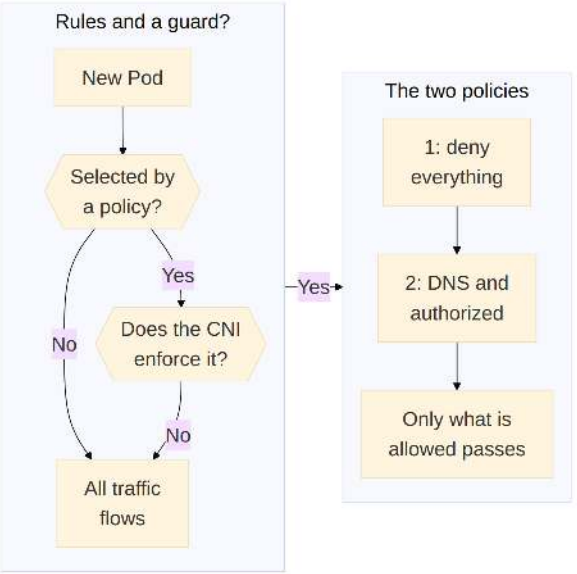


Figure 1.6. The complex's rules: with no policies, or with no CNI to enforce them, everything flows in and out; with one policy that denies everything and another that opens just what is needed, only what is allowed gets through.

Bibliography

- Kubernetes documentation, Network Policies https://kubernetes.io/docs/concepts/services-networking/network-policies/
- Aqua Security, kube-bench documentation, Running kube-bench https://github.com/aquasecurity/kube-bench/blob/main/docs/running.md
- CNCF, CKS Exam Curriculum v1.34 (Cluster Setup domain) https://github.com/cncf/curriculum
- kind v0.24.0 release notes (kindnet NetworkPolicy support) https://github.com/kubernetes-sigs/kind/releases/tag/v0.24.0