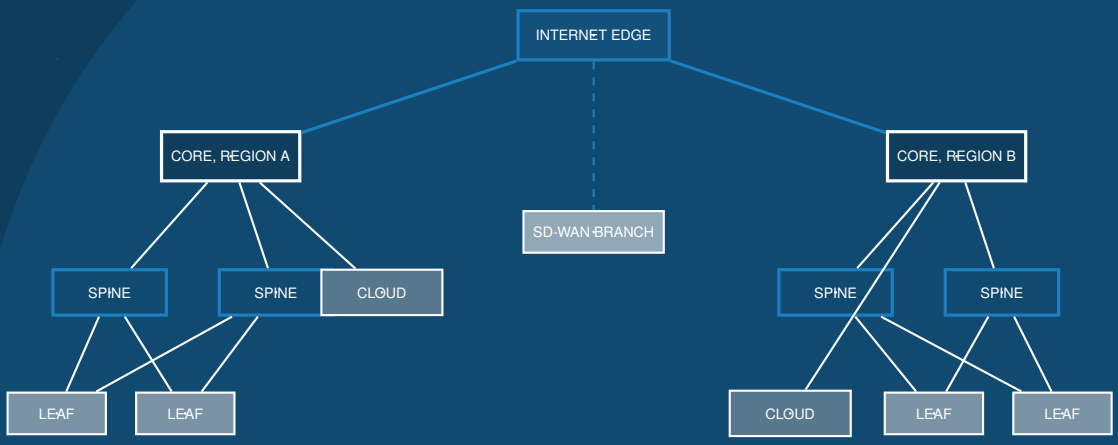
CODELIVLY

The Codelivly Network Engineer Series

LEVEL 3

NETWORK ENGINEER: L3

Advanced Enterprise Architecture, BGP, Data Center Networking, Cloud, Automation, Security and Network Engineering



BGP & ARCHITECTURE — DATA CENTER — CLOUD — SECURITY — AUTOMATION

Written by Rocky

Published by Codelivly

codelivly.com

Copyright

Network Engineer: L3 Advanced Enterprise Architecture, BGP, Data Center Networking, Cloud, Automation, Security and Network Engineering

Copyright 2026 Codelivly. All rights reserved.

Written by Rocky. Published by Codelivly, codelivly.com.

First edition, 2026.

No part of this publication may be reproduced, distributed, or transmitted in any form without the prior written permission of the publisher, except for brief quotations used in reviews and other noncommercial uses permitted by copyright law.

Disclaimer. This book is provided for educational purposes. Networking technologies, platforms, and vendor behavior change over time, so always verify configuration steps and architectural guidance against current vendor documentation before applying them in a production environment. Command examples are illustrative and labeled by platform: Linux, Windows, Cisco style command line interface, Python, Ansible, YAML, JSON, or REST API. Where output is illustrative rather than captured from a live system, it is explicitly labeled as illustrative example output.

Trademarks. This book is an independent publication and is not affiliated with, endorsed by, or sponsored by Cisco, Juniper, Arista, Amazon Web Services, Microsoft, Google, Red Hat, or any other vendor or certification body mentioned in the text. All trademarks are the property of their respective owners. Where behavior differs between vendors, the text says so explicitly rather than presenting one vendor's implementation as a universal standard.

Contact. For corrections and practical resources related to this book, visit codelivly.com.

About Codelivly

Codelivly is a technical learning platform focused on practical cybersecurity, networking, and IT education. Codelivly material is built around how skills are actually used on the job: real architecture decisions, real design tradeoffs, and exercises readers can reproduce themselves, rather than theory presented in isolation.

This book is the third and most advanced title in the Codelivly Network Engineer series, following Network Engineer L1 and Network Engineer L2. You can find more guides and learning resources at codelivly.com.

About the Author

Rocky is a technical writer and instructor at Codelivly focused on networking and cybersecurity. His approach centers on engineering reasoning: understanding the tradeoffs behind a design decision, not just the mechanics of a protocol, and reinforcing that understanding with realistic scenarios, labs, and case studies. This book applies that approach at the level of enterprise architecture.

How to Use This Book

How L3 differs from L1 and L2. L1 built your foundation. L2 took you into enterprise switching, routing, and operations. L3 assumes both and shifts the entire lens: instead of asking "how do I configure this," you are asked "why would I design it this way, and what happens when it fails." Expect fewer step-by-step configuration walkthroughs and more architecture reasoning, tradeoffs, and case studies.

Recommended study approach. Read the parts in order on a first pass. Part I establishes the senior engineering mindset that the rest of the book depends on; skipping it makes later architecture chapters feel like a list of facts rather than a way of thinking.

Using architecture diagrams. Every diagram in this book is numbered and captioned, and is meant to be studied, not glanced at. Before reading a diagram's caption, try to explain to yourself what it shows and why it is drawn that way.

Performing labs. Part XIX contains twenty five advanced labs. Build them in a free simulator or virtualization platform as you reach the matching chapter. The goal is not memorizing commands, it is seeing the architectural concept behave the way this book describes.

Approaching design scenarios and case studies. Part XVIII's ten case studies and the capstone in Part XXIV are meant to be worked, not read. Attempt your own design before reading the reference approach, then compare your reasoning, not just your final diagram, against it.

Using the troubleshooting scenarios. Part XX's seventy five scenarios follow evidence, reasoning, and root cause in that order. Read only the incident and symptoms, decide what you would check first, then compare.

Using the interview and assessment sections. Revisit these after finishing the relevant part, then again after finishing the book, as spaced review rather than a single read through.

Completing the capstone. The capstone in Part XXIV is the single best evidence of what you have learned. Treat it as a real design review: justify every decision, not just describe it.

Prerequisites

Before starting this book, you should already be comfortable with the material covered in Network Engineer L1 and Network Engineer L2, specifically:

IPv4 and IPv6 addressing and subnetting. VLANs, switching, and spanning tree. Static and OSPF routing fundamentals, and a basic understanding of BGP. Access control lists, firewalls, and VPN concepts. Network monitoring fundamentals and basic packet analysis with Wireshark. Basic Python and basic network automation concepts.

If any of these feel unfamiliar, working through Network Engineer L1 and L2 first will make this book significantly more useful. This book does not reteach these fundamentals, it builds directly on top of them, and moves quickly into architecture, scale, and engineering tradeoffs.

Network Architecture at Scale

This part establishes the mindset the rest of this book depends on. Chapter 1 covers how senior engineers think differently from configuration focused engineers. Chapter 2 surveys the architecture patterns you will choose between throughout the book. Chapter 3 covers designing deliberately for scale, using failure domains and blast radius as the organizing concepts.

1 Thinking Like a Senior Network Engineer

The difference between an engineer who configures networks and one who engineers them is not years of experience alone, it is a shift in what questions get asked before a change is made.

What You Will Learn

You will learn the practical difference between configuration work and engineering work. You will learn how senior engineers reason about tradeoffs, risk, reliability, and operational simplicity. You will learn why technical debt is a deliberate, tracked decision rather than an accident. You will learn how documentation, change management, and incident ownership fit into engineering judgment rather than being administrative afterthoughts.

1.1 Configuration Versus Engineering

Configuring a network means correctly applying a known solution to a known problem: adding a VLAN, writing an access list, bringing up a BGP neighbor. Engineering a network means deciding what the right solution is in the first place, under real constraints: budget, existing infrastructure, team skill, growth projections, and risk tolerance. An L1 or L2 engineer is usually handed the design and executes it correctly. An L3 engineer is expected to produce the design, defend it, and live with its consequences years later.

Architecture Note

A useful test: if a task can be fully specified as "do X to device Y," it is configuration work. If it requires answering "what should we do, and why," it is engineering work. Most senior engineering interviews probe for the second skill specifically, because the first is comparatively easy to verify and comparatively easy to automate.

1.2 Engineering Tradeoffs and Risk Management

Every meaningful design decision trades one property for another: redundancy against cost, simplicity against flexibility, centralization against blast radius. Senior engineers make these tradeoffs explicit rather than pretending an optimal answer exists that satisfies every property at once. **Risk management** means identifying what could go wrong with a decision before making it, sizing how bad that outcome would be, and deciding whether the mitigation is worth its own cost.

Property pair	The tension
Redundancy vs cost	More redundancy protects availability but consumes budget that could go elsewhere
Simplicity vs flexibility	A simple design is easy to operate but may not accommodate every future requirement
Centralization vs blast radius	Centralizing control simplifies operations but concentrates the impact of a failure
Automation vs manual control	Automation scales consistent changes but a flawed automated change scales just as fast

1.3 Reliability, Scalability, and Operational Simplicity

Reliability is the property that a system behaves correctly and predictably over time, including during partial failure. **Scalability** is the property that a design can grow, in users, sites, or traffic, without a fundamental redesign. **Operational simplicity** is how easily the team that has to run the network day to day can understand, change, and troubleshoot it. A design can be reliable and scalable on paper and still fail in practice if it is too complex for its own operators to run safely.

Engineering Decision

Given a choice between a marginally more efficient design that only one engineer on the team fully understands, and a slightly less efficient design the whole team can operate confidently, the second is usually correct. A design's real reliability includes the reliability of the humans operating it.

1.4 Technical Debt as a Deliberate Decision

Technical debt is any gap between the ideal design and what is actually running, taken on deliberately to meet a deadline, cost constraint, or urgent need. The senior engineering discipline is not avoiding technical debt entirely, that is unrealistic, it is tracking it explicitly: what shortcut was taken, why, and what condition would trigger paying it down. Undocumented, untracked technical debt is what turns into a mystery outage two years later.

1.5 Documentation, Change Management, and Incident Ownership

Documentation, change management discipline, and taking ownership of incidents through to a real root cause are not administrative overhead layered on top of engineering, they are part of the engineering work itself. A design that is not documented cannot be safely operated by anyone other than its author. A change made without a plan and a rollback path is a gamble, not an engineering action. An incident that is closed without understanding its root cause will recur.

Copyright 2026 Codelivly • Written by Rocky

Remember

A brilliant design nobody can safely operate, change, or recover is not actually a good design. Operability is a design requirement, not an inconvenience.

1.6 Chapter Summary

Senior network engineering is distinguished by the questions asked before a change: what problem is being solved, what tradeoffs are being made, what can fail, and how bad would that failure be. Reliability, scalability, and operational simplicity are properties that must be balanced against each other, not maximized independently. Technical debt, documentation, change management, and incident ownership are core engineering responsibilities, not administrative afterthoughts.

Architecture Takeaways. Configuration executes a decision; engineering makes the decision and owns its consequences. Every design choice is a tradeoff, made explicit. Operability is part of the design, not a downstream concern.

Key Terms. Configuration work, engineering work, risk management, reliability, scalability, operational simplicity, technical debt.

Quick Check

1. Explain the practical difference between configuration work and engineering work with an example.
2. Why is untracked technical debt more dangerous than tracked technical debt.
3. Give an example of a tradeoff between two of the properties covered in this chapter.

Practical Exercise

Describe a network change you have made or observed. Identify what tradeoff was made, whether it was made explicitly, and what technical debt, if any, it introduced.

Interview Question

Describe a situation where you would deliberately choose a less efficient design because of operational simplicity. What makes that the right call.

Further Practice. Review a design decision from a previous role or project and write one paragraph identifying the tradeoff it represents, using the vocabulary from this chapter.

Copyright 2026 Codelivly • Written by Rocky

2 Enterprise Architecture Patterns

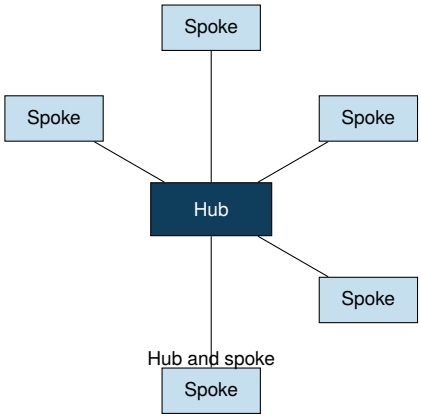
Every enterprise network is built from a small set of recurring architectural patterns. Recognizing which pattern is in play, and why, is faster and more reliable than reasoning about a network from scratch every time.

What You Will Learn

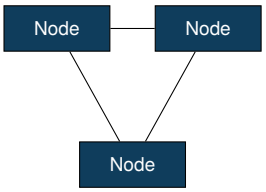
You will review the three tier hierarchical model and collapsed core from L2, then extend into spine-leaf, hub-and-spoke, partial mesh, and full mesh patterns. You will learn hierarchical and modular design as organizing principles. You will learn how data center, campus, branch, and internet edge architectures differ, and when each pattern applies.

2.1 The Pattern Library

Pattern	Optimized for	Typical use
Three tier hierarchical	North to south campus traffic	Large campus networks
Collapsed core	North to south, smaller scale	Single building or small campus
Spine-leaf	East to west data center traffic	Data centers
Hub-and-spoke	Centralized policy and simplicity	Branch networks, SD-WAN
Partial mesh	Redundancy without full mesh cost	Core and distribution links
Full mesh	Maximum redundancy	Small, highly critical cores



Hub and spoke



Full mesh

Figure 2.1. Hub and spoke centralizes policy and simplifies the spokes at the cost of routing everything through the hub. Full mesh maximizes redundancy at a cost that grows as the square of the node count.

2.2 Hierarchical and Modular Design as Principles

Every pattern above is an instance of two deeper principles. **Hierarchical design** separates a network into layers with narrow responsibilities, access, distribution, core, so that a change or failure in one layer has a bounded effect elsewhere. **Modular design** builds the network from repeatable, interchangeable blocks, a distribution block, a data center pod, a branch template, so that adding capacity means replicating a proven block rather than inventing a new design each time.

Architecture Note

When evaluating an unfamiliar network, identify its hierarchy and its modules before evaluating anything else. A network without clear hierarchy or repeatable modules is usually harder to operate and scale regardless of which specific technologies it uses.

2.3 Data Center, Campus, Branch, and Internet Edge

These four environments are architecturally different because their traffic patterns and risk profiles differ. **Data center** architecture optimizes for east to west, server to server traffic and workload mobility. **Campus** architecture optimizes for north to south user traffic and physical scale across buildings. **Branch** architecture optimizes for cost and simplicity at a large number of small, similar sites, favoring hub-and-spoke and centralized policy. **Internet edge** architecture optimizes for controlled, secure, and resilient connectivity to the outside world, and is where firewalls, NAT, and inter-domain routing concentrate.

Environment	Dominant traffic	Typical pattern
Data center	East to west	Spine-leaf
Campus	North to south	Three tier or collapsed core
Branch	North to south, small scale	Hub and spoke, SD-WAN
Internet edge	Inbound and outbound to the internet	Dual redundant edge routers and firewalls

2.4 Chapter Summary

Enterprise architecture is built from a small library of recurring patterns: three tier hierarchical, collapsed core, spine-leaf, hub-and-spoke, partial mesh, and full mesh, each suited to a specific traffic pattern and scale. Hierarchical and modular design are the deeper principles behind all of them. Data center, campus, branch, and internet edge environments each favor different patterns because their traffic and risk profiles genuinely differ.

Copyright 2026 Codelivly • Written by Rocky

Architecture Takeaways. Recognize the pattern before analyzing the network. Hierarchy bounds the effect of failure; modularity makes growth repeatable. Match the pattern to the traffic, not the other way around.

Key Terms. Hierarchical design, modular design, hub and spoke, partial mesh, full mesh.

Quick Check

1. Why does full mesh become impractical as node count grows.
2. What traffic pattern does spine-leaf optimize for, and why does that differ from campus networks.
3. Why is hub-and-spoke a common choice for branch and SD-WAN architectures.

Interview Question

Given a network you have never seen before, describe the first two things you would look for to understand its architecture.

Copyright 2026 Codelivly • Written by Rocky

3 Designing for Scale

Scale is not just "more of the same." A design that works cleanly for ten sites can fail in ways that were invisible at that size once it reaches one hundred.

What You Will Learn

You will learn failure domains and blast radius as the primary tools for reasoning about scale. You will learn how capacity planning, addressing strategy, and routing domain boundaries must be decided deliberately as a network grows. You will work through architecture case studies applying these concepts.

3.1 Failure Domains and Blast Radius

A **failure domain** is the boundary within which a given failure is contained. **Blast radius** is the actual scope of impact, how many users, sites, or services, when something inside that domain fails. Scaling a network safely means shrinking blast radius as the network grows, even as the total number of things that could fail increases. A network where every failure has the same blast radius as it did at one tenth the size has not actually been engineered for scale, it has just gotten bigger.

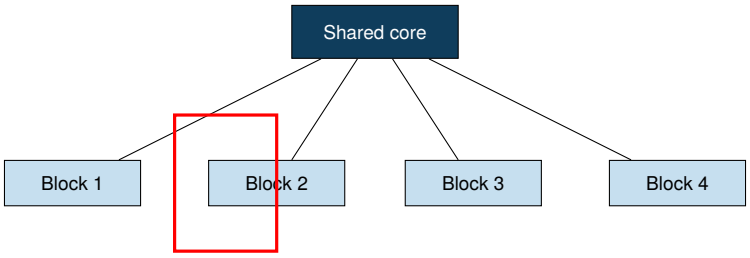


Figure 3.1. A failure inside Block 2, outlined in red, stays contained to that block. The other three blocks and the shared core are unaffected, an example of blast radius engineered to stay small as blocks are added.

3.2 Capacity Planning at Scale

At small scale, capacity planning is informal: someone notices a link is busy and upgrades it. At large scale, capacity planning must be systematic, using utilization trends across the whole estate to forecast where upgrades are needed months in advance, since lead times for circuits and hardware do not shrink just because the network grew. Under provisioning at scale does not fail gracefully, it fails as a widespread, hard to diagnose performance problem across many sites at once.

3.3 Addressing Strategy and Routing Domains

Addressing strategy, covered as hierarchical addressing in L2, becomes existential at scale: a network with hundreds of sites and no coherent addressing plan cannot be summarized, cannot be firewalled sanely, and cannot be reasoned about during an incident. **Routing domains** must be deliberately bounded, deciding where one OSPF area or BGP autonomous system ends and another begins, so that a routing event in one domain does not require every router in the company to recompute anything.

Engineering Decision

Splitting a single flat routing domain into several deliberately bounded domains costs upfront design effort and some ongoing complexity at the boundaries. It buys contained convergence and a smaller blast radius for routing events. At scale, this tradeoff almost always favors bounded domains.

3.4 Administrative Boundaries and Operational Scalability

Administrative boundaries define who owns and can change a given part of the network, a distinction that matters increasingly as organizations grow multiple network teams, business units, or acquired companies with their own infrastructure. **Operational scalability** asks whether the processes and tooling used to run the network, not just the network itself, will hold up as the network grows: can the team make one hundred sites' worth of changes as reliably as they make one site's worth.

3.5 Chapter Summary

Scale changes the character of network problems, not just their size. Failure domains and blast radius are the central tools for reasoning about safety at scale. Capacity planning, addressing strategy, and routing domain boundaries must all become deliberate and systematic rather than informal. Administrative boundaries and operational scalability, the ability of people and process to keep up, matter as much as the technical design itself.

Architecture Takeaways. Scale should shrink blast radius, not just grow the network. Addressing and routing domain boundaries are scale decisions, made deliberately and early. Operational scalability, the team's ability to run it, is as real a constraint as any technical one.

Key Terms. Failure domain, blast radius, capacity planning, routing domain, administrative boundary.

Quick Check

1. Explain the difference between a failure domain and blast radius.
2. Why does capacity planning need to become systematic rather than informal at scale.
3. Why do routing domain boundaries matter more as a network grows.

Copyright 2026 Codelivly • Written by Rocky