



GHOST
PROTOCOL

CL

THE FIELD LIBRARY

DEEP TRACK

SOMEONE ELSE'S COMPUTER

Cloud and Containers

The cloud model, IAM, storage and secrets,
serverless and the metadata service,
containers and Kubernetes.



Ghost Protocol (Pvt) Ltd · Colombo · ghosts.lk

Security internship training · The Field Library · Edition 1 · 2026

THE GHOST PROTOCOL FIELD LIBRARY · DEEP TRACK

Cloud and Containers

The cloud model, IAM, storage and secrets,
serverless and the metadata service, containers
and Kubernetes.

Ghost Protocol (Pvt) Ltd · Colombo

The Field Library, Edition 1 · 2026

Copyright 2026 Ghost Protocol (Private) Limited. All rights reserved.

Sold as a PDF for the buyer's own reading. Please do not share or resell the file.

The library

The Field Library is a whole shelf, not one book. The Core Path is the spine everyone reads. The rest is depth and breadth, taken on as the work calls for it. The shelf is complete now, and you are reading the one marked here.

A · THE CORE PATH

O The Oath BUILT

I Groundwork BUILT

II The Web BUILT

III The Wider Field BUILT

IV The Work BUILT

V The Craft BUILT

B · DEEP TRACKS

AI and Agents BUILT

Reading Code BUILT

Internal and Active Directory BUILT

The Web, Deeper BUILT

Cloud and Containers YOU ARE HERE

Mobile and APIs BUILT

Reverse Engineering BUILT

Hardware and the Edges BUILT

C · THE OPERATIONS

Red and Purple Team BUILT

OSINT and Recon BUILT

Report Writing Mastery BUILT

The Consulting Craft BUILT

Bug Bounty BUILT

D · THE OTHER SIDE

Detection and Response BUILT

Threat Hunting and Forensics BUILT

Hardening and Architecture BUILT

E · THE GROUND RULES

The Law, Deeper BUILT

Standards and Compliance BUILT

Ethics and Disclosure BUILT

F · COMPANIONS

The Reference Pack BUILT

The Lab Workbook BUILT

Case Files BUILT

The Bestiary BUILT

The Glossary and the Ledger BUILT

Contents

- Opening 5
- 1** The Cloud Model 6
- 2** Identity and IAM 11
- 3** Storage, Secrets and Misconfiguration 17
- 4** Serverless and the Metadata Service 22
- 5** Containers and Kubernetes 28
- Worked answers 35

Someone else's computer



The client had said it on the call, the way clients do: we moved to the cloud, so the security is handled. The scope was an account on a provider, a handful of buckets, some functions, a cluster. Blink repeated the line back, half believing it.

BLINK "It is handled though. The provider runs it."

SOL "The provider secures the floor and the walls. What you put in the room is yours. The cloud is someone else's computer, and the lock on your part of it is a configuration you can leave open by accident: a role with too much reach, one bucket set to public, one function that trusts the address next door. The provider will not save a client from their own settings."

Wyrm did not open its eyes. It had already read the account's policy, and one line in it had a star where a name should be.

WYRM "a star. no."

That is the book. Not breaking the provider, which you will not do and are not paid to try, but reading the tenant: identity, storage, the metadata service, the containers, and every place a default was left exactly as it shipped.

CHAPTER ONE

The Cloud Model

Before any attack, learn where the security line falls. The provider owns one side, the customer owns the other, and the line moves depending on how much of the stack the customer rents. Get this right and every later chapter has a place to stand.

Volume III taught cloud at working awareness: the vulnerability is usually a setting, you read the account rather than break it, and most incidents land on the customer's side of a shared line. This book is where you go deeper, into identity, serverless, and containers, at a depth a professional actually tests at. We start with the model itself, because the single most common reason an intern gets lost in a cloud engagement is that they never worked out who owns what. Once the ownership line is clear, the attack surface stops being a fog and becomes a short list.

Three ways to rent a computer

Cloud is renting someone else's computers and services instead of running your own. How much you rent decides how much you still have to secure. There are three broad shapes, and the names are worth knowing exactly because the shared-responsibility line is drawn against them.

Infrastructure as a service (IaaS)

You rent raw compute, storage, and network, and you run everything above it. A virtual machine (Amazon EC2, Azure Virtual Machines, Google Compute Engine) is the classic example. The provider runs the datacentre, the hardware, and the hypervisor. You own the operating system, its patches, the firewall rules, the software you install, and of course your data and who can reach it. This is the most control and the most to get wrong.

Platform as a service (PaaS)

You rent a managed runtime or a managed data store and hand your code or your data to it. A managed database (Amazon RDS, Azure SQL Database, Google Cloud SQL), a managed app runtime (AWS App Runner, Azure App Service, Google App Engine), a serverless function (AWS Lambda, Azure Functions, Google Cloud Functions). The provider takes the operating system and the patching off your plate. You still own your data, your access rules, your application code, and whatever secrets that code carries.

Software as a service (SaaS)

You rent a finished application and only ever configure it. Microsoft 365, Google Workspace, Salesforce, a hosted ticketing tool. The provider owns nearly the whole stack. Your job shrinks to who you grant accounts to, what those accounts can do, how you connect other systems to it, and the data you put in. The surface is small, but it is never zero, and it is always the same small thing.

The shared-responsibility line

Every major provider publishes a **shared-responsibility model**, and they all say the same thing in slightly different words. The provider is responsible for the security of the cloud: the physical sites, the hardware, the hypervisor, the internals of the managed services they sell. The customer is responsible for security in the cloud: their data, their identities and access, their network configuration, their application. The provider secures the thing you rent. You secure the way you use it.

The important part is that the line is not in a fixed place. It slides with the service model. The more of the stack you rent, the more the provider secures, and the less remains yours to misconfigure.

LAYER	ON-PREM (FOR CONTRAST)	IAAS	PAAS	SAAS
Physical, hardware, hypervisor	Customer	Provider	Provider	Provider
Operating system, patching	Customer	Customer	Provider	Provider
Runtime, application code	Customer	Customer	Customer	Provider
Network and firewall config	Customer	Customer	Shared	Provider
Identity and access (who can do what)	Customer	Customer	Customer	Customer
Data (classification, exposure, keys)	Customer	Customer	Customer	Customer

Who secures what, by service model. "Customer" is where your findings live.

Read the last two rows across every column. No matter how much of the stack the provider takes, two things never leave the customer: **identity** (who is allowed to do what) and **data** (what you store, how it is exposed, and who holds the keys). A provider will never configure your permissions for you or decide which bucket is public. That is why those two are the spine of this whole book. Identity is Chapter Two. Data, storage, and the misconfigurations around them are Chapter Three. Everything else is a variation.

■ NOTE

A tempting misreading of shared responsibility is "the provider handles security". They handle their half, and their half is genuinely excellent, which is exactly why attackers stopped trying to break the hypervisor and started reading customer configurations instead. The provider securing the cloud does not secure your cloud. The large majority of reported cloud incidents are a customer-side setting, not a provider breach.

The blast-radius container: account, subscription, project

Each provider gives you a top-level boundary that holds your resources and your identities. Knowing its name matters, because it is the unit a scope is written against and the unit a compromise can spread across.

- **AWS:** the **account**, with many accounts grouped under AWS Organizations. Identity is AWS IAM.

- **Azure:** resources live in **resource groups** inside a **subscription**, and identities live in a **Microsoft Entra ID** tenant (the service formerly called Azure Active Directory). A tenant can own several subscriptions.
- **Google Cloud:** resources live in a **project**, and projects nest under folders and an organisation. Identity is Google Cloud IAM.

The practical consequence: a single broken identity policy can reach every resource in that container, not one application. The blast radius of a cloud mistake is the account, not the box. This is why identity is the perimeter and why Chapter Two is the longest.

What cloud testing actually is

Because the failures moved into configuration, cloud testing is mostly a **configuration review**, and it is almost always authenticated and read-only. The client gives you a set of read-only credentials (a reviewer role) and you walk the account's own settings with the provider's tooling and open auditors, looking for the permission granted too widely, the storage left reachable, the key committed by mistake, the log never switched on. You rarely "break in". You read the account and report what is actually exposed. That does not make it less valuable. A clear, true account of what a single leaked key could reach is worth more to a client than any exploit demo.

There is a second mode, the external or assumed-breach view, where you start with no credentials, or with one low-value foothold, and see how far it reaches. That is where the chains in this book live: a web Server-Side Request Forgery that reaches the metadata service (Chapter Four), a function whose role is too strong (Chapter Four), a pod that can read every secret in the cluster (Chapter Five). Both modes are legitimate. Both need the same thing first: written authorisation, and certainty about which account you are allowed to touch.

How the cloud is actually built, in one concrete example

It helps to have a real stack in mind rather than an abstraction. Ghost Protocol's own services run on Cloudflare's edge, which is a useful teaching case because it shows the model clearly. The static site is served as plain files; the dynamic slices run as serverless functions (Workers); data sits in managed stores (a SQLite service, a key-value store, an object store) reached through **bindings** rather than over the network; and secrets are set with a dedicated secrets command, never written into the configuration file or the code. That last habit, secrets out of config, is the same rule you will be checking a client against in Chapter Three, and the "reach data through a binding, not an ambient network credential" pattern is a preview of why the edge serverless model in Chapter Four does not even have a metadata endpoint to steal from. Different provider, same two owners: identity and data.

■ THE LINE

Two authorisation facts govern every cloud engagement, and they apply from the first command. First, the account, subscription, or project must be the client's own, confirmed in writing, and you verify the identifier before you touch it, because a cloud account is trivial to misidentify and a neighbour's resource can sit one identifier away from your target. Second, the providers publish their own testing policies and you stay inside both the policy and the Rules of Engagement at once. The big three now let a customer test their own account's resources without asking first, but never the provider's shared infrastructure, and never anything resembling denial-of-service or load testing. Read the provider's current policy before the engagement, because it is theirs to change. The law and why this is binary is Volume 0. The paperwork that records the scope is Volume IV. Both apply, every command.



SOL SAYS

The first hour on a cloud engagement is not spent attacking. It is spent drawing the line on paper: which service model is this, which account is in scope, what did the client actually grant me, and what does the provider forbid. An intern who skips that hour spends the next three days confused about why nothing behaves like a web app, and occasionally testing something they were never allowed to touch. Draw the line first. Everything after it is just reading the account carefully.

■ EXERCISE

On your own free-tier cloud account only, and before anything else, set a billing alarm on it at a small amount, so a lab you forget to tear down tells you before it costs you. Then pick three services you can see in the console and sort each into IaaS, PaaS, or SaaS, writing one sentence on where the shared-responsibility line falls for it. Then open the provider's own shared-responsibility document and its penetration-testing policy (all three publish both) and note two things you are allowed to test on your own account and two you are forbidden from touching. Finally, find your account, subscription, or project identifier and write it down the way a Rules of Engagement document would record the in-scope target. You are practising the habit that opens every cloud test: know the model, know the scope, know the exact identifier.

■ VERIFY IT

Before you accept that an account is in scope, prove the identifier matches the one in the written authorisation, not just the display name a console shows. Account names are chosen by humans and are easy to confuse; the numeric account ID, the subscription GUID, or the project ID is the thing a scope is written against. The lead is "the client said this is theirs". The evidence is the identifier in the signed scope matching the identifier the credentials actually resolve to. If they do not match, you stop and ask, because the one unrecoverable mistake in cloud work is testing the wrong account.



WHAT WYRM KEEPS

The provider secures the cloud. The customer secures what they put in it. The line slides with the service model, but identity and data never leave the customer's side.

- IaaS, PaaS, SaaS: the more of the stack you rent, the more the provider secures and the less remains yours to misconfigure. Whatever the model, the customer always owns identity and data.
- The blast-radius container is the AWS account, the Azure subscription and Entra tenant, or the GCP project. A broken identity policy reaches the whole container, not one box.
- Cloud testing is mostly authenticated, read-only configuration review, plus an assumed-breach view for the chains in this book. Both need written authorisation and the exact account identifier first.
- Two rule sets apply at once: the client's Rules of Engagement and the provider's own testing policy. Never the provider's shared infrastructure, never denial-of-service.