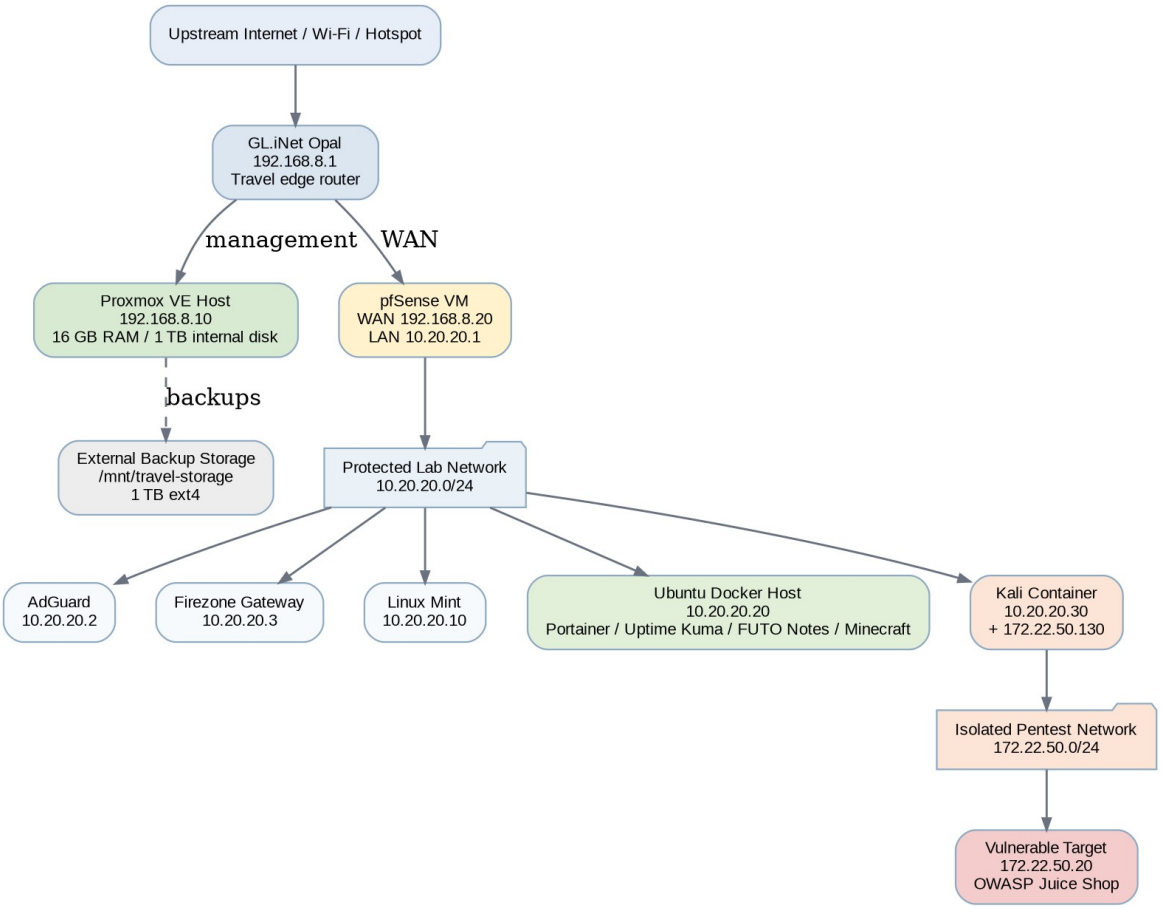
DATA CENTER IN A BAG

Complete Build Guide

A portable Proxmox environment for virtualization, networking, security testing, monitoring, backup, and automation.



Based on the 12-part Data Center in a Bag project series

About This Guide

The Data Center in a Bag began as a simple portable Proxmox experiment and grew into a complete lab for infrastructure, networking, self-hosting, cybersecurity study, disaster recovery, and day-to-day operations. This guide condenses the finished design into a repeatable build sequence.

> ***Addressing note: The IP addresses, ports, and network names in this guide match the finished project. They are a known-good reference, not mandatory values. Adapt them to your environment and verify that they do not overlap with an upstream network you expect to use.***

Before You Start

- A dedicated laptop, mini PC, or small-form-factor system that can run Proxmox VE.
- A travel router or other edge device that can provide a predictable management subnet.
- At least one wired network interface available to the Proxmox host; additional virtual networks are created as Linux bridges.
- A separate USB drive if you want the backup and recovery workflow described later in this guide.
- Comfort with basic IP addressing, Linux shell commands, and web-based administration. The guide explains the design decisions, but it assumes you can follow vendor installation prompts safely.

Terminology Used Throughout the Guide

- Upstream network - the hotel, home, office, or hotspot network that provides Internet access to the travel router.
- Management network - 192.168.8.0/24 in this build; used for the travel router, Proxmox management, and pfSense WAN.
- Protected lab - 10.20.20.0/24; the primary network behind pfSense for normal services and workloads.
- Isolated pentest network - 172.22.50.0/24; a deliberately separated network for authorized security testing.
- Dual-homed - a system with interfaces on two networks. Kali is dual-homed so it can reach both the protected lab and the isolated pentest network.

Who this is for

Home-lab builders, systems administrators, network engineers, cybersecurity students, and anyone who wants a portable environment that can operate behind changing upstream networks while keeping its own internal architecture consistent.

What You Will Build

- A headless Proxmox VE host on repurposed laptop or mini PC hardware.

- A GL.iNet travel router edge with a stable management network.
- A pfSense VM protecting a separate 10.20.20.0/24 lab network.
- Core services including AdGuard, Firezone, Docker/Portainer, FUTO Notes, and Minecraft.
- A persistent Kali container connected to both the normal lab and an isolated 172.22.50.0/24 pentest network.
- Selective VPN policy routing through Private Internet Access.
- External Proxmox backup storage with tested restore capability.
- Uptime Kuma monitoring with Discord notifications.
- Scheduled Proxmox backups and automated Ubuntu/Docker maintenance reporting.

Guide Map

1 Architecture & requirements

2 Proxmox and network foundation

3 pfSense and static routing

4 Core services and Docker

5 Kali and isolated pentest lab

6 VPN policy routing

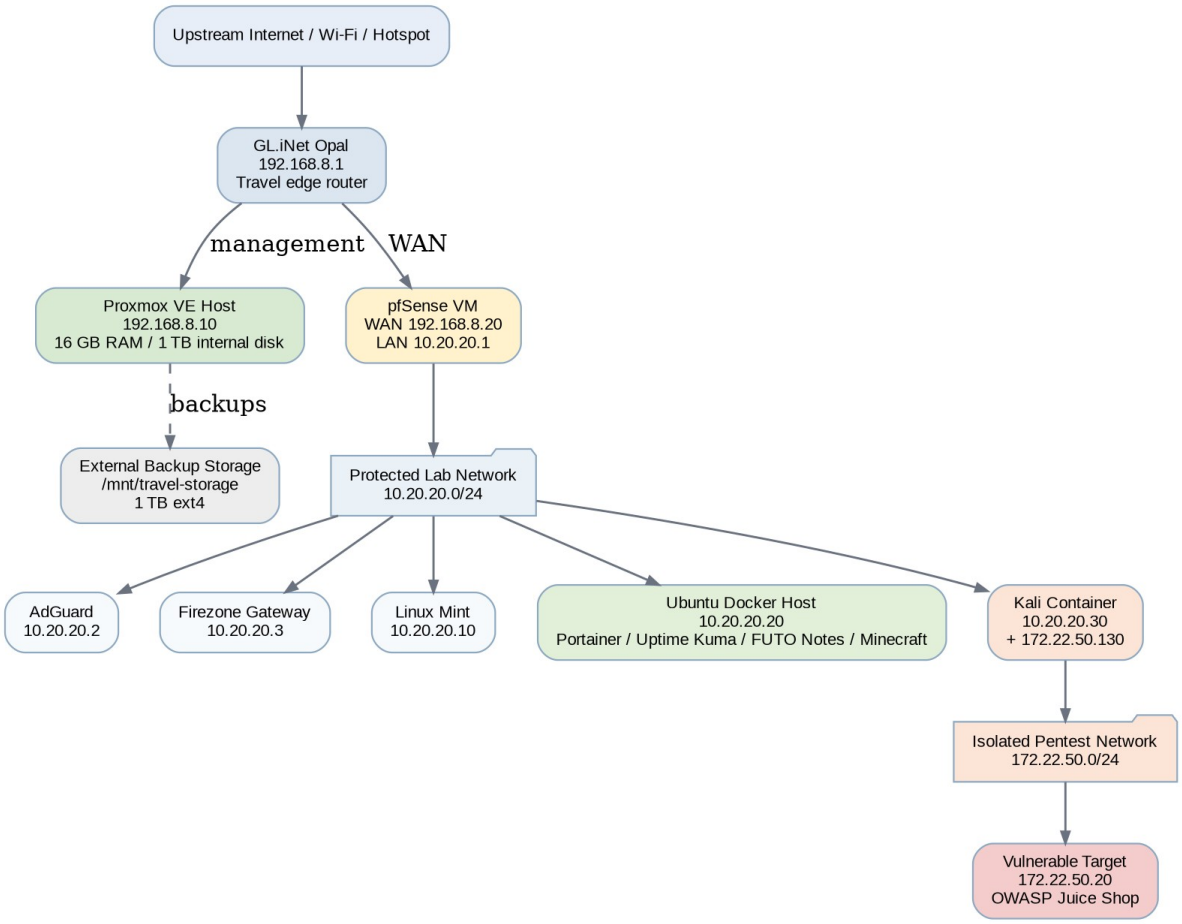
7 Backup and recovery

8 Monitoring and alerts

9 Automation and maintenance

10 Validation, hardening & lessons

1. Architecture and Requirements



Final logical architecture of the portable lab.

Hardware Baseline

The original build uses an older HP laptop with an AMD A10-9600P, 16 GB of RAM, and a 1 TB hard drive. A GL.iNet Opal serves as the travel router, and a separate 1 TB USB drive provides backup storage.

Component	Original build	Recommended starting point
Compute	Older HP laptop, 16 GB RAM	4+ cores, 16 GB RAM minimum, 32 GB preferred
Primary storage	1 TB HDD	SSD strongly preferred
Network edge	GL.iNet Opal	Any GL.iNet/OpenWrt-class travel router with Ethernet
Backup storage	1 TB USB drive	1 TB+ USB 3.x SSD/HDD

Component	Original build	Recommended starting point
Access	Headless	Laptop/phone browser + SSH

Address Plan

Role	Address / subnet	Notes
Travel router LAN	192.168.8.1/24	Stable management/upstream network
Proxmox host	192.168.8.10/24	Management
pfSense WAN	192.168.8.20/24	Connected to travel router side
pfSense LAN	10.20.20.1/24	Protected lab gateway
AdGuard	10.20.20.2	DNS filtering
Firezone Gateway	10.20.20.3	Remote access gateway
Linux Mint	10.20.20.10	GUI utility workstation
Ubuntu Docker	10.20.20.20	Application host
Kali	10.20.20.30 + 172.22.50.130	Dual-homed
Vulnerable target	172.22.50.20	No normal Internet path

Design principle

Keep upstream connectivity and the internal lab separate. The travel router can change how it reaches the Internet, while the Proxmox and pfSense networks remain consistent.

2. Build the Proxmox and Network Foundation

Install Proxmox VE

Install Proxmox VE directly on the laptop or mini PC and assign a static management address on the travel router LAN. In the original build, Proxmox uses 192.168.8.10/24 with 192.168.8.1 as its gateway. Keep the management address outside any DHCP range you expect the travel router to hand out, or create a reservation, so the address remains predictable.

> ***Validation checkpoint: Before creating pfSense or additional bridges, confirm that the Proxmox web UI is reachable from another device on the 192.168.8.0/24 network and that the Proxmox host itself can reach its configured gateway.***

```
Proxmox management
IP:      192.168.8.10/24
Gateway: 192.168.8.1
```

Create the Internal Bridge

Keep the management bridge connected to the physical Ethernet interface. Create a second Linux bridge for the pfSense LAN; this bridge does not need a physical port because it exists only inside the Proxmox host. In this design, the protected network is 10.20.20.0/24. A third isolated bridge is later used for the pentest network. Bridge names such as vmbr0, vmbr2, or vmbr3 are local implementation details - the important part is which network each bridge represents.

- Management/upstream bridge -> 192.168.8.0/24
- Protected pfSense LAN bridge -> 10.20.20.0/24
- Isolated pentest bridge -> 172.22.50.0/24

Travel Router

Configure the GL.iNet router so its LAN is 192.168.8.0/24. This is the portable management layer. The router can use Ethernet, Wi-Fi repeater mode, or a hotspot as its upstream connection without changing the lab addressing behind it.

> **Why a travel router matters**
>
> It gives the lab a predictable edge network. Hotel Wi-Fi, home Internet, or a phone hotspot becomes "upstream" rather than something every VM has to understand.

3. Deploy pfSense and Configure Routing

pfSense VM Interfaces

```
WAN: 192.168.8.20/24
LAN: 10.20.20.1/24
```

Attach the pfSense WAN NIC to the management/upstream bridge and the LAN NIC to the protected internal bridge. Give each interface the address shown in the plan, then confirm pfSense can reach the travel router gateway before opening access from the protected LAN. Configure firewall rules intentionally: start with the minimum required traffic and add exceptions as services are introduced.

> ***Validation checkpoint: From a protected host, verify the default gateway is 10.20.20.1, DNS behaves as expected, and Internet access works before adding the static route on the travel router. This isolates routing problems from firewall or DNS problems.***

Prefer Routing Over Excessive NAT

The final design uses a static route on the GL.iNet router so clients on 192.168.8.0/24 know that pfSense is responsible for 10.20.20.0/24.

```
uci add network route
uci set network.@route[-1].interface='lan'
uci set network.@route[-1].target='10.20.20.0'
uci set network.@route[-1].netmask='255.255.255.0'
uci set network.@route[-1].gateway='192.168.8.20'
uci commit network
```

Validation

From a travel router client, verify that an allowed 10.20.20.x service is reachable without a port-forward. Then verify that pfSense firewall policy, not accidental routing, determines what is permitted.

4. Build the Core Services Layer

AdGuard and Firezone

Deploy core infrastructure first. AdGuard at 10.20.20.2 provides DNS filtering. Firezone at 10.20.20.3 provides a remote-access path when needed. Keep both on predictable static addresses.

Ubuntu Docker Host

The Ubuntu VM at 10.20.20.20 becomes the application platform. Install Docker and Portainer, then deploy application stacks with persistent volumes and Compose files rather than one-off manual containers. Keep Compose definitions and any environment files in a documented location so a container can be recreated without relying on Portainer history alone. Do not place passwords or API tokens directly in screenshots or public repository examples.

- Portainer for container management
- FUTO Notes for self-hosted notes
- Uptime Kuma for monitoring
- Minecraft Survival and Creative servers
- Custom Kali image and related tooling

Minecraft Networking

The Bedrock servers use distinct TCP and NetherNet UDP ranges. Direct routing from the travel router network into the protected lab proved cleaner than maintaining multiple layers of NAT.

Server	TCP	UDP
Survival	19132	19200-19215
Creative	19142	19216-19231

5. Add Kali and the Isolated Pentest Lab

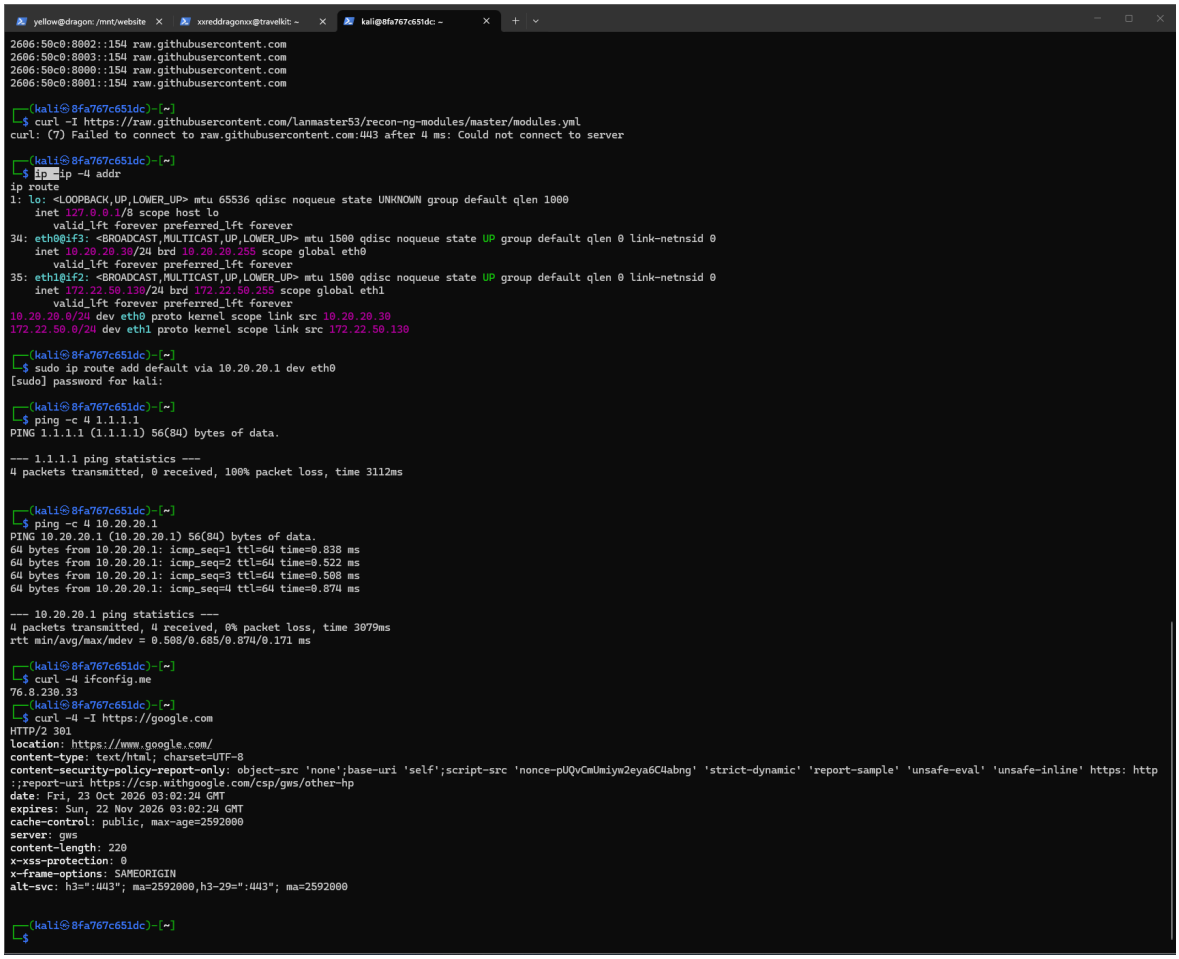
A custom Kali Docker image provides a portable CLI security workstation without the overhead of another full VM. The image includes common reconnaissance and network tools and exposes SSH for controlled management. This design is optimized for command-line work. GUI-heavy tools are better run from a separate desktop system or VM when a graphical session is required.

Persistent Tooling

- kali-tools-top10
- Recon-ng (source-installed)
- dnsenum
- subfinder
- assetfinder
- whois
- theHarvester
- tcpdump
- OpenSSH

Dual-Homed Network Design

Kali connects to both the protected travel LAN and a separate isolated pentest network. The vulnerable target remains on 172.22.50.0/24 and does not need a normal default gateway.



Kali route troubleshooting and validation during the dual-homed build.

```
Kali normal interface:  10.20.20.30
Kali pentest interface: 172.22.50.130
Target:                 172.22.50.20
```

Default Route

The travel LAN must remain Kali's preferred default path. After one rebuild, the container had only connected routes and lost Internet access. The final behavior should be:

```
default via 10.20.20.1
10.20.20.0/24 directly connected
172.22.50.0/24 directly connected
```

> **Safety boundary**
>
> Use the isolated target network only for systems you own or are explicitly authorized to test.

Intentionally vulnerable applications should not be exposed to the public Internet.

6. Add Selective VPN Policy Routing

Private Internet Access was added to pfSense using OpenVPN. Instead of routing the entire lab through the tunnel, host-specific firewall policy determines which systems use the VPN.

Example Policy

- Create the PIA OpenVPN client and confirm the tunnel is connected.
- Assign the tunnel as an interface/gateway in pfSense.
- Create outbound NAT for the selected source address.
- Add a LAN rule that sends the selected host to the PIA gateway.
- Add a block rule beneath it as a kill switch so the host cannot fall back to normal WAN if the VPN disappears.
- Place a local-LAN bypass rule above the policy route so internal management still works.

Result

A selected host can use the commercial VPN while the rest of the travel environment continues to use the normal pfSense WAN. If the VPN fails, that host loses Internet access rather than leaking onto the normal path.

7. Add Backup Storage and Prove Recovery

Connect the external 1 TB drive to the Proxmox host. Identify it by size and model before running any destructive command. The original drive happened to be /dev/sdb and was reformatted from an old APFS layout to GPT + ext4, but your device name may be different.

Destructive step: Never copy /dev/sdb from this guide without verifying your own device. Run `lsblk -o NAME,SIZE,MODEL,FSTYPE,MOUNTPOINTS` and confirm the external drive by model and capacity before wiping or partitioning it.

```
lsblk -o NAME,SIZE,MODEL,FSTYPE,MOUNTPOINTS
# Verify the correct external device first. Example below assumes /dev/sdb.
wipefs -a /dev/sdb
fdisk /dev/sdb       # create GPT + one full-disk partition
mkfs.ext4 -L travel-storage /dev/sdb1
mkdir -p /mnt/travel-storage
mount /dev/sdb1 /mnt/travel-storage
```

Persistent Mount

Use the filesystem UUID in /etc/fstab and include nofail because this is removable storage. After editing /etc/fstab, reload systemd and test the entry before rebooting: unmount the drive, run `mount -a`, and confirm the expected filesystem is mounted at /mnt/travel-storage.

> ***Validation checkpoint: Use `mount | grep travel-storage` and `df -h /mnt/travel-storage`. A successful mount test is safer than discovering an fstab typo during a later reboot.***

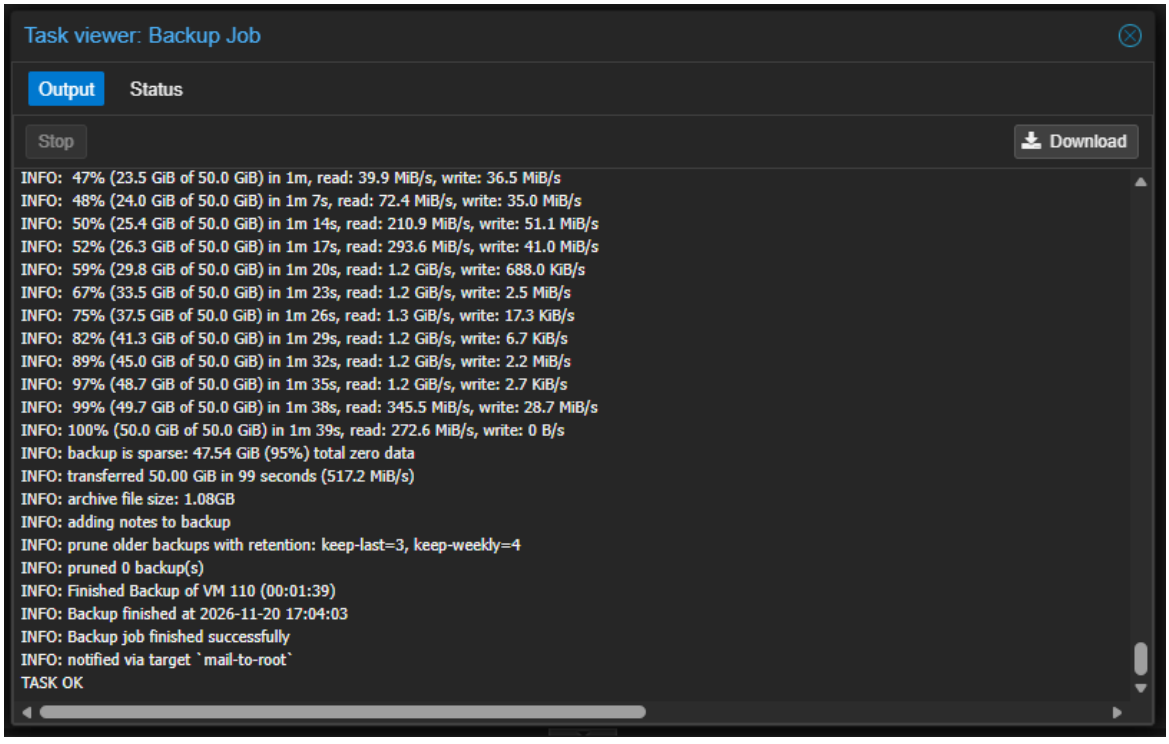
```
UUID=<your-filesystem-uuid> /mnt/travel-storage ext4 defaults,nofail 0 2
```

Proxmox Storage Role

- Add /mnt/travel-storage as Directory storage named travel-storage.
- Enable Backup, ISO image, and Container template content types.
- Keep live VM/LXC disks on the internal storage.

Restore Test

Back up a small container, restore it as a temporary new ID, and leave the restored copy powered off until you review its network configuration. Change its IP only when the original system is still online, then confirm the restored guest boots, its files/configuration are present, and it is reachable from the expected network. A side-by-side test may expose hard-coded application bindings; in a true DR replacement, retaining the original IP may be exactly what you want.



Validated Proxmox backup job completing with TASK OK.

Rule

A backup is not proven until a restore has been tested.

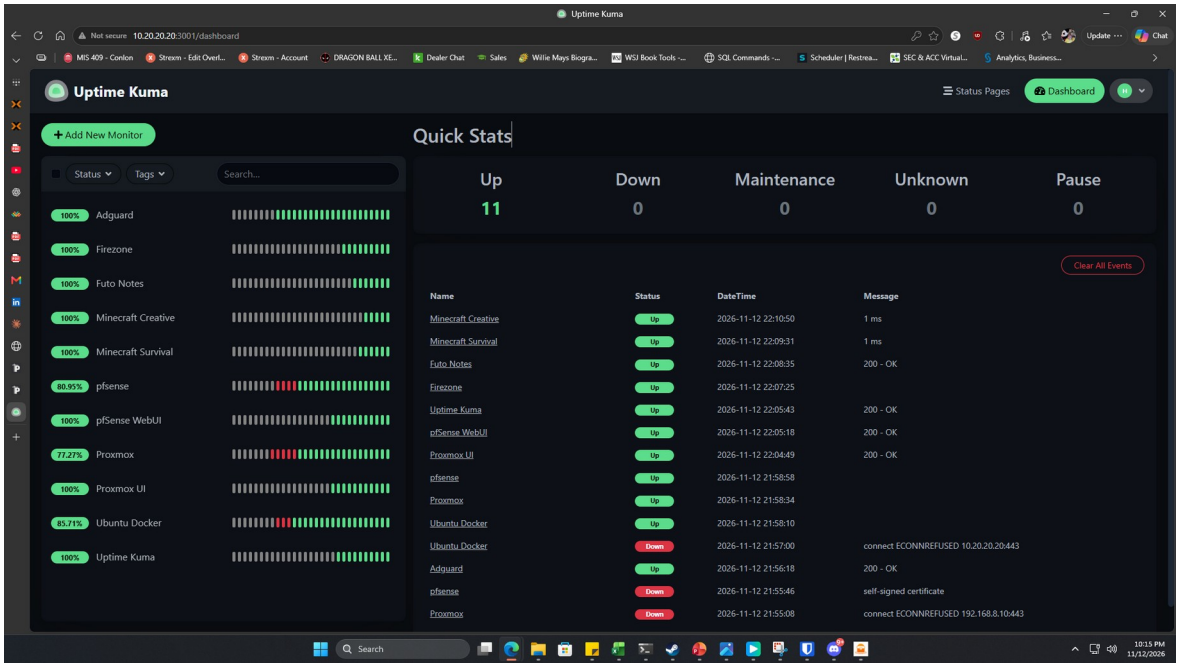
8. Add Monitoring and Alerts

Deploy Uptime Kuma v2 on the Ubuntu Docker host and expose it on port 3001. Use both host-level and application-level checks so the dashboard can distinguish a live machine from a failed service. For internal HTTPS services such as Proxmox and pfSense, configure the monitor appropriately for locally issued or self-signed certificates rather than treating the certificate warning as an outage.

```
Uptime Kuma: http://10.20.20.20:3001
```

Suggested Monitors

System / service	Check
Proxmox	Ping + HTTPS :8006
pfSense	Ping + HTTPS
AdGuard	Ping / HTTP as appropriate
Ubuntu Docker	Ping
Uptime Kuma	HTTP :3001
Firezone	Reachability/service check
FUTO Notes	HTTP
Minecraft Survival	TCP :19132
Minecraft Creative	TCP :19142



Uptime Kuma with all 11 monitored services up.

Use Discord webhook notifications for the services that deserve immediate attention. Avoid attaching alerts to every redundant check or a single host failure may generate several notifications.

> ***Minecraft note: the TCP monitors used here provide a lightweight availability signal. Bedrock/NetherNet also depends on UDP, so a green TCP check does not prove that every game-network path is healthy.***

9. Automate Backups and Maintenance

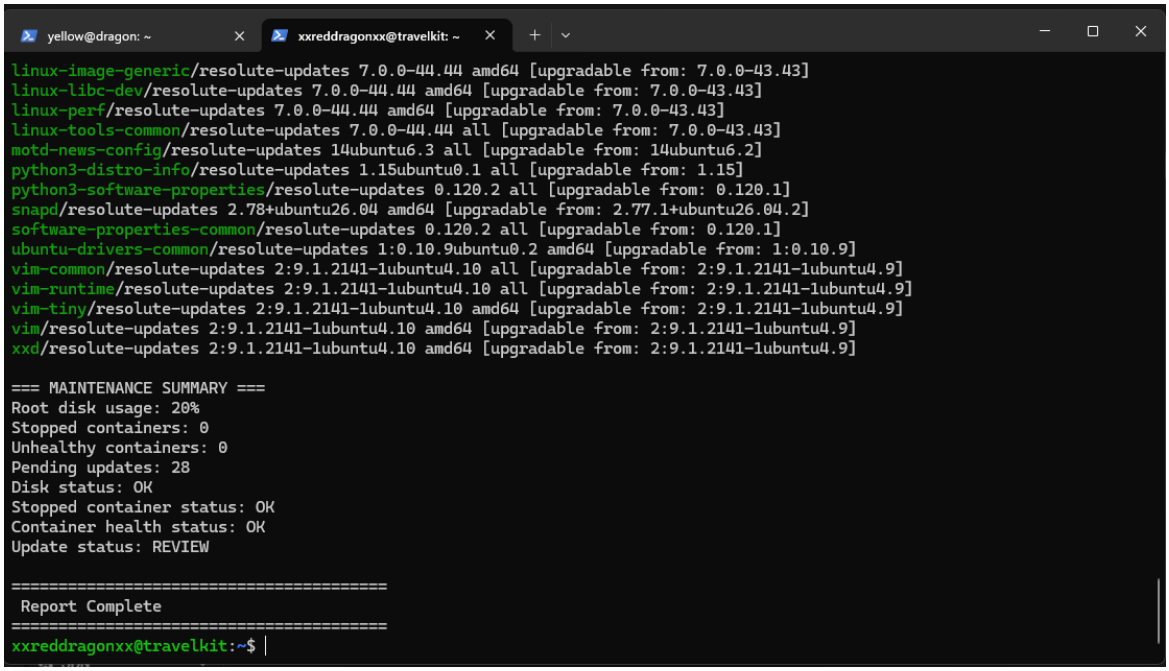
Scheduled Proxmox Backups

After manual backup and restore validation, schedule the job. The project uses a weekly Sunday 01:00 backup to travel-storage with a simple retention policy.

```
Schedule:    Sunday 01:00
Storage:     travel-storage
Mode:        Snapshot
Compression: ZSTD
Keep last:   3
Keep weekly: 4
```

Ubuntu/Docker Maintenance Report

A small script at /usr/local/sbin/travelkit-maintenance-report.sh reports disk usage, Docker storage, container health, and available package updates. It deliberately reports rather than automatically changing the system. The daily cron schedule uses the Ubuntu VM's local system timezone, so adjust the schedule if the VM is intentionally configured for UTC.



Daily maintenance summary: healthy containers, low disk usage, updates flagged for review.

Cron

```
0 7 * * * /usr/local/sbin/travelkit-maintenance-report.sh > /var/log/travelkit-
maintenance/report-$(date +\%F).log 2>&1
15 7 * * * find /var/log/travelkit-maintenance -type f -name 'report-*.log'
-mtime +30 -delete
```

Operating philosophy

Automation detects. A human reviews. A human approves. Automatic patching, pruning, and restarts are intentionally left out of the core design.

10. Validation, Hardening, and Lessons Learned

Final Validation Checklist

☐ Proxmox web UI reachable on the travel router management network.

☐ pfSense WAN and LAN routing are correct.

☐ 192.168.8.0/24 clients can reach only the protected services permitted by firewall policy.

☐ AdGuard resolves and filters DNS as intended.

☐ Ubuntu Docker services survive container/host restarts.

☐ Kali has a default route through 10.20.20.1 and direct access to 172.22.50.0/24.

☐ The vulnerable target cannot reach the normal travel LAN or Internet unless intentionally allowed.

☐ VPN-routed hosts lose Internet rather than falling back to WAN when the tunnel is unavailable.

☐ Backups write to the external disk and at least one restore test succeeds.

☐ Uptime Kuma reports both host and service health.

☐ Discord notifications deliver successfully.

☐ Scheduled backups and maintenance reports run without manual intervention.

Hardening Priorities

- Keep pfSense rules narrow and document exceptions.
- Do not expose vulnerable pentest targets to upstream/public networks.
- Use unique credentials and remove unnecessary default accounts.
- Patch Proxmox, pfSense, Ubuntu, containers, and network appliances on a controlled schedule.
- Review Docker images and reclaimable storage before large Kali rebuilds.
- Keep backup media disconnected when practical if the kit is not in use.
- Periodically repeat a restore test rather than assuming old backups remain usable.
- Keep Portainer, Proxmox, pfSense, Uptime Kuma, and other management interfaces on trusted networks; do not expose them directly to the public Internet unless a deliberate secure-access design requires it.
- Store secrets outside public Compose examples and documentation. Use environment files, Docker secrets, or another controlled secret-management method where practical.

Lessons Learned

Route cleanly	Direct routing between the travel network and protected lab was easier to reason about than stacking NAT rules.
Plan storage early	Docker image builds can temporarily need much more space than the final image.
Monitoring should arrive sooner	Once several services exist, a dashboard becomes operationally important.
Backups are not enough	Restore testing revealed environment-specific details that a successful backup alone would never show.
Automation should be conservative	Reporting and scheduling provide value without allowing unattended changes to break the lab.
Portable does not mean simple	A small laptop can still teach the same routing, virtualization, monitoring, DR, and operations concepts found in larger

environments.

Appendix A - Recommended Build Order

1. Install Proxmox and establish management connectivity.
2. Configure the travel router and stable 192.168.8.0/24 edge network.
3. Deploy pfSense with WAN and LAN interfaces.
4. Create the 10.20.20.0/24 protected network and static route.
5. Deploy AdGuard and the Ubuntu Docker host.
6. Install Docker and Portainer; add normal application workloads.
7. Deploy Kali and build the isolated 172.22.50.0/24 pentest network.
8. Add vulnerable training targets and validate isolation.
9. Add VPN policy routing if desired.
10. Add external backup storage; perform backup and restore testing.
11. Deploy Uptime Kuma and notifications.
12. Schedule backups and maintenance reporting.
13. Document the final architecture and keep the configuration under change control.

Appendix B - Port and Service Quick Reference

Service	Address / port
Proxmox	192.168.8.10:8006
pfSense	10.20.20.1 HTTPS
Uptime Kuma	10.20.20.20:3001
FUTO Notes	10.20.20.20:3005
Minecraft Survival	10.20.20.20:19132 + UDP 19200-19215
Minecraft Creative	10.20.20.20:19142 + UDP 19216-19231
Kali SSH	10.20.20.30:22
Juice Shop target	172.22.50.20:3000

Closing Notes

This guide captures the stable end state of the project rather than every detour taken to get there. The companion Data Center in a Bag series documents the troubleshooting, experiments, failures, and design changes behind the final build. Use this document as the implementation reference, and use the series when you want the deeper engineering context behind a decision.

> **The goal**
>
> Build a portable environment that can keep evolving - without losing the discipline of segmentation, recovery, monitoring, and controlled operations.

HBTECHSOLUTIONS

Data Center in a Bag - Complete Build Guide