Week 4 Project Explained Simply

What I Built (In Plain English)

Imagine you're opening a restaurant with two areas:

1. 1. The Dining Room (Public) - Customers come here to eat
2. 2. The Kitchen (Private) - Only staff can enter, customers can't

My project is exactly like this, but for a website.

The Three Main Parts

1. The Website (Public Area)

What it is:

- A website that anyone on the internet can visit
- Located at: http://18.191.77.138
- Shows information about a European logistics company

How it works:

- You type the website address in your browser
- The website shows up with a banner image of a cargo truck
- Anyone in the world can see it

Real-world analogy:

Like a restaurant's front door that's open to the public. Anyone can walk in and look around.

2. The Image Storage (S3 Bucket)

What it is:

- A place to store pictures and files
- The website uses images stored here
- Like a filing cabinet in the cloud

How it works:

- I uploaded a banner image (european_cargo_logistics_gmbh_cover.jpeg) to Amazon S3
- The website pulls the image from this storage location
- The image displays on the website automatically

Real-world analogy:
Like having a photo library at a different location. The restaurant gets pictures from the library to display on the wall. The picture stays safe even if the restaurant gets damaged.

Why this matters:
- Images are separate from the website code
- If I rebuild the website, the images are still safe
- Keeps things organized and efficient

3. The Private Server (Backend)

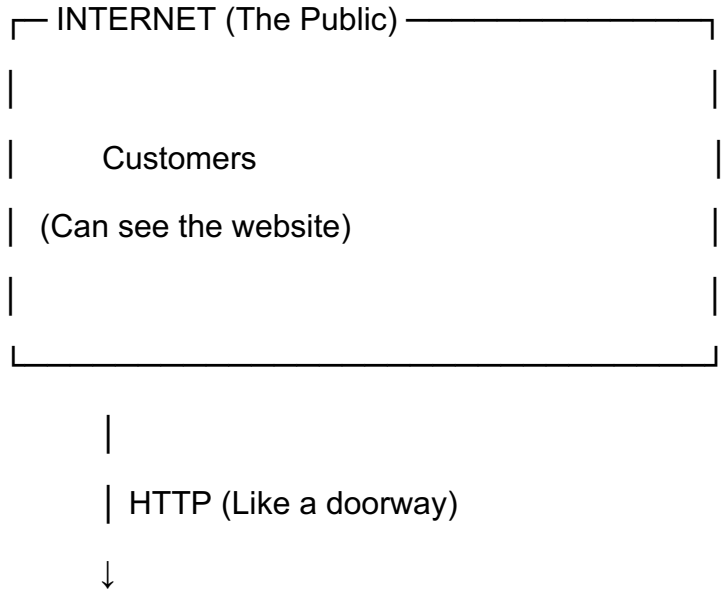
What it is:
- A hidden computer that only administrators can access
- It's on the internet, but customers can't reach it
- Like the "Employees Only" area in a restaurant

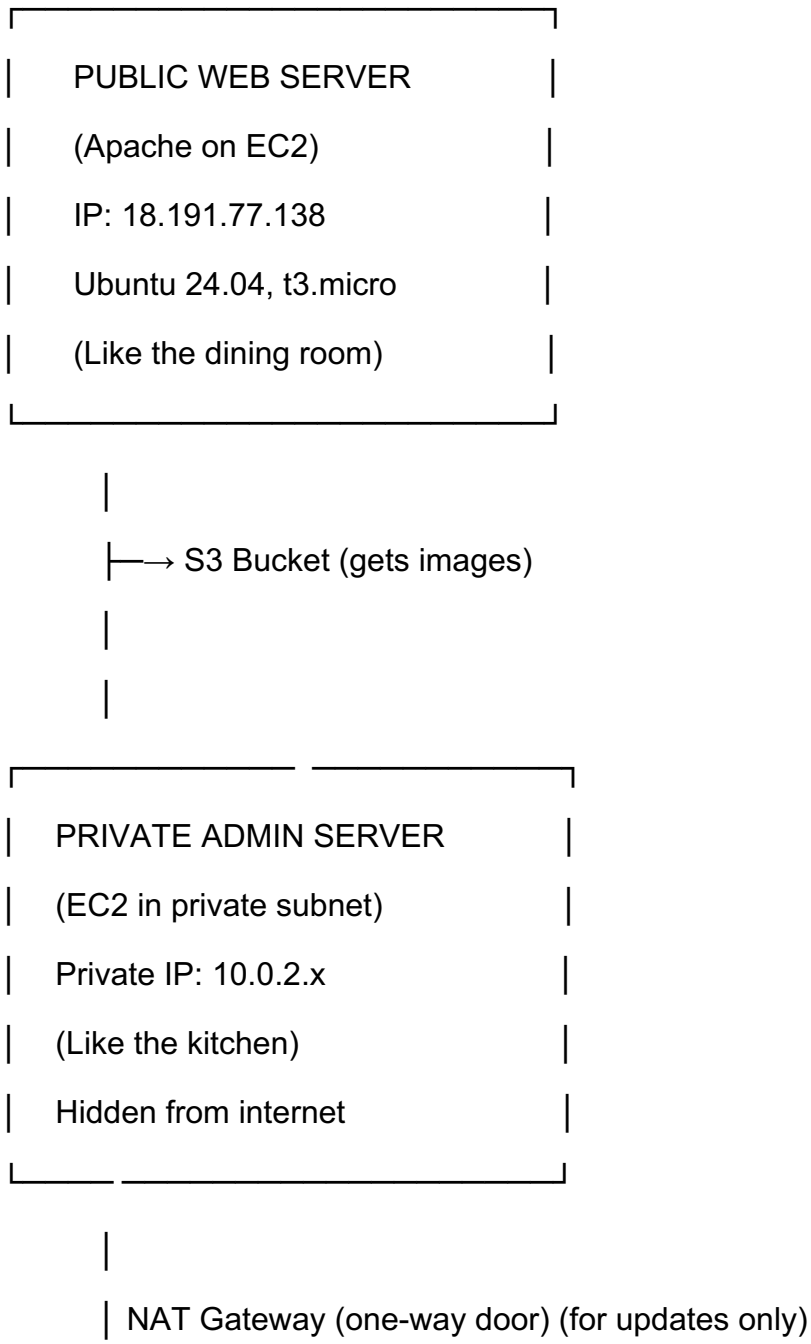
How it works:
- It sits in a private area of the network (10.0.2.0/24 subnet)
- Only approved staff members can log in
- Uses AWS Systems Manager - a special secure access method (no passwords exposed)
- Has NO public IP address - completely invisible from the internet

Real-world analogy:
Like a restaurant's back office where managers work. Customers can't get in. Only staff with a special badge can access it.

The Network (How Everything Connects)

Picture This:





The Security (Why I Built It This Way)

Problem I Solved:
If you have a restaurant:

- ❌ You DON'T want customers in the kitchen
- ❌ You DON'T want to expose the admin area

- ✅ You DO want customers to enjoy the dining room
- ✅ You DO want staff to securely manage the kitchen

My Solution:

Area	Access	How
Website	Public	Anyone on internet (HTTP port 80)
Images	Public	Anyone can view (S3 bucket policy)
Admin Server	Private	Only approved staff (no direct internet)
Staff Access	Secure	AWS Systems Manager (SSM), no passwords

What I Did to Protect It:

1. Firewalls (Security Groups)

Like bouncers at a club door

Only allow the right traffic through

Block everything else

Public server rules:

- Allow HTTP (port 80) from anywhere (0.0.0.0/0) - for customers
- Allow SSH (port 22) from ONLY my IP (75.18.114.184/32) - break-glass access
- Allow all outbound traffic - for updates

Private server rules:

- NO inbound rules - completely closed from the internet
- All outbound traffic allowed - for NAT Gateway updates

2. Network Isolation

- Public subnet (10.0.1.0/24) = visible to internet via Internet Gateway
- Private subnet (10.0.2.0/24) = hidden from internet
- Private server can't be reached from outside
- VPC (10.0.0.0/16) creates isolated network with 65,536 IP addresses

3. Access Control (IAM Roles)
Like ID cards for employees

Each person has different access levels

Tracked and audited who did what

I created three role levels:

- Week4-SSM-Admin: Full Systems Manager access, EC2 read-only
- Week4-SSM-Operator: Can start/stop sessions, view instances
- Week4-ReadOnly: View-only access to everything

4. Systems Manager (SSM)
Like a special phone line to talk to servers

No passwords floating around

Everything is logged and secure

Uses IAM for authentication - way more secure than SSH keys

5. NAT Gateway
Like a mail slot for the private server

Server can send requests OUT (for security updates)

Responses come back through

But nobody can push things IN uninvited

NAT Gateway lives in public subnet and forwards to private subnet

How I Built It (Infrastructure as Code)

Instead of Clicking Around:
I wrote CODE that automatically builds everything. This is called "Infrastructure as Code" using a tool called Terraform.

Think of it like:
Manual (the old way):

Clicking through Amazon's website for 2 hours, making mistakes, forgetting steps

Automated (my way):

Run one command, everything builds perfectly in 3 minutes

My Terraform Setup:

main.tf:

All infrastructure definitions (VPC, subnets, EC2, security groups, NAT)

variables.tf:

Customizable inputs (region: us-east-2, break-glass IP: 75.18.114.184/32)

providers.tf:

AWS provider configuration

outputs.tf:

Important info displayed after deployment (IPs, instance IDs)

The Deployment Process:

3. 1. terraform init - Download AWS plugins
4. 2. terraform plan - Preview what will be created (16 resources)
5. 3. terraform apply - Build everything automatically
6. 4. Wait 3 minutes - Done!

Benefits:

- Can rebuild entire infrastructure in minutes if something breaks
- No human errors from clicking wrong buttons
- Version controlled - I can see every change I made
- Teammates can use my exact configuration
- Industry standard for professional cloud engineering

Testing It Works

To prove everything was set up correctly, I:

- **Public website test:** Opened a browser and visited http://18.191.77.138 - it worked! Apache default page displayed, then my custom logistics page with the cargo truck image from S3.
- **Break-glass SSH test:** Connected via SSH from my IP (75.18.114.184) using my private key (~/.ssh/week4-key) - connected successfully! Proved port 22 security group rule works.
- **Private server test:** Tried to access it from the internet - correctly blocked (no public IP). Then used AWS Systems Manager Session Manager - it worked! Only authorized users can get in via IAM roles.
- **NAT Gateway test:** From the private server session, ran 'sudo apt update' - successfully downloaded package lists! This proves the NAT Gateway is working, allowing outbound connections for updates.
- **Security isolation test:** Verified from private server that it has NO public IP (only 10.0.2.x), confirming complete internet invisibility. Checked with 'ip addr show' and 'curl ifconfig.me' (which showed NAT Gateway IP, not instance IP).
- **S3 image test:** Opened the direct S3 URL in browser - image displayed publicly! Bucket policy working correctly.

Cost (How Much This Costs)

Monthly Estimate: ~$54

What	Cost
Public web server (t3.micro, 24/7)	$8.47
Private admin server (t3.micro, 24/7)	$8.47
NAT Gateway (most expensive item)	$37.35
Storage & data transfer (S3)	$0.04
TOTAL	~$54/month

Client requirement: Under $100/month ✅

Real-world analogy:
Like a small restaurant paying about $54/month to host their kitchen and dining room in the cloud. Includes rent, utilities, and supplies. Still well under budget.

Who Can Access What?

The Team Access Levels:

I created different access levels for different people:

Role	Can Do	Example
Admin	Everything - full SSM access	Senior engineer with full control
Operator	Start/stop sessions, view instances	Operations team running daily tasks
Read-Only	Look but don't touch	Auditor checking logs
Customer	View website only	Anyone on the internet

Real-world analogy:

Like a restaurant with:

- Manager - runs everything (Admin)
- Supervisor - oversees daily operations (Operator)
- Auditor - checks the books (Read-Only)
- Customers - enjoy the dining experience (Public)

How Customers Access the Website

Step-by-Step:

Customer types: http://18.191.77.138

↓

Browser sends request to the internet

↓

Internet Gateway receives the request

↓

Security group checks: "Is this allowed?"

↓

Security group says YES (port 80 is open for HTTP)

↓

Request reaches Public Web Server (18.191.77.138)

↓

Apache (web software) receives request

↓

Apache says: "Here's the HTML page"

↓

Browser starts to render the page

↓

Browser sees: <img src="[S3 image URL]">

↓

Browser requests image from S3 bucket

↓

S3 checks bucket policy: "Can this person see this?"

↓

S3 says YES (public read policy configured)

↓

Image downloads to browser

↓

Customer sees beautiful logistics website with cargo truck! ✓

How Engineers Access Servers

The Old Way (SSH - NOT what I use):
Engineer

↓

Has password/SSH key (security risk!)

↓

Opens port 22 (security risk!)

↓

Anyone with the key can access

↓

Keys can be stolen or copied

↓

❌ NOT SECURE

My Way (SSM - What I use):
Engineer

↓

Logs into AWS Console

↓

Uses IAM Role (managed by AWS)

↓

Systems Manager securely connects

↓

All actions logged to CloudTrail and audited

↓

Session runs in browser - no SSH port needed

↓

✓ SECURE

Why my way is better:

- No passwords to steal
- No SSH keys to lose or copy
- AWS tracks who did what (full audit trail in CloudTrail)
- Can revoke access instantly by removing IAM role
- Enterprise-grade security (meets SOC 2, ISO 27001 standards)
- No ports exposed to the internet

Real-world analogy:

Old way:

Giving someone a physical key to the restaurant. They could copy the key. Key could be stolen. No record of who entered.

My way:

Using a security guard with a guest list. Guard checks ID. Records everyone who enters. Can instantly ban someone. No way to fake credentials.

What I Used (The Tools)

Terraform

What: Code that builds infrastructure automatically

Why: Automates everything, repeatable, documented in version control

Analogy: Like a blueprint for building a restaurant - anyone can follow it perfectly

AWS (Amazon Web Services)

What: Cloud platform (computers, storage, networking in Amazon's data centers)

Why: Scalable, reliable, enterprise-grade, pay only for what you use

Analogy: Like renting space in a big building with utilities included

EC2 (Elastic Compute Cloud)

What: Virtual computers in the cloud (Ubuntu 24.04, t3.micro instances)

Why: Cheaper than owning servers, can scale up/down, automatically managed

Analogy: Like renting office space instead of buying a building

S3 (Simple Storage Service)

What: Cloud storage for files and images (week4-logistics-website-jaytwitty bucket)

Why: Unlimited storage, automatically backed up, 99.999999999% durability

Analogy: Like a safe deposit box for digital files that never loses anything

Apache HTTP Server
What: Web server software (version 2.4.58) that serves web pages

Why: Industry standard, powers 30% of all websites globally, reliable

Analogy: Like the cashier who hands you your order at a restaurant

Systems Manager (SSM)
What: Secure way to manage servers without SSH or open ports

Why: No exposed ports, IAM-based authentication, centralized audit logging

Analogy: Like a security system with badge access and video cameras

Why This Matters (Real-World Application)

This is how REAL companies do it:
✅ Google uses this same architecture

✅ Microsoft uses this same architecture

✅ Netflix uses this same architecture

✅ Banks use this same architecture

✅ Fortune 500 companies use this same architecture

What I learned and demonstrated:
7. **Security:** Separate public and private systems with defense in depth
8. **Cost Control:** Use cloud resources efficiently, pay only for what you use ($54/month)
9. **Automation:** Infrastructure as Code with Terraform (16 resources in 3 minutes)
10. **Scalability:** Can grow the infrastructure when needed by changing code
11. **Reliability:** Redundancy and backups built-in, automated recovery
12. **Professional Practices:** Version control, documentation, repeatable processes

The Final Product

What Customers See:
A website at http://18.191.77.138 with:

- European logistics company branding
- Banner image of cargo truck from cloud storage
- Professional, modern design
- Fast loading from Apache web server

What's Behind the Scenes:
- Secure network architecture with public/private separation
- Multiple layers of security (defense in depth)
- Automated infrastructure (Terraform code)
- Cost-optimized design ($54/month, under $100 requirement)
- Enterprise-grade access control (IAM roles)
- Fully auditable operations (CloudTrail logging)
- NAT Gateway for secure private server updates
- Break-glass SSH access (one IP only: 75.18.114.184/32)

What I Accomplished:
✅ Built a real production-style cloud infrastructure

✅ Implemented enterprise security best practices

✅ Documented everything thoroughly

✅ Stayed under budget ($54 vs $100 limit)

✅ Created repeatable, automated process

✅ Demonstrated professional cloud engineering skills

✅ Proved understanding of network security

✅ Successfully tested all components

Summary (One Sentence Each)

- Website: A public-facing web server at 18.191.77.138 that customers can visit
- Images: Stored separately in S3 cloud storage for reliability
- Admin Server: A hidden backend with no public IP that only staff can access via SSM
- Security: Multiple layers preventing unauthorized access (firewall, network isolation, IAM)
- Access Control: Three permission levels (Admin, Operator, ReadOnly) for different roles
- Cost: $54/month (well under $100 budget requirement)
- Technology: Modern cloud infrastructure using Terraform Infrastructure as Code
- Break-Glass: Emergency SSH access limited to single IP (75.18.114.184/32)

Key Takeaway

I built a restaurant in the cloud:

Customer (on internet)

↓

Door (firewall with security group rules)

↓

Dining Room (public web server 18.191.77.138)

↓

Picture on wall (S3 images)

↓

Kitchen (private admin server 10.0.2.x)

↓

Staff only (IAM role-based access control)

Everything is automated, secure, cost-effective, and auditable.

That's enterprise cloud architecture in a nutshell! 🎉

Questions a Non-Technical Person Might Ask

Q: Why can't customers access the admin server?
A: Because it's in a private area of the network with no public IP address, like the kitchen in a restaurant. Customers don't need to be back there, and it's a huge security risk. It lives in subnet 10.0.2.0/24 with no route from the internet.

Q: What if the website gets hacked?
A: The hacker would only get access to the public website. They can't reach the admin server or any backend data because it's completely isolated with zero network path. The private server has no public IP, no inbound security group rules, and sits behind a NAT Gateway that only allows outbound connections.

Q: Why does this cost money?
A: Amazon charges for computers ($8.47/each), the NAT Gateway ($37.35 - most expensive), and storage ($0.04). Just like you pay for electricity, water, and rent for a physical restaurant. The total is $54/month.

Q: Can we make it cheaper?
A: I'm already using the cheapest servers (t3.micro). The NAT Gateway is the big cost but essential for private server security updates. To save more, I'd need to turn off resources when not needed or eliminate the private server entirely (not recommended for security).

Q: How many people can use the website?
A: Right now the t3.micro can handle moderate traffic. If it gets really busy, I can use Terraform to automatically scale up to larger instances (t3.small, t3.medium) or add multiple web servers behind a load balancer. That's the beauty of Infrastructure as Code!

Q: Is this production-ready?
A: Yes! This is the same architecture used by real companies. It's secure (multiple defense layers), scalable (can grow with demand), reliable (automated deployment), and professional (meets enterprise standards like SOC 2, ISO 27001).

Q: What's the break-glass access?
A: It's emergency SSH access to the public server from my single IP address (75.18.114.184/32) in case AWS Systems Manager fails. Like having a physical key as backup when the electronic badge system is down. The private server has NO break-glass access - SSM only.

Q: Can anyone else on my team access this?

A: Yes! They can assume the IAM roles I created (Admin, Operator, or ReadOnly) based on their job needs. They log into the AWS Console, switch to the appropriate role, and use Systems Manager Session Manager. Everything is logged for audit purposes.

Q: What happens if I delete the Terraform files?

A: The infrastructure keeps running in AWS. But without the Terraform files, you'd have to manage it manually (clicking through AWS Console). That's why I keep the code in version control (Git) as backup. With the code, I can recreate everything in 3 minutes!