Coolify on AWS: A Complete Setup & Deployment Guide

10 min read · Jan 22, 2025

Julio Vaught Follow

Listen Share More

Before we dive into creating our Coolify cluster, let's make sure we have all our AWS infrastructure prerequisites in place. This guide will walk you through each component needed and then show you how to launch your spot instances efficiently.

Prerequisites Checklist

Before creating our spot instances, ensure you have:

1. **Docker Hub Account Setup**

- Active Docker Hub account
- Created repository for your images
- Remember your credentials (you'll need them later)

2. VPC Setup

- A Virtual Private Cloud (VPC)
- At least one public subnet
- Internet Gateway attached to your VPC
- Route tables configured to use the Internet Gateway

3. Security Group Configuration

Create a security group with these inbound rules:

		Port	Source	Description
		22	Your IP	SSH from your location
		8000	Your IP	Coolify Server Port
Custom TCP	TCP	6001	Your IP	WebSocket connections
Custom TCP	TCP	6002	Your IP	Terminal connections
HTTP	TCP	80	0.0.0.0/0	HTTP traffic
HTTPS	TCP	443	0.0.0.0/0	HTTPS traffic
Custom TCP	TCP	2377	Security Group ID	Docker Swarm Communication
Custom TCP	TCP	7946	Security Group ID	Overlay network discovery (TCP)
Custom UDP	UDP	7946	Security Group ID	Overlay network discovery (UDP)
Custom UDP	UDP	4789	Security Group ID	Overlay network traffic
SSH	TCP	22	Security Group ID	Inter-node SSH for Coolify/Swarm

💡 **Security Group Self-Reference:** For rules referencing “Security Group ID”, use the ID of the security group you’re creating. This allows nodes within the group to communicate with each other.

4. SSH Keys

- Generate or use existing SSH key pair
- Ensure the key is registered in AWS EC2

5. IAM Permissions

- EC2 Full Access
- VPC Full Access
- Spot Fleet Request permissions
- IAM role for spot fleet management

Creating the Spot Fleet Request

We’ll use AWS Spot Fleet to launch all four instances simultaneously. Here’s the configuration template:

```
{
    "IamFleetRole": "arn:aws:iam::[YOUR-ACCOUNT-ID]:role/aws-ec2-spot-fleet-tag
                    ategy": "priceCapacityOptimized",
                    y": 4,
                    2025-01-21T18:10:58.000Z",
                    "2026-01-21T18:10:58.000Z",
                    ancesWithExpiration": true,
                    st",
    "TargetCapacityUnitType": "units",
    "LaunchSpecifications": [
        {
            "ImageId": "[UBUNTU-AMI-ID]",  // Latest Ubuntu 24.04 LTS ami-07d26
            "KeyName": "[YOUR-KEY-NAME]",
            "BlockDeviceMappings": [
                {
                    "DeviceName": "/dev/sda1",
                    "Ebs": {
                        "DeleteOnTermination": true,
                        "Iops": 15000,
                        "VolumeSize": 30,
                        "VolumeType": "gp3",
                        "Encrypted": false
                    }
                }
            ],
            "SubnetId": "[YOUR-SUBNET-ID]",
            "SecurityGroups": [
                {
                    "GroupId": "[YOUR-SECURITY-GROUP-ID]"
                }
            ],
            "InstanceRequirements": {
                "VCpuCount": {
                    "Min": 2,
                    "Max": 2
                },
                "MemoryMiB": {
                    "Min": 2048,
                    "Max": 4096
                }
            }
        }
    ]
}
```

Understanding the Spot Fleet Configuration

Let’s break down the key components:

1. **Fleet Role:** Your IAM role for spot fleet management
2. **Allocation Strategy:** `priceCapacityOptimized` ensures best price while ity

instances (one each for Coolify, Manager, Worker1, Worker2)

ıents:

- 2 vCPUs
- 2–4 GB RAM
- 30GB GP3 storage for optimal performance

Launching the Fleet

You can launch this fleet using either:

1. AWS Console:

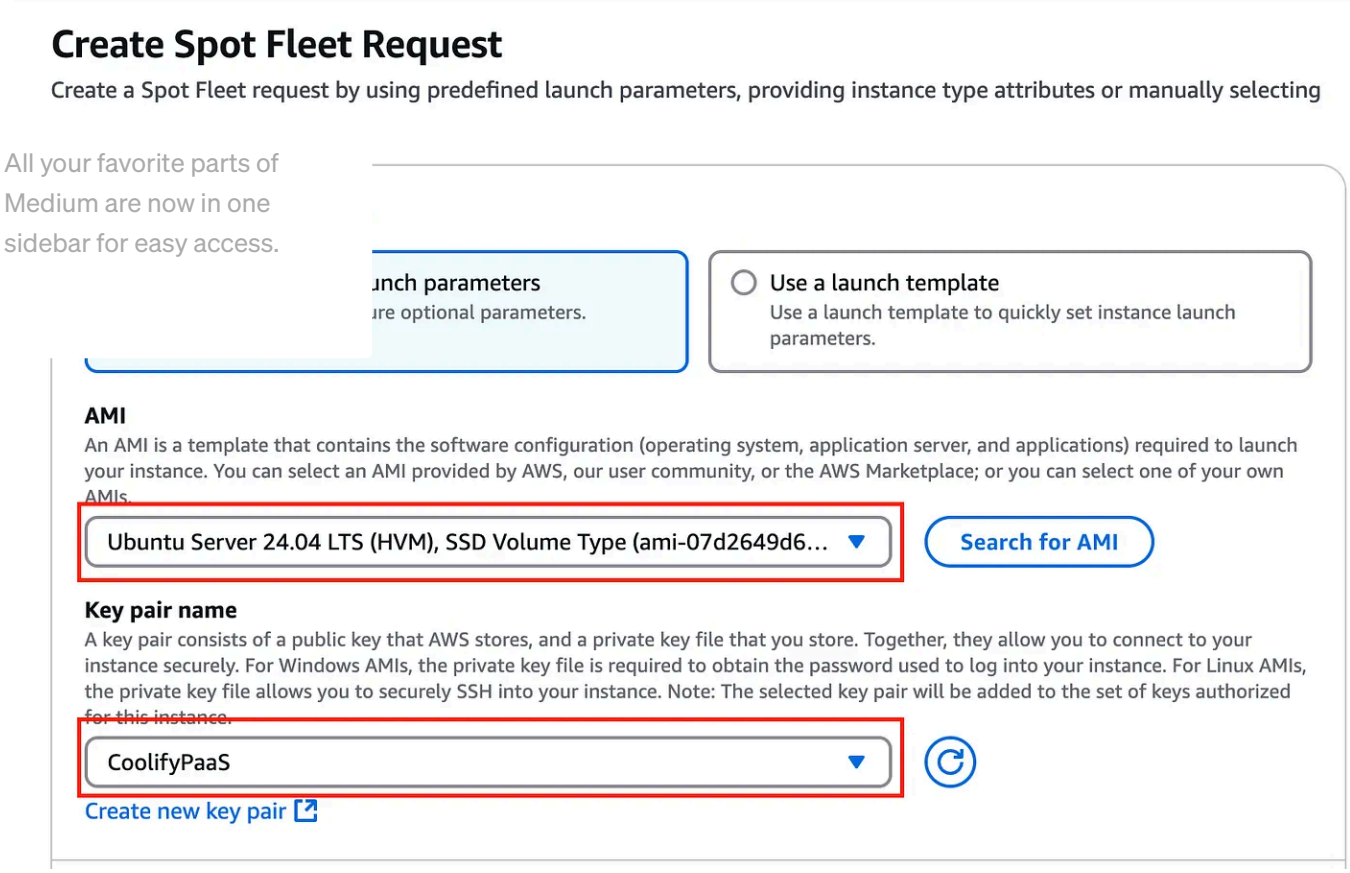
```
EC2 Dashboard → Spot Requests → Request Spot Instances →
Configure with JSON → Paste configuration
```

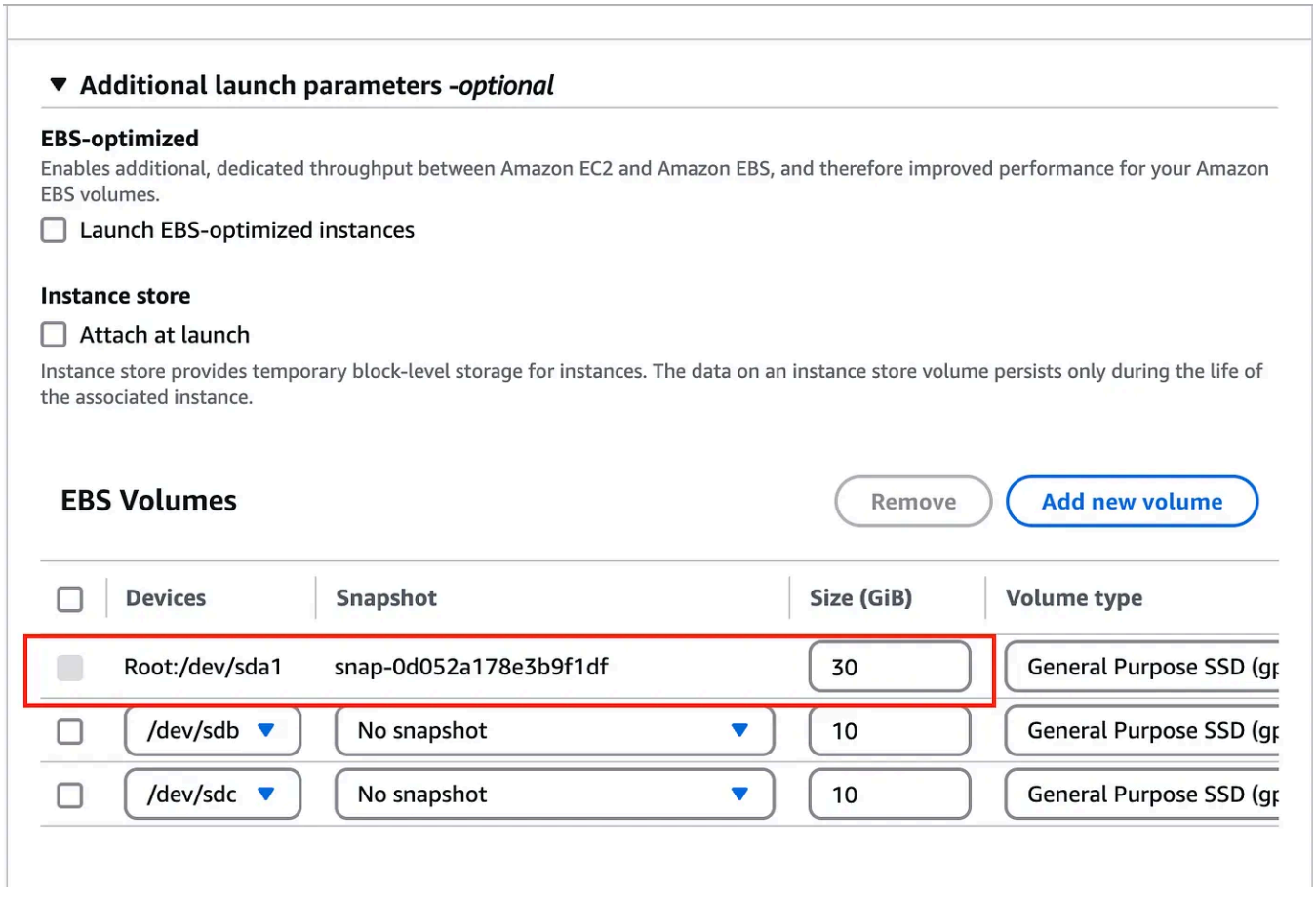
2. AWS CLI:

```
aws ec2 request-spot-fleet --spot-fleet-request-config file://spot-fleet-config
```

3. AWS GUI:

- Follow the json template and fill in the necessay sections.


Create Spot Fleet Request
Create a Spot Fleet request by using predefined launch parameters, providing instance type attributes or manually selecting
All your favorite parts of Medium are now in one sidebar for easy access.
unch parameters
ure optional parameters.
Use a launch template
Use a launch template to quickly set instance launch parameters.
AMI
An AMI is a template that contains the software configuration (operating system, application server, and applications) required to launch your instance. You can select an AMI provided by AWS, our user community, or the AWS Marketplace; or you can select one of your own AMIs.
Ubuntu Server 24.04 LTS (HVM), SSD Volume Type (ami-07d2649d6...
Search for AMI
Key pair name
A key pair consists of a public key that AWS stores, and a private key file that you store. Together, they allow you to connect to your instance securely. For Windows AMIs, the private key file is required to obtain the password used to log into your instance. For Linux AMIs, the private key file allows you to securely SSH into your instance. Note: The selected key pair will be added to the set of keys authorized for this instance.
CoolifyPaaS
Create new key pair


Additional launch parameters -optional
EBS-optimized
Enables additional, dedicated throughput between Amazon EC2 and Amazon EBS, and therefore improved performance for your Amazon EBS volumes.
Launch EBS-optimized instances
Instance store
Attach at launch
Instance store provides temporary block-level storage for instances. The data on an instance store volume persists only during the life of the associated instance.
EBS Volumes
Remove
Add new volume
Devices
Snapshot
Size (GiB)
Volume type
Root:/dev/sda1
snap-0d052a178e3b9f1df
30
General Purpose SSD (gp
/dev/sdb
No snapshot
10
General Purpose SSD (gp
/dev/sdc
No snapshot
10
General Purpose SSD (gp

Monitoring
Monitor, collect, and analyze instance metrics through Amazon CloudWatch. The default is free, basic monitoring, where data is available in 5-minute periods. You can enable detailed monitoring, where data is available in 1-minute periods (additional charges apply).

ed monitoring

es on physical servers fully dedicated for your use, or on a shared hardware.

vare instance

Security groups
A security group is a set of firewall rules that control the traffic for your instance. Note: If you are using EC2-VPC, you must use security groups created specifically for your VPC.

- [ ] default
- [ ] DockerSwarm
- [x] DockerSwarm-LabVPC
- [ ] launch-wizard-1
- [ ] launch-wizard-2
- [ ] launch-wizard-3

Create new security group

Auto-assign IPv4 public IP
Auto-assign a public IPv4 IP address to your instance(s) at launch to make it reachable from the Internet

Use subnet setting

IAM instance profile
An instance profile is a container for an IAM role and enables you to pass role information to an Amazon EC2 instance when the instance starts.

(optional)

Create new IAM profile

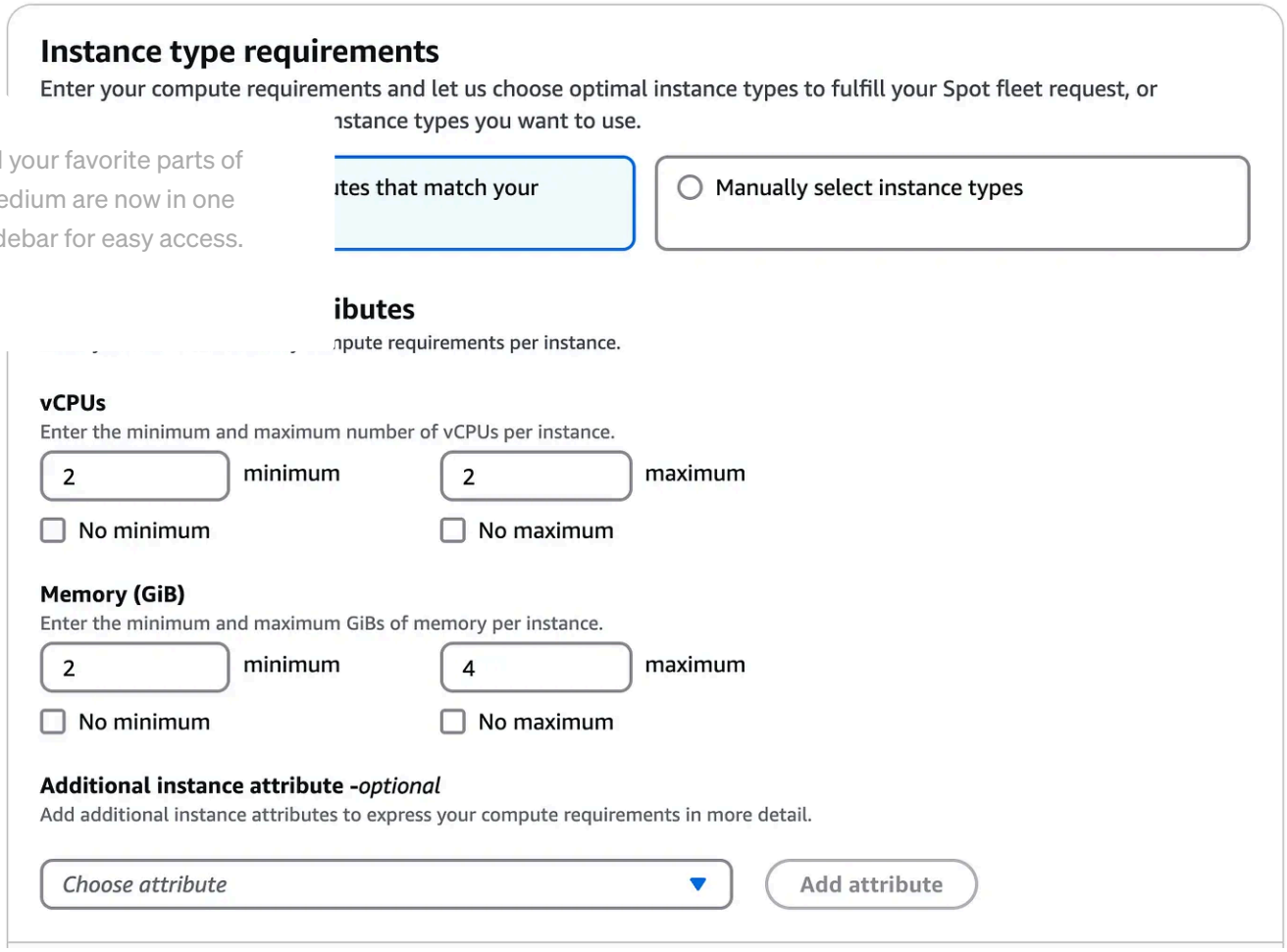
User data

Target capacity

Total target capacity
Set your total target capacity (number of instances or vCPUs) to launch. If you specified a launch template, you can allocate part of the target capacity as On-Demand. The number of On-Demand Instances always persists, while Spot Instances can be scaled.

4 instances

- [ ] Include On-Demand base capacity
  Allocate part of target capacity as On-Demand instances
- [ ] Maintain target capacity
  Automatically replace interrupted Spot Instances
- [ ] Set maximum cost for Spot Instances
  Set the maximum amount per hour that you're willing to pay for all the Spot Instances in your fleet



Post-Launch Setup

Once your instances are running:

1. Tag your instances for easy identification:

- Coolify Server: `Name=coolify-main, Role=coolify-server`
- Swarm Manager: `Name=swarm-manager, Role=manager`
- Workers: `Name=worker-1, Role=worker` and `Name=worker-2, Role=worker`

2. Note the private and public IPs of all instances:

```
# Create a file to store your instance details
echo "Coolify Server: [PUBLIC_IP] ([PRIVATE_IP])" >> cluster-info.txt
echo "Swarm Manager: [PUBLIC_IP] ([PRIVATE_IP])" >> cluster-info.txt
echo "Worker 1: [PUBLIC_IP] ([PRIVATE_IP])" >> cluster-info.txt
echo "Worker 2: [PUBLIC_IP] ([PRIVATE_IP])" >> cluster-info.txt
```

1. Test SSH access to all instances:


```
əm] ubuntu@[PUBLIC_IP]
```

Next Steps

Now that we have our infrastructure ready, in the next section we'll:

1. Install Docker on all instances
2. Set up Coolify on the main server
3. Initialize our Docker Swarm
4. Join our nodes to the swarm

Make sure to keep your `cluster-info.txt` file handy - we'll need those IPs in the next steps!

Setting Up Docker Swarm

With our infrastructure in place, let's set up our Docker Swarm cluster.

Initialize the Swarm Manager

1. SSH into your designated manager node:

```
ssh -i your-key.pem ubuntu@manager-ip/dns
```

1. Install Docker:

```
curl -fsSL https://get.docker.com -o get-docker.sh
sudo sh get-docker.sh
```

1. Initialize the swarm:

```
sudo docker swarm init
```


— you'll need it for workers:

```
# You'll see output like:
sudo docker swarm join --token SWMTKN-1-49nj1cmql0jkz5s9242349dsadasdEXAMPLEoex
```

Join Worker Nodes

1. SSH into each worker node:

```
ssh -i your-key.pem ubuntu@worker-ip/dns
```

1. Install Docker:

```
curl -fsSL https://get.docker.com -o get-docker.sh
sudo sh get-docker.sh
```

1. Run the join command from step 3 above:

```
sudo docker swarm join --token SWMTKN-1... 192.168.99.100:2377
```

Verify Cluster Status

On the manager node, run:


```
s
```

You should see all nodes listed with `Ready` status.

Installing and Configuring Coolify

Now that we have our infrastructure ready, let's get Coolify up and running! This section will walk you through the initial installation and configuration.

Updating Your Servers

First things first, let's make sure all our servers are up-to-date. Connect to each server and run:

```
sudo apt-get update && sudo apt-get upgrade -y
```

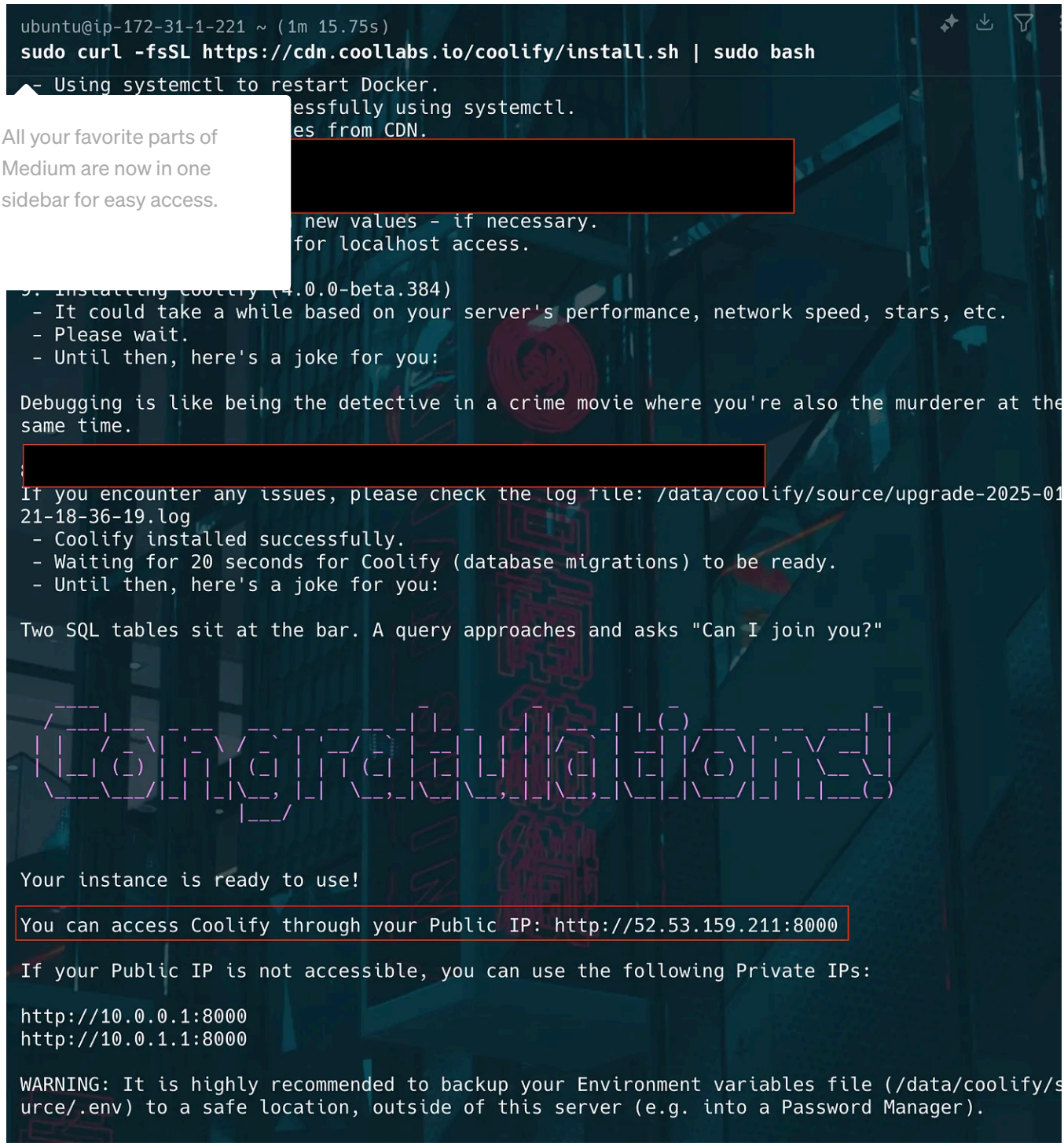
Installing Coolify

Now, let's focus on our Coolify server. SSH into it and run the installation script:

```
sudo curl -fsSL https://cdn.coollabs.io/coolify/install.sh | sudo bash
```

Sit back and enjoy your coffee while the magic happens! Once the installation completes, you'll see a success message with your Coolify URL:

```
🚀 Installation completed successfully!
You can access Coolify through your Public IP: http://yourpublicip:8000
```



Initial Setup

1. Accessing the Dashboard

Navigate to `http://yourpublicip:8000` in your browser. You'll be greeted with the Coolify setup wizard.

2. Creating Your Admin Account

- Enter your email address
- Set a strong password
- Keep these credentials safe — you'll need them for future logins

- When prompted for environment selection, choose “localhost”

[illegible]eys

[illegible]d, navigate to “Keys and Tokens”

You have two options for SSH authentication:

Option 1: Use Existing SSH Keys (Recommended)

1. Display your existing SSH private key:

```
cat path/to/your/ssh/key.pem
```

2. In Coolify:

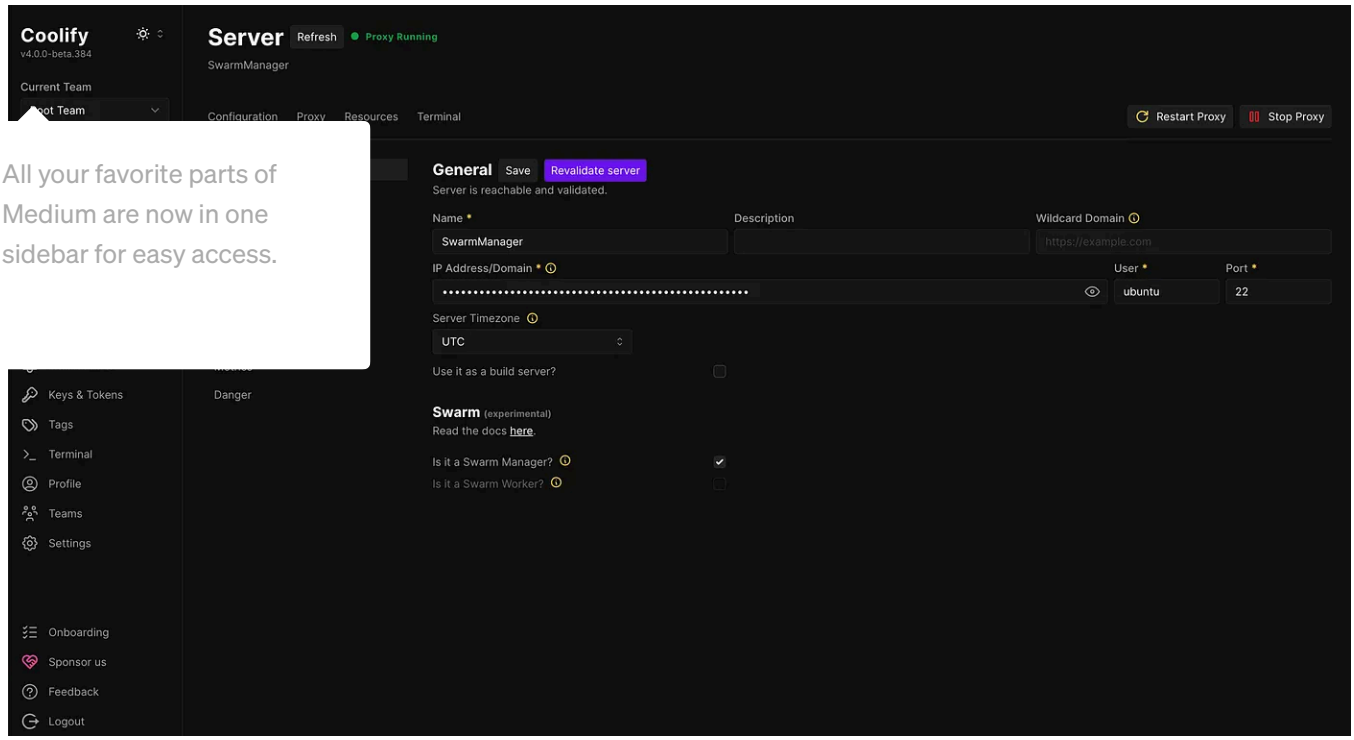
- Choose “Use Existing SSH Key”
- Paste your private key content
- Give your key a memorable name (e.g., “cluster-key”)

Option 2: Generate New Keys in Coolify

1. Select “Generate New SSH Key”
2. Follow the prompts to add the generated public key to your nodes

4. Adding the Swarm Manager

1. Navigate to “Servers” in the dashboard
2. Click “Add”
3. Enter your Swarm Manager’s details
4. Important: Check the “Swarm Manager” checkbox
5. Use the SSH key you just configured
6. Complete the connection process



Important Notes

- **Keep SSH Keys Safe:** Back up any generated keys securely
- **Server Access:** Make sure port 8000 is open in your security group for Coolify dashboard access
- **Authentication:** Store your admin credentials in a secure password manager

Next Steps

With Coolify installed and your Swarm Manager connected, we're ready to:

1. Add worker nodes
2. Set up our first deployment

Keep your terminal handy — in the next section, we'll be configuring the proxy for our swarm environment!

Adding Worker Nodes to Coolify

Before we start deploying stuff, we need to add our worker nodes to Coolify. This is a crucial step that allows Coolify to manage container lifecycles and clean up resources efficiently.

Access Server Management

1. From your Coolify dashboard, click on "Servers" in the navigation menu

2. Click the “Add New Server” button

rver” form:

NS name:

```
ec2-worker1-example.us-west-1.compute.amazonaws.com
```

- Set the username to `ubuntu` (or your EC2 instance’s username)

2. Under SSH Authentication:

- If using existing SSH key (recommended):
- Select “**Use Existing SSH Key**” — Choose the key you added during initial setup
- If using Coolify-generated keys:
- Select “**Use Generated Key**” — Make sure you’ve added the public key to the worker’s **authorized_keys**`

NOTE: Here is where you select “It is a swarm Worker”. You need to do this!

1. Click “Validate”
2. Wait for Coolify to:

- Verify SSH connection
- Install necessary dependencies
- Configure the node

NOTE: YOU Can continue repeat steps to add the second worker node.

Step 4: Verify Node Status

1. Once both workers are added, you should see all three servers in your dashboard:

- Swarm Manager

Worker 1

er shows:

- ✅ Server is reachable and validated

Important Notes

- **Username Configuration:** The default username for Ubuntu EC2 instances is `ubuntu` . If you used a different AMI or modified the user, make sure to use the correct username.
- **SSH Key Management:** Keep your SSH keys secure and backed up. These are crucial for server management.
- **Resource Cleanup:** Having workers properly connected to Coolify ensures efficient resource cleanup when scaling down or removing applications.

What's Next?

Now that we have all our nodes added to Coolify, we're ready to:

1. Verify Setup
2. Ensure Swarm Networking is working
3. Deploy our first test application

Keep going — we're getting to the good part where all this setup starts to pay off!

Remember: A properly configured swarm cluster is the foundation of a reliable container orchestration system. Take a moment to verify everything is working before moving on to

uration and restarting the proxy, let's verify everything is

1. SSH into your swarm manager node
2. Check the service status:

```
sudo docker service ls
```

1. You should see something like:

```
```
ID   NAME   MODE   REPLICAS   IMAGE
nv6e81x5s0oo   coolify-proxy_traefik   replicated   1/1   traefik:v3.1
```
```

1. Check the service details:

```
```
sudo docker service ps nv6e81x5s0oo (this is the ID of the service in the above
```
```

1. Look for output like:

```
```
ID   NAME   IMAGE   NODE   DESIRE
```
```

```
```


ou see `1/1` replicas in the service list, your proxy is running

Verify Docker Swarm Networking

This is a **crucial** part. If Docker swarm networking is not working then you cannot deploy resources on worker nodes, you cannot get to worker nodes from the proxy amongst other things.

To verify networking you should:
On the manager node run:

```
sudo docker network ls
```

You should see something like:

```
NETWORK ID     NAME              DRIVER    SCOPE
11f8a1153e17   bridge            bridge    local
jhkg6c1e0h26   coolify           overlay   swarm
6quwjum9wus2   coolify-overlay   overlay   swarm
8ed3b5518ee8   docker_gwbridge   bridge    local
15b271139fcb   host              host      local
uqgrenyllp2w   ingress           overlay   swarm
7eb222a2c7be   none              null      local
```

> 💡 **Pro Tip:** The `overlay` driver with `swarm` scope means the network can span multiple Docker hosts, enabling communication between containers across your entire cluster. This is essential for Coolify's proxy to route traffic to services running on any node.

Run the same command on your worker nodes:

```
sudo docker network ls
```


ıe overlay networks (`coolify` and `ingress`) on each worker
g or show as `local` scope instead of `swarm` , your swarm
networking isn't configured correctly.

Key Points to Remember

- Coolify network needs to be operating in the swarm scope NOT local scope
- All proxy configurations must be done through Coolify's interface
- The proxy service should always show as running with 1/1 replicas
- Any changes to the proxy configuration require a restart of the service

Next Steps

With our swarm cluster properly connected to Coolify and our proxy running, we're ready to:

1. Connect to Docker Hub for container registry access
2. Link our GitHub repository(optional)
3. Deploy our first application across the swarm

Final Steps: Setting Up Your Repository and First Deployment

We're in the home stretch! Let's get your first application deployed across your swarm cluster.

Option A: Private GitHub Repository Setup

Skip this section if you're using a public repository

1. Navigate to "Projects" in Coolify
2. Create a new project (e.g., "Coolify Demo")
3. Click "Add Resource"
4. Choose "Private GitHub Repository"
5. Select your Swarm Manager

6. Follow the GitHub authorization steps:

v”

ıb prompts

access (all or select repositories)

Option B: Public Repository Setup

1. Create a new project
2. Click “Add Resource”
3. Select “Public Repository”
4. Choose your Swarm Manager as the server
5. Use this test repository: `https://github.com/jvaught01/postman`
6. Click “Check Repository”
7. For Build Pack, select “Nixpacks”

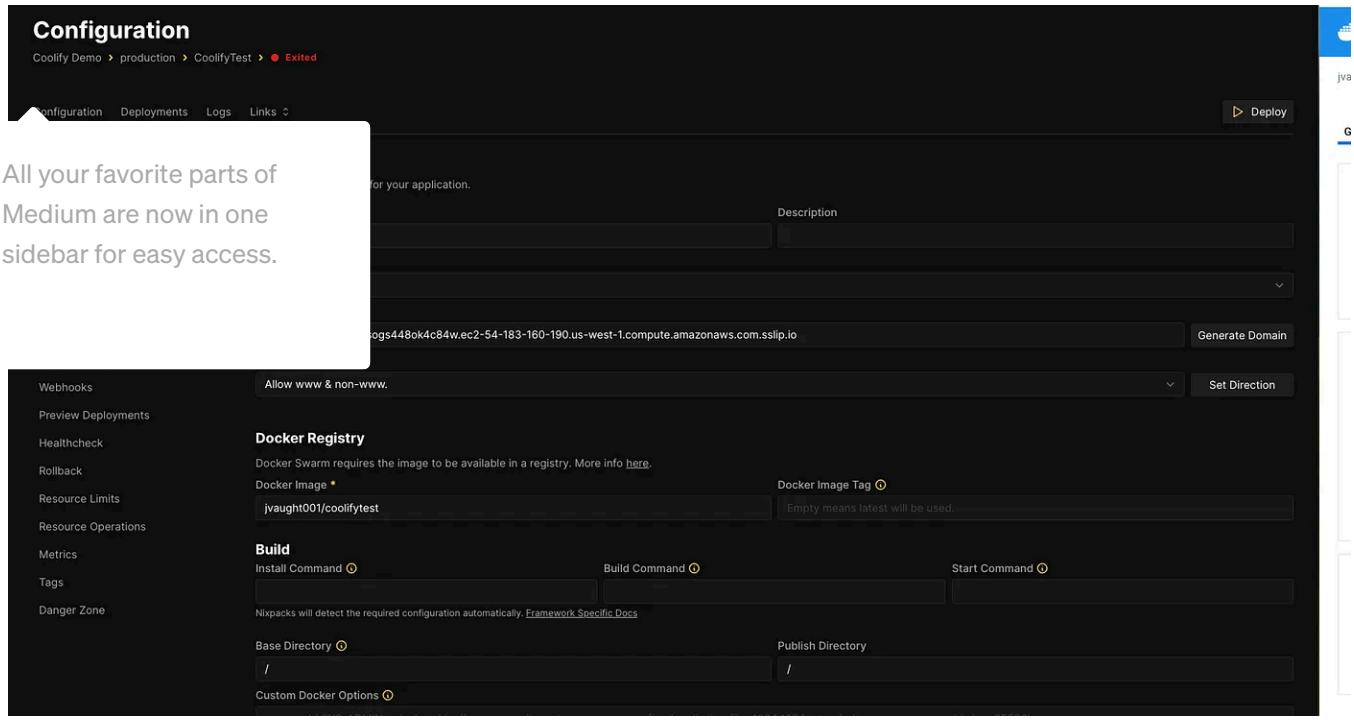
> 💡 **What’s Nixpacks?** Think of it as your deployment Swiss Army knife — it automatically detects your application type and sets up the perfect build environment. It’s what makes one-click deployments possible!

Docker Hub Authentication

1. SSH into each of your servers (Coolify, Manager, and both Workers)
2. On each server, run:
3. `docker login`
4. Enter your Docker Hub credentials when prompted

Configuring Your Deployment

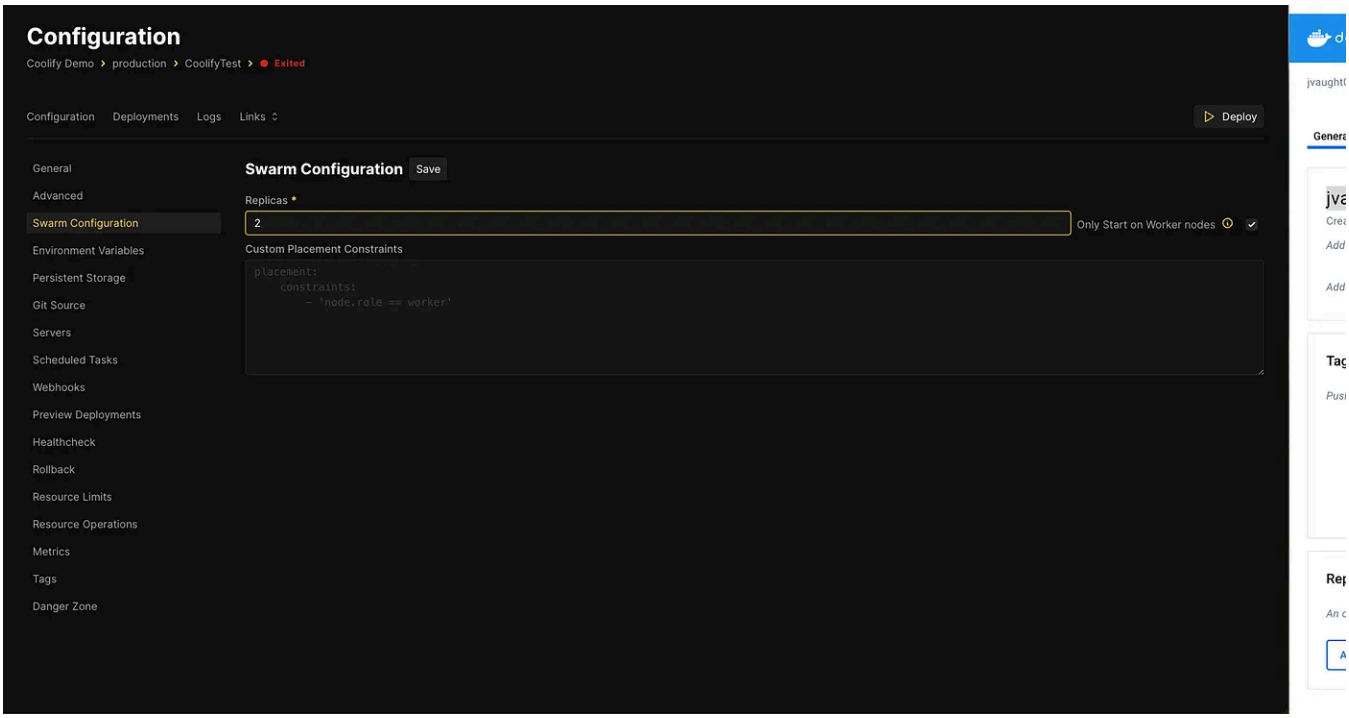
1. In your deployment settings, scroll to “Docker Registry”
2. Enter your Docker image name:
3. `yourusername/yourrepo`



⚠️ **IMPORTANT:** Click “Save” at the top of the page! Missing this step will cause deployment issues.

4. (Optional) Under “Swarm Configuration”:

- Set your desired number of replicas
- Configure any other swarm-specific settings



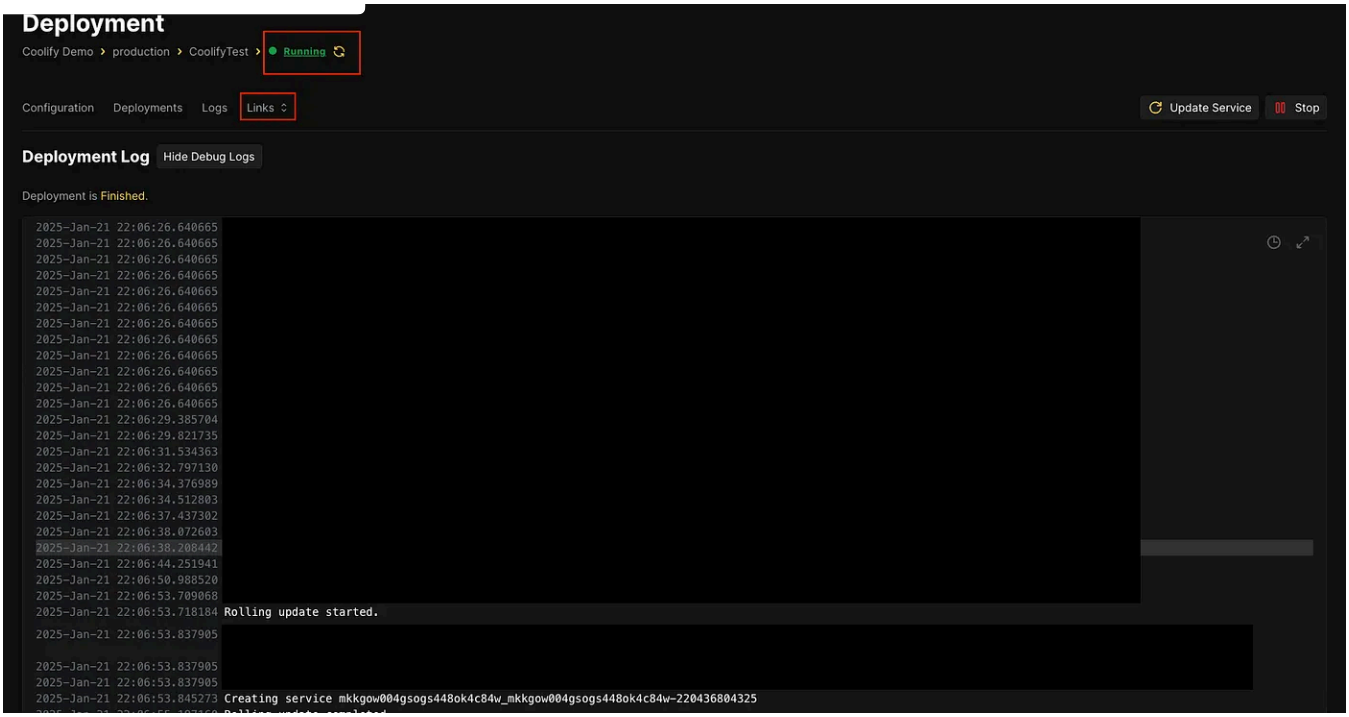
Swarm Replicas

The Moment of Truth: Deployment

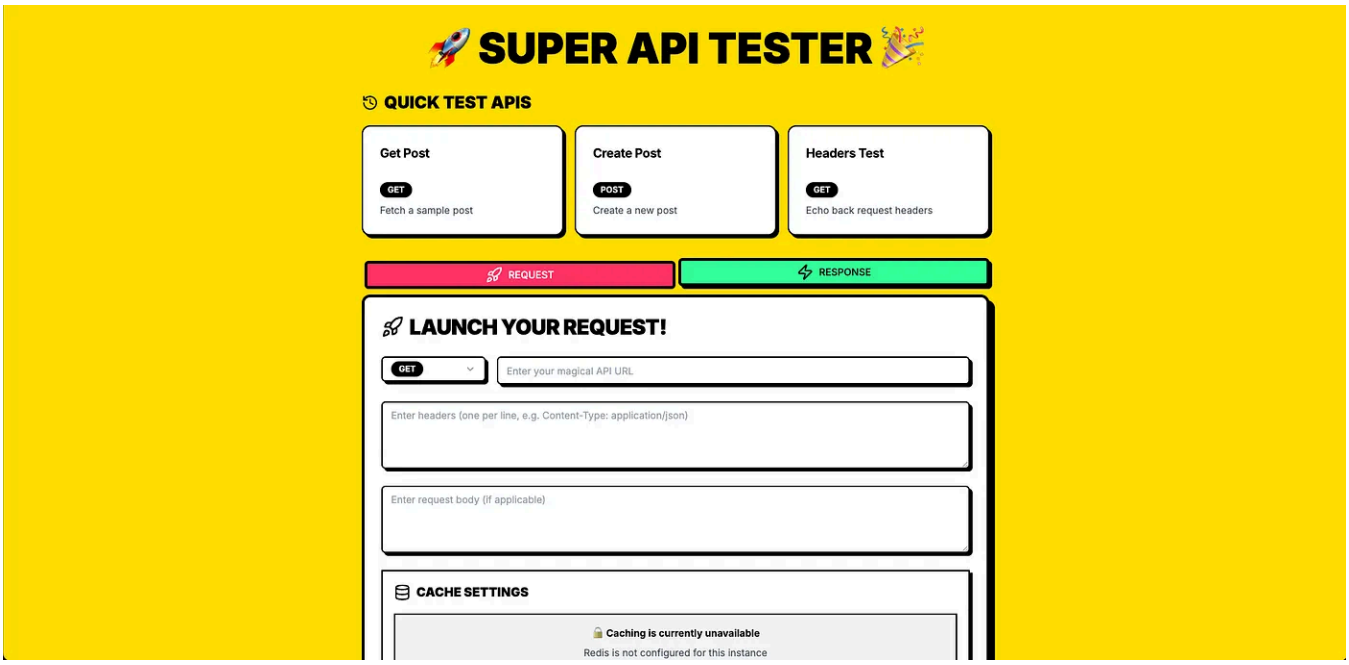
1. Click the “Deploy” button

2. Watch the build logs — you’ll see something like this:


```
g to registry... Deploying to swarm... Rolling Update
```



Deployment



Custom Postman With caching :)

1. Wait for the status indicator to turn green (usually takes 1–2 minutes)

2. Once green, click on “Links”
3. Visit your newly deployed application! 🎉

ONS! 🎊

successfully:

- Set up a **Docker Swarm cluster**
- Configured **Coolify Server**
- Connected your repository
- **Deployed** your first application

Your application is now running across your swarm cluster, with all the benefits of:

- High availability
- Load balancing
- Easy scaling
- Full deployment control

What’s Next?

Now that you have your infrastructure set up, you can:

- Deploy more applications
- Set up continuous deployment
- Configure custom domains
- Explore Coolify’s monitoring features

Remember, you can always scale your application by adjusting the number of replicas in the swarm configuration!

If you found this guide helpful, feel free to **drop a comment, share your experience, or ask any questions** — I’d love to hear how your setup went! **Also, check out my postman test site here: https://superapitester.com**

Happy hosting! 🚀🔥

-JulioV

LINKEDIN HERE

Docker Swarm Cluster Vercel

Follow

Written by Julio Vaught

2 followers · 1 following

All things Engineering. Pretty good at Elite Dangerous.

No responses yet

Eduardo Sánchez

What are your thoughts?

Recommended from Medium

In Coding Nexus by Code Coup

How Adding One Header Cut Our AWS S3 Costs by 80%

We'd been looking at our AWS bill for months, frustrated but also somewhat confused. S3 alone was costing us about $2,400 each month...

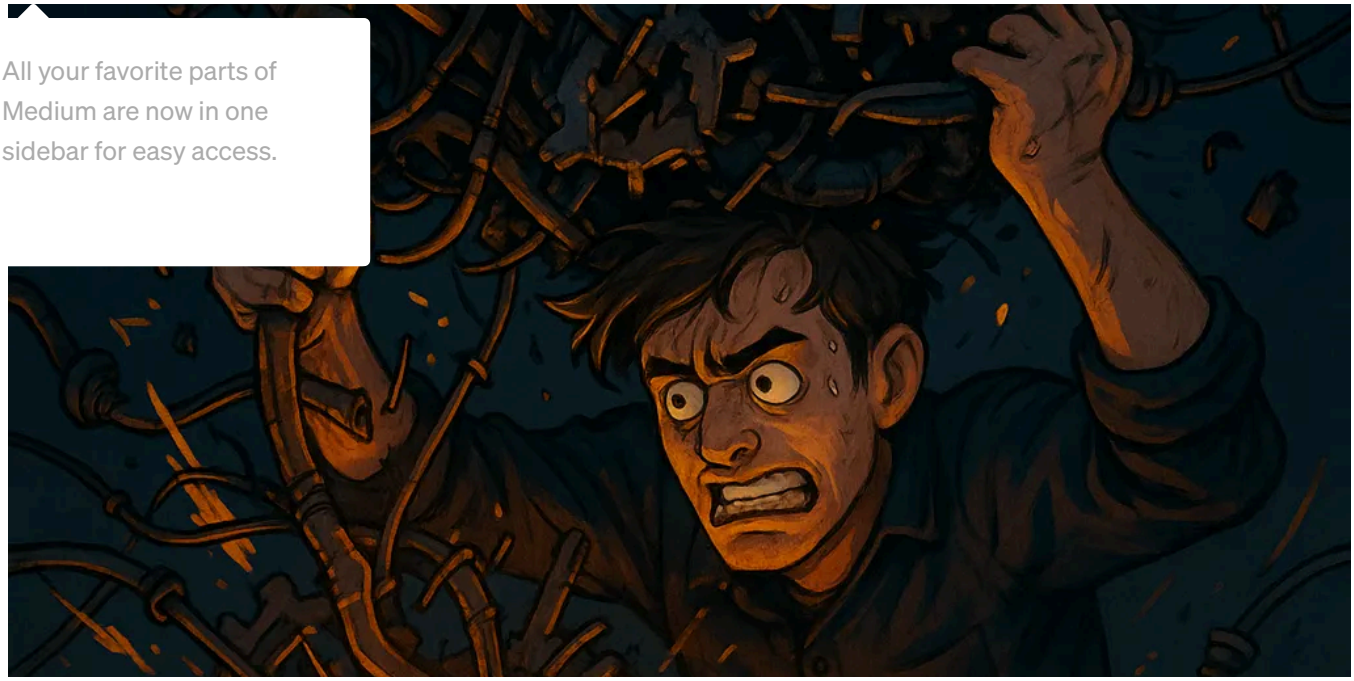
Nov 21 217 2

TAYO

My Linux Workflow - DevOps Toolkit

The Toolkit I use to handle my daily tasks

Nov 12 53

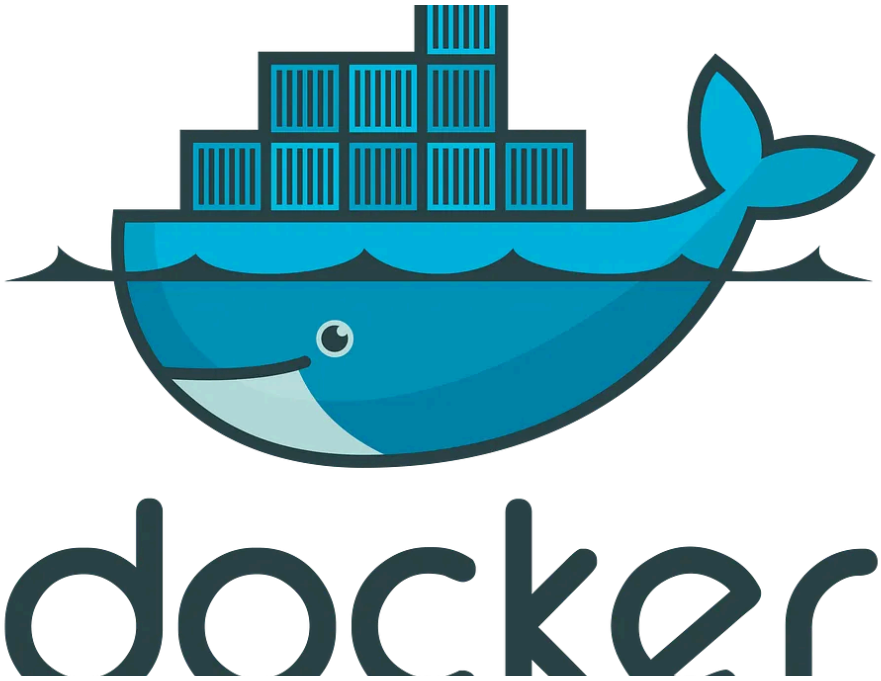


In AWS in Plain English by Adonis

Kubernetes on AWS Is a Mistake—And Here's the Data to Prove It

Kubernetes promised freedom, scale, and simplicity. What most teams actually get is a cost bomb, operational chaos, and a platform nobody...

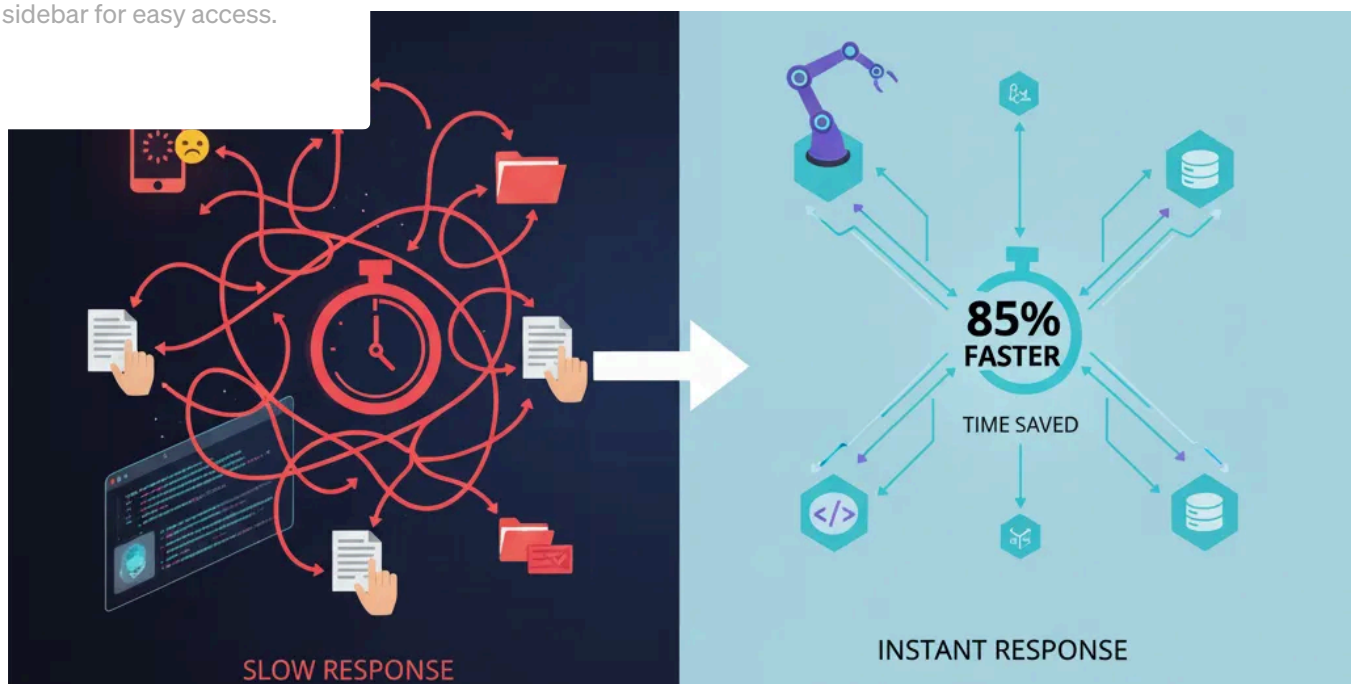
6d ago 26 5



Abhinav

Docker Is Dead—And It's About Time

Docker changed the game when it launched in 2013, making containers accessible and turning “Dockerize it” into a developer catchphrase.

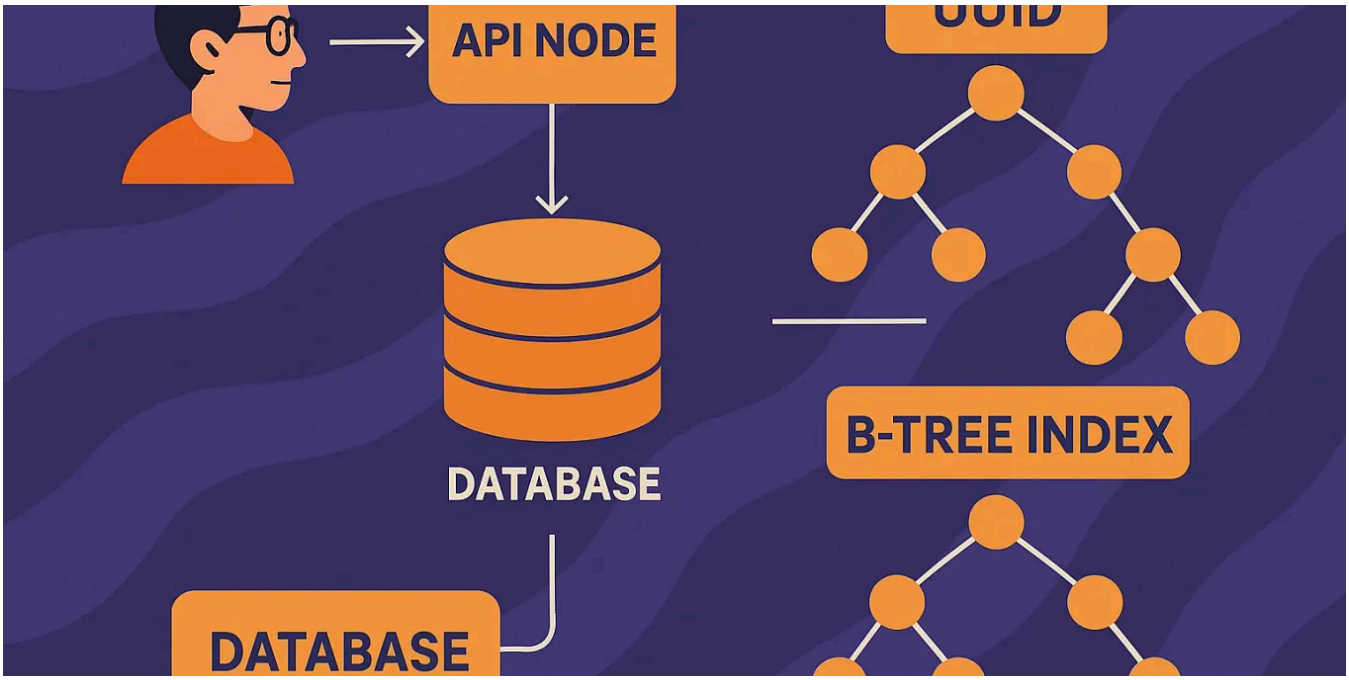


Bhavyansh

12 Small Automation Scripts That Saved Me Hours Every Week

The boring scripts nobody talks about that eliminate daily friction from your development workflow

Nov 22 90 3



The Concurrent Mind

Stop Using UUIDs as Primary Keys. It Is Killing Your Database Throughput

because it is old. It is slow because every new row lands in a random ɔt predict.