Utho Cloud Deployment Handbook

Complete Guide for Vibecoders: From Vercel/Render to Utho VM

Table of Contents

1. Introduction: Your Journey from Managed to Self-Managed

What You're Used To (The Easy Life)

Vercel (Frontend)

- Push code to GitHub → Auto-deploy ✨
- Zero configuration needed
- Automatic SSL, CDN, scaling

Render (Backend + Database)

- Connect repository → Auto-deploy ✨
- Managed PostgreSQL (just works)
- Environment variables via UI

What You're Moving To (More Power, More Responsibility)

Utho VM

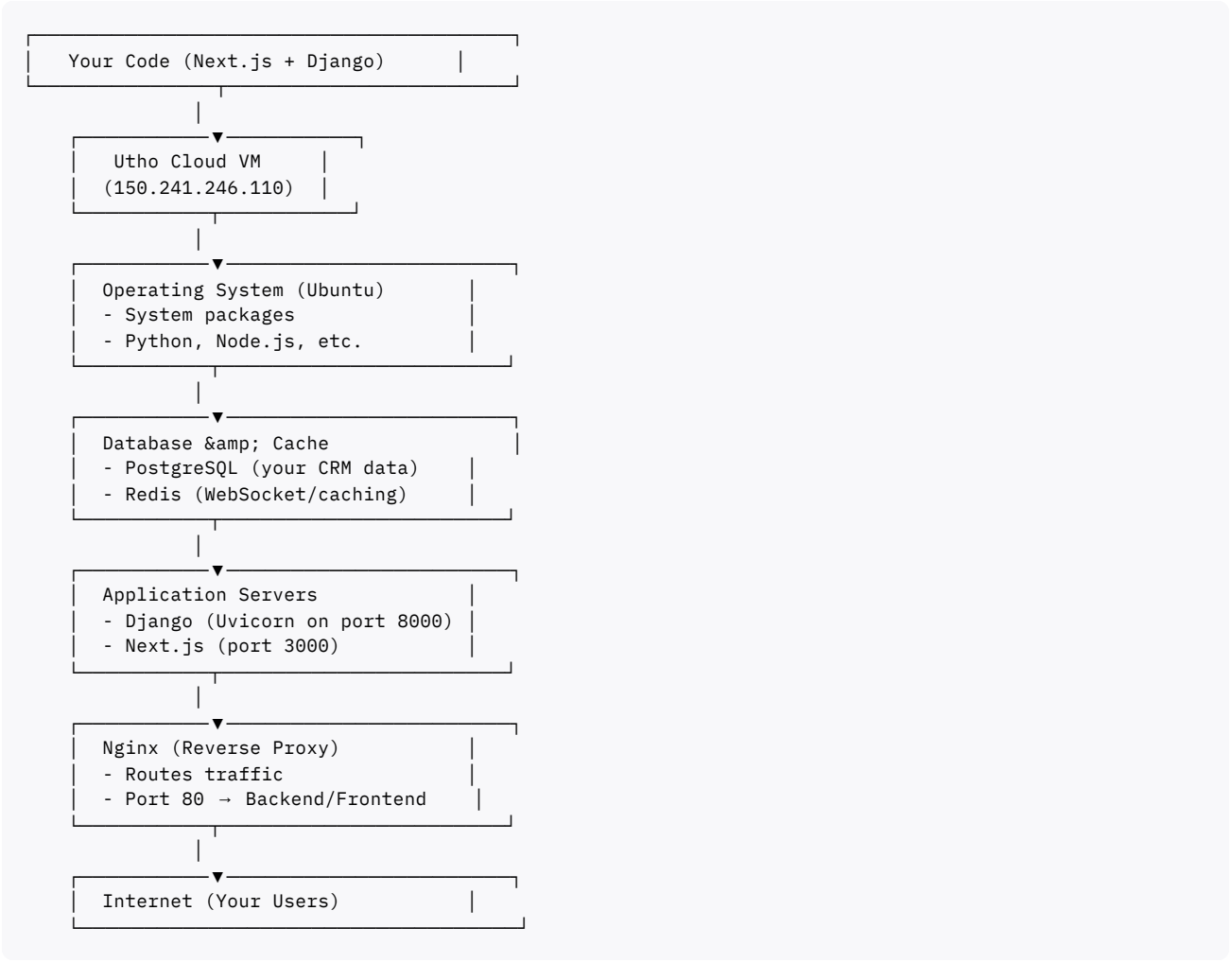
- You are the system administrator now
- Manual setup initially, but full control
- More cost-effective (up to 60% savings)[64][67]
- Deploy anywhere in India with low latency[56][60]

Think of it like:

- Before: Living in a hotel (everything managed)
- Now: Owning an apartment (you manage, but you control everything)

2. Understanding the Big Picture

The Traditional Deployment Stack



Why Each Component?

Component	Purpose	Alternative (What You're Used To)
Ubuntu	Operating System	Managed by Vercel/Render
PostgreSQL	Your CRM database	Render PostgreSQL
Redis	WebSocket support, caching	Managed Redis services
Python/Node	Run your code	Built into Vercel/Render
Nginx	Web server, routes traffic	Handled automatically
Systemd	Auto-start services on boot	Platform handles this

3. Utho Cloud Platform Overview

What is Utho?

Utho is India's leading public cloud provider, offering simple, affordable, and scalable cloud solutions[65][67]. Key features:

- **Deploy in 30 seconds** from their UI[^67]
- **2x faster** than traditional cloud servers[^67]
- **Made in India** with data centers across India and US[64][70]
- **60% cheaper** than hyperscale providers like AWS[64][70]
- **24/7 human support** (not just tickets)[64][70]

Utho Services You'll Use

1. **Cloud Instances (VMs)**: Your virtual server[56][60]
2. **VPC (Virtual Private Cloud)**: Network isolation[63][64]
3. **Elastic Block Storage**: Additional disk space[56][60]
4. **Firewalls**: Security rules[56][60]
5. **Snapshots**: Backups[56][60]

Your Deployment Options on Utho

Option 1: Use Utho UI (Easiest)

- Login to Utho dashboard
- Click "Deploy Cloud Instance"[56][60]
- Select OS (Ubuntu 22.04)
- Choose plan (Basic/CPU/Memory optimized)
- Configure authentication (SSH key recommended)
- Deploy in 30 seconds[62][67]

Option 2: Manual VM Setup (What this guide covers)

- More control over every component
- Better understanding of production architecture
- Easier debugging when issues occur

4. Step-by-Step Deployment Algorithm

Phase 1: Initial VM Setup (10 minutes)

Step 1.1: Get Your VM from Utho

Via Utho Dashboard[56][60]:

1. Login to https://console.utho.com
2. Navigate to "Cloud Instances"
3. Click "Deploy New"
4. Configure:
   - **Datacenter**: Choose closest to your users (Mumbai, Delhi, etc.)
   - **OS**: Ubuntu 22.04 LTS
   - **Plan**: Basic (2 vCPU, 4GB RAM minimum for your CRM)
   - **Storage**: General Storage (SSD)
   - **Auth**: SSH Key (generate in Utho or upload yours)
5. Deploy → Get IP address (e.g., 150.241.246.110)

Step 1.2: Connect to Your VM

What you're doing: Opening a secure terminal connection to your server

Command:

```
ssh root@150.241.246.110
```

Layman explanation:

- `ssh` = Secure Shell (encrypted remote connection)[75][81]
- `root` = Admin user with full control[^75]
- `150.241.246.110` = Your VM's address on the internet

Think of it as: Opening a terminal window that runs commands on your server instead of your computer

First time connection:

- You'll see a warning about host authenticity → Type `yes`
- If using password: Enter the password from Utho dashboard
- If using SSH key: Should connect automatically

Phase 2: Install System Dependencies (15 minutes)

Step 2.1: Update System Packages

Command:

```
apt update &amp;amp;&amp;amp; apt upgrade -y
```

What each part does:

- `apt` = Ubuntu's app store (package manager)[74][76]
- `update` = Refresh list of available software
- `upgrade` = Install newer versions
- `-y` = Say "yes" to all prompts automatically
- `&amp;&amp;` = Run second command only if first succeeds

Why for: SYSTEM - Keeps your server secure and up-to-date[76][79]

Think of it as: Windows Update for Ubuntu

Expected output: List of packages being updated, takes 2-5 minutes

Step 2.2: Install Basic Tools

Command:

```
apt install -y curl wget git build-essential
```

What you're installing:

- `curl` = Download files from internet (like a browser for terminal)[^75]
- `wget` = Another download tool
- `git` = Version control (same as your local Git)
- `build-essential` = Compilers needed to build software from source

Why for: SYSTEM - Basic utilities needed by almost all software

Think of it as: Installing WinRAR, Notepad++, and other essentials on Windows

Step 2.3: Install PostgreSQL Database

Command:

```
apt install -y postgresql postgresql-contrib libpq-dev
```

What you're installing:

- `postgresql` = The database server itself[76][79][^82]
- `postgresql-contrib` = Extra features and extensions
- `libpq-dev` = Development files (needed for Python to talk to PostgreSQL)[76][88]

Why for: BACKEND - Your Django CRM needs a database to store customer data, sales records, etc.

Think of it as: Installing PostgreSQL locally, but on your server

Verification:

```
systemctl status postgresql
```

Should show "active (running)"[76][79]

Step 2.4: Start and Enable PostgreSQL

Command:

```
systemctl enable postgresql
systemctl start postgresql
```

What each part does:

- `systemctl` = System service manager (like Windows Services)[74][77][^80]
- `enable` = Auto-start when server boots[74][80][^83]
- `start` = Start right now[74][77]
- `postgresql` = Name of the service

Why for: BACKEND - Ensures database runs automatically after server restarts

Think of it as: Setting PostgreSQL to "Start with Windows"

Step 2.5: Create Database and User

Command:

```
sudo -u postgres psql &lt;&lt; EOF
CREATE DATABASE jewellery_crm;
CREATE USER crm_user WITH PASSWORD 'SecurePassword123!';
ALTER ROLE crm_user SET client_encoding TO 'utf8';
ALTER ROLE crm_user SET default_transaction_isolation TO 'read committed';
ALTER ROLE crm_user SET timezone TO 'UTC';
GRANT ALL PRIVILEGES ON DATABASE jewellery_crm TO crm_user;
\q
EOF
```

What each part does:

- `sudo -u postgres` = Run command as the "postgres" admin user[76][85]
- `psql` = PostgreSQL command-line interface[76][85]
- `CREATE DATABASE` = Makes new database for your CRM
- `CREATE USER` = Creates login credentials for Django

- `GRANT ALL PRIVILEGES` = Gives user full access to the database[85][91]

Why for: BACKEND - Django needs a dedicated database and user to store data

Think of it as: What you'd do in pgAdmin or Render's database UI, but via commands

Step 2.6: Install Redis

Command:

```
apt install -y redis-server
```

What you're installing:

- `redis-server` = In-memory data store (cache)[^79]

Why for: BACKEND - Your CRM uses Django Channels for telecalling WebSockets. Redis is required for channel layers.

Think of it as: Installing Redis locally for Django Channels development

Step 2.7: Configure Redis

Commands:

```
sed -i 's/supervised no/supervised systemd/' /etc/redis/redis.conf
sed -i 's/# maxmemory &lt;bytes&gt;/maxmemory 256mb/' /etc/redis/redis.conf
sed -i 's/# maxmemory-policy noeviction/maxmemory-policy allkeys-lru/' /etc/redis/redis.conf
```

What each part does:

- `sed` = Stream editor (find and replace in files)
- `-i` = Edit file in-place (save changes)
- `s/old/new/` = Replace "old text" with "new text"
- `/etc/redis/redis.conf` = Redis configuration file

Why for: BACKEND - Configures Redis for production use with systemd and memory limits

Think of it as: Editing settings.py, but using commands instead of opening the file

Step 2.8: Start and Enable Redis

Commands:

```
systemctl enable redis-server
systemctl start redis-server
```

Why for: BACKEND - Auto-starts Redis on boot

Verification:

```
redis-cli ping
```

Should return "PONG"

Step 2.9: Install Python 3.11

Commands:

```
add-apt-repository -y ppa:deadsnakes/ppa
apt update
apt install -y python3.11 python3.11-venv python3.11-dev
```

What each part does:

- `add-apt-repository` = Adds external software source[^82]
- `ppa:deadsnakes/ppa` = Repository with newer Python versions
- `python3.11-venv` = Virtual environment support
- `python3.11-dev` = Development headers for pip packages[^82]

Why for: BACKEND - Django runs on Python. You need 3.11 for modern features.

Think of it as: Installing Python on Windows, but ensuring correct version

Step 2.10: Set Python 3.11 as Default

Command:

```
update-alternatives --install /usr/bin/python3 python3 /usr/bin/python3.11 1
```

Why for: BACKEND - Makes `python3` command use version 3.11

Think of it as: Setting default Python version in VS Code

Verification:

```
python3 --version
```

Should show "Python 3.11.x"

Step 2.11: Install Node.js 20

Commands:

```
curl -fsSL https://deb.nodesource.com/setup_20.x | bash -
apt install -y nodejs
```

What you're doing:

- First line: Downloads and runs Node.js setup script[1] [2]
- Second line: Installs Node.js and npm

Why for: FRONTEND - Next.js requires Node.js to build and run

Think of it as: Installing Node.js on your PC, but on the server

Verification:

```
node --version  # Should show v20.x
npm --version
```

Step 2.12: Install Nginx

Commands:

```
apt install -y nginx
systemctl enable nginx
systemctl start nginx
```

What you're installing:

- `nginx` = Web server and reverse proxy[3] [^44]

Why for: BOTH - Routes internet traffic to Django (port 8000) and Next.js (port 3000)

Why you need Nginx:

1. **Security**: Protects your apps from malicious traffic[4] [5]
2. **Performance**: Serves static files 10-20x faster[6]
3. **Single Entry Point**: All traffic goes through port 80, then routes internally[7] [8]
4. **WebSocket Support**: Your telecalling feature needs proper WebSocket routing[9] [10]

Think of it as: A smart router that directs visitors to the right app

Verification:

```
systemctl status nginx
```

Should show "active (running)"

Open browser: http://150.241.246.110 → Should see "Welcome to Nginx"

Phase 3: Transfer Your Code (10 minutes)

Step 3.1: Clone from GitHub (Recommended)

Commands:

```
cd /var/www
git clone https://github.com/yourusername/CRM_FINAL.git
```

What you're doing:

- `cd /var/www` = Change to standard web application directory
- `git clone` = Download your code from GitHub (same as in VS Code)[^66]

Why for: BOTH - Gets your code onto the server

Alternative: Upload from your Windows PC

If not using Git:

```
# From Windows PowerShell:
scp -r K:\Master\CRM_FINAL root@150.241.246.110:/var/www/
```

What SCP does:

- `scp` = Secure Copy (like FTP, but encrypted)[^75]
- `-r` = Recursive (copy entire folder)
- Copies from your PC to server[^75]

Step 3.2: Set Permissions

Commands:

```
cd /var/www/CRM_FINAL
chown -R root:root .
chmod -R 755 .
```

What each part does:

- `chown` = Change owner
- `chmod` = Change permissions (read/write/execute)[75][78]
- `755` = Owner can do everything, others can read/execute

Why for: SYSTEM - Ensures proper file permissions for security

Phase 4: Backend Setup (20 minutes)

Step 4.1: Create Virtual Environment

Commands:

```
cd /var/www/CRM_FINAL/backend
python3.11 -m venv venv
source venv/bin/activate
```

What you're doing:

- Creates isolated Python environment (same as in Cursor)
- `source venv/bin/activate` = Activates venv (Windows: `venv\Scripts\activate`)

Why for: BACKEND - Isolates Django dependencies from system Python

Think of it as: Exactly same as your local venv workflow

Visual cue: Your prompt should now show `(venv)` at the start

Step 4.2: Install Python Dependencies

Commands:

```
pip install --upgrade pip
pip install -r requirements.txt
pip install gunicorn uvicorn[standard]
```

What you're installing:

- `requirements.txt` = Your Django project dependencies
- `gunicorn` = Production WSGI server (serves Django HTTP) [11]
- `uvicorn` = Production ASGI server (serves Django with WebSocket support) [11] [^47]

Why for: BACKEND - Django needs these to run in production (Uvicorn for your telecalling WebSockets)

Think of it as: `pip install` like you do in Cursor, but on server

Expected time: 2-5 minutes depending on dependencies

Step 4.3: Create Environment File

Command:

```
cat &gt; /var/www/CRM_FINAL/backend/.env &lt;&lt; 'EOF'
DEBUG=False
SECRET_KEY=jewelry-crm-2024-production-secure-key-8f7e6d5c4b3a2918
ALLOWED_HOSTS=150.241.246.110,localhost,127.0.0.1

DB_ENGINE=django.db.backends.postgresql
DB_NAME=jewellery_crm
DB_USER=crm_user
DB_PASSWORD=SecurePassword123!
DB_HOST=localhost
DB_PORT=5432

REDIS_HOST=127.0.0.1
REDIS_PORT=6379

CORS_ALLOWED_ORIGINS=http://150.241.246.110,http://localhost:3000
CSRF_TRUSTED_ORIGINS=http://150.241.246.110,http://localhost:3000

STATIC_URL=/static/
STATIC_ROOT=/var/www/CRM_FINAL/backend/staticfiles
MEDIA_URL=/media/
```

```
MEDIA_ROOT=/var/www/CRM_FINAL/backend/media
EOF
```

What you're doing:

- `cat >` = Write text to file
- `<< 'EOF'` = Multi-line input until you type `EOF`

Why for: BACKEND - Django needs these settings to connect to database, Redis, etc.

Think of it as: Creating .env file (like in Cursor, but via terminal)

Important settings explained:

- `DEBUG=False` = Production mode (never True in production!)
- `ALLOWED_HOSTS` = IP addresses allowed to access Django
- `DB_*` = PostgreSQL connection details (from Step 2.5)
- `REDIS_*` = Redis connection for Django Channels
- `CORS_*` = Allow Next.js to call Django API

Step 4.4: Run Database Migrations

Commands:

```
python manage.py migrate
```

What you're doing:

- Creates database tables from your Django models

Why for: BACKEND - Sets up your CRM database schema

Think of it as: Same Django command you run locally

Expected output: List of migrations being applied (e.g., "Applying contenttypes.0001...")

Step 4.5: Create Admin User

Command:

```
python manage.py createsuperuser --username admin --email admin@jewelrycrm.com
```

What you're doing:

- Creates Django admin account

Why for: BACKEND - Access /admin panel to manage data

Think of it as: Same as running locally, but for production

You'll be prompted: Enter password (type it twice)

Step 4.6: Collect Static Files

Command:

```
python manage.py collectstatic --noinput
```

What you're doing:

- Gathers all CSS/JS/images from Django and apps into one folder
- Nginx will serve these files directly (faster than Django)

Why for: BACKEND - Static files (admin CSS, etc.) need to be accessible to Nginx

Think of it as: Building Django's static assets for production

Step 4.7: Create Directories

Commands:

```
mkdir -p /var/www/CRM_FINAL/backend/logs
mkdir -p /var/www/CRM_FINAL/backend/media
chmod -R 755 /var/www/CRM_FINAL/backend/staticfiles
chmod -R 755 /var/www/CRM_FINAL/backend/media
```

Why for: BACKEND - Logs and user-uploaded files need dedicated folders

Phase 5: Frontend Setup (15 minutes)

Step 5.1: Install Dependencies

Commands:

```
cd /var/www/CRM_FINAL/jewellery-crm
npm install --production=false
```

What you're doing:

- Installs all Node modules (same as in Cursor)
- `--production=false` = Installs dev dependencies too (needed for build)

Why for: FRONTEND - Next.js needs dependencies to build

Expected time: 3-5 minutes

Step 5.2: Create Environment File

Command:

```
cat &gt; /var/www/CRM_FINAL/jewellery-crm/.env.local &lt;&lt; 'EOF'
NEXT_PUBLIC_API_URL=http://150.241.246.110:8000
NODE_ENV=production
EOF
```

Why for: FRONTEND - Next.js needs to know where Django backend is

Key settings:

- `NEXT_PUBLIC_API_URL` = Django API endpoint
- `NODE_ENV=production` = Production mode

Step 5.3: Build Next.js

Command:

```
npm run build
```

What you're doing:

- Builds Next.js for production (what Vercel does automatically) [1] [12]
- Creates optimized bundles, static pages, etc.

Why for: FRONTEND - Production build is much faster than dev mode

Expected time: 2-5 minutes

Expected output: "Build completed successfully" with page routes

Phase 6: Systemd Services (15 minutes)

What are Systemd Services?

- Background processes that auto-start on boot[74][77][^80]
- Like Windows Services, but for Linux
- Systemd will restart your apps if they crash[74][80]

Step 6.1: Create Backend Service

Command:

```
cat &gt; /etc/systemd/system/crm-backend.service &lt;&lt; 'EOF'
[Unit]
Description=Jewellery CRM Backend (Django ASGI with Uvicorn)
After=network.target postgresql.service redis-server.service
Requires=postgresql.service redis-server.service

[Service]
Type=notify
User=root
Group=root
WorkingDirectory=/var/www/CRM_FINAL/backend
Environment="PATH=/var/www/CRM_FINAL/backend/venv/bin"
EnvironmentFile=/var/www/CRM_FINAL/backend/.env
ExecStart=/var/www/CRM_FINAL/backend/venv/bin/uvicorn core.asgi:application \
    --host 0.0.0.0 \
    --port 8000 \
    --workers 2 \
    --proxy-headers \
    --log-level warning
Restart=always
RestartSec=10
StandardOutput=append:/var/www/CRM_FINAL/backend/logs/backend.log
StandardError=append:/var/www/CRM_FINAL/backend/logs/backend-error.log

[Install]
WantedBy=multi-user.target
EOF
```

What this file does:

- Tells systemd: "Run Django with Uvicorn, restart if it crashes"[74][77][^86]
- `After=...` = Wait for network and databases to be ready[74][80]
- `ExecStart=...` = Command to run Django
- `Restart=always` = Auto-restart on failure[74][80]
- `--workers 2` = Run 2 Uvicorn processes (for parallel requests)

Why for: BACKEND - Keeps Django running 24/7, auto-restarts on crash

Think of it as: Installing Django as a Windows Service

Step 6.2: Start Backend Service

Commands:

```
systemctl daemon-reload
systemctl enable crm-backend.service
systemctl start crm-backend.service
systemctl status crm-backend.service
```

What each command does:

- `daemon-reload` = Refresh systemd's service list[74][80][^83]
- `enable` = Auto-start on boot[74][77][^80]
- `start` = Start right now[74][77]
- `status` = Check if it's running[74][77]

Expected output: Should show "active (running)" in green

If it fails: Check logs:

```
journalctl -u crm-backend.service -n 50
```

Step 6.3: Create Frontend Service

Command:

```
cat &gt; /etc/systemd/system/crm-frontend.service &lt;&lt; 'EOF'
[Unit]
Description=Jewellery CRM Frontend (Next.js)
After=network.target

[Service]
Type=simple
User=root
Group=root
WorkingDirectory=/var/www/CRM_FINAL/jewellery-crm
Environment="PATH=/usr/bin:/usr/local/bin"
Environment="NODE_ENV=production"
ExecStart=/usr/bin/npm start
Restart=always
RestartSec=10
StandardOutput=append:/var/www/CRM_FINAL/jewellery-crm/logs/frontend.log
StandardError=append:/var/www/CRM_FINAL/jewellery-crm/logs/frontend-error.log

[Install]
WantedBy=multi-user.target
EOF
```

Why for: FRONTEND - Keeps Next.js running 24/7

Step 6.4: Start Frontend Service

Commands:

```
mkdir -p /var/www/CRM_FINAL/jewellery-crm/logs
systemctl daemon-reload
systemctl enable crm-frontend.service
systemctl start crm-frontend.service
systemctl status crm-frontend.service
```

Expected output: Should show "active (running)"

Phase 7: Nginx Configuration (15 minutes)

Step 7.1: Create Nginx Config

Command:

```
cat &gt; /etc/nginx/sites-available/crm-app &lt;&lt; 'EOF'
upstream django_backend {
    server 127.0.0.1:8000;
}

upstream nextjs_frontend {
```

```
    server 127.0.0.1:3000;
}

server {
    listen 80;
    server_name 150.241.246.110;

    client_max_body_size 100M;

    # Backend API
    location /api/ {
        proxy_pass http://django_backend;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto $scheme;
        proxy_redirect off;
    }

    # Django Admin
    location /admin/ {
        proxy_pass http://django_backend;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto $scheme;
    }

    # WebSocket for telecalling
    location /ws/ {
        proxy_pass http://django_backend;
        proxy_http_version 1.1;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection "upgrade";
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto $scheme;
        proxy_read_timeout 86400;
    }

    # Static files
    location /static/ {
        alias /var/www/CRM_FINAL/backend/staticfiles/;
        expires 30d;
        add_header Cache-Control "public, immutable";
    }

    # Media files
    location /media/ {
        alias /var/www/CRM_FINAL/backend/media/;
        expires 30d;
        add_header Cache-Control "public";
    }

    # Next.js Frontend
    location / {
        proxy_pass http://nextjs_frontend;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto $scheme;
        proxy_redirect off;
    }
}
EOF
```

What this does:

- Listens on port 80 (public internet)[3][^44]
- Routes `/api/*` to Django (port 8000)[7][8]

- Routes `/ws/*` to Django WebSockets[9] [10][^47]
- Routes `/` to Next.js (port 3000)[7] [8]
- Serves static files directly (fast!)[6]

Why for: BOTH - Single entry point for all traffic, routes to correct app

Think of it as: Smart traffic director

Step 7.2: Enable and Test Nginx

Commands:

```
ln -sf /etc/nginx/sites-available/crm-app /etc/nginx/sites-enabled/
rm -f /etc/nginx/sites-enabled/default
nginx -t
systemctl reload nginx
```

What each command does:

- `ln -sf` = Create symbolic link (like a shortcut)
- `rm` = Remove default Nginx site
- `nginx -t` = Test configuration for errors
- `reload` = Apply new configuration[^83]

Expected output: "nginx: configuration file test is successful"

Phase 8: Firewall Setup (5 minutes)

Step 8.1: Configure UFW

Commands:

```
apt install -y ufw
ufw default deny incoming
ufw default allow outgoing
ufw allow 22/tcp
ufw allow 80/tcp
ufw allow 443/tcp
ufw --force enable
ufw status verbose
```

What you're doing:

- Installing firewall
- Block all incoming traffic except SSH (22), HTTP (80), HTTPS (443)
- Allow all outgoing traffic

Why for: SYSTEM - Security! Only allow necessary ports

Critical: Allow port 22 (SSH) first, or you'll lock yourself out!

Expected output: List of allowed ports

Phase 9: Testing & Verification (10 minutes)

Test 1: Backend API

```
curl http://localhost:8000/api/health/
curl http://150.241.246.110/api/health/
```

Expected: JSON response from Django

Test 2: Frontend

```
curl http://localhost:3000
curl http://150.241.246.110/
```

Expected: HTML from Next.js

Test 3: Admin Panel

Open browser: http://150.241.246.110/admin/
Login with superuser credentials from Step 4.5

Test 4: Database

```
sudo -u postgres psql jewellery_crm
\dt  # List tables
\q   # Exit
```

Test 5: Redis

```
redis-cli ping
```

Expected: "PONG"

Phase 10: Monitoring & Logs

Check Service Status

```
systemctl status crm-backend.service crm-frontend.service nginx postgresql redis-server
```

View Backend Logs

```
tail -f /var/www/CRM_FINAL/backend/logs/backend.log
# Or
journalctl -u crm-backend.service -f
```

View Frontend Logs

```
tail -f /var/www/CRM_FINAL/jewellery-crm/logs/frontend.log
# Or
journalctl -u crm-frontend.service -f
```

View Nginx Logs

```
tail -f /var/log/nginx/access.log
tail -f /var/log/nginx/error.log
```

5. Complete Command Reference

Essential System Commands

Command	What It Does	Example
`ssh user@ip`	Connect to remote server[^75][81]	`ssh root@150.241.246.110`
`exit`	Disconnect from server[^75]	`exit`
`ls`	List files[^78]	`ls -la`
`cd`	Change directory[^78]	`cd /var/www`
`pwd`	Show current directory[^78]	`pwd`
`mkdir`	Create directory[^78]	`mkdir logs`
`rm`	Delete file/folder[^78]	`rm file.txt`
`cp`	Copy file[^78]	`cp file.txt backup.txt`
`mv`	Move/rename file[^78]	`mv old.txt new.txt`
`cat`	View file contents[^78]	`cat .env`
`nano`	Edit file[^78]	`nano settings.py`
`chmod`	Change permissions[^75][78]	`chmod 755 file.sh`
`chown`	Change owner[^75][78]	`chown root:root file`

Systemd Service Commands[^74][77][^80][83]

Command	What It Does	Example
`systemctl start SERVICE`	Start service now	`systemctl start crm-backend`
`systemctl stop SERVICE`	Stop service	`systemctl stop crm-backend`
`systemctl restart SERVICE`	Restart service	`systemctl restart crm-backend`
`systemctl status SERVICE`	Check service status	`systemctl status crm-backend`
`systemctl enable SERVICE`	Auto-start on boot	`systemctl enable crm-backend`
`systemctl disable SERVICE`	Don't auto-start	`systemctl disable crm-backend`
`systemctl daemon-reload`	Refresh service files	After editing .service files
`journalctl -u SERVICE -f`	View live logs	`journalctl -u crm-backend -f`
`journalctl -u SERVICE -n 50`	Last 50 log lines	For troubleshooting

PostgreSQL Commands[76][79][^85]

Command	What It Does	Example
`sudo -u postgres psql`	Open PostgreSQL shell	Access as postgres user
`\l`	List databases	Inside psql
`\dt`	List tables	Inside psql

Command	What It Does	Example
`\du`	List users	Inside psql
`\q`	Exit psql	Exit shell
`CREATE DATABASE name;`	Create database	In psql
`CREATE USER name WITH PASSWORD 'pass';`	Create user	In psql
`GRANT ALL PRIVILEGES ON DATABASE db TO user;`	Grant permissions	In psql

Nginx Commands[3][44][83]

Command	What It Does	Example
`nginx -t`	Test configuration	Before reloading
`systemctl reload nginx`	Apply config changes	After editing config
`systemctl restart nginx`	Full restart	If reload doesn't work
`tail -f /var/log/nginx/error.log`	View error logs	Troubleshooting

Git Commands (For Updates)

Command	What It Does	Example
`git pull origin main`	Get latest code	From GitHub
`git status`	Check current state	See changes
`git log`	View commit history	Recent changes

6. CI/CD: Manual vs Automated

Current State: 100% Manual ✖

Every time you update code:

1. SSH into server
2. `cd /var/www/CRM_FINAL`
3. `git pull origin main`
4. Backend: Install dependencies, migrate, collectstatic, restart
5. Frontend: Install dependencies, build, restart
6. **Takes 10-15 minutes, error-prone**

Solution: GitHub Actions (Recommended for You) ✓

Why GitHub Actions?

- Familiar workflow (push to GitHub, like Vercel)[13][^43]
- Visual feedback in GitHub UI[^43]
- Runs automatically on push[13]

- Free for public repos, 2000 minutes/month for private [13]

Setting Up GitHub Actions

Step 1: Generate SSH Key on Server

```
ssh-keygen -t ed25519 -C "github-deploy" -f ~/.ssh/github_deploy
cat ~/.ssh/github_deploy.pub &gt;&gt; ~/.ssh/authorized_keys
cat ~/.ssh/github_deploy
```

Copy the private key output (starts with `-----BEGIN OPENSSH PRIVATE KEY-----`)

Step 2: Add to GitHub Secrets

1. Go to your repo on GitHub
2. Settings → Secrets and variables → Actions
3. Click "New repository secret"
4. Add three secrets:

Name	Value
`SSH_PRIVATE_KEY`	The private key you copied
`SERVER_IP`	`150.241.246.110`
`SERVER_USER`	`root`

Step 3: Create Workflow File

In your repo, create `.github/workflows/deploy.yml`:

```
name: Deploy to Utho VM

on:
  push:
    branches: [main]

jobs:
  deploy:
    runs-on: ubuntu-latest

    steps:
      - name: Deploy to Production
        uses: appleboy/ssh-action@v1.0.0
        with:
          host: ${{ secrets.SERVER_IP }}
          username: ${{ secrets.SERVER_USER }}
          key: ${{ secrets.SSH_PRIVATE_KEY }}
          script: |
            cd /var/www/CRM_FINAL
            git pull origin main

            # Backend
            cd backend
            source venv/bin/activate
            pip install -q -r requirements.txt
            python manage.py migrate
            python manage.py collectstatic --noinput
            sudo systemctl restart crm-backend.service

            # Frontend
            cd ../jewellery-crm
            npm install --silent
            npm run build
            sudo systemctl restart crm-frontend.service
```

```
            echo "✅ Deployment completed!"
```

Step 4: Test It!

1. Commit and push this file to GitHub
2. Make any code change
3. Push to `main` branch
4. Go to GitHub → Actions tab
5. Watch deployment happen automatically!

Your new workflow:

```
You (in Cursor) → git push → GitHub Actions → Auto-deploy to Utho
```

Same as Vercel/Render!

Alternative: Quick Deploy Script (Simpler)

If you prefer manual SSH but faster:

On Server:

```
cat &gt; /usr/local/bin/deploy-crm.sh &lt;&lt; 'EOF'
#!/bin/bash
set -e

echo " Starting deployment..."

cd /var/www/CRM_FINAL
git pull origin main

# Backend
echo " Deploying Backend..."
cd backend
source venv/bin/activate
pip install -q -r requirements.txt
python manage.py migrate --noinput
python manage.py collectstatic --noinput
systemctl restart crm-backend.service
echo "✅ Backend deployed"

# Frontend
echo " Deploying Frontend..."
cd ../jewellery-crm
npm install --silent
npm run build
systemctl restart crm-frontend.service
echo "✅ Frontend deployed"

echo " Deployment completed!"
EOF

chmod +x /usr/local/bin/deploy-crm.sh
```

Usage:

```
ssh root@150.241.246.110
deploy-crm.sh
```

One command instead of 11! ✨

7. Deployment Flowchart

[See attached flowchart image showing complete deployment process from login to verification]

Key Decision Points

1. **Storage Type**: General vs EBS → Affects authentication setup flow[56][60]
2. **Image Type**: OS vs Marketplace → Determines pre-installed software[56][60]
3. **Authentication**: Password vs SSH Key → SSH key more secure[56][60]

8. Troubleshooting Guide

Service Won't Start

Symptom: `systemctl status` shows "failed"

Solution:

```
journalctl -u SERVICE_NAME -n 50
```

Common issues:

- Wrong file paths in .service file
- Environment variables missing
- Port already in use

Port Already in Use

Symptom: "Address already in use"

Solution:

```
netstat -tuln | grep :8000
kill -9 &lt;PID&gt;
```

Database Connection Failed

Symptom: Django can't connect to PostgreSQL

Solution:

```
sudo -u postgres psql
\l  # Check database exists
\du # Check user exists
```

Check `.env` file has correct credentials

Nginx 502 Bad Gateway

Symptom: Browser shows "502 Bad Gateway"

Solution:

```
systemctl status crm-backend
systemctl status crm-frontend
```

One of them is probably down. Check logs:

```
journalctl -u crm-backend -n 50
```

Can't SSH Into Server

Symptom: "Connection refused"

Solution:

1. Check Utho dashboard - is VM running?[53][54]
2. Check firewall allows port 22:

```
ufw status
```

3. Try password if SSH key fails

9. Quick Reference Cards

Daily Operations

Task	Command
Check all services	`systemctl status crm-backend crm-frontend nginx`
View backend logs	`journalctl -u crm-backend -f`
Restart backend	`systemctl restart crm-backend`
Deploy updates	`deploy-crm.sh` (if using script)
Check disk space	`df -h`
Check memory	`free -h`

Access URLs

Service	URL
Frontend	http://150.241.246.110/
Django Admin	http://150.241.246.110/admin/
API Health Check	http://150.241.246.110/api/health/

Important File Locations

Purpose	Path
Backend code	`/var/www/CRM_FINAL/backend/`
Frontend code	`/var/www/CRM_FINAL/jewellery-crm/`
Backend logs	`/var/www/CRM_FINAL/backend/logs/`
Frontend logs	`/var/www/CRM_FINAL/jewellery-crm/logs/`

Purpose	Path
Nginx config	`/etc/nginx/sites-available/crm-app`
Nginx logs	`/var/log/nginx/`
Backend service	`/etc/systemd/system/crm-backend.service`
Frontend service	`/etc/systemd/system/crm-frontend.service`
Environment vars	`/var/www/CRM_FINAL/backend/.env`

10. Additional Resources

Utho Documentation

- Utho Cloud Platform: https://console.utho.com[56][60]
- Quick Start Guide: https://utho.com/docs/products/compute/cloud/getting-started/quick-start/[56][71]
- Deploy Cloud Instance: https://utho.com/docs/products/compute/cloud/how-tos/deploy-cloud-instance/[^60]
- VPC Setup: https://www.youtube.com/watch?v=GcbjJmdBi9s[^63]
- Complete Walkaround: https://www.youtube.com/watch?v=-atpBlDM1CI[^59]

Linux/SSH Guides

- SSH Commands: https://phoenixnap.com/kb/linux-ssh-commands[^75]
- Systemd Management: https://www.digitalocean.com/community/tutorials/how-to-use-systemctl-to-manage-systemd-services-and-units[^83]

Database Guides

- PostgreSQL Ubuntu Setup: https://www.liquidweb.com/blog/install-postgresql-ubuntu/[^79]

Deployment Best Practices

- Django + Next.js: https://www.reddit.com/r/nextjs/comments/zu8iqc/how_to_deploy_a_website_with_django_backend_and/ [7]
- Production Deployment: https://www.microtica.com/blog/mastering-production-deployments [13]

Conclusion

What You've Achieved

You've gone from clicking "Deploy" on Vercel/Render to:

✅ Understanding full production architecture
✅ Managing your own Linux server
✅ Setting up databases, caching, and web servers
✅ Configuring systemd services
✅ Setting up CI/CD automation
✅ 60% cost savings[64][67]

Your New Workflow

Before (Managed):

```
Cursor → Git Push → Vercel/Render Auto-Deploy
```

Now (Self-Managed with CI/CD):

```
Cursor → Git Push → GitHub Actions → Auto-Deploy to Utho
```

Same convenience, more control, less cost!

Next Steps

1. **Add SSL Certificate** (Let's Encrypt) for HTTPS
2. **Set up automated backups** (PostgreSQL + media files)
3. **Add monitoring** (Uptime checks, error alerts)
4. **Scale horizontally** when traffic grows (add more VMs)

Need Help?

- Utho Support: 24/7 human support[64][70]
- Email: support@utho.com
- Dashboard: https://console.utho.com[^56]

Version: 1.0 | **Date:** October 22, 2025 | **Made for Vibecoders**
[14] [15] [16] [17] [18] [19] [20] [21] [22] [23] [24] [25] [26] [27] [28] [29] [30] [31] [32] [33] [34] [35] [36] [37] [38] [39] [40] [41] [42]

⁂

1. https://utho.com/docs/products/compute/cloud/getting-started/quick-start/
2. https://www.youtube.com/watch?v=AWN9bxFDTd4
3. https://github.com/JimPresting/Oracle-Cloud-VM-Setup
4. https://docs.fedoraproject.org/en-US/quick-docs/systemd-understanding-and-administering/
5. https://last9.io/blog/systemctl-guide/
6. https://phoenixnap.com/kb/linux-ssh-commands
7. https://www.youtube.com/watch?v=MKLC_kmYJug
8. https://www.youtube.com/watch?v=-atpBlDM1CI
9. https://www.digitalocean.com/community/tutorials/ssh-essentials-working-with-ssh-servers-clients-and-keys
10. https://www.digitalocean.com/community/tutorials/how-to-use-ssh-to-connect-to-a-remote-server
11. https://www.geeksforgeeks.org/linux-unix/ssh-command-in-linux-with-examples/
12. https://www.cherryservers.com/blog/how-to-install-and-setup-postgresql-server-on-ubuntu-20-04
13. https://www.postgresql.org/download/linux/ubuntu/
14. https://dev.to/latchudevops/launching-a-virtual-machine-on-utho-cloud-step-by-step-guide-125i
15. https://www.youtube.com/watch?v=GcbjJmdBi9s
16. https://cxotoday.com/press-release/utho-cloud-unveils-new-tools-for-secure-migration-cutting-cloud-costs-by-up-to-60/
17. https://in.linkedin.com/company/uthocloud
18. https://utho.com
19. https://krishnapaul.in/utho-cdk/
20. https://cyfuture.cloud/kb/howto/how-to-set-up-a-virtual-machine-in-cloud-computing-step-by-step
21. https://businessreviewlive.com/utho-cloud-launches-vpc-and-waf-to-enable-hassle-free-and-secure-cloud-migration/
22. https://utho.com/docs/products/compute/cloud/getting-started/
23. https://www.linkedin.com/pulse/vpn-split-tunneling-detailed-guide-utho-cloud-uthocloud-w76qc

24. https://utho.com/docs/products/compute/cloud/how-tos/deploy-cloud-instance/
25. https://utho.com/docs/products/compute/cloud/getting-started/quick-start/
26. https://www.youtube.com/watch?v=tducLYZzElo
27. https://www.geeksforgeeks.org/linux-unix/systemctl-in-unix/
28. https://www.cybrosys.com/blog/basic-ssh-commands-in-linux
29. https://www.liquidweb.com/blog/install-postgresql-ubuntu/
30. https://documentation.suse.com/smart/systems-management/html/systemd-management/index.html
31. https://www.g2.com/products/utho/features
32. https://www.ssh.com/academy/ssh/command
33. https://www.digitalocean.com/community/tutorials/how-to-use-systemctl-to-manage-systemd-services-and-units
34. https://www.tecmint.com/install-postgresql-on-ubuntu/
35. https://docs.redhat.com/en/documentation/red_hat_enterprise_linux/7/html/system_administrators_guide/chap-managing_services_with_systemd
36. https://www.ovhcloud.com/en-in/community/tutorials/how-to-install-pg-ubuntu/
37. https://aws.plainenglish.io/systemd-systemctl-explained-linux-service-management-for-devops-768cc466d812
38. https://www.inmotionhosting.com/support/server/ssh/ssh-tutorial-for-beginners/
39. https://www.youtube.com/watch?v=8QCB_YNXE2U
40. https://www.linkedin.com/pulse/driving-innovation-uthos-role-smb-transformation-across-india-gicfc
41. https://utho.com/docs/products/compute/cloud/how-tos/deploy-cloud-instance/
42. https://www.youtube.com/watch?v=0ag13CIMgrw