Claw Club: Multi-Tenant OpenClaw Deployment Guide

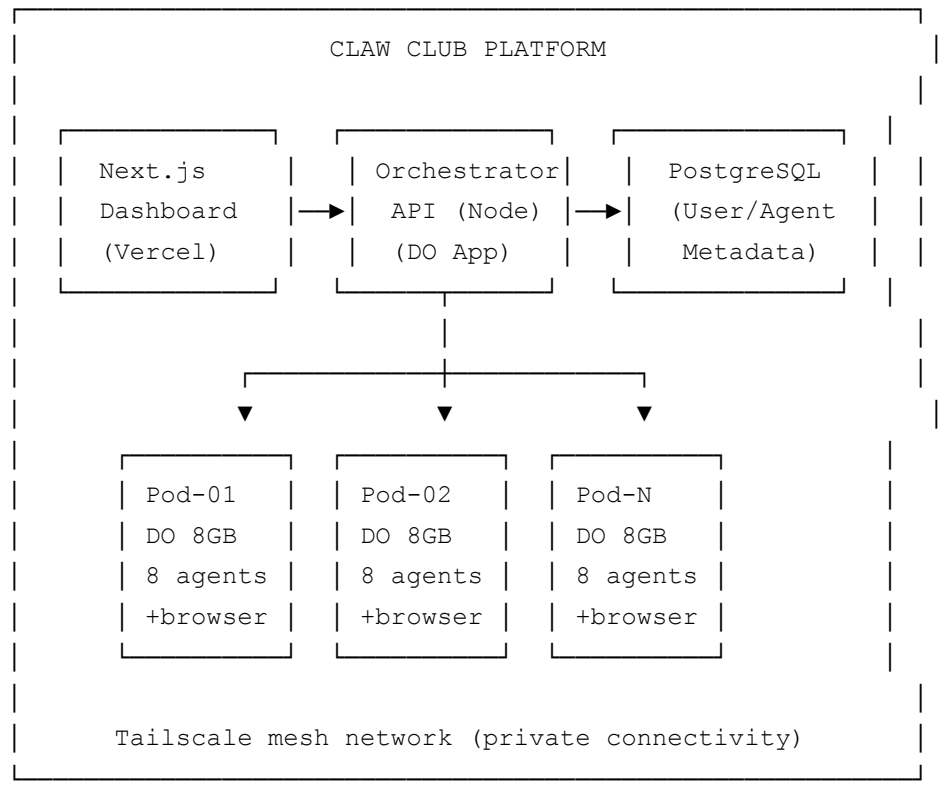
Architecture Overview — 100+ Users, Custom UI, Full Isolation

The Problem with Single-Instance Approaches

Railway and single-Droplet setups run one OpenClaw gateway with multiple agents in `agents.list[]`. This works for personal use but breaks at scale because all agents share the same gateway process, one crash can take everyone down, and you can't independently restart or upgrade individual user agents.

The Architecture: Pod-Based Multi-Tenancy

Instead of cramming 100 agents onto one box, we use a **pod architecture** where each DigitalOcean Droplet hosts 8-10 user agents, and a central **Orchestrator API** (your custom backend) manages provisioning, monitoring, and the user-facing dashboard.



Phase 1: Foundation Infrastructure

Step 1.1 — DigitalOcean Account & API Setup

1. Create a DigitalOcean account at https://cloud.digitalocean.com
2. Generate a Personal Access Token:
   - Go to **API > Tokens > Generate New Token**
   - Name it `claw-club-orchestrator`
   - Grant **Read + Write** scope
   - Save the token securely (you'll need it for the Orchestrator API)
3. Create an SSH key pair for automated Droplet access:

```
ssh-keygen -t ed25519 -C "claw-club-deploy" -f ~/.ssh/claw-club-deploy
# Upload the public key to DO: Settings > Security > SSH Keys
```

4. Note your SSH key fingerprint from the DO dashboard — the Orchestrator API needs it.

Step 1.2 — Set Up the Database

Create a DigitalOcean Managed PostgreSQL database (or use Supabase/Neon for lower cost):

```
-- Core tables for multi-tenant management
CREATE TABLE users (
  id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
  email TEXT UNIQUE NOT NULL,
  stripe_customer_id TEXT,
  usdc_wallet_address TEXT,
  plan TEXT DEFAULT 'basic',           -- basic, pro, enterprise
  created_at TIMESTAMPTZ DEFAULT NOW()
);

CREATE TABLE pods (
  id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
  droplet_id BIGINT UNIQUE NOT NULL,   -- DigitalOcean Droplet ID
  ip_address INET NOT NULL,
  tailscale_ip INET,
  region TEXT NOT NULL,                 -- nyc1, sfo2, sgp1, etc.
  capacity INTEGER DEFAULT 8,          -- max agents per pod
  current_load INTEGER DEFAULT 0,
```

```
  status TEXT DEFAULT 'provisioning',   -- provisioning, active, draining, dead
  created_at TIMESTAMPTZ DEFAULT NOW()
);

CREATE TABLE agents (
  id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
  user_id UUID REFERENCES users(id) ON DELETE CASCADE,
  pod_id UUID REFERENCES pods(id),
  agent_name TEXT NOT NULL,             -- unique per pod
  gateway_token TEXT NOT NULL,          -- per-agent auth token
  status TEXT DEFAULT 'provisioning',   -- provisioning, active, paused, error
  config JSONB DEFAULT '{}',           -- sandbox config, model prefs, channels
  last_heartbeat TIMESTAMPTZ,
  created_at TIMESTAMPTZ DEFAULT NOW(),
  UNIQUE(pod_id, agent_name)
);

CREATE TABLE billing_events (
  id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
  user_id UUID REFERENCES users(id),
  event_type TEXT NOT NULL,             -- subscription, usage, topup
  amount_usd DECIMAL(10,2),
  payment_method TEXT,                  -- stripe, usdc
  tx_hash TEXT,                         -- for USDC payments
  created_at TIMESTAMPTZ DEFAULT NOW()
);
```

Step 1.3 — Tailscale Network Setup

Every pod and the orchestrator join a shared Tailscale network for private, encrypted communication. No OpenClaw gateway is ever exposed to the public internet.

1. Create a Tailscale account at https://tailscale.com
2. Generate an auth key:
   - Go to **Settings > Keys > Generate auth key**
   - Enable **Reusable** and **Ephemeral**
   - Tag it `tag:claw-pod`
3. Set up ACL rules in Tailscale admin:

```
{
  "acls": [
    {
```

```
      "action": "accept",
      "src": ["tag:claw-orchestrator"],
      "dst": ["tag:claw-pod:*"]
    },
    {
      "action": "accept",
      "src": ["tag:claw-pod"],
      "dst": ["tag:claw-pod:*"]
    }
  ],
  "tagOwners": {
    "tag:claw-orchestrator": ["autogroup:admin"],
    "tag:claw-pod": ["autogroup:admin"]
  }
}
```

Phase 2: Pod Image & Provisioning

Step 2.1 — Create the Golden Pod Image

Instead of using the DO 1-Click (which is single-agent), build a custom image that supports multi-agent + browser automation out of the box.

Create a Packer template or use a cloud-init script. Here's the cloud-init approach:

```
# pod-cloud-init.yaml
#cloud-config
package_update: true
package_upgrade: true

packages:
  - docker.io
  - docker-compose-v2
  - jq
  - curl
  - ufw
  - fail2ban

write_files:
  - path: /opt/claw-pod/docker-compose.yml
    content: |
      version: "3.8"
      services:
        openclaw-gateway:
```

```
    image: openclaw:local
    build:
      context: /opt/claw-pod/openclaw
      dockerfile: Dockerfile
    restart: unless-stopped
    ports:
      - "127.0.0.1:18789:18789"
    volumes:
      - openclaw-config:/home/node/.openclaw
      - openclaw-workspace:/home/node/openclaw/workspace
      - /var/run/docker.sock:/var/run/docker.sock:ro
    environment:
      - OPENCLAW_GATEWAY_TOKEN=${GATEWAY_TOKEN}
    networks:
      - claw-net

  browser:
    image: kasmweb/chrome:1.16.1
    restart: unless-stopped
    shm_size: "2g"
    ports:
      - "127.0.0.1:6901:6901"
      - "127.0.0.1:9222:9222"
    environment:
      - VNC_PW=${BROWSER_VNC_PW}
    networks:
      - claw-net

  health-reporter:
    image: curlimages/curl:latest
    restart: unless-stopped
    entrypoint: /bin/sh
    command: >
      -c 'while true; do
        LOAD=$(docker ps --filter "label=openclaw-sandbox" -q | wc -l);
        curl -s -X POST http://${ORCHESTRATOR_IP}:3001/api/pods/heartbeat \
          -H "Authorization: Bearer ${POD_SECRET}" \
          -H "Content-Type: application/json" \
          -d "{\"pod_id\":\"${POD_ID}\",\"load\":$LOAD,\"status\":\"active\
        sleep 30;
      done'
    networks:
      - claw-net

volumes:
  openclaw-config:
  openclaw-workspace:
```

```
    networks:
      claw-net:
        driver: bridge

- path: /opt/claw-pod/setup-firewall.sh
  permissions: "0755"
  content: |
    #!/bin/bash
    ufw default deny incoming
    ufw default allow outgoing
    ufw allow 22/tcp          # SSH (will be removed after Tailscale)
    ufw allow 41641/udp       # Tailscale
    ufw --force enable
    # After Tailscale is up, lock down SSH to Tailscale only:
    # ufw delete allow 22/tcp
    # ufw allow in on tailscale0 to any port 22

- path: /opt/claw-pod/install-tailscale.sh
  permissions: "0755"
  content: |
    #!/bin/bash
    curl -fsSL https://tailscale.com/install.sh | sh
    tailscale up --authkey=${TAILSCALE_AUTH_KEY} --advertise-tags=tag:claw-pod

runcmd:
  - systemctl enable docker
  - systemctl start docker
  - usermod -aG docker ubuntu
  - /opt/claw-pod/setup-firewall.sh
  - /opt/claw-pod/install-tailscale.sh
```

Step 2.2 — Automated Pod Provisioning via DO API

The Orchestrator API creates new pods when capacity runs low. Here's the core provisioning logic:

```
// orchestrator/src/services/pod-provisioner.ts

import { createClient } from "@digitalocean/api-client";

interface PodConfig {
  region: string;
  size: string;       // "s-4vcpu-8gb" for General Purpose
  tailscaleKey: string;
  podId: string;
```

```
}

export async function provisionPod(config: PodConfig) {
  const doClient = createClient({ token: process.env.DO_API_TOKEN! });

  // 1. Create the Droplet
  const droplet = await doClient.droplets.create({
    name: `claw-pod-${config.podId}`,
    region: config.region,
    size: config.size,
    image: "ubuntu-24-04-x64",
    ssh_keys: [process.env.DO_SSH_KEY_FINGERPRINT!],
    user_data: generateCloudInit(config),
    tags: ["claw-pod", `region-${config.region}`],
    monitoring: true,
    ipv6: true,
  });

  // 2. Wait for Droplet to be active
  const readyDroplet = await waitForDroplet(doClient, droplet.id);

  // 3. Store pod metadata
  await db.pods.insert({
    id: config.podId,
    droplet_id: readyDroplet.id,
    ip_address: readyDroplet.networks.v4[0].ip_address,
    region: config.region,
    capacity: 8,
    current_load: 0,
    status: "provisioning",
  });

  // 4. Wait for Tailscale to come online (poll for ~2 min)
  const tailscaleIp = await waitForTailscale(config.podId);

  // 5. SSH in via Tailscale and build OpenClaw from source
  await sshExec(tailscaleIp, `
    cd /opt/claw-pod &&
    git clone https://github.com/openclaw/openclaw.git &&
    cd openclaw &&
    git checkout $(git describe --tags --abbrev=0) &&
    docker build -t openclaw:local -f Dockerfile . &&
    docker compose up -d
  `);

  // 6. Mark pod as active
  await db.pods.update(config.podId, {
```

```
    status: "active",
    tailscale_ip: tailscaleIp
  });

  return { podId: config.podId, tailscaleIp };
}

// Auto-scale: check every 5 minutes if we need more pods
export async function checkAndScale() {
  const pods = await db.pods.findAll({ status: "active" });
  const totalCapacity = pods.reduce((sum, p) => sum + p.capacity, 0);
  const totalLoad = pods.reduce((sum, p) => sum + p.current_load, 0);

  // If we're at 80% capacity, provision a new pod
  if (totalLoad / totalCapacity > 0.8) {
    const region = selectBestRegion(pods); // least loaded region
    await provisionPod({
      region,
      size: "s-4vcpu-8gb",
      tailscaleKey: process.env.TAILSCALE_AUTH_KEY!,
      podId: crypto.randomUUID(),
    });
  }
}
```

Step 2.3 — Agent Provisioning per User

When a Claw Club member signs up and pays, the orchestrator assigns them to a pod and creates their agent:

```
// orchestrator/src/services/agent-provisioner.ts

export async function provisionAgent(userId: string, preferences: AgentPreference
  // 1. Find the best pod (least loaded, closest region)
  const pod = await db.pods.findOne({
    where: { status: "active" },
    orderBy: { current_load: "asc" },
    having: sql`current_load < capacity`,
  });

  if (!pod) throw new Error("No available pods — scaling triggered");

  // 2. Generate unique agent credentials
  const agentName = `agent-${userId.slice(0, 8)}`;
  const gatewayToken = crypto.randomBytes(32).toString("hex");
```

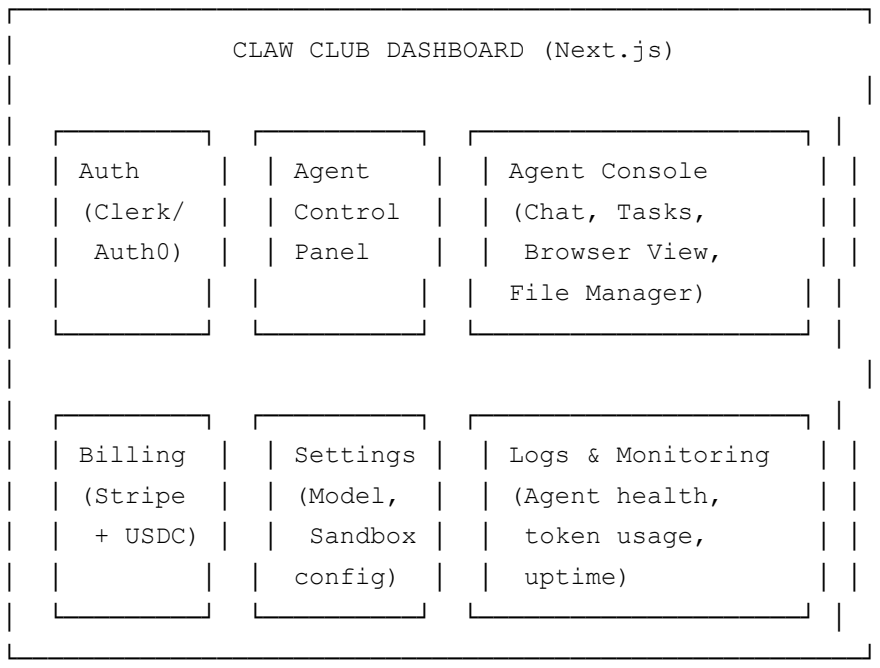
```
  // 3. SSH into the pod and configure the new agent
  await sshExec(pod.tailscale_ip, `
    # Add agent to OpenClaw config
    cd /opt/claw-pod/openclaw

    # Update the config JSON to add the new agent
    cat <<'AGENT_CONFIG' | python3 -c "
import json, sys
config_path = '/home/node/.openclaw/config.json'
new_agent = json.load(sys.stdin)
with open(config_path) as f:
    config = json.load(f)
if 'agents' not in config:
    config['agents'] = {'list': []}
config['agents']['list'].append(new_agent)
with open(config_path, 'w') as f:
    json.dump(config, f, indent=2)
"
    {
      "id": "${agentName}",
      "name": "${preferences.displayName || 'My Agent'}",
      "model": "${preferences.model || 'claude-sonnet-4-5-20250929'}",
      "sandbox": {
        "mode": "all",
        "scope": "agent",
        "docker": {
          "memory": "1g",
          "cpus": 0.5,
          "network": "bridge",
          "pidsLimit": 128
        }
      },
      "tools": {
        "sandbox": {
          "tools": {
            "allow": ["exec", "read", "write", "edit", "browser"]
          }
        }
      }
    }
AGENT_CONFIG

    # Restart gateway to pick up new config
    docker compose restart openclaw-gateway
  `);

  // 4. Store agent record
```

```
  await db.agents.insert({
    user_id: userId,
    pod_id: pod.id,
    agent_name: agentName,
    gateway_token: gatewayToken,
    status: "active",
    config: preferences,
  });

  // 5. Update pod load
  await db.pods.increment(pod.id, "current_load");

  return {
    agentName,
    gatewayToken,
    dashboardUrl: `https://${pod.tailscale_ip}:18789`,
    podRegion: pod.region,
  };
}
```

Phase 3: The Claw Club Dashboard (Custom UI)

Step 3.1 — Dashboard Architecture

Your custom UI is the main interface members use. It wraps around the OpenClaw gateway API and adds Claw Club-specific features.



Step 3.2 — Key Dashboard Pages

1. Onboarding Flow

```
Sign Up → Choose Plan → Payment (Stripe/USDC) → Agent Provisioning →
Connect Channels (WhatsApp/Telegram/Discord) → Dashboard
```

The provisioning happens via your Orchestrator API — the user sees a loading screen while the agent is being created on an available pod.

2. Agent Console

This is the killer feature. Instead of making users SSH into their server or use OpenClaw's raw TUI, you proxy the gateway WebSocket through your dashboard:

```
// dashboard/src/app/agent/[id]/page.tsx

"use client";
import { useEffect, useRef, useState } from "react";

export default function AgentConsole({ params }: { params: { id: string } }) {
  const [messages, setMessages] = useState<Message[]>([]);
  const wsRef = useRef<WebSocket | null>(null);

  useEffect(() => {
    // Fetch agent connection details from your API
    fetch(`/api/agents/${params.id}/connection`)
      .then(res => res.json())
      .then(({ gatewayUrl, token }) => {
        // Connect to the OpenClaw gateway WebSocket via your proxy
        const ws = new WebSocket(
          `wss://proxy.clawclub.io/ws/${params.id}?token=${token}`
        );
        ws.onmessage = (event) => {
          const data = JSON.parse(event.data);
          setMessages(prev => [...prev, data]);
        };
        wsRef.current = ws;
      });

    return () => wsRef.current?.close();
  }, [params.id]);

  const sendMessage = (text: string) => {
    wsRef.current?.send(JSON.stringify({
```

```
      type: "chat",
      content: text,
      agent: params.id,
    }));
  };

  return (
    <div className="flex h-screen">
      {/* Chat panel */}
      <div className="flex-1 flex flex-col">
        <MessageList messages={messages} />
        <ChatInput onSend={sendMessage} />
      </div>
      {/* Side panel: agent status, running tasks, browser view */}
      <div className="w-80 border-l">
        <AgentStatus agentId={params.id} />
        <RunningTasks agentId={params.id} />
        <BrowserPreview agentId={params.id} />
      </div>
    </div>
  );
}
```

3. Browser Automation Viewer

Each pod runs kasmweb/chrome. Your dashboard can embed the noVNC viewer so users can watch their agent browse the web in real-time:

```
// Embed the noVNC iframe for the user's pod browser
<iframe
  src={`https://${agent.podTailscaleIp}:6901/vnc.html?autoconnect=true&password=$
  className="w-full h-96 rounded-lg border"
/>
```

4. Billing Integration

```
// orchestrator/src/routes/billing.ts

// Stripe webhook for subscription events
app.post("/webhooks/stripe", async (req, res) => {
  const event = stripe.webhooks.constructEvent(/*...*/);

  switch (event.type) {
    case "customer.subscription.created":
      // Provision agent
      const user = await db.users.findByStripeId(event.data.object.customer);
```

```
      await provisionAgent(user.id, { model: getPlanModel(event.data.object) });
      break;

    case "customer.subscription.deleted":
      // Deprovision agent
      await deprovisionAgent(user.id);
      break;
  }
});

// USDC payment verification (Solana)
app.post("/api/billing/verify-usdc", async (req, res) => {
  const { txHash, userId } = req.body;

  // Verify the transaction on Solana
  const tx = await solanaConnection.getTransaction(txHash);
  const isValid = verifyUsdcTransfer(tx, TREASURY_WALLET, MONTHLY_PRICE_USDC);

  if (isValid) {
    await db.billingEvents.insert({
      user_id: userId,
      event_type: "subscription",
      amount_usd: MONTHLY_PRICE_USDC,
      payment_method: "usdc",
      tx_hash: txHash,
    });

    // Provision or extend agent
    await provisionOrExtendAgent(userId);
  }
});
```

Phase 4: Capacity Planning for 100+ Users

Droplet Sizing Math

Each agent needs:

- **Sandbox container**: 1GB RAM, 0.5 CPU
- **Shared browser sidecar**: 2GB RAM (shared per pod)
- **Gateway overhead**: ~500MB per pod

Per pod (General Purpose 4vCPU/8GB — $63/mo):

- Usable RAM after OS + Docker: ~7GB
- Browser sidecar: 2GB
- Gateway: 0.5GB
- Available for sandboxes: 4.5GB
- **Agents per pod: 4-5 (with headroom)**

For conservative sizing, use **8GB pods with 5 agents each**:

Users	Pods Needed	Droplet Size	Monthly Infra Cost
25	5	4vCPU/8GB	$315
50	10	4vCPU/8GB	$630
100	20	4vCPU/8GB	$1,260
150	15	8vCPU/16GB	$1,425 (fewer, bigger pods)

Add to this:

- Managed PostgreSQL (db-s-1vcpu-1gb): $15/mo
- Orchestrator API (App Platform basic): $12/mo
- Tailscale (free for <100 devices, $6/user after)
- Dashboard hosting (Vercel Pro): $20/mo

Total for 100 users: ~$1,350/mo infrastructure

At $29/user/month pricing, that's $2,900 revenue vs $1,350 infra = **53% margins before LLM API costs**.

Auto-Scaling Rules

```
// orchestrator/src/services/autoscaler.ts

const SCALE_UP_THRESHOLD = 0.75;   // 75% pod capacity
const SCALE_DOWN_THRESHOLD = 0.25; // 25% pod capacity
const MIN_PODS = 2;                // always keep 2 running
const MAX_PODS = 50;               // hard ceiling

export async function autoScale() {
  const pods = await db.pods.findAll({ status: "active" });
```

```
  const totalCapacity = pods.reduce((s, p) => s + p.capacity, 0);
  const totalLoad = pods.reduce((s, p) => s + p.current_load, 0);
  const utilization = totalLoad / totalCapacity;

  if (utilization > SCALE_UP_THRESHOLD && pods.length < MAX_PODS) {
    // Scale up: provision new pod in least-loaded region
    const region = selectLeastLoadedRegion(pods);
    await provisionPod({ region, size: "s-4vcpu-8gb", /* ... */ });

  } else if (utilization < SCALE_DOWN_THRESHOLD && pods.length > MIN_PODS) {
    // Scale down: drain least-loaded pod
    const targetPod = pods.sort((a, b) => a.current_load - b.current_load)[0];
    await drainPod(targetPod.id); // migrate agents, then destroy
  }
}

// Run every 5 minutes
setInterval(autoScale, 5 * 60 * 1000);
```

Phase 5: Execution Checklist

Week 1: Foundation

- Set up DigitalOcean account + API token
- Create PostgreSQL database (DO Managed or Supabase)
- Set up Tailscale account + auth keys + ACLs
- Create the cloud-init pod template
- Manually provision first pod and verify OpenClaw + browser work
- Test multi-agent config on single pod (3 agents)

Week 2: Orchestrator API

- Scaffold Node.js/TypeScript Orchestrator API
- Implement pod provisioning via DO API
- Implement agent provisioning (SSH + config injection)
- Build heartbeat/health monitoring system
- Implement auto-scaler logic

- Deploy Orchestrator to DO App Platform

Week 3: Dashboard & Billing

- Scaffold Next.js dashboard
- Implement auth (Clerk or Auth0)
- Build onboarding flow (sign up → pay → provision)
- Build agent console with WebSocket proxy
- Embed browser automation viewer (noVNC)
- Integrate Stripe subscriptions
- Integrate USDC payments (Solana)

Week 4: Hardening & Launch

- Load test: provision 20 agents across 4 pods
- Set up monitoring (DO built-in + custom dashboards)
- Configure alerting (pod down, high memory, agent crash)
- Write member documentation
- Set up backup strategy (DO Droplet snapshots weekly)
- Security audit: verify no public gateway exposure
- Beta launch with 10-20 users
- Iterate based on feedback, then open to 100+

Phase 6: Making It Better Than Railway

What Railway Can't Do (Your Competitive Moat)

Feature	Railway	Claw Club on DO
Per-user Docker sandboxes	No Docker-in-Docker	Full Docker daemon
Browser automation	Broken/unsupported	Native kasmweb/chrome sidecar
Watch agent browse live	Not possible	noVNC embedded in dashboard

Custom management UI	Raw OpenClaw TUI	Polished Next.js dashboard
Agent isolation	Shared single container	Per-user sandbox containers
Multi-region	Single region	NYC, SFO, SGP, LON, etc.
Stripe + USDC billing	Manual	Integrated payment flow
Auto-scaling	Manual resize	Auto-provision new pods
Private networking	Public URLs	Tailscale mesh (zero-trust)
Uptime guarantee	Best-effort	DO 99.99% SLA per pod
White-label ready	Not possible	Full custom branding

Key Differentiators for Claw Club Members

1. **One-click agent deployment** — Members sign up, pay, and get a fully provisioned agent in under 3 minutes. No SSH, no terminal, no config files.
2. **Visual browser control** — Members can watch their agent browse the web in real-time through the embedded noVNC viewer, and even take over manually.
3. **Channel marketplace** — Pre-configured WhatsApp, Telegram, Discord, Slack integrations that members can toggle on/off from the dashboard.
4. **Skill store** — Curated AgentSkills that members can install with one click (email management, calendar, GitHub, trading tools, etc.).
5. **Usage analytics** — Token consumption, task completion rates, uptime, and cost tracking per agent.
6. **Data sovereignty** — Each member's agent runs in an isolated Docker container with its own workspace. No data mixing between users.

Quick Reference: Key Commands

```
# Provision a new pod manually
curl -X POST "https://api.digitalocean.com/v2/droplets" \
  -H "Authorization: Bearer $DO_TOKEN" \
  -H "Content-Type: application/json" \
  -d '{"name":"claw-pod-01","region":"nyc1","size":"s-4vcpu-8gb","image":"ubuntu-
```

```
# SSH into a pod via Tailscale
ssh ubuntu@100.x.y.z

# Check agent sandboxes running on a pod
docker ps --filter "label=openclaw-sandbox"

# View OpenClaw gateway logs
docker compose -f /opt/claw-pod/docker-compose.yml logs -f openclaw-gateway

# Restart a specific agent's sandbox
docker restart sandbox-agent-abc12345

# Check pod resource usage
docker stats --no-stream

# Backup a pod's config
tar -czf pod-backup-$(date +%Y%m%d).tar.gz /home/node/.openclaw/
```

Pricing Strategy Recommendation

Plan	Price/mo	What They Get
Starter	$19	1 agent, basic model (Haiku), 2 channels, no browser
Pro	$39	1 agent, Sonnet model, all channels, browser automation
Power	$79	1 agent, Opus model, priority pod placement, premium support

At 100 Pro users ($39/mo each):

- Revenue: $3,900/mo
- Infra: ~$1,350/mo
- LLM API pass-through: billed separately or included with caps
- **Net margin: ~65% before API costs**

Members bring their own API keys (BYOK model) to keep your margins clean and avoid unpredictable LLM costs.