把 AI 助手放进 VPS：OpenClaw 入门与轻量运维实战

从 Gateway、模型、微信通道和工具权限讲起，记录如何在一台 1 GiB 内存的 VPS 上部署 OpenClaw，并用固定命令、人工审批与原子发布安全地管理 Freshmark 和 Tranquil Reader。

2026年7月30日 · OpenClaw · VPS · systemd · 微信 · 自动化运维

一台 VPS 上的服务逐渐增多以后，日常运维通常会出现许多短小却重复的操作：查看 CPU 和内存、确认网站是否在线、拉取新版本、重新构建、切换发布目录，以及在故障时回滚。它们并不复杂，但每一步都要求操作者记住正确的目录、服务名和检查顺序。

我希望得到的不是一个可以任意执行命令的“聊天机器人”，而是一个更窄的运维入口：

```
微信消息
  |
  ▼
OpenClaw 微信通道
  |  识别用户、建立会话
  ▼
DeepSeek 模型
  |  理解“查看状态”或“部署 Freshmark”
  ▼
OpenClaw 工具策略与执行审批
  |
  ▼
固定的只读状态命令 / 原子部署脚本
  |
  ▼
systemd、Nginx 与两个网站
```

最终，这套系统能够在微信中回答服务器状态，列出 CPU 和内存占用最高的进程，也能理解部署和回滚请求。与此同时，模型没有获得任意 root shell：状态查询可以自动执行，部署和回滚必须经过人工批准，脚本之外的 root 操作则会被拒绝。

本文从计算机基础概念讲起，记录这套轻量实现的结构、配置思路与安全边界。

OpenClaw 是什么

OpenClaw 可以把大语言模型、聊天软件和本机工具连接在一起。与普通网页聊天不同，它不仅生成文字，还可以在策略允许时调用工具，例如读取文件、执行命令或发送消息。

理解 OpenClaw，可以先区分五个部分：

部分	作用
Gateway	长期运行的核心进程，管理会话、通道、插件和工具调用
Model Provider	提供大语言模型，例如 DeepSeek
Channel	用户发送消息的入口，例如 WebChat、微信或 Telegram
Agent	接收上下文、调用模型并决定是否使用工具的执行主体
Skill	写给 Agent 的操作说明，描述有哪些命令、何时使用以及怎样解释结果

其中，模型负责理解语言，却不应直接决定自己拥有什么权限。真正的权限来自 OpenClaw 配置、操作系统用户、`sudoers` 规则和部署脚本。这种分层很重要：模型可能误解问题，也可能受到提示注入影响；只要底层边界足够窄，错误判断就不应自动变成任意系统操作。

OpenClaw 官方也把它定位为**单一可信操作者的个人助手**，而不是让互不信任的多个用户共享的隔离平台。如果要服务多个彼此不信任的用户，应当分别运行 Gateway，并使用不同的系统用户或主机。

本次部署环境

这台 VPS 运行 Ubuntu 24.04，资源并不宽裕：

资源	配置
CPU	2 核
内存	1 GiB
Swap	约 545 MiB
磁盘	约 19 GiB

服务器原本已经运行 Nginx、Freshmark API、Freshmark 邮件桥接、Tranquil Reader 同步服务、Postfix、OpenDKIM 和 Xray。为了避免在 1 GiB 内存上再引入容器守护进程，本次直接安装 OpenClaw，没有使用 Docker，也没有启用浏览器自动化、本地大模型和多 Agent 并发。

实测空闲时 OpenClaw Gateway 的 RSS 会在约 230–350 MiB 之间变化。执行一次模型与工具调用时，还会短暂出现约 250 MiB 的 Agent 进程；Freshmark 构建期间，负责构建的 Node 进程一度达到约 374 MiB。因此，1 GiB 内存能够运行这套精简配置，但已经接近适合的下限；如果继续加入数据库、无头 Chromium、Docker 构建或本地模型，升级到 2 GiB 会稳妥得多。

RSS 是 Resident Set Size，即进程当前驻留在物理内存中的页面总量。它可能包含共享页面，所以不能简单把所有进程的 RSS 相加并当作系统真实占用；判断 Linux 内存压力时，还应优先观察 `available`，而不是只看 `free`。

安装：先把 OpenClaw 与 root 分开

官方提供安装脚本、npm、容器等多种方式。本文环境使用系统级 Node.js 和 npm 安装 OpenClaw，并让它作为 systemd 服务长期运行。无论选择哪种安装方式，都应先确认版本与诊断结果：

```
openclaw --version
openclaw doctor
openclaw gateway status
```

这台服务器没有让 Gateway 直接以 root 身份运行，而是建立了专用用户：

```
用户：openclaw
主目录：/var/lib/openclaw
状态目录：/var/lib/openclaw/.openclaw
工作区：/var/lib/openclaw/.openclaw/workspace
```

systemd 服务的思路可以简化为：

```
[Service]
User=openclaw
Group=openclaw
Environment=HOME=/var/lib/openclaw
Environment=OPENCLAW_STATE_DIR=/var/lib/openclaw/.openclaw
EnvironmentFile=/etc/openclaw/openclaw.env
ExecStart=/usr/bin/openclaw gateway run --bind loopback --port 18789 --auth token
Restart=on-failure
MemoryHigh=384M
MemoryMax=512M
PrivateTmp=true
ProtectSystem=full
ProtectHome=true
```

`MemoryHigh` 是内核开始积极回收内存的软阈值，`MemoryMax` 是 cgroup 的硬上限。子进程默认也会继承这组限制，所以不能直接让 Gateway 进程树承担一次完整的前端构建；后文会说明如何把已批准的构建放进独立的临时 systemd unit。

模型密钥和 Gateway Token 不写入仓库，也不放在聊天记录或普通配置文件中。环境文件只允许 root 读取；systemd 的系统管理器先读取它，再以 `openclaw` 用户启动服务。模型凭据则保存为 OpenClaw 的认证配置。安装插件时固定版本，并设置显式插件白名单，避免服务器在不知情的情况下加载其他插件。

专用用户并不能解决所有安全问题，但它建立了第一层边界：即使 Agent 可以执行普通命令，也不会天然拥有 root 的完整权限。

Gateway 只监听本机

Gateway 的管理界面没有通过 Nginx 暴露到公网，而是只监听回环地址：

```
127.0.0.1:18789
```

回环地址只能从服务器本机访问。需要打开控制界面时，可以在自己的电脑上建立 SSH 隧道：

```
ssh -N -L 18789:127.0.0.1:18789 root@203.0.113.10
```

然后访问：

```
http://127.0.0.1:18789/
```

这里的 `203.0.113.10` 是文档示例地址，不是实际服务器地址。SSH 会把本机的 18789 端口加密转发到 VPS 的回环端口，因此无需再维护一个公开的管理域名、反向代理和公网登录入口。

服务器的 root SSH 同样只允许密钥登录，密码登录已经关闭。这样，Gateway、SSH 和网站服务形成了彼此独立的入口：

- 网站由 Nginx 对公网提供；
- 微信通道主动连接腾讯接口；
- Gateway 管理界面只通过 SSH 隧道访问；
- root 权限只通过 SSH 密钥或精确的 `sudoers` 规则获得。

接入 DeepSeek：模型不等于权限

本次使用 DeepSeek 模型，并把默认并发限制为 1、推理等级设为较低水平，同时关闭没有必要的心跳任务。这些设置一方面节省模型调用费用，另一方面也减少小内存服务器上的并发进程。

模型可以看到 Skill 中的说明，也可以请求调用 OpenClaw 提供的工具。但“模型提出调用”与“系统允许执行”是两件事。例如，模型可能生成：

```
请执行：systemctl restart nginx
```

如果工具策略和操作系统权限没有允许这条命令，它就不会因为文字看起来合理而自动获得 root 权限。因此，选择能力较强的模型有助于正确理解意图，却不能替代最小权限、审批和脚本校验。

用微信访问 OpenClaw

OpenClaw 通过腾讯微信团队维护的外部插件接入微信。本文部署时使用 OpenClaw `2026.7.1-2` 和微信插件 `2.4.6`；版本兼容关系以后可能变化，应以微信通道文档为准。

安装时固定插件版本：

```
openclaw plugins install "@tencent-weixin/openclaw-weixin@2.4.6"
openclaw config set plugins.entries.openclaw-weixin.enabled true
```

然后在运行 Gateway 的同一台服务器上启动登录：

```
openclaw channels login --channel openclaw-weixin
```

终端会显示二维码和备用链接。用微信扫码并确认后，插件把授权凭据保存在 OpenClaw 状态目录。二维码有效期较短，过期后重新运行登录命令即可，不需要删除插件。

本次登录完成后，命令行提示它无法在当前路径中读取 systemd 注入的 Gateway Token，因此没有自动热重载通道。授权本身已经成功，只需由 root 重启正式服务：

```
systemctl restart openclaw-gateway
```

随后检查：

```
openclaw channels status --probe
```

正常状态应包含 `configured` 和 `running`。微信插件通过长轮询接收消息，不要求给 VPS 增加新的公网入站端口。

私聊策略被明确设置为：

```
dmPolicy = pairing
```

未知用户首次发送消息时，应当先完成配对；扫码授权的所有者账号可以直接对话，所以不会收到自己的配对码。需要审批其他用户时，可以使用：

```
openclaw pairing list openclaw-weixin
openclaw pairing approve openclaw-weixin <配对码>
```

微信只是一个消息入口。接入微信不会自动扩大 Agent 的工具权限，部署审批和操作系统权限仍由后面的几层控制。

不把任意 shell 交给模型

如果直接允许模型以 root 身份运行任意命令，那么一句含糊的“清理一下服务器”就可能产生不可逆后果。更安全的办法是把操作收敛为有限集合：

```
查看系统状态
查看 Freshmark 状态
查看 Tranquil Reader 状态
部署 Freshmark
回滚 Freshmark
部署 Tranquil Reader
回滚 Tranquil Reader
```

为此，服务器提供三个面向 Agent 的命令：

```
/usr/local/bin/ops-status system
/usr/local/bin/ops-deploy freshmark
/usr/local/bin/ops-rollback freshmark
```

它们不是把参数直接拼进 shell，而是先检查参数是否属于固定枚举，再调用 root 拥有的分发器。分发器同样只接受确定的“服务—动作”组合。即使模型尝试添加额外参数、替换服务名或调用其他 sudo 命令，也会被拒绝。

sudoers 只允许 OpenClaw 用户执行这一条分发器，而不是授予通用的 root shell。换句话说，权限关系是：

```
openclaw 用户
  └ 可以 sudo 调用固定分发器
      ├ system + status
      ├ freshmark + status/deploy/rollback
      └ tranquil-reader + status/deploy/rollback
```

OpenClaw 的执行审批再提供一层交互边界：

- `ops-status` 加入允许列表，可以自动运行；
- `ops-deploy` 和 `ops-rollback` 不在自动允许列表；
- 真正部署或回滚时，操作者必须选择 `allow-once`，即只允许本次执行。

这里需要准确理解审批的作用：它是防止助手误解操作者意图的护栏，不是用于隔离恶意租户的完整沙箱。真正可靠的边界仍然来自专用系统用户、固定命令、严格参数检查和原子部署脚本。

审批 ID 从哪里来

最初的 Skill 要求 Agent “请求一次性批准”，但没有明确要求它先调用工具。于是 Agent 只生成了下面这段文字：

> 准备执行部署：`/usr/local/bin/ops-deploy freshmark`。
>
> 请确认是否允许执行？回复 `/approve` 并选择 **allow once**。

这看起来像一条审批请求，实际上只是普通的模型回复。Gateway 尚未收到 exec 调用，自然没有创建审批记录。此时发送裸命令 `/approve`，得到的真实回复是：

```
Usage: /approve <id> <decision>
(see the pending approval message for available decisions)
```

修复后的规则是：用户已经明确要求部署时，Agent 必须立即调用固定包装命令，不能再索要第二次文字确认。只有这次工具调用触发 allowlist miss 以后，Gateway 才会创建真实审批 ID。

审批转发也被显式设为当前会话：

```
{
  "approvals": {
    "exec": {
      "enabled": true,
      "mode": "session",
      "agentFilter": ["main"]
    }
  }
}
```

微信收到的待审批消息会给出 ID 和可用 decision。正确回复类似：

```
/approve 01234567-89ab-cdef-0123-456789abcdef allow-once
```

其中 ID 必须原样使用真实消息中的值，不能自行编造。`allow-always` 会把该命令持久加入允许列表，不适合部署和回滚；如果本次执行失败，再次尝试也应产生一个新的审批 ID。

Skill：把运维知识写给 Agent

只有命令还不够。模型需要知道何时使用命令、怎样解释字段，以及哪些结论不能从一次采样中得到。为此，我建立了一个名为 `vps-operations` 的自定义 Skill。

它记录了：

- 两个项目的源代码目录、分支和服务名；
- 状态、部署与回滚的唯一合法命令；
- 健康状态应当满足的 HTTP 状态码；
- CPU、负载、内存、Swap、磁盘和 inode 的解释方法；

- 单次快照不能推断长期趋势；
- 发现异常时只能报告，不能临时编造修复命令。

例如，系统状态命令会输出：

```
cpu_count=2
cpu_used_percent_1s=1.0
load_1m=0.07
memory_available_mib=457
disk_root_used_percent=62%
failed_systemd_unit_names=networking.service
unit_nginx_service=active
```

Skill 告诉 Agent：

- `cpu_used_percent_1s` 只是 1 秒采样；
- 平均负载应与 CPU 核数比较；
- Linux 会主动利用空闲内存做缓存，应优先看 `available`；
- Swap 非零不等于故障；
- 磁盘或 inode 超过 80% 才需要警告，超过 90% 应当视为紧急；
- 一次快照不能说明使用率正在上升或下降。

这样，模型不只是把数字重新排列，而是按照预先约定的标准解释数字。

查看 CPU 与内存占用最高的进程

基础状态只能告诉我“内存用了多少”，却不能回答“被谁使用”。因此，状态脚本后来加入了两个 Top 5 列表。

CPU 列表来自 `top` 的第二次刷新，以约 1 秒的间隔采样：

```
top_cpu_process_1=pid:266904,name:openclaw-gatewa,cpu_percent_1s:1.0
```

内存列表按照 RSS 排序：

```
top_memory_process_1=pid:266904,name:openclaw-gatewa,memory_percent:34.7,rss_mib:356.1
top_memory_process_2=pid:255795,name:node,memory_percent:4.6,rss_mib:48.0
```

输出只包含 PID、经过清洗的进程名和资源数值，不包含完整命令行、启动参数、环境变量或日志。这一点尤其重要：API Key、Bearer Token 和数据库地址经常以环境变量或参数形式进入进程，所谓“详细状态”不应顺便把这些秘密交给模型。

在一次真实测试中，OpenClaw 能把这些字段整理为表格，并指出活跃的 Agent 进程会短暂增加内存占用。测试结束后，临时进程退出，内存也随之回落。CPU 几乎空闲时，Top 5 可能全部显示 `0.0%`；这并非脚本失效，而是采样窗口内没有可观测的 CPU 消耗。

第二次 Freshmark 部署期间，状态命令捕捉到了真实的构建负载。OpenClaw 的回复包括：

> **CPU TOP 5**：node（PID 268941）达到 **128.7%**，openclaw-gatewa 约为 1.0%。
>
> **内存 TOP 5**：node 使用 **373.5 MiB**，Gateway 使用 **230.9 MiB**，同时可以看到 esbuild 与 npm。
>
> 内存使用率为 **84.5%**，可用内存约 **159 MiB**；Swap 使用约 **305 MiB**。

这里的 128.7% 使用 top 的 Irix 表示方式，约等于占用 1.29 个逻辑 CPU；在 2 核服务器上不等于整机超过 100%。进程名、npm 和 esbuild 同时出现，也为“正在构建”提供了较强证据，但这种判断仍然是对同一时刻多个字段的综合推断。

构建结束后，可用内存恢复到约 512 MiB，内存使用率回落到约 50%。Swap 从约 305 MiB 降到约 238 MiB，但没有立即清零。Linux 会保留已经换出的冷页面，单独看到非零 Swap 不代表仍有故障。

两个项目如何实现原子部署

让 Agent 执行 `git pull && npm run build` 并不等于可靠部署。构建中途失败、SSH 断开或服务重启过早，都可能让线上目录处于半新半旧的状态。

Freshmark 和 Tranquil Reader 都使用“发布目录 + 当前软链接”的结构：

```
/var/www/freshmark/
  ├─ releases/
  │   ├─ 20260729T152651Z-35b2d023dd78/
  │   └─ ...
  └─ current -> releases/20260729T152651Z-35b2d023dd78/
```

部署过程依次完成：

1. 在控制仓库中获取指定远端分支；
2. 要求本地工作区干净，并只接受快进更新；
3. 在新的 staging 目录中构建；
4. 检查生成文件和持久化资源；
5. 把 staging 改名为正式 release；
6. 原子切换 `current` 软链接；
7. 重启或重载对应服务；
8. 检查本地健康接口与公网 HTTPS；
9. 若检查失败，自动切回上一版本。

“原子”并不表示整个构建瞬间完成，而是指线上入口从一个完整版本切换到另一个完整版本，不会长期指向构建到一半的目录。

Freshmark 的账号、会话、评论、访问量和图片缓存位于 release 目录之外。Tranquil Reader 的 PDF 也保存在持久化目录，通过链接提供给每次发布。这样，回滚应用代码时不会回滚或删除用户数据。

因此，OpenClaw 所做的只是选择并请求一个已经设计好的部署动作。真正决定发布是否安全的，是部署脚本中的校验、软链接切换、健康检查和自动回滚。

为什么构建要离开 Gateway 的 cgroup

第一次通过微信批准部署时，命令确实经过了固定的 root 分发器，但部署进程仍是 Gateway 的后代，继承了 512 MiB 的 `MemoryMax` 和 `ProtectHome=true`。后者使 root 身份的 npm 也无法访问 `/root/.npm`。

失败后的真实回复是：

> npm 更新尝试（10.9.8 → 12.0.2）失败了，退出码 254。
>
> 错误原因：npm 尝试写入 `/root/.npm/_logs` 日志目录时遇到权限问题。

这段回复准确复述了日志目录错误，却把 npm 的版本更新提示误判成了“正在升级 npm”。部署脚本实际执行的是 `npm ci`，并没有运行 `npm install -g npm@12.0.2`。npm 只是在失败输出末尾提示有新版本可用。

直接给 `/root` 改权限并不能解决 systemd 的挂载隔离，也会扩大不必要的访问范围。最终修复分为两层。

首先，部署脚本把 npm 缓存和日志固定到专用目录，并关闭与构建无关的更新提示：

```
NPM_CONFIG_CACHE=/var/cache/freshmark-npm
NPM_CONFIG_UPDATE_NOTIFIER=false
```

该目录由 root 创建，权限为 `0700`。脚本还拒绝把缓存目录配置成 `/`、`/var` 或 `/var/cache` 这样的宽泛路径。

其次，root 分发器不再直接把构建留在 Gateway 进程树中，而是启动一个临时 unit：

```
微信审批
  │
  ▼
OpenClaw Gateway（MemoryMax=512M）
  │  sudo：仅允许固定服务与动作
  ▼
root 分发器
  │  systemd-run --wait --pipe --collect
  ▼
openclaw-freshmark-deploy.service
  │  独立 cgroup；PrivateTmp / ProtectHome / ProtectSystem
```

```
   ▼
Freshmark 原子部署脚本
```

`--wait` 让 OpenClaw 等待真实退出码，`--pipe` 把构建输出交还给审批会话，`--collect` 则在任务结束后回收临时 unit。构建获得独立的 cgroup，同时仍然受到部署脚本、`sudoers` 和固定分发器的约束；它没有变成任意 root shell。

第一次失败发生在构建阶段，尚未切换 `current`，因此线上旧版本一直保持健康。修复后重新产生审批 ID 并选择 `allow-once`，部署成功切换到：

```
revision=70794709dbebdfc2f5c04fea8a6764f9a67ef3f7
release=20260730T020915Z-70794709dbeb
api=active
health_http=200
public_http=200
article_http=200
```

这次测试同时验证了一个重要性质：批准只负责允许命令开始运行，发布成功仍必须由退出码、活动 revision、API 健康检查和公网 HTTP 结果共同证明。

安全审计与仍然存在的边界

每次修改通道或工具配置后，都应运行：

```
openclaw security audit
openclaw security audit --deep
```

本次配置关闭了浏览器工具、提权工具、心跳任务和不需要的内置 Skill，只允许 `read` 与受控的 `exec`，并使用插件白名单。Gateway 只绑定回环地址，微信私聊使用 pairing，部署动作要求审批。

不过，这套配置没有使用 Docker 沙箱，而 `exec` 本质上仍然是 shell。即使 OpenClaw 的文件工具被限制，shell 也可能读写它有权限访问的路径。当前方案依靠以下边界降低风险：

- OpenClaw 使用独立的非 root 用户；
- Agent 只知道经过筛选的运维命令；
- 自动允许列表中只有只读状态脚本；
- root 分发器拒绝任意参数；
- 具有副作用的部署和回滚需要人工批准；
- Gateway 与控制界面不暴露到公网；
- 服务器属于单一可信操作者。

如果未来要允许陌生用户访问、执行任意构建命令或操作更多服务，就不应继续沿用同一个信任边界，而应考虑容器沙箱、独立 Gateway、独立系统用户，甚至独立主机。

还应注意模型可能“解释得过头”。例如，进程名 node 并不能单独证明它运行的是哪个项目；一次 1 秒 CPU 采样也不能证明服务器一整天都很空闲。状态脚本主动避免输出命令行后，Agent 就应承认无法从现有字段确定进程的完整用途，而不是根据经验补全事实。

验证清单

一套运维助手是否真正可用，不能只看它能否回复“你好”。本次部署依次验证了：

```
[✓] OpenClaw Gateway 由 systemd 启动并保持 active
[✓] DeepSeek Provider 返回有效模型响应
[✓] Gateway 只监听回环地址
[✓] 任意 sudo、非法动作和额外参数被拒绝
[✓] 只读状态命令可以自动执行
[✓] 部署与回滚命令触发人工审批
[✓] 微信收到真实审批 ID，allow-once 只批准一次命令
[✓] 构建在独立临时 systemd unit 中运行并返回真实退出码
[✓] npm 缓存不访问 /root，构建结束后临时 unit 被回收
[✓] Freshmark 状态、健康接口和公网首页正常
[✓] Freshmark 从微信完成一次真实原子部署，失败尝试未切换旧版本
[✓] Tranquil Reader 状态、清单和 PDF 资源正常
[✓] 微信扫码登录后通道保持 running
[✓] 微信消息能够触发一次模型与工具调用并收到回复
[✓] 状态报告包含 CPU / 内存 Top 5，且不包含命令行和秘密
```

这份清单也揭示了一个常见误区：安装成功只是开始。通道、模型、工具和服务分别正常，并不自动保证整条链路正常；必须从真实消息出发，完成一次端到端测试。

结语

OpenClaw 的价值不在于“让 AI 获得服务器 root 权限”，而在于把自然语言放在已有自动化之上。模型负责理解“帮我看看服务器是否健康”，Skill 负责提供领域知识，固定脚本负责收集可信数据，操作系统负责执行权限，人工审批则守住具有副作用的动作。

对于只有一两台 VPS 的个人项目，这种结构比完整监控与发布平台轻得多，也比把 SSH 命令随手交给模型可靠。它不能代替备份、监控、权限设计和部署脚本，却可以把这些能力组织成一个随时可用的统一入口。

真正值得自动化的并不是每一条 shell 命令，而是那些边界清楚、结果可验证、失败可以回滚的操作。

参考

- OpenClaw 安装文档
- OpenClaw Gateway 安全文档

- OpenClaw Exec Approvals
- OpenClaw 微信通道
- 腾讯微信 OpenClaw 插件