PX4 虚拟多机 + 容器化 vuav_agent + 群控脚本里程碑说明

（内部技术文档草稿）

2025-12-16

目录

1 整体目标与当前里程碑 **2**

1.1 项目背景与目标简述 2

2 容器镜像与部署方式 **3**

2.1 当前镜像版本与命名约定 3

2.2 辅助脚本与配置文件 4

2.3 推荐运行方式：vuav-net + 群控容器 4

2.4 镜像备份与恢复建议 5

3 容器内代理程序 vuav_agent.py 概述 **5**

3.1 整体功能 5

3.2 mavlink_connect() 与心跳确认 6

3.3 通用命令发送封装：send_command_long() 6

3.4 命令处理逻辑 process_request() 7

3.5 TCP 会话处理 handle_client() 与主循环 7

4 群控脚本 swarm_controller.py 概述 **8**

4.1 设计思路 8

4.2 vUAV 列表与配置加载 8

4.3 广播与监控函数 9

4.4 高度曲线监控 monitor_altitude_curve 9

5 当前一轮运行结果解读 **10**

5.1 SITL 初始化与 Preflight 告警 10

5.2 群控脚本在容器内完整运行的一次典型结果 10

6 当前进展与下一步工作 **11**

6.1 当前进展小结 11

6.2 已知限制与改进方向 11

6.3 下一步工作建议（供后续迭代参考） 12

1 整体目标与当前里程碑

1.1 项目背景与目标简述

本项目面向后续“虚拟无人机集群 + 可信计算/TEE 防护”方向的教学与科研需求，尝试构建一个尽量轻量、易扩展的基础平台：

- 使用 PX4 SITL（当前采用 SIH 模式）作为虚拟无人机飞控内核；
- 每个虚拟无人机（vUAV）封装在一个 Docker 容器中，容器内同时运行 PX4 与一个自定义的 MAVLink 代理脚本 `vuav_agent.py`；
- 群控脚本 `swarm_controller.py` 也以单独容器形式运行，通过 Docker 自定义网络 `vuav-net` 统一连接多个 vUAV 的 agent，实现“探活、状态查询、起飞命令、位置/高度监控”等群控操作；
- 所有 vUAV 通信参数抽离到 JSON 配置文件 `vuav_cluster_config.json`，弱化网络/端口与业务逻辑之间的耦合，为后续从单机到云上大规模扩展预留空间；
- 后续在此基础上逐步叠加：OFFBOARD 控制、编队轨迹规划、遥测缓存与观测、可信执行环境中的控制决策等。

截至 2025-12-16，目前平台已完成的工作可以视为一个阶段性里程碑：

1) **容器镜像构建与复现路径打通（升级到 v6）**
   - 基于 Ubuntu + PX4 源码 + Python 运行时，构建出自包含的镜像；
   - 镜像内可以直接启动 PX4 SIH 模式，并通过 `pymavlink` 连接到 `udp:127.0.0.1:14540`；
   - 将 `vuav_agent.py` 与 PX4 一起打包进镜像，形成统一的 vUAV 运行环境；
   - 当前最新镜像为 `px4-vuav-agent:v6-20251215`。
2) **多容器虚拟无人机（vUAV）实例可稳定启动**
   - 使用脚本 `create_vuav_net.sh` 创建专用 Docker 网络 `vuav-net`；
   - 使用 `run_vuav1.sh` 可以单独启动 `vuav1` 容器，用于单机调试；
   - 使用 `start_vuav_cluster.sh` 可以在 `vuav-net` 上启动多台 vUAV 容器（当前默认三机 `vuav1/2/3`），每个容器内都运行 PX4 + `vuav_agent.py`，对外提供统一的 JSON/TCP 控制接口。
3) **容器内 MAVLink 代理脚本 `vuav_agent.py` 升级**
   - 负责：
     (a) 与本容器内 PX4 建立 MAVLink 连接（当前为 `udp:127.0.0.1:14540`），在启动阶段等待 HEARTBEAT；
     (b) 在容器内监听 TCP 端口 `6000`，接收来自 swarm 控制器的 JSON 命令；
     (c) 将 JSON 命令翻译为 MAVLink 消息(如 `COMMAND_LONG`、`SET_POSITION_TARGET_LOCAL_NED` 等)，并将结果封装为 JSON 返回；
     (d) 维护最近一次姿态/位置等遥测的缓存，避免每次查询都阻塞式等待 MAVLink 消息，提高多机场景下的响应稳定性；
     (e) 提供基本的“探活、姿态、起飞、局部位置、速度控制占位”等能力。
4) **群控脚本 `swarm_controller.py` 容器化并配置外置**
   - `swarm_controller.py` 不再在宿主机裸跑，而是通过 `run_swarm_controller.sh` 启动一个专门的 `swarm-controller` 容器，挂载当前目录并加入 `vuav-net`；
   - vUAV 列表不再硬编码为 `"127.0.0.1:1600X"`，而是从 `vuav_cluster_config.json` 中

加载,按名称(如 vuav1)映射到容器级别的 host:port(例如 "host": "vuav1", "port": 6000);

- 这样一来，集群规模、命名规则、端口等都可以通过修改 JSON 配置调整，而无需改 Python 源码，满足“网络配置松耦合”的目标。

5) **当前可用的基本功能**

- **容器/网络层面**:
  - 创建/启动/停止 vUAV 容器;
  - 使用专用 Docker 网络 vuav-net 将所有 vUAV 与 swarm-controller 容器互联;
  - 通过 docker logs 观测 PX4 日志（包括 Preflight 检查告警、SITL 初始化信息等)。
- **MAVLink/控制层面**:
  - 通过 vuav_agent.py 将以下能力以 JSON RPC 的形式暴露给群控容器:
    - ping 探活;
    - get_status 读取 ATTITUDE;
    - takeoff（MAV_CMD_NAV_TAKEOFF，当前能收到 COMMAND_ACK，result=0);
    - get_local_position 读取 LOCAL_POSITION_NED，用于姿态/高度监控（支持从最近缓存返回，减少阻塞);
    - set_velocity_ned/broadcast_velocity_ned:基于 SET_POSITION_TARGET_LOCAL_NED 的速度控制占位接口，为后续 OFFBOARD 控制和编队轨迹铺垫;
    - goto_local_ned 等航点类接口的占位实现，为之后路径规划/任务层逻辑预留扩展点。
  - swarm_controller.py 可以对集群做统一的 ping、状态轮询、起飞命令分发、位置/高度监控以及简单的速度实验。

2 容器镜像与部署方式

2.1 当前镜像版本与命名约定

当前使用的最新镜像名为:

```
px4-vuav-agent:v6-20251215
```

该镜像主要包含:

- Ubuntu 基础系统（版本见 Dockerfile);
- 编译好的 PX4 SITL（本项目采用 SIH 模式以简化外部仿真依赖);
- Python3 运行时与依赖包（如 pymavlink 等);
- 项目目录 /px4 下的:
  - PX4 可执行程序及其配置;
  - MAVLink 代理脚本 vuav_agent.py（带缓存与统一 JSON 接口);
  - 容器内启动脚本（例如用于在容器启动时并行拉起 PX4 与 agent)。

镜像版本号后缀 v6-20251215 用于记录:

- 第 6 次较大的镜像迭代;
- 包含当前版本的 vuav_agent.py、依赖库以及适配 swarm_controller.py 的运行环境。

2.2 辅助脚本与配置文件

目前项目中与部署相关的主要脚本和配置文件为：

- `create_vuav_net.sh`：检查并创建 Docker 网络 `vuav-net`，作为所有 vUAV 与群控容器的统一二层网络；
- `run_vuav1.sh`：在 `vuav-net` 上启动单个 `vuav1` 容器，用于单机调试（PX4 + `vuav_agent`）；
- `start_vuav_cluster.sh`：在 `vuav-net` 上批量启动 vUAV 集群，当前默认创建 `vuav1`, `vuav2`, `vuav3` 三个容器；
- `swarm_controller.py`：群控脚本本体，负责读取集群配置、建立 TCP 连接并下发 JSON 命令；
- `run_swarm_controller.sh`：以 `px4-vuav-agent` 镜像为基础，创建 `swarm-controller` 容器，加入 `vuav-net`，挂载当前目录并运行 `swarm_controller.py`；
- `vuav_cluster_config.json`：集群配置文件，描述各个 vUAV 的逻辑名称与对应的 `host:port`，例如：

```
{
  "vuav1": { "host": "vuav1", "port": 6000 },
  "vuav2": { "host": "vuav2", "port": 6000 },
  "vuav3": { "host": "vuav3", "port": 6000 }
}
```

2.3 推荐运行方式：vuav-net + 群控容器

推荐的启动顺序为：

1) 在项目根目录（例如 `$HOME/px4-docker`）下创建网络：

```
./create_vuav_net.sh
```

2) 启动 vUAV 集群：

```
./start_vuav_cluster.sh
```

3) 启动群控容器并运行 `swarm_controller.py`：

```
./run_swarm_controller.sh
```

此时整体链路为：

`swarm-controller` 容器 ⇒ `vuav-net` ⇒ 各 `vuavX` 容器（TCP 6000 上的 `vuav_agent`）⇒ 本容器内 PX4 (MAVLink: `udp:127.0.0.1:14540`)

宿主机只负责启动/管理容器，并不直接参与控制数据的转发。相比早期通过 `-p 1600X:6000` 暴露到宿主机再访问的方式，这种“全容器化 + 专用网络”的结构：

- 网络路径清晰、可控；
- 方便迁移到云环境中对接 overlay 网络或 CNI；
- 避免依赖宿主机端口映射的细节，减少难以复现的端口超时问题。

2.4 镜像备份与恢复建议

镜像备份

为了方便日后在其他机器或环境中快速复现当前状态，建议定期导出镜像为 .tar 包，例如：

```
# 导出当前镜像（注意版本号保持和上文一致）
docker save -o px4-vuav-agent_v6-20251215.tar px4-vuav-agent:v6-20251215
```

该 .tar 文件可放置在：

- Git 仓库的 Release（注意 2 GB 以上不适合直接放 GitHub）；
- 内部 NAS/网盘；
- 或通过硬盘拷贝作为长期备份。

镜像恢复

在新环境或重新安装系统后，只要有上述 px4-vuav-agent_v6-20251215.tar 备份文件，即可通过以下命令恢复：

```
# 从 tar 文件加载镜像
docker load -i px4-vuav-agent_v6-20251215.tar

# 加载完成后，可以通过 docker images 查看
docker images | grep px4-vuav-agent
```

只要镜像名和标签仍为 px4-vuav-agent:v6-20251215，上文启动命令即可直接复用，无需重新构建。

3 容器内代理程序 vuav_agent.py 概述

3.1 整体功能

vuav_agent.py 的职责是：

1) 连接本容器内的 PX4 SITL，使用 MAVLink 建立稳定会话：
   - 当前使用 URL：udp:127.0.0.1:14540；
   - 通过 mavutil.mavlink_connection 建立连接；
   - 调用 wait_heartbeat() 确认 PX4 正常启动；
   - 在单独线程中循环接收 MAVLink 消息，并更新内部缓存（如最近一次 ATTITUDE 与 LOCAL_POSITION_NED）。
2) 在 TCP 端口 6000 上监听来自群控容器的连接请求：
   - 每收到一个新连接，创建一个新的线程处理；
   - 按行解析 JSON（以换行符 "\n" 为分包边界）；
   - 每条 JSON 请求对应一次 process_request 调用，并返回一条 JSON 响应。
3) 将上层 JSON 命令翻译为 MAVLink 消息，并从 MAVLink/缓存中收集返回结果，封装为 JSON 返回给上层。

3.2 mavlink_connect() 与心跳确认

核心代码结构如下（只保留关键片段）：

```
MAVLINK_URL = "udp:127.0.0.1:14540"

def mavlink_connect():
    print(f"[agent] connecting to PX4 via {MAVLINK_URL} ...")
    m = mavutil.mavlink_connection(MAVLINK_URL)
    m.wait_heartbeat(timeout=10)
    print(f"[agent] HEARTBEAT from PX4: sysid={m.target_system}, compid={m.
        target_component}")
    return m
```

说明：

- wait_heartbeat 有超时（10 秒），若 PX4 未正常启动会直接抛异常/退出；
- 成功收到心跳后，打印系统 ID 和组件 ID，方便区分多机场景；
- 内部后续会启用循环读取线程，持续接收 ATTITUDE、LOCAL_POSITION_NED 等消息并写入缓存。

3.3 通用命令发送封装：send_command_long()

```
def send_command_long(mav, command, params, timeout=2.0):
    """
    发送 MAV_CMD_*，并等待 COMMAND_ACK。
    params: 长度为 7 的 list/tuple -> param1..param7
    """
    mav.mav.command_long_send(
        mav.target_system,
        mav.target_component,
        command,
        0,
        *params
    )
    ack = mav.recv_match(type='COMMAND_ACK', blocking=True, timeout=timeout)
    if not ack:
        return {"ok": False, "error": "no COMMAND_ACK"}

    ok = (ack.result == mavutil.mavlink.MAV_RESULT_ACCEPTED)
    return {
        "ok": ok,
        "result": int(ack.result),
        "command": int(ack.command),
    }
```

该函数被用来封装例如：

- 解锁/上锁：MAV_CMD_COMPONENT_ARM_DISARM；

- 起飞：MAV_CMD_NAV_TAKEOFF；
- 模式切换等命令（后续可扩展 MAV_CMD_DO_SET_MODE）。

3.4 命令处理逻辑 process_request()

process_request(req, mav) 是 agent 的核心逻辑入口，负责根据 JSON 请求中的 "cmd" 字段分发到不同 MAVLink 操作。整体模式是：

- 解析 cmd = req.get("cmd")；
- 针对不同字符串执行不同分支；
- 若不认识该命令，统一返回 "unknown cmd"。

当前支持的命令集包括（节选）：

- ping：简单探活，直接返回 "pong"；
- get_attitude：从最近 ATTITUDE 缓存中读取姿态，返回欧拉角（度），若缓存过旧则阻塞等待一条新消息；
- get_status：基于 ATTITUDE 和系统状态封装为 "status" 结构；
- arm/disarm：分别调用 MAV_CMD_COMPONENT_ARM_DISARM 的 1/0 版本；
- takeoff：调用 MAV_CMD_NAV_TAKEOFF，支持通过 "alt" 参数指定目标高度；
- get_local_position：优先从最近一条 LOCAL_POSITION_NED 缓存返回 (x, y, z, vx, vy, vz)，其中 $z > 0$ 表示向下；必要时也可阻塞等待；
- set_velocity_ned：基于 SET_POSITION_TARGET_LOCAL_NED 发送简单速度控制，只填充 v_x, v_y, v_z 和偏航角速度 yaw_rate，其它字段通过 type_mask 屏蔽（实验性接口，需要 PX4 处于 OFFBOARD 等合适模式）；
- goto_local_ned：航点接口占位，目前仅解析目标 (x, y, z) 并返回 “未实现”，为后续路径规划/航点导航逻辑预留扩展点；
- 其它未实现命令统一返回 "unknown cmd"。

3.5 TCP 会话处理 handle_client() 与主循环

handle_client(conn, addr, mav) 的职责是：

- 从 conn.recv 中持续读入字节流，缓存在 buf 中；
- 每当发现换行符 "\n"，就将前面的内容视为一条 JSON；
- 尝试解析 JSON，调用 process_request，并将响应 JSON 加上换行后发回客户端；
- 任意异常都会被捕获并转换为 "handler exception"，避免直接断开连接导致调试困难。

主循环 agent_main() 逻辑：

1) 调用 mavlink_connect() 与 PX4 建立连接；
2) 启动 MAVLink 消息接收线程，负责更新内部缓存；
3) 打开 TCP 监听 socket：

```
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
s.bind(("0.0.0.0", 6000))
s.listen(5)
```

4) 每接收到一个新连接，就创建一个守护线程，调用 handle_client 处理；

5) 在容器退出前，主线程不会主动关闭该 socket。

4 群控脚本 swarm_controller.py 概述

4.1 设计思路

swarm_controller.py 主要是为了在一个专门的控制容器中统一操作多个 vUAV 的 agent。其设计目标包括：

- 将 vUAV 的 IP/端口配置放在外部 JSON 文件中，避免硬编码，便于扩展为 4 机、6 机甚至更多；
- 提供一个通用的 call(name, cmd) 接口，简化对单机的 JSON RPC 调用；
- 提供若干 “群体操作” 函数，例如广播 ping、广播起飞、轮询状态等；
- 提供简单的观测工具，例如 LOCAL_POSITION_NED 的采样和高度曲线监控；
- 整体运行在 px4-vuav-agent 镜像环境内，便于未来将控制逻辑迁移到 TEE 等受保护环境。

4.2 vUAV 列表与配置加载

当前版本不再硬编码 VUS 字典，而是提供类似如下的加载函数：

```
def load_vus(config_path=None):
    if config_path is None:
        config_path = "vuav_cluster_config.json"
    with open(config_path, "r", encoding="utf-8") as f:
        cfg = json.load(f)
    vus = {}
    for name, item in cfg.items():
        host = item.get("host", "localhost")
        port = int(item.get("port", 6000))
        vus[name] = (host, port)
    return vus

VUS = load_vus()
```

配置文件 vuav_cluster_config.json 中，每个 vUAV 记录其逻辑名称、容器内地址和端口。脚本运行时通过 load_vus 读取，并为后续所有呼叫提供统一的地址解析。

通用发送函数与之前基本一致：

```
def send_cmd(addr, cmd, timeout=3.0):
    s = socket.create_connection(addr, timeout=timeout)
    s.sendall((json.dumps(cmd) + "\n").encode())
    data = s.recv(4096)
    s.close()
    return json.loads(data.decode("utf-8"))

def call(name, cmd):
    addr = VUS[name]
    try:
```

```
        resp = send_cmd(addr, cmd)
        print(f"[{name}] cmd={cmd} resp={resp}")
        return resp
    except Exception as e:
        print(f"[{name}] ERROR sending {cmd}: {e}")
        return {"ok": False, "error": str(e)}
```

4.3 广播与监控函数

整体设计上：

- 广播函数：
  - `broadcast(cmd)`：对所有 vUAV 调用 `call`；
- 状态监控函数：
  - `poll_status_once()`：对所有 vUAV 调用 `get_status`，查看姿态；
  - `monitor_local_position(ticks, interval)`：周期性读取 `LOCAL_POSITION_NED` 并打印位置与速度全状态；
  - `monitor_altitude_curve(ticks, interval)`：只关注高度 $\text{alt} = -z$，输出高度时间序列与 min/max 汇总，用于“飞没飞起来”的快速检查；
- 控制函数：
  - `broadcast_takeoff(alt)`：对所有 vUAV 发送 `takeoff` 命令；
  - `broadcast_velocity_ned(vx, vy, vz, yaw_rate)`:通过代理的 `set_velocity_ned` 接口向所有 vUAV 下发统一速度指令（实验性接口，依赖 PX4 处于合适的 `OFFBOARD` 等模式）；
  - `experiment_velocity_forward(...)`：作为简单的“向前匀速飞一段时间”样例，用于验证速度指令链路是否畅通。
- 简单入口：
  - 在 `__main__` 中串联：广播 ping → 轮询状态 → 广播 arm → 再次轮询 → 广播起飞 → 位置监控与高度曲线监控 → 速度试验。

4.4 高度曲线监控 `monitor_altitude_curve`

`monitor_altitude_curve(ticks, interval)` 的逻辑是：

1) 对每个 vUAV 维护一个高度列表；
2) 每次循环对各机分别调用 `get_local_position`；
3) 从返回中取出 z，计算 $\text{alt} = -z$（NED 坐标系下 z 向下为正，这里取负号作为“向上高度”）；
4) 打印当前 tick 的高度，同时追加到各自的列表；
5) 所有 tick 完成后，对每个 vUAV 打印最小/最大高度。

这在实践中主要用来回答一个问题：“**虽然 PX4 接受了起飞命令，但它到底有没有真正离地？**”

5 当前一轮运行结果解读

5.1 SITL 初始化与 Preflight 告警

在 px4-vuav-agent:v6-20251215 中，PX4 SIH 启动日志大致为：

- 载入参数，初始化数据管理器 dataman；
- 初始化 SIH 仿真循环（250 Hz）；
- 启动多路 MAVLink 端口（包括与 14540 的本地 Onboard 链接）；
- Logger 启动并创建 .ulg 日志文件；
- 随后可以看到多条 Preflight 告警，例如：
  - Preflight Fail: Accel 0 uncalibrated
  - Preflight Fail: barometer 0 missing
  - Preflight Fail: Found 0 compass (required: 1)

这些告警说明当前配置下，PX4 认为部分传感器尚未完成标定/初始化，因此会阻止实际解锁/起飞（或者在 SITL 中表现为高度与姿态受限），这与后续看到的高度仅在厘米级波动相互印证。

5.2 群控脚本在容器内完整运行的一次典型结果

在当前代码和脚本下，启动 vUAV 集群与群控容器后，在 swarm-controller 容器内执行：

```
python3 swarm_controller.py
```

观察到的典型输出结构（节选）大致为：

1) **广播 ping**：所有配置在 vuav_cluster_config.json 中的机体均返回 "ok": true, "reply": "pong"；
2) **状态轮询**：大部分机体的 get_status 返回正常 ATTITUDE，偶尔可能有个别出现 "no ATTITUDE"（启动初期尚未产生该消息）；
3) **广播 arm**：当前由于 Preflight 检查未通过，返回中通常可见 "ok": false, "result": 1；
4) **广播 takeoff**：
   - 对每台 vUAV，PX4 返回 COMMAND_ACK 且 result=0；
   - 表明命令层面 “起飞指令被接受”。
5) **LOCAL_POSITION_NED 位置监控**：
   - 对每台 vUAV，在多个 tick 中可以看到 x,y,z,vx,vy,vz 缓慢变化；
   - 例如 z 从约 -0.021 变化到约 -0.023~-0.025 等（单位米）。
6) **高度曲线监控 monitor_altitude_curve**：
   - 高度定义为 alt $= -z$；
   - 运行中，几台机的高度大致在 0.02 m to 0.03 m 左右波动；
   - 最终 summary 形如：

```
[vuav1] alt min=0.02 m, max=0.03 m
[vuav2] alt min=0.02 m, max=0.02 m
[vuav3] alt min=0.02 m, max=0.02 m
```

总结：

- 从 **通信链路**视角：

  - 群控容器到各 vUAV 的 TCP 连接稳定，JSON 命令可以往返；
  - 状态/位置命令均正常返回；
  - 起飞命令被 PX4 接受（`COMMAND_ACK result=0`）。
- 从 **物理行为**视角：
  - LOCAL_POSITION_NED 中 `x,y,z` 随时间缓慢变化，同时高度监控 `monitor_altitude_curve` 给出的 $\text{alt} = -z$ 约在 0.02 m to 0.03 m 之间波动，证明“平台链路完整、仿真器位姿在演化”；
  - 但高度始终停留在厘米量级，结合 PX4 日志中的 Preflight 告警，可判断当前仍处于“未真正离地”的安全保护状态，需要后续补全传感器标定与解锁/模式切换（如 `OFFBOARD`）流程。

6 当前进展与下一步工作

6.1 当前进展小结

综合来看，当前平台已经完成：

- **容器与网络层**：
  - 基于 `px4-vuav-agent:v6-20251215` 镜像，可稳定启动多实例 vUAV；
  - 通过 `vuav-net` 将所有 vUAV 与群控容器互联，摆脱对宿主机端口映射的依赖；
  - 启动/停止集群有相对规范的脚本入口（`create_vuav_net.sh`、`start_vuav_cluster.sh` 等）。
- **MAVLink 代理层**：
  - `vuav_agent.py` 已能对接 PX4 心跳、发送 `COMMAND_LONG`、接收 ATTITUDE 与 LOCAL_POSITION_NED 等消息；
  - 内部增加了基本的遥测缓存逻辑，避免每次查询都阻塞等待新消息；
  - 对上层暴露统一的 JSON 命令接口，便于后续 TEE/密态环境直接复用。
- **群控层**：
  - `swarm_controller.py` 已可在容器内对多机进行统一的 ping、状态轮询、起飞命令分发、局部位置与高度曲线监控，并提供了速度控制广播的占位函数；
  - 集群拓扑与网络地址抽象在 `vuav_cluster_config.json` 中，实现了网络配置与控制逻辑的一步解耦。

这些内容可以视作后续“可信计算保护下的无人机集群控制”的一块基础：只要能在受保护环境中运行与 `swarm_controller.py` 类似的逻辑，通过 JSON 调用 agent，即可对 vUAV 集群进行统一调度。

6.2 已知限制与改进方向

目前仍然存在的主要问题和限制包括：

- **PX4 预飞行检查未完全通过**：
  - 日志中存在加速度计未标定、气压计缺失、磁罗盘不存在等告警；
  - 这会限制解锁/起飞的行为，使得当前高度仅在厘米级范围内微调；
  - 需要在 SIH 配置或参数层面进一步排查与调整。

- **缺乏标准化 OFFBOARD 控制流程**:
  - 当前仅实现了基于 SET_POSITION_TARGET_LOCAL_NED 的速度控制占位接口(set_velocity_ned/broa
  - 尚未设计完整的 “模式切换 → OFFBOARD 心跳 → 路径/速度命令序列” 逻辑。
- **编队与任务层逻辑尚未引入**:
  - 目前更多是 “多机单体控制”;
  - 尚未实现编队路径规划、碰撞检测、任务分配等高级功能。
- **观测与缓存仍然较基础**:
  - 目前遥测缓存仅存在于每个 vuav_agent 的进程内存中;
  - 尚未有集中式的缓存/队列系统（例如 Redis）来聚合多机状态，便于后续云上观测、记录与分析。

6.3 下一步工作建议（供后续迭代参考）

(A) **完善 PX4 传感器/参数配置**:
- 在 SIH 模式下，查阅官方文档与示例配置，尽可能消除 Preflight 告警;
- 确认解锁/起飞流程的正确配置，使高度曲线能真实反映 “离地” 状态。

(B) **系统化 OFFBOARD 流程**:
- 在 vuav_agent.py 中增加模式切换命令（例如 MAV_CMD_DO_SET_MODE 或对应参数操作);
- 在 swarm_controller.py 中封装 “进入 OFFBOARD → 周期性发送 setpoint → 退出” 的模式;
- 将 broadcast_velocity_ned 与 OFFBOARD 流程结合，形成一个可演示的 “直线飞行” 或 “绕圈” 样例。

(C) **逐步引入编队与任务层逻辑**:
- 从最简单的 “三机水平间隔飞行” 开始，设计一个离散的参考轨迹;
- 逐步考虑碰撞避免、轨迹跟踪误差等指标;
- 为将来与 TEE 内的控制算法对接预留清晰的接口(例如将编队控制封装为 “给 swarm_controller 提供一个轨迹列表” 等)。

(D) **网络与集群扩展规划**:
- 在保持 vuav-net 结构的前提下，验证 4/6/8 机规模的运行情况，评估 CPU/内存与网络开销;
- 调整 vuav_cluster_config.json 以支持更灵活的命名与分组（例如分编队、小队);
- 为迁移到云环境（Kubernetes/多主机 Docker）预先设计好 overlay 网络映射与服务发现方案。

(E) **状态缓存与观测系统**:
- 在保持每个 vuav_agent 本地缓存的基础上，评估引入轻量级集中缓存/消息队列（如 Redis）的必要性;
- 设计统一的状态上报通道和数据结构，为后续日志分析、可视化以及 TEE 内推理提供数据基础。

(F) **文档与教学材料完善**:
- 将当前文档整理成对学生/合作者友好的 “快速上手指南” 版本，明确 “三步启动”（建网、起集群、起群控);

- 增加架构图（容器/agent/PX4/群控之间的关系）、数据流图等；
- 在后续迭代中逐渐补充更多 “典型场景” 示例（如单机调试、多机编队 demo、云上部署样例）。

本文档旨在对当前平台状态做一次阶段性 “对账”，后续每次较大迭代（例如引入 OFFBOARD、编队控制、TEE 集成、集中缓存等）都可以在此基础上增量更新，形成一份持续演化的项目说明。