一、基于 LXD 的实验室公共 GPU 服务器搭建：

1. LXC：全称是 Linux Container（linux 容器），LXC 是一种内核虚拟化技术，可以提供轻量级的虚拟化，以便隔离进程和资源（通过一个 linux 内核在一个受控主机上虚拟地运行多个 linux 系统）；容器有效地将由单个操作系统管理的资源划分到孤立的组中，以更好地在孤立的组之间平衡有冲突的资源使用需求；与传统虚拟化技术相比，它的优势在于：其和宿主机使用同一个内核，性能损耗小；
2. LXD：LXD 是一个系统级的容器，类似于虚拟机或物理机，它是对 LXC 的基础打包和升级，可在内部运行完整的操作系统；解决了 LXC 中存在的一些缺点，比如无法有效支持跨主机之间的容器迁移、管理复杂等；
   A. 容器的构建和管理：LXD（将现有的容器打包成镜像等）；
   B. 容器的存储：ZFS 是一种现代 Linux 文件系统，其可以汇集和管理来自多个物理存储设备的数据，再将数据聚合到 zpool（存储池）中，然后将其组织起来以供操作系统有效使用；与任何其他文件系统一样，ZFS 使用最先进的数据保护来管理存储的数据和文件，以保持数据完整性；注：在 LXD 初始化时，指定创建的 zfs-pool 为存储池；
   C. 网桥（桥接上网-Netplan：实现在实验室的其他电脑访问服务器中的 LXD 容器）：
      1) 概述：使用桥接模式，每个容器的 IP 与实验室的 IP 域相同，物理外部、局域网内部的机器可以直接 ssh 容器的 IP，不需要端口号就可以登陆。
      2) 配置方法：容器、宿主机配置好之后，之间重启服务即可；
   D. 移动硬盘挂载：mount 挂载（到特定的目录下），以便后面的共享文件和 samba 的部署；
   E. 文件如何共享（共享目录 from 宿主机 to 容器）：
      使用 lxc 命令：sudo lxc config device add <container> <device-name> disk path=/home/xxx/share source=/home/xxx/share；
   F. 如何访问机械硬盘（内网通过 samba：局域网下在 linux 和 windows 系统之间共享文件；外网通过 frp 的文件访问服务）：
      1) 本地登录（samba 部署完毕，链接到移动硬盘挂在的目录）：本地“win+R”输入\\192.168.1.200\share 即可；
      2) frp 的内网穿透：通过 plguin_local_path 指定移动硬盘挂载的目录（直接给定远程端口即可）；
   G. 内网穿透：frp（基于的腾讯云的轻量应用服务器）：
      1) frp 概述：是一个高性能的反向代理应用，可以帮助用户轻松地进行内

网穿透，对外网提供服务，支持 tcp, http, https 等协议类型；其采用 C/S 模式，将服务端部署在具有公网 IP 机器上，客户端部署在内网或防火墙内的机器上，通过访问暴露在服务器上的端口（端口映射），反向代理到处于内网的服务；

2) frps 配置在公网服务器、frpc 配置在容器上即可；

H. 容器如何访问（win 下、linux 下、vscode 中）：

1) 命令行界面访问：win（PowerShell）和 linux 均可通过 ssh（加密的网络传输协议）远程登录；

2) 图形化界面访问：win（mstsc）进行远程桌面连接；

3) IDE 下：vscode 的 remote-ssh 插件远程连接（在 config 文件中配置 ip 和端口即可，且通过 ssh-keygen -t rsa 生成密钥文件可实现免密登录）；

3. Docker：Docker 是一个开源的应用容器引擎，让开发者可以打包他们的应用以及依赖包到一个可移植的镜像中，然后发布到任何流行的 Linux 或 Windows 操作系统的机器上，也可以实现虚拟化；Docker 是一个用于在集中式平台上创建、部署和运行应用程序的开源工具；这使得主机的操作系统通过容器运行具有相同 linux 内核的应用程序（公用内核），而不是创建一个完整的虚拟机；

4. LXD 和 Docker 的区别：

A. 前者是一个系统级容器、后者是一个应用程序容器；

B. 前者只支持 linux、后者支持 linux、windows、macOS；

C. 前者支持可视化页面、后者仅支持命令行页面；

5. 传统虚拟机和操作系统虚拟化（共享内核）的区别：

A. 传统的虚拟机使用了一种叫做 hypervisor（虚拟机监视器）的东西，它运行在内核之上，该管理程序通过监视其资源使用情况和访问模式，为在其上运行的应用程序提供虚拟化；这会导致大量开销，导致不必要的性能损失；

B. 操作系统虚拟化的工作方式与其不同；它使用内核的 namespace（进程命名空间）和 cgroup（进程按组管理）技术来限制应用程序的功能，包括资源的使用（资源隔离、进程隔离），这是 linux 内核提供的一个特性，几乎没有开销；

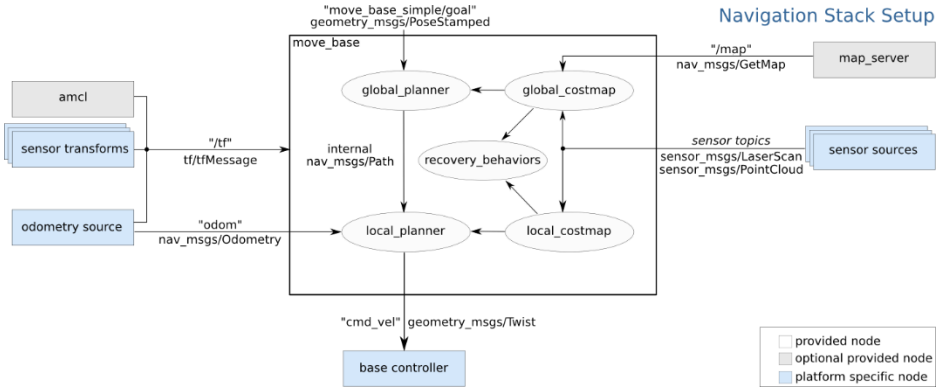
C. 总结：Docker 和 LXD 与主机操作系统共享一个内核，并利用它来创建独立的进程，主要区别在于 Docker 运行单个应用程序 / 流程，而 LXD 运行完整的操作系统，这为它们支持的工作负载类型提供了灵活性；

二、无人车编队：

1. 系统架构：

A. navigation：ROS 的二维导航功能包，其根据输入的里程计和激光雷达等传感器的信息流，通过导航算法，计算得出安全可靠的机器人速度控制指令；

B. move_base：navigation 中的核心包，其提供了 action 动作的实现（actionlib 包），即给定一个世界系下的目标位置（actionlib 包控制目标点的给定和取消），机器人会试图移动到该位置；另外，move_base 节点中包含了两个代价地图（全局、局部），以及一个全局规划器和一个局部规划器，以便实现导航任务；其需要的输入如下（数据输入如下）：



C. 全局路径规划器：asr_navfn 全局路径规划包（内置迪杰斯特拉算法，ros 的 navigation 包中默认全局路径规划器（navfn）的升级版本）；

D. 局部路径规划器：TEB（针对阿克曼模型的底盘）/ DWA（针对差速底盘）；前者将路径看作橡皮筋，然后用外力控制形变（目标点的引力和障碍物的斥力）、后者是一种贪心的算法，通过可选速度、可选角速度的组合，模拟出很多局部轨迹，然后选择最优的（采样）；

2. 硬件通信（底盘、手柄）：

A. 解析手柄：根据 SBUS 通道结构体（转串口数据）进行解析；

B. 底盘通信：can 协议、USB 协议（根据厂家提供的底盘来确定）；将来自 TEB 规划出来的速度或手柄遥控给出的速度转换底盘需要的数据（将帧发送到 can 总线上）；

C. 解析惯导：跟据 GPCHC 协议进行解析；

3. 编队通信：

A. 如何实现：前车每前进 3m，将上一时刻的位置发送给后车，作为后车的目标点，而头车的路径点是第三方软件（地图编辑器）给定的（电台搭建组网，使得三辆车处于同一局域网下，为 UDP 通信创造前提）；

B. UDP 通信：解析定义的 UDP 通信的协议，数据的发送和接收；

4. 遇到的问题：

A. 全局路径规划器：动态障碍物移至在已知规划的目标点上，使得规划器崩溃，车停止不动，采用 navfn 的升级版本（asr_navfn），其优势在于：它使用 Dijkstra 算法计算到达目标的最短（加权）路径，然后检查目标是否在障碍物中，如果是这种情况，则从目标向后遍历路径，直到找到一个不属

于障碍物的点（在栅格地图上）；

B. UDP 通信丢包（不适合 TCP）：低增益电台（维护组网质量）；软件层面开辟多线程或增加系统收发的缓冲区；

C. 用到的数据结构：根据硬件要求定义相关的结构体（惯导滤波用到了队列（维护 10 个经纬度的队列）和数组，然后取中位数，区分直道弯道，设置超调参数）；vector（用于从配置文件中读取相关参数）；

D. 坐标系（大体坐标系）：tf 获取当前位置（map 到 base_center 的变换）；

E. 为什么采用 GPS：消除绝对误差（slam 的里程计的误差是累计的），改变了小车当前位姿的获取方式（传感器来源被改变，用激光雷达变为 GPS 惯导，坐标系也随之改变）；

F. 导航是基于栅格地图（基于占据的思想，订阅雷达原始点云数据）；

G. 规划器的切换时机：读取路径点交给全局路径规划器，点于点之间的规划由全局路径规划器负责；在局部代价地图内，由局部路径规划器负责避障（点云栅格地图）；