一、CMDB 项目大体攻略：

1.先建目录

bin #可执行文件

config #配置文件

lib #公共的模块

src #业务逻辑

2.配置文件

settings 一般都是大写，放自定制配值文件

自定义配置文件

默认配置文件

在 lib 里面写个目录 conf 在目录中建立一个 config.py 和 global_settings.py 第一个文件是放逻辑的，让二、自定义配置文件和默认配置文件结合起来，

****整合技巧

PS:

import os

os.environ['USER_SETTINGS']="config.settings"

3.开发插件（可插拔）

公司采集资产有差别

默认采集：

basic #主机名,系统版本,系统类型

board #主板

cpu

disk

memory #内存

nic #网卡

定义插件：

class xxxx(object):

def process(self):

return '1331321321321'

把这些插件都放到 settings 里面

4.向 API 获取发送数据

AGENT：

向 API 发送资产信息

SSH、SALT：

整个流程的开始应该是从 API 接口开始，先从这里拿到没有采集的主机列表获取未采集的主机列表：[c1.com,c2.com]

for host in 主机列表:

host 采集资产，发送到 API

三、为什么要开发 CMDB

Excel 维护资产信息，资产变更时难以保证 Excel 表正确性；信息交换不方便

自动采集资产工具，目标：自动汇报，保存变更记录

最终目标:实现运维自动化

四、CMDB 架构：

资产采集

API(两个功能:接收数据保存入库,对外提供数据接口比如有人访问指定的 url 就能拿到指定主机的详细信息都展示出来)

后台管理(对资产进行增删改查)

五、开发程序时你负责做什么

周期：将近三个月

三个人负责，我负责资产采集(那个简单说那个)

调研提出三种方案：agent paramiko saltstack 三种方案，三种都兼容，为了提高

扩展性，配置文件(自定义和默认)和中间件(反射)参考 Django:

六、有没有遇到难题(技术的难题不是难题,业务的难题才是难题)

Linux 不太熟，命令都不知道该用啥

唯一标识：大问题，一开始用的是主板的 SN 号，当初我们认为他就是唯一标识，

物理机的话 SN 号是唯一的，openstack 虚拟机和物理机的 SN 号是一样的

所以在实现运维自动化之前就要先做标准话,后来改为主机名，主机名不能重复，用主机名作唯一标识

主机名是唯一标识，依赖本地文件

七、最终流程：

标准化：主机名不重复，流程的标准化，装机的时候 CMDB 中主机名已经设定好了

服务器上的资产采集（AGENT）:

a.第一次采集，文件不存在或者内容为空，采集资产要把主机名写入文件，将采集信息发送到 API

b.第 N 次，采集资产,主机名从文件中获取，

八、SSH 和 Salt 不存在这种情况：

这个架构开始是从中控机获取没有采集过的主机名(主机名跟 IP 是绑定的)，而流程一开始主机名已经被录入 cmdb 了

九、为什么要进行 API 验证：

在数据传输过程中保证数据不被篡改，为了保证数据安全，数据只给通过验证的人看

十、API 验证是如何设计的：

这个设计我借鉴了 Tornado 中的加密 Cookie 来实现的，根据 client_key+time 进行 md5 创建动态的 KEY，总共有三个限制：第一个，是时间，如果时间超过服务端两秒直接 return；第二个，把客户端的 ctime 取出来，和把事先存到服务端的字符串和客户端一样的那个字符串进行 md5 加密，用这个加密的结果和客户端发来的加密结果进行对比，如果一样就通过，否则就 return；第三个，我们在服务端维护了一个字典，字典中放的是最近 2s 内访问的 client_md5_ctime_key 和客户时间+2s 组成的键值对，如果穿过来的这个值在字典中有的话我们就 return，在 for 循环查字典的时候顺便把超时的删掉，如果通过，就把他在放到字典里面。

还有一个问题就是如果黑客的网速比我快的话(外网会发生)，会比我先到 API 这个时候我需要对我的请求进行加密，这个是借鉴的微信的加密方式，也就是 AES 加密方式。