云联万维
YUNION

OneCloud V3容器化之

系统服务容器化

01

背景介绍

02

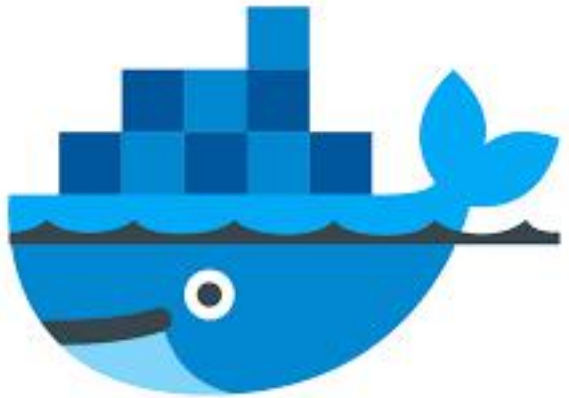
问题与解决方案

01

为什么做计算节点容器化

为什么要容器化

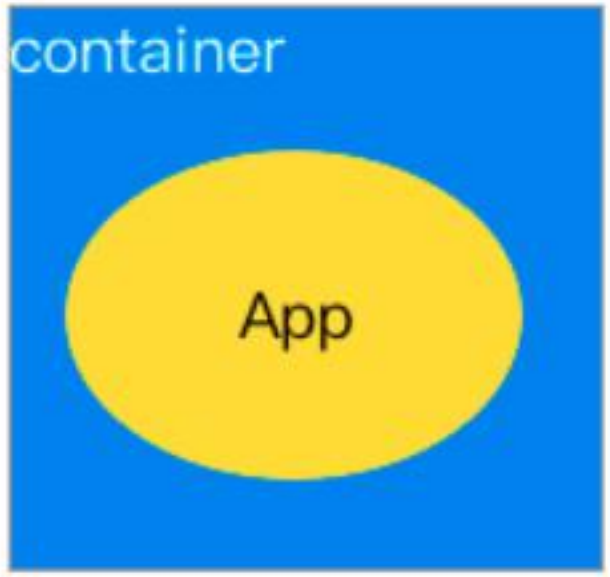
- 减少运维成本，服务升级回滚更为方便
- 日志收集管理
- 服务配置统一管理
- 统一的部署方式
- ……



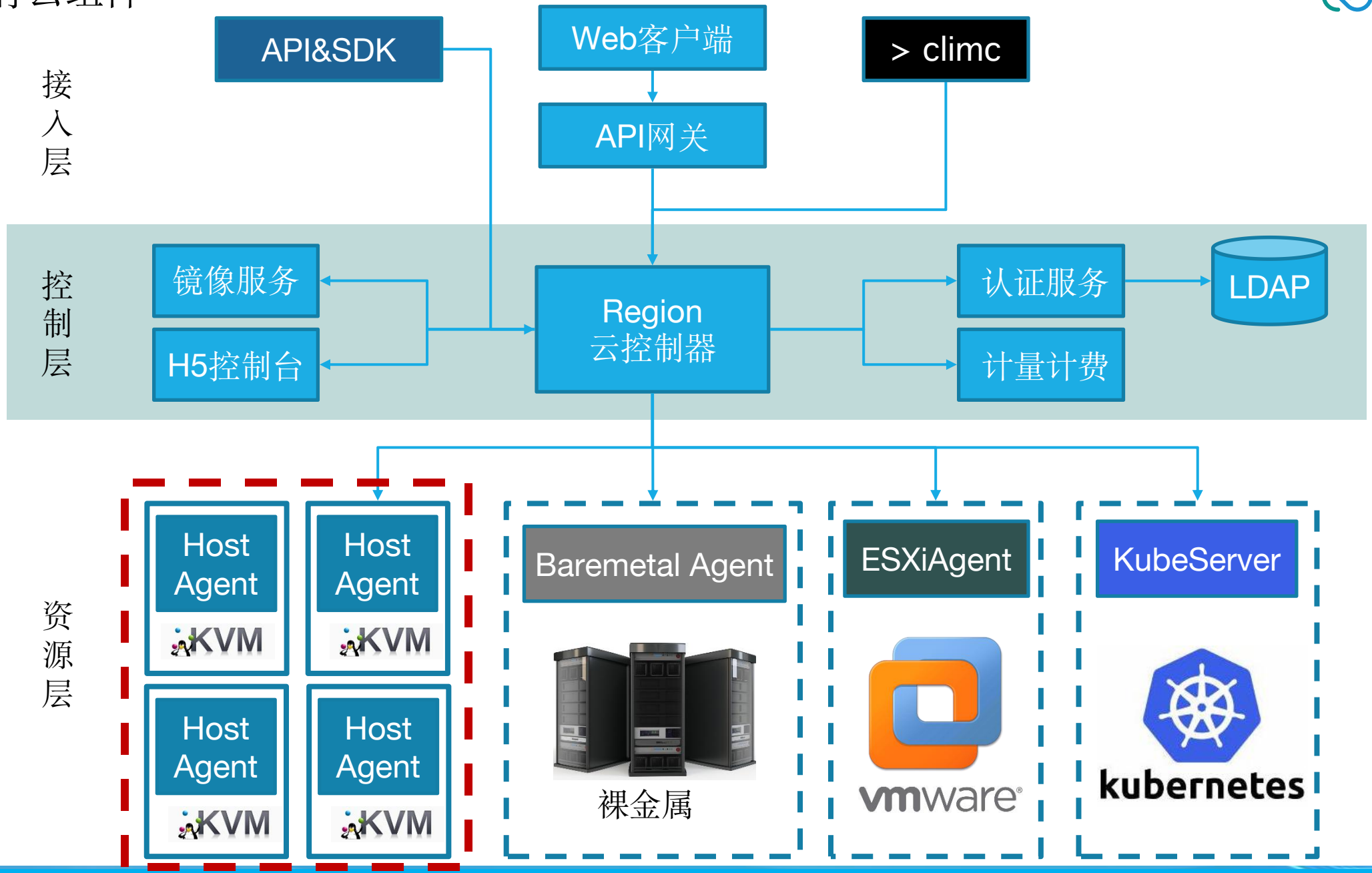
容器化系统服务面临的问题

由于计算节点上的服务的一些特性，在容器化的过程中会面临各种各样的问题

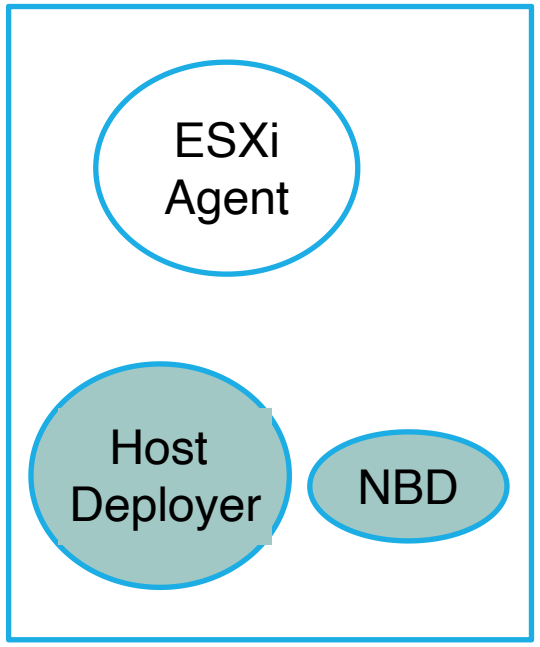
- 依赖宿主机操作系统内核模块
- 需要直接访问宿主机设备
- 在镜像制作上不是一个简单的二进制，依赖繁多
- 一些程序一定要在宿主机上运行
- 配置不能完全统一
- ……



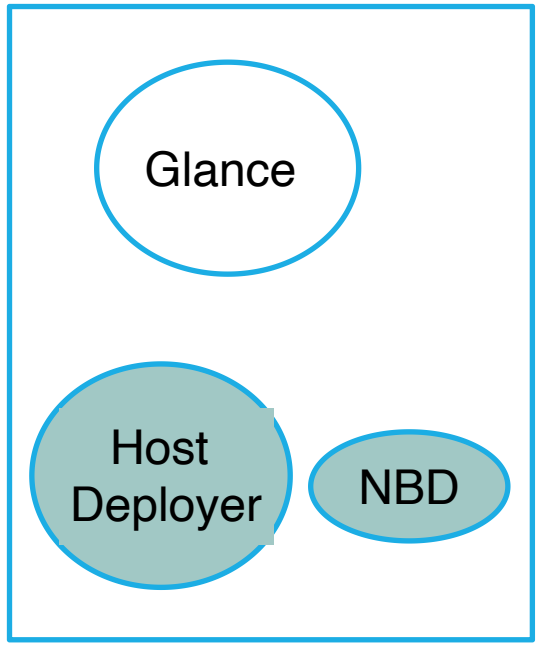
私有云组件


接入层
API&SDK
Web客户端
> climc
API网关
控制层
镜像服务
H5控制台
Region
云控制器
认证服务
计量计费
LDAP
资源层
Host Agent
KVM
Host Agent
KVM
Host Agent
KVM
Host Agent
KVM
Baremetal Agent
裸金属
ESXiAgent
vmware
KubeServer
kubernetes

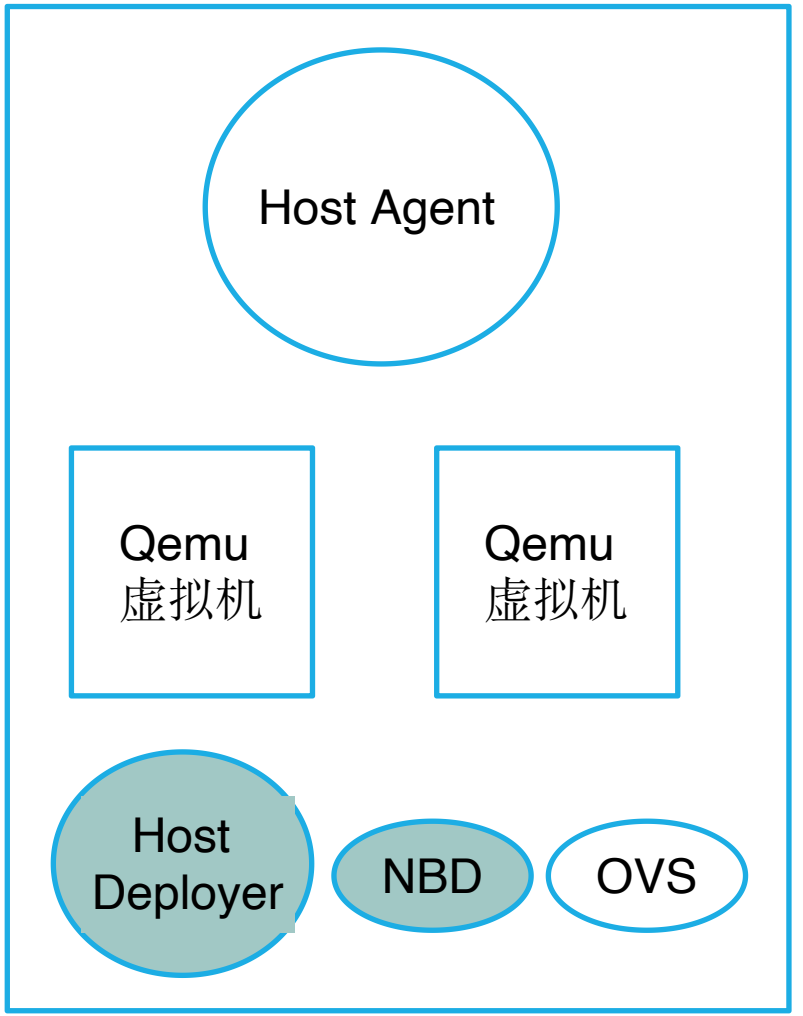
私有云计算节点相关组件



ESXiAgent



镜像服务



计算节点服务

02

容器化带来的问题与解决方案

系统服务容器化遇到的问题

- 如何在容器里运行系统服务？
- 如何制作系统服务的容器镜像？
- 如何选择合适的K8S资源模型？

如何在容器里运行系统服务？

问题：

- 需要在容器内加载宿主机的内核模块
- 虚拟机的Qemu进程必须要在宿主机上运行，不能运行在容器内部
- 无法控制宿主机的systemd服务进程

方案：

command executor:

- 作为宿主机上的代理，帮助容器进程在宿主机上执行命令

主要包括qemu相关命令、内核模块探测、系统服务检查等。

host-agent

并且 command executor 对外提供的 rpc 接口是与 golang 的标准库 os/exec 在行为上和接口是都是一致的.

```
type Cmd interface {
    StdinPipe() (io.WriteCloser, error)
    StdoutPipe() (io.ReadCloser, error)
    StderrPipe() (io.ReadCloser, error)
    CombinedOutput() ([]byte, error)
    Start() error
    Wait() error
    Run() error
    Kill() error
}
```

- 调用 os/exec 的地方可以根据不同的 driver 在容器内执行命令或者在宿主机上执行命令
https://github.com/yunionio/onecloud/tree/master/pkg/util/procutils
https://github.com/yunionio/executor

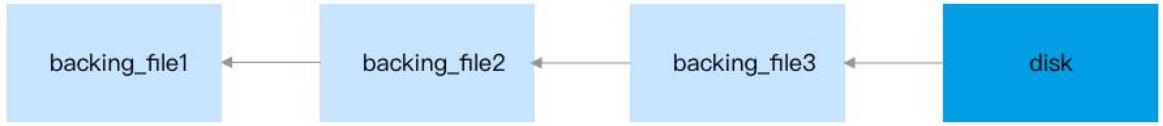
系统服务容器化遇到的问题

- 如何在容器里运行系统服务？
- 如何制作系统服务的容器镜像？
- 如何选择合适的K8S资源模型？

镜像制作

- base镜像使用的是alpine

一些二进制在编译一个alpine版本的很困难，比如host-image



host-image主要功能是展开qcow2磁盘，提供读取磁盘数据的能力，主要是为了实现远程挂载磁盘的功能

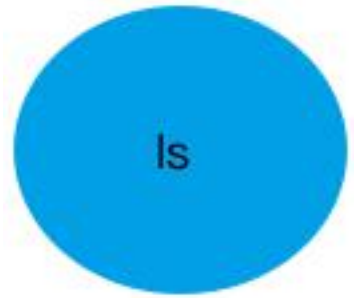
镜像制作

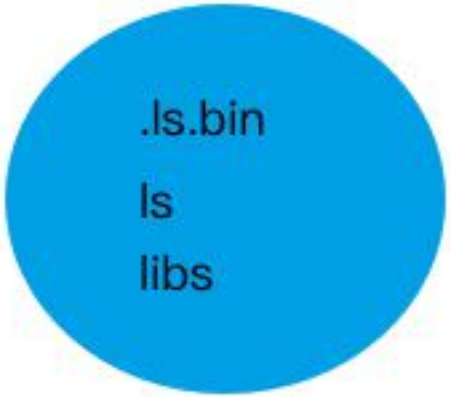
在镜像制作上引入了bundle_libraries.sh:
　将二进制程序和它依赖的动态库打包到指定的目录，这样能够尽量的避免对运行环境的依赖.

　bundle-libraries.sh是源自OpenWrt项目的一个脚本。bundle-libraries.sh直接调用ld.so，控制二进制的加载过程。尽量保证在目标环境内加载二进制的过程和原来的环境是一样的。

镜像制作

```
root@wyq-test-allinone ls # ldd /bin/ls
        linux-vdso.so.1 =>  (0x00007ffeb93fb000)
        libselinux.so.1 => /lib64/libselinux.so.1 (0x00007fb35c54a000)
        libcap.so.2 => /lib64/libcap.so.2 (0x00007fb35c345000)
        libacl.so.1 => /lib64/libacl.so.1 (0x00007fb35c13b000)
        libc.so.6 => /lib64/libc.so.6 (0x00007fb35bd6d000)
        libpcre.so.1 => /lib64/libpcre.so.1 (0x00007fb35bb0b000)
        libdl.so.2 => /lib64/libdl.so.2 (0x00007fb35b906000)
        /lib64/ld-linux-x86-64.so.2 (0x00007fb35c780000)
        libattr.so.1 => /lib64/libattr.so.1 (0x00007fb35b701000)
        libpthread.so.0 => /lib64/libpthread.so.0 (0x00007fb35b4e5000)
root@wyq-test-allinone ls #
```





镜像制作

```
root@wyq-test-allinone ls # ./bundle_libraries.sh ./bundles/ ./ls
Bundling ls
root@wyq-test-allinone ls # ls -al
总用量 140
drwxr-xr-x   3 root root   4096 3月  27 11:49 .
dr-xr-x---. 16 root root   4096 3月  27 11:44 ..
-rwxr-xr-x   1 root root   4307 3月  27 11:44 bundle_libraries.sh
drwxr-xr-x   4 root root   4096 3月  27 11:49 bundles
-rwxr-xr-x   1 root root    213 3月  27 11:49 ls
-rwxr-xr-x   1 root root 117656 3月  27 11:44 .ls.bin
root@wyq-test-allinone ls #
```

当前的ls命令被替换成一个如下脚本:

```
root@wyq-test-allinone ls # cat ls
#! /usr/bin/env sh
dir="$(dirname "$0")"
export RUNAS_ARG0="$0"
export LD_PRELOAD="$dir//bundles/lib/runas.so"
exec "$dir//bundles/lib/ld-linux-x86-64.so.2" --library-path "$dir//bundles/lib/" "$dir/.ls.bin" "$@"
root@wyq-test-allinone ls #
```

```
root@wyq-test-allinone ls # ls bundles/lib/
ld-linux-x86-64.so.2  libattr.so.1  libc.so.6   libpcre.so.1     libselinux.so.1
libacl.so.1           libcap.so.2   libdl.so.2  libpthread.so.0  runas.so
root@wyq-test-allinone ls #
```

镜像制作

除了ls依赖的动态库之外，还多了一个runas.so

```
root@wyq-test-allinone ls # ls bundles/lib/
ld-linux-x86-64.so.2  libattr.so.1  libc.so.6   libpcre.so.1     libselinux.so.1
libacl.so.1           libcap.so.2   libdl.so.2  libpthread.so.0  runas.so
root@wyq-test-allinone ls #
```

runas.so的作用就是还原程序的arg0

```c
int mangle_arg0(int argc, char **argv, char **env) {
        char *arg0 = getenv("RUNAS_ARG0");

        if (arg0) {
                argv[0] = arg0;
                unsetenv("RUNAS_ARG0");
        }

        return 0;
}
```

镜像制作

使用bundle_libraries.sh 制作可执行包的时候遇到的问题：

- 如果你的代码中有对环境依赖，如在代码中检查某个动态库是否存在

- glibc 2.20之前的版本ld.so有bug, 如果你的代码中使用了__attribute__((constructor)), constructor function 会被调用两次, 可执行文件init_array中的函数被ld.so先执行了一次，其后又在libc_start_main中再被执行一次。

- 可执行程序执行chroot命令后执行其他命令会找不到runas.so

系统服务容器化遇到的问题

- 如何在容器里运行系统服务？
- 如何制作系统服务的容器镜像？
- 如何选择合适的K8S资源模型？

系统服务容器化遇到的问题

常用的资源模型

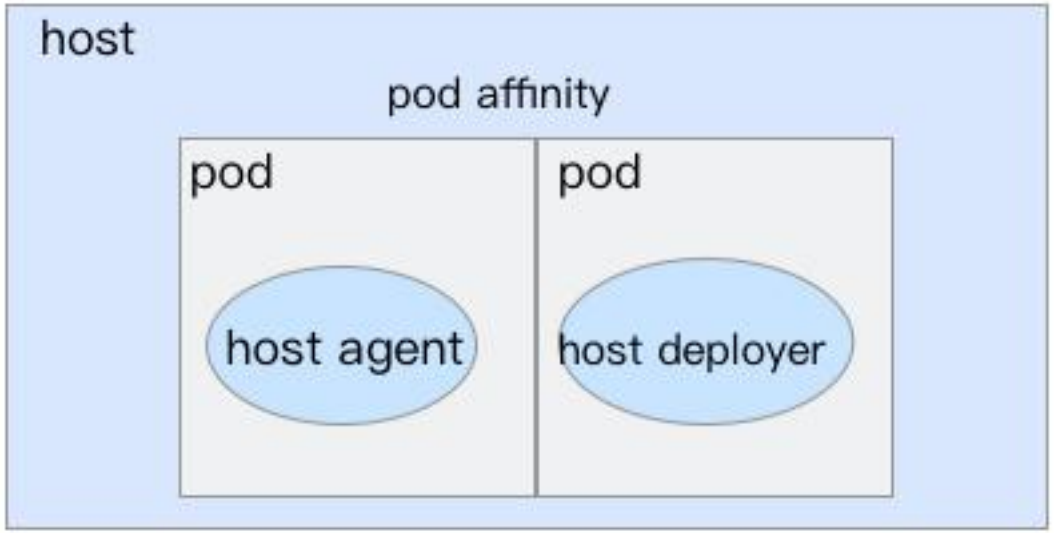
- deployment
- statefulset
- daemonset
- cronjob

容器部署实践

- 在k8s中我们将host-agent和host-deployer定义为daemonset

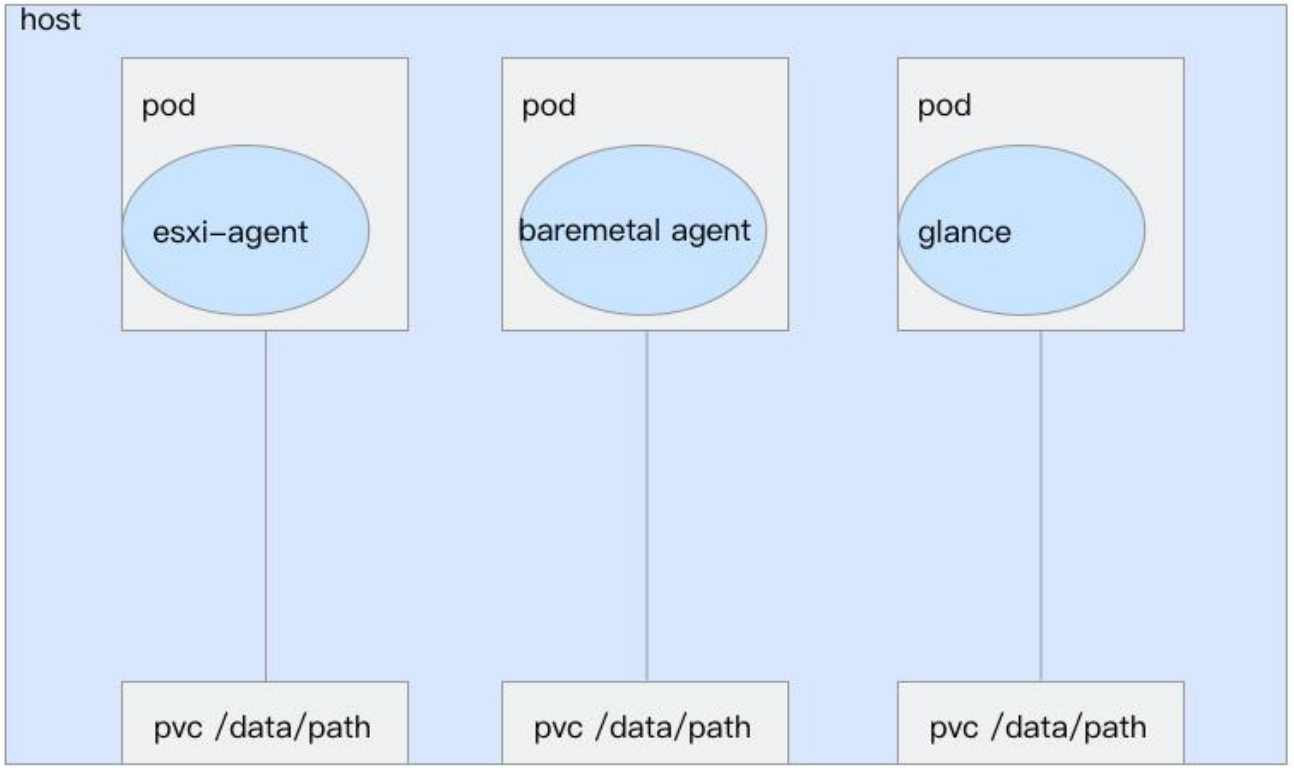
- host-agent使用node affinity来控制哪些节点启用host-agent(onecloud.yunion.io/host=enable)

- host-deployer则是由pod affinity来控制哪些节点需要部署host-deployer。



容器部署实践

- glance和esxi-agent在k8s中定义为deployment, 他们都对宿主机的存储有依赖。
  他们都挂载了一个local-path的pvc, 防止重启后调度到别的节点



云联万维
YUNION

Q&A

请填写调查问卷，协助我们把活动办得更好、更符合您的需求，谢

欢迎加入OneCloud技术交流群
加云联小助手ID：yunionio

携手同行 共赢未来

智能多云领导者