QINGTENG

容器安全技术研究成果

高斌
产品技术部
2019/12/04

目录 Contents

QINGTENG

一. 容器基本概念
- 容器与容器引擎
- 容器编排工具
- 容器平台
- 容器操作系统

二. 容器技术原理
- 容器文件系统
- 容器资源隔离

三. 容器安全产品形态
- 产品形态
- 主机部署下的安全能力
- 容器化部署：Sidecar和Daemonset
- 容器化部署下的安全能力

四. 容器生命周期安全
- 容器生命周期安全
- Docker安全机制
- Kubernetes安全机制

五. 容器安全产品介绍
- Clair
- cAdvisor
- Sysdig
- Falco
- Anchore (Anchore Engine & Anchore Image Validator)

容器基本概念

- 容器与容器引擎
- 容器编排工具
- 容器平台
- 容器操作系统

容器与容器引擎

QINGTENG

容器

- 容器是一种沙盒技术，通过linux内核的namespace特性，使容器拥有隔离运行环境(pid,net,usr,mnt,uts,ipc)。
- 容器从镜像启动。镜像是存储和分发的实体。(类似iso和运行态操作系统的关系)。

容器的特性

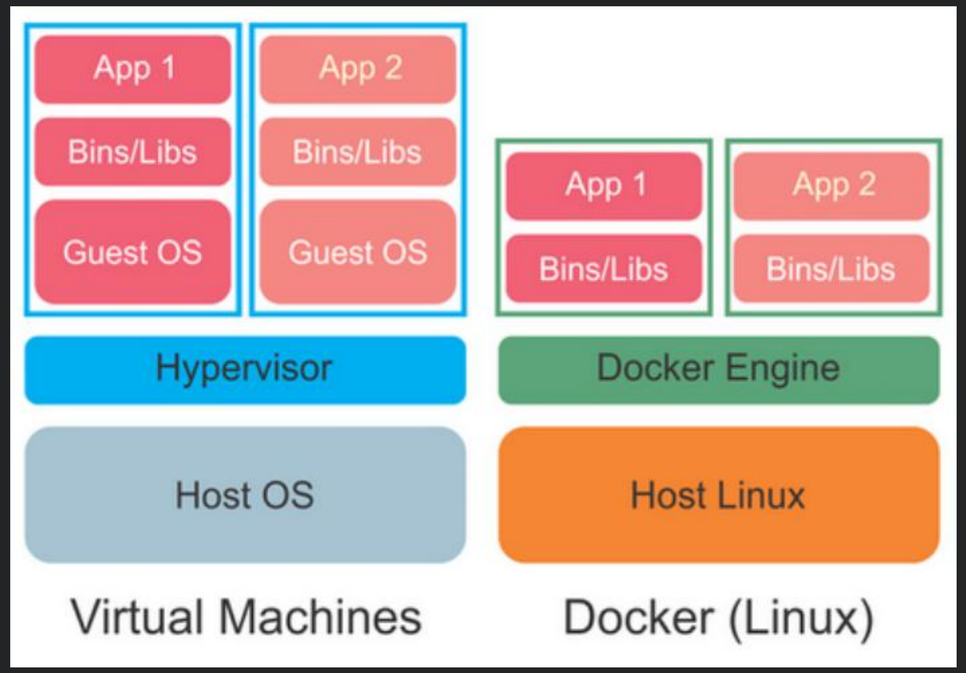
- 轻量级：容器共用宿主机内核，快速启动。
- 灵活性：将复杂的应用和环境集装箱化，快速部署，随意迁移，扩展维护方便。
- 无状态：镜像打包应用和库，不打包数据。
- DevOps的最佳解决方案。

容器与容器引擎

QINGTENG

容器引擎-Docker

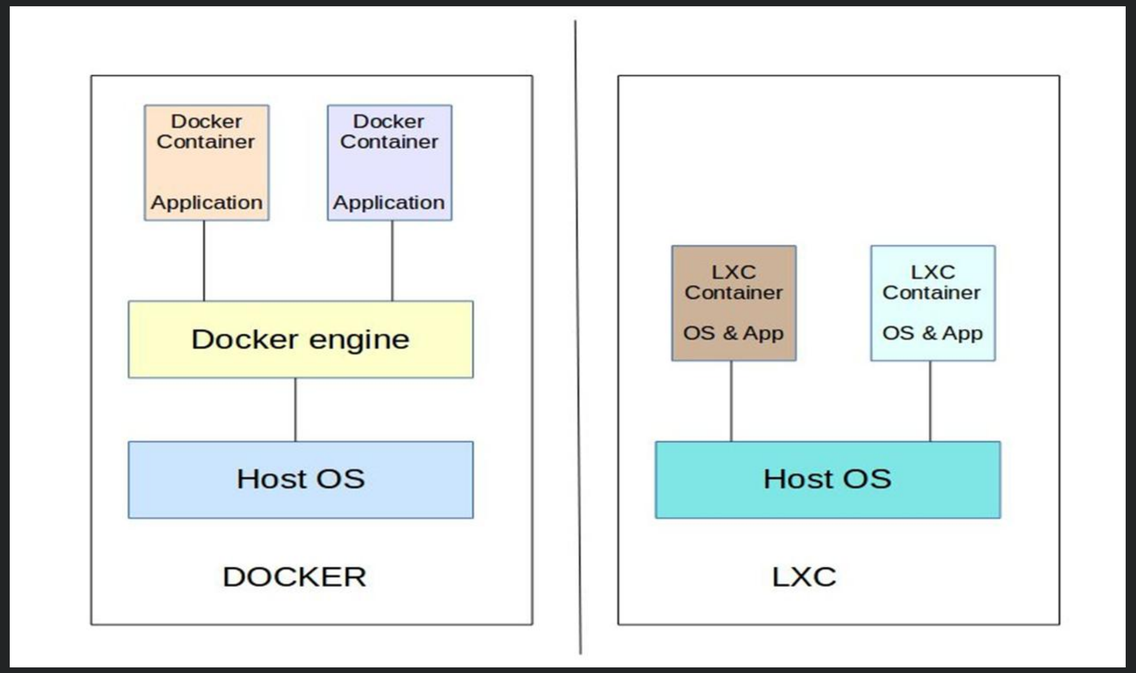
- Docker是一个开源的应用容器引擎。
- Docker镜像打包应用和依赖库。



虚拟机与Docker对比

容器引擎-LXC

- LXC是系统容器。
- LXC镜像打包了操作系统，提供完整的操作系统体验



Docker与LXC对比

容器编排工具

QINGTENG

容器编排工具用于跨多个主机协调创建，管理和更新多个容器。

Swarm

Docker自己的编排工具，现在与Docker Engine完全集成，并使用标准API和网络。Swarm模式内置于Docker CLI中，无需额外安装，并且易于获取新的Swarm命令。部署服务可以像使用docker service create命令一样简单。

Kubernetes（k8s)

Google对容器管理的开源解决方案。它适用于多种生产环境，包括裸机，内部部署虚拟机，大多数云提供商，以及三者的组合/混合。

Amazon Elastic Container Service（ECS）

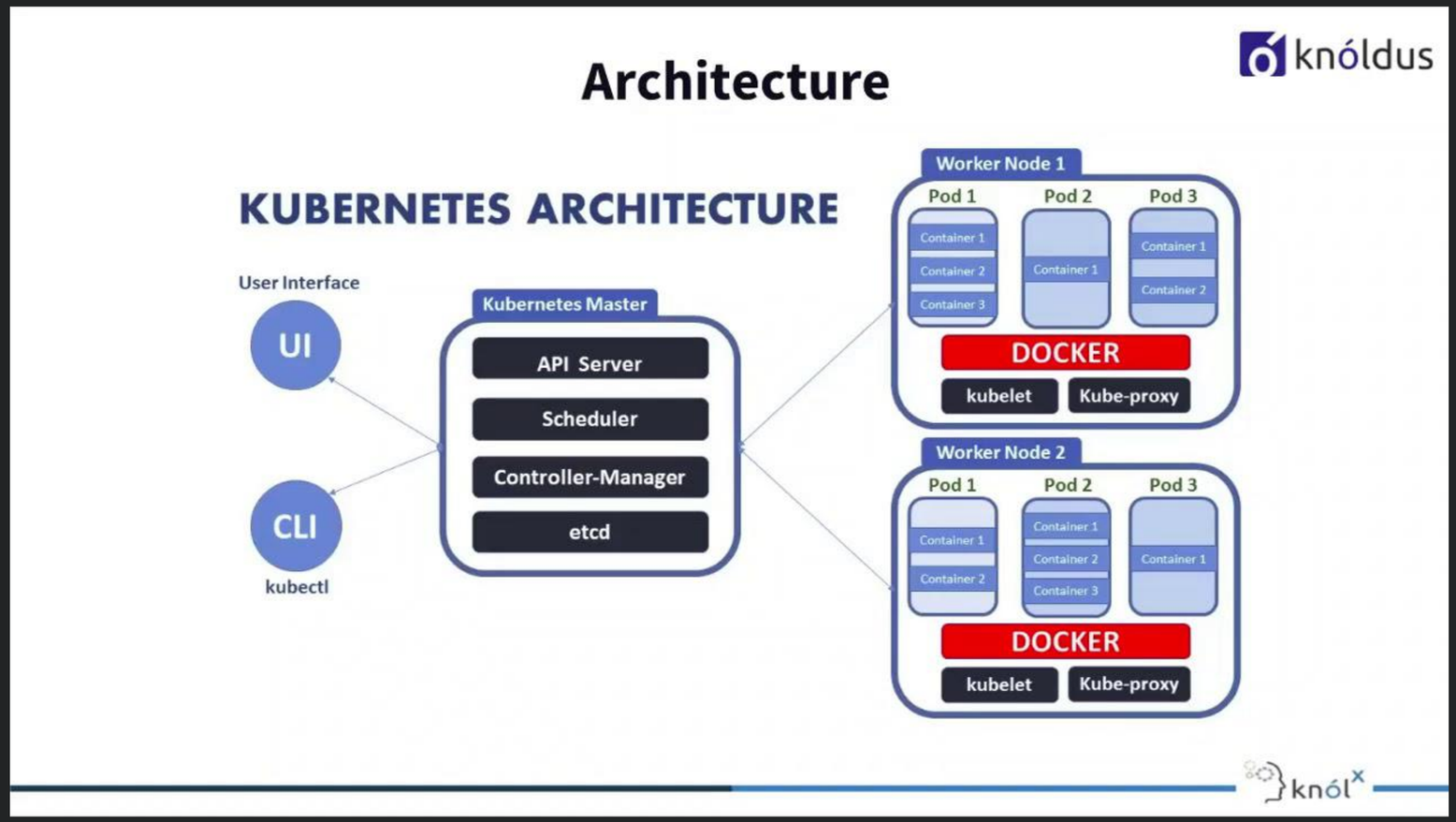
亚马逊专有的容器调度程序，旨在与其他AWS服务协调工作

容器与容器引擎

QINGTENG

- k8s的主机角色分为和Master和Node。
- Master：主节点，承载 k8s 的控制和管理整个集群系统的控制面板。
- Node(工作节点，运行实际应用)。
- k8s调度的最小单元是pod，一个POD中的所有容器都必须被调度到同一台机器上。
- 一个pod包含一个或多个容器，一个pod内的容器共享网络和端口。

Kubernetes（k8s) 架构

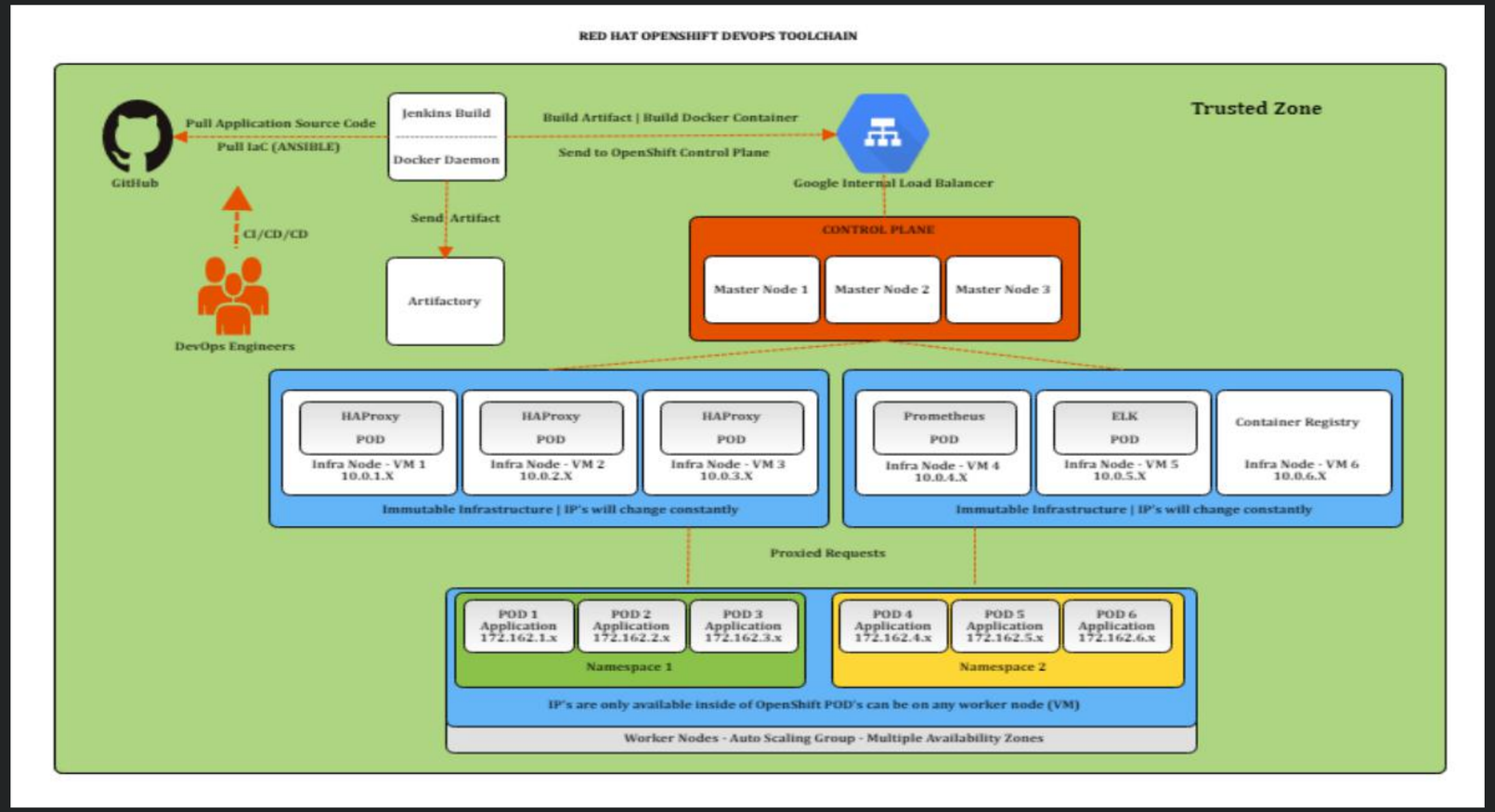


容器平台

容器编排工具用于跨多个主机协调创建，管理和更新多个容器。

OpenShift

- 以Docker作为容器引擎驱动。
- 以Kubernetes作为容器编排引擎组件。
- 提供开发语言、中间件、自动化流程工具及界面等元素，提供了一套完整的基于容器的应用云平台。
- 是基于Kubernetes管理的平台即服务（PaaS）。



灵雀云/DaoCloud等

容器操作系统

QINGTENG

容器操作系统是用来运行容器的专用操作系统（区别于镜像操作系统）

CoreOS

基于Chrome OS再定制的轻量级Linux发行版本，剔除了其他对于服务器系统非核心的软件，比如GUI和包管理器。在CoreOS 中，所有应用程序都被装在一个个 Docker容器中。

RancherOS

RancherOS是一种极简的Linux发行版，用于运行Docker容器。它可以直接在内核上运行Docker，取代了初始化(init)系统，并将Linux服务作为容器来交付，这些系统服务包括udev 和rsyslog。RancherOS仅仅包括最少运行Docker所需要的软件。

Photon OS

Photon OS是一种极简的Linux操作系统，面向针对VMware平台优化的云原生应用程序。它使用采用多种格式(包括Docker、Rkt和Garden)的容器，运行分布式应用程序。

Project Atomic

Project Atomic是红帽公司针对部署和扩展容器化应用程序的许多开源基础设施项目的统称。它为一套Linux Docker Kubernetes(LDK)应用程序架构提供了一种操作系统平台，基于Fedora、CentOS和红帽企业级Linux(RHEL)。

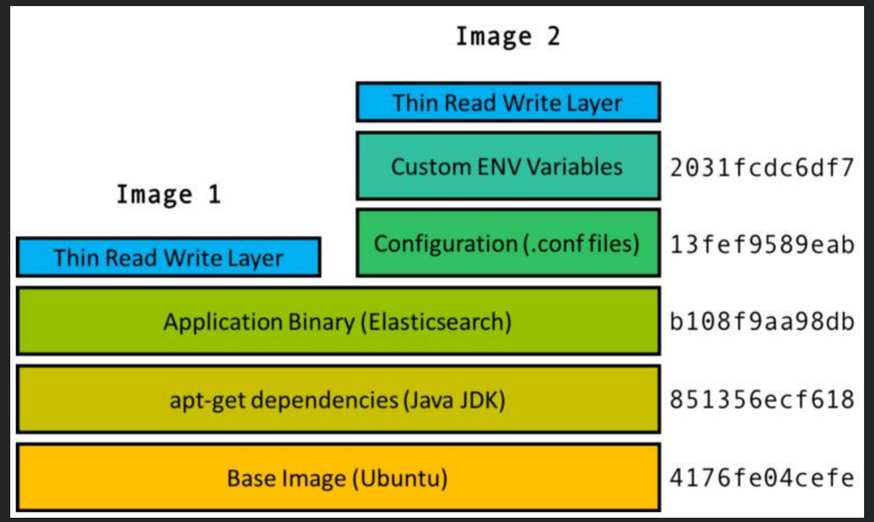
容器技术原理

- 容器文件系统
- 容器资源隔离

容器文件系统

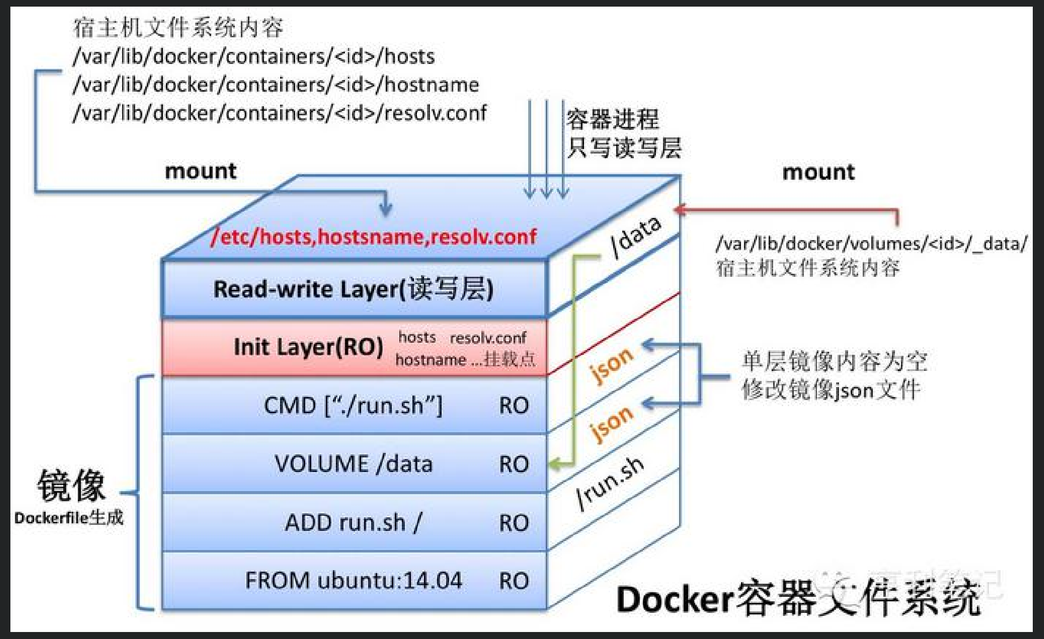
QINGTENG

- 容器由镜像启动，镜像按层存储。每一层包括文件系统变更和json描述文件
- 层新增/修改文件，例/etc/shadow表示覆盖下层的/etc/shadow文件
- 层删除文件以.wh.开头，例/etc/.wh.shadow表示删除/etc/shadow
- 容器内的修改只对读写层生效



Dockerfile 示例

```
FROM ubuntu:14.04
 ADD run.sh /
 VOLUME /data
 CMD ["./run.sh"]
```



容器资源隔离

QINGTENG

Linux Namespace

- pid namespace: 容器拥有独立的/proc文件系统
- net namespace: 容器拥有独立的网络栈
- usr namespace: 容器拥有独立的用户和组
- mnt namespace: 提供磁盘挂载点和文件系统的隔离能力
- uts namespace: 容器拥有独立的主机名(nodename)和域名(domainname)
- ipc namespace: 提供进程间通信的隔离能力

容器默认会创建以上六个namespace，创建容器时可以通过 --pid/--network/--uts/--ipc=host or container:<name|id>
设置使用主机或其他容器的命名空间

CGroup

Linux Namespace 帮助进程隔离出自己的单独空间，而 Cgroups 则可以限制每个空间的大小。Cgroups 提供了对一组进程及将来子进程的资源限制、控制和统计的能力。

容器安全产品形态

- 产品形态
- 主机部署下的安全能力
- 容器化部署：Sidecar & Daemonset
- 容器化部署下的安全能力

容器安全产品形态

QINGTENG

运行在容器内

部署 方式一：Agent打包到镜像，随镜像启动；
方式二.：镜像启动容器后，Agent注入容器

缺点 每个容器内部署一个Agent比较耗资源，对容器环境的侵入性较高。

运行在主机上

部署 Agent部署到主机上，按需检测容器

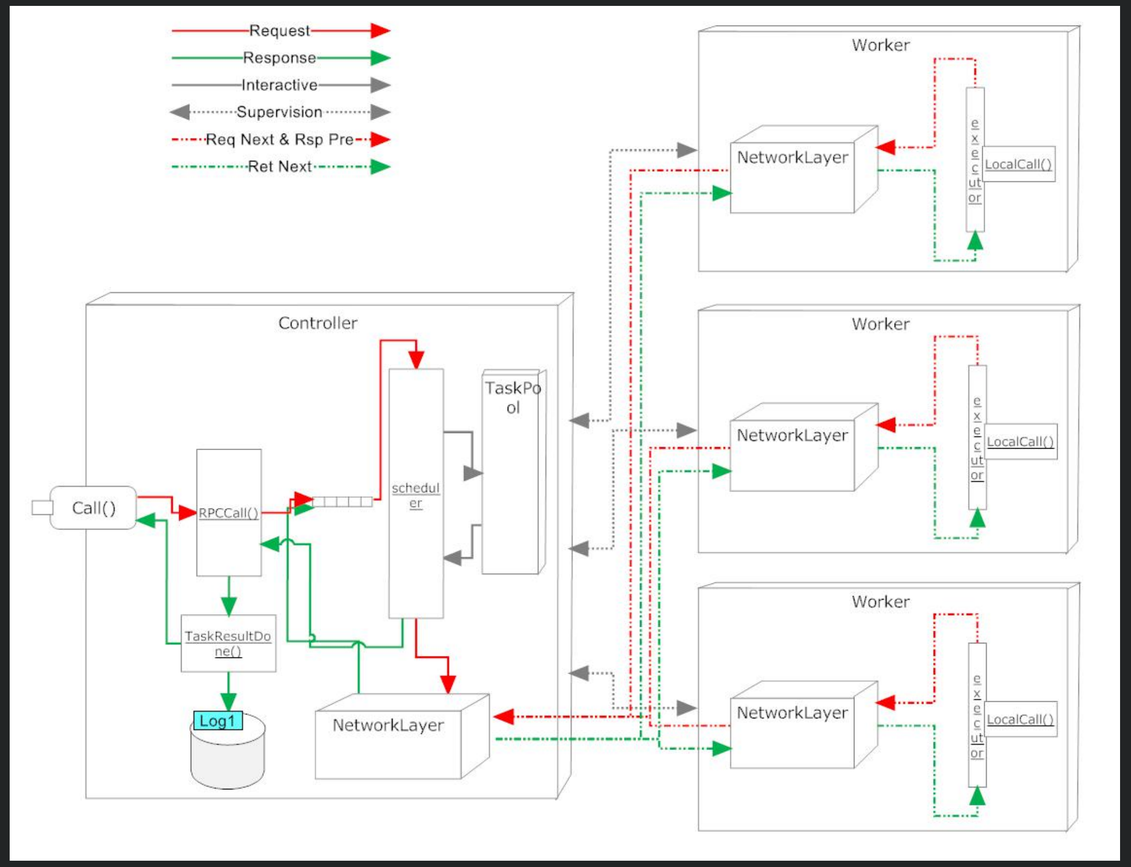
缺点 无法在CoreOS,RancherOS,Atomic操作系统上运行。

容器化部署

部署 包括Sidecar和Daemonset模式

主机部署

- 业务逻辑在主机执行
- API调用在容器内执行（RPC调用）



以反弹SHELL功能为例：

```
modified   src/check_reverse_shell.lua
@@ -23,11 +23,15 @@ local confchecker = agent.require("confchecker")
 local agent_netstat = agent.require("agent_netstat")
 local audit = agent.require("audit")
 local string_api = agent.require("agent.platform.linux.string_api")
+local docker_api = agent.require("agent.platform.linux.docker_api")
+local snippet_api = agent.require("agent.platform.linux.snippet_api")
 local libfs = agent.require "libfs"
 local titan_ip_list = nil
 local titan_ip_list_get_time = nil
 local listen_pid = {}

+local snippet_vm = nil
+
 local function check_ip_is_exist(ip, ip_info_list)
     local is_exist = false
     if ip_info_list == nil then
@@ -107,7 +111,7 @@ local function check_ip_list()
 end

 local function get_fd_info(proc_pid, fd)
-    local ret, info = audit.audit_get_fd_info2(proc_pid, fd)
+    local ret, info = snippet_vm:run("audit.audit_get_fd_info2", proc_pid, fd)
     if ret and info ~= nil and type(info) == "table" then
         --agent.print_r(info)
         if info.type == "tcp" or info.type == "udp" then
@@ -148,7 +152,7 @@ end

 local function check_socket_status(pid)
 -- 获取端口状态
-    local listen_ret, listen_msg = agent_netstat.get_netstat("tcp_listen")
+    local listen_ret, listen_msg = snippet_vm:run("agent_netstat.get_netstat", "tcp_listen")
     if listen_ret ~= 0 then
         agent.debug_log(string.format("check_socket_status listen_ret = %d ", listen_ret))
         return false
@@ -1017,6 +1021,9 @@ local function check_reverse_shell_rule(data, param)
         return true
     end

+    local container_id = docker_api.get_container_id_from_pid(data.proc_pid)
+    snippet_vm = snippet_api.init(container_id)
+
```

容器化部署 - Sidecar & Daemonset

QINGTENG

Sidecar和Daemonset是k8s的两种容器模式。

Sidecar

每个Pod内部署一个Agent容器，一个Pod内可以有多个不同的Sidecar容器。

优点：减少容器间通信延迟
缺点：1. 有侵入性。2. Sidecar容器意外退出会导致整个pod重启

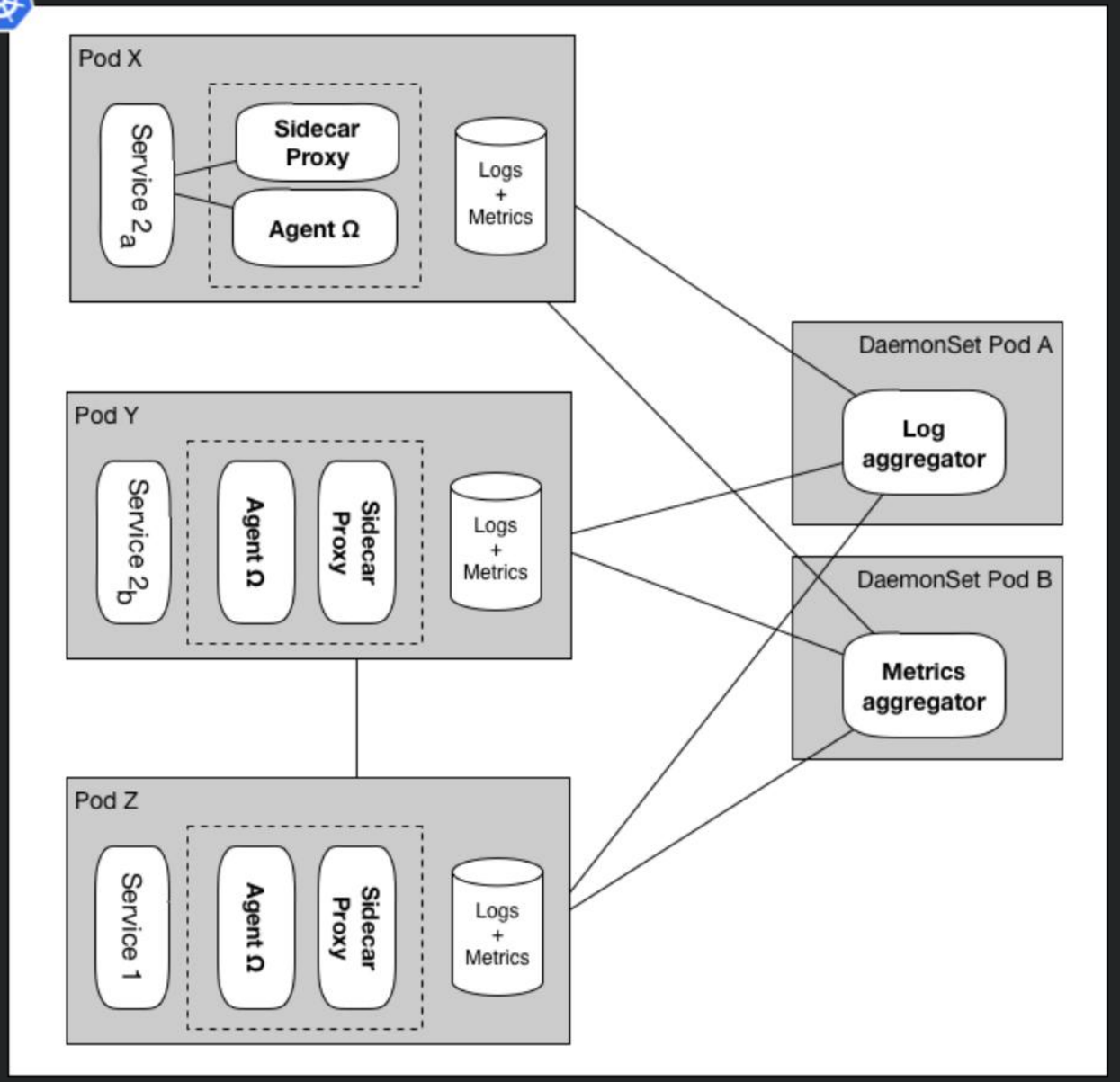
代表产品：Aqua Enforcer

Daemonset

每个Node(主机)部署一个Agent容器，一个Node上可以部署多个不同类型的Daemonset容器。

优点：1. 无侵入性。2. 资源占用低

代表产品：StackRox, NeuVector, Twistlock



容器化部署 – 能力及特权

QINGTENG

Docker在镜像启动时，可以为容器添加能力(Capability)及特权

--cap-add: Add Linux capabilities
--cap-drop: Drop Linux capabilities
--pid: PID namespace to use
--network: NET namespace to use
--privileged=false: Give extended privileges to this container
--device=[]: Allows you to run devices inside the container without the --privileged flag.

linux内核的Capabilities大部分是不区分namespace的，如果容器内进程拥有某个Capability，就和主机上进程有相同的能力。
docker为了管控容器进程的能力，默认删除了容器的部分Capability。

通过 --cap-add 可以为容器添加能力。
通过 –device 可以允许容器访问设备。
通过 --pid=host 设置容器使用主机的pid namespace，可以看到主机/proc目录，strace或gdb主机进程。
通过 --network=host 设置容器使用主机的net namespace，可以共用主机的网络栈。

容器化部署 – 默认能力

默认拥有能力

Capability Key	Capability Description
SETPCAP	Modify process capabilities.
MKNOD	Create special files using mknod(2).
AUDIT_WRITE	Write records to kernel auditing log.
CHOWN	Make arbitrary changes to file UIDs and GIDs (see chown(2)).
NET_RAW	Use RAW and PACKET sockets.
DAC_OVERRIDE	Bypass file read, write, and execute permission checks.
FOWNER	Bypass permission checks on operations that normally require the file system UID of the process to match the UID of the file.
FSETID	Don't clear set-user-ID and set-group-ID permission bits when a file is modified.
KILL	Bypass permission checks for sending signals.
SETGID	Make arbitrary manipulations of process GIDs and supplementary GID list.
SETUID	Make arbitrary manipulations of process UIDs.
NET_BIND_SERVICE	Bind a socket to internet domain privileged ports (port numbers less than 1024).
SYS_CHROOT	Use chroot(2), change root directory.
SETFCAP	Set file capabilities.

默认禁用能力

Capability Key	Capability Description
SYS_MODULE	Load and unload kernel modules.
SYS_RAWIO	Perform I/O port operations (iopl(2) and ioperm(2)).
SYS_PACCT	Use acct(2), switch process accounting on or off.
SYS_ADMIN	Perform a range of system administration operations.
SYS_NICE	Raise process nice value (nice(2), setpriority(2)) and change the nice value for arbitrary processes.
SYS_RESOURCE	Override resource Limits.
SYS_TIME	Set system clock (settimeofday(2), stime(2), adjtimex(2)); set real-time (hardware) clock.
SYS_TTY_CONFIG	Use vhangup(2); employ various privileged ioctl(2) operations on virtual terminals.
AUDIT_CONTROL	Enable and disable kernel auditing; change auditing filter rules; retrieve auditing status and filtering rules.
MAC_ADMIN	Allow MAC configuration or state changes. Implemented for the Smack LSM.
MAC_OVERRIDE	Override Mandatory Access Control (MAC). Implemented for the Smack Linux Security Module (LSM).
NET_ADMIN	Perform various network-related operations.
SYSLOG	Perform privileged syslog(2) operations.
DAC_READ_SEARCH	Bypass file read permission checks and directory read and execute permission checks.
LINUX_IMMUTABLE	Set the FS_APPEND_FL and FS_IMMUTABLE_FL i-node flags.
NET_BROADCAST	Make socket broadcasts, and listen to multicasts.
IPC_LOCK	Lock memory (mlock(2), mlockall(2), mmap(2), shmctl(2)).
IPC_OWNER	Bypass permission checks for operations on System V IPC objects.
SYS_PTRACE	Trace arbitrary processes using ptrace(2).
SYS_BOOT	Use reboot(2) and kexec_load(2), reboot and load a new kernel for later execution.
LEASE	Establish leases on arbitrary files (see fcntl(2)).
WAKE_ALARM	Trigger something that will wake up the system.
BLOCK_SUSPEND	Employ features that can block system suspend.

容器化部署安全能力 – 非特权容器

QINGTENG

非特权容器，启动时添加必要的能力(Capability)，映射必要的主机目录，可实现大部分的安全功能。

容器启动时添加相关能力（--cap-add）

- SYS_ADMIN: 包括很多管理能力，包括mount、setdomainname等，在容器内执行的相关操作在主机上生效。
- NET_ADMIN: 能在容器内控制主机防火墙、iptable等。
- AUDIT_CONTROL: 实现Audit监控，能在容器内监控主机上所有Audit事件。
- SYS_PTRACE: 能在容器内ptrace主机进程(包括其他容器内的进程)。

容器启动时映射主机路径（-v）

- /： 读取主机文件系统
- /proc: 获取所有进程信息
- /var/lib/docker: 读取镜像和容器文件系统 (实现镜像扫描、容器扫描)
- /sys/fs/cgroup: 监控容器启动，获取容器资源占用(实现资源监控)
- /var/run/docker.sock: 访问docker api (实现docker资产清点)
- /usr/bin/docker: ptrace docker cli操作 (实现docker操作审计)

代表产品：

cAdvisor, Neuvector， Twistlock, StackRox

例：Twistlock

```
"CapAdd": [
    "NET_ADMIN",
    "SYS_ADMIN",
    "SYS_PTRACE",
    "AUDIT_CONTROL"
],
"Privileged": false,
```

- /var/lib/docker — Required for accessing Docker runtime.
- /var/run/docker.sock — Required for accessing Docker runtime.
- /usr/bin/docker — Required for capturing Docker CLI access.
- /var/lib/twistlock — Required for storing Twistlock data.
- /dev/log — Required for writing to syslog.
- [optional] /usr/lib/systemd/system/docker-registry.service
- [optional] /usr/lib/systemd/system/docker.service
- [optional] /etc/default/docker

容器化部署安全能力 – 特权容器

QINGTENG

以docker run --privileged运行的容器，特权包括：

- 开启后具有所有设备的读写权限（默认不允许访问设备）
- 配置 AppArmor & SELinux
- 加载内核驱动

例：sysdig

```
docker run -i -t --name sysdig --privileged
  -v /var/run/docker.sock:/host/var/run/docker.sock
  -v /dev:/host/dev
  -v /proc:/host/proc:ro
  -v /boot:/host/boot:ro
  -v /lib/modules:/host/lib/modules:ro
  -v /usr:/host/usr:ro
  sysdig/sysdig
```

代表产品： Sysdig, Falco

容器生命周期安全

- Docker 安全机制
- Kubernetes 安全机制
- 容器生命周期安全

Docker 安全机制

QINGTENG

Docker 内容信任机制（DCT）

- 开启 DCT 的情况下，镜像会在推送时自动被签名
- push、pull 、run 时自动校验签名，build 时自动校验基础镜像签名

授权插件（authz）

Docker插件是增强Docker引擎功能的进程外扩展。包括授权（authz），卷驱动（VolumnDriver），网络驱动（NetworkDriver），Ipam驱动（IpamDrvier）。
授权插件允许接管Docker守护进程和其远程调用接口的认证和授权。

能力：审计 & 阻断 docker api 操作
代表产品：Twistlock

HTTP 代理

Docker Engine通过 /var/run/docker.sock 或者 IP:PORT 提供 api 访问。通过自定义的sock文件或者端口，实现api的代理和转发

能力：审计 & 阻断 docker api 操作
代表产品：Twistlock

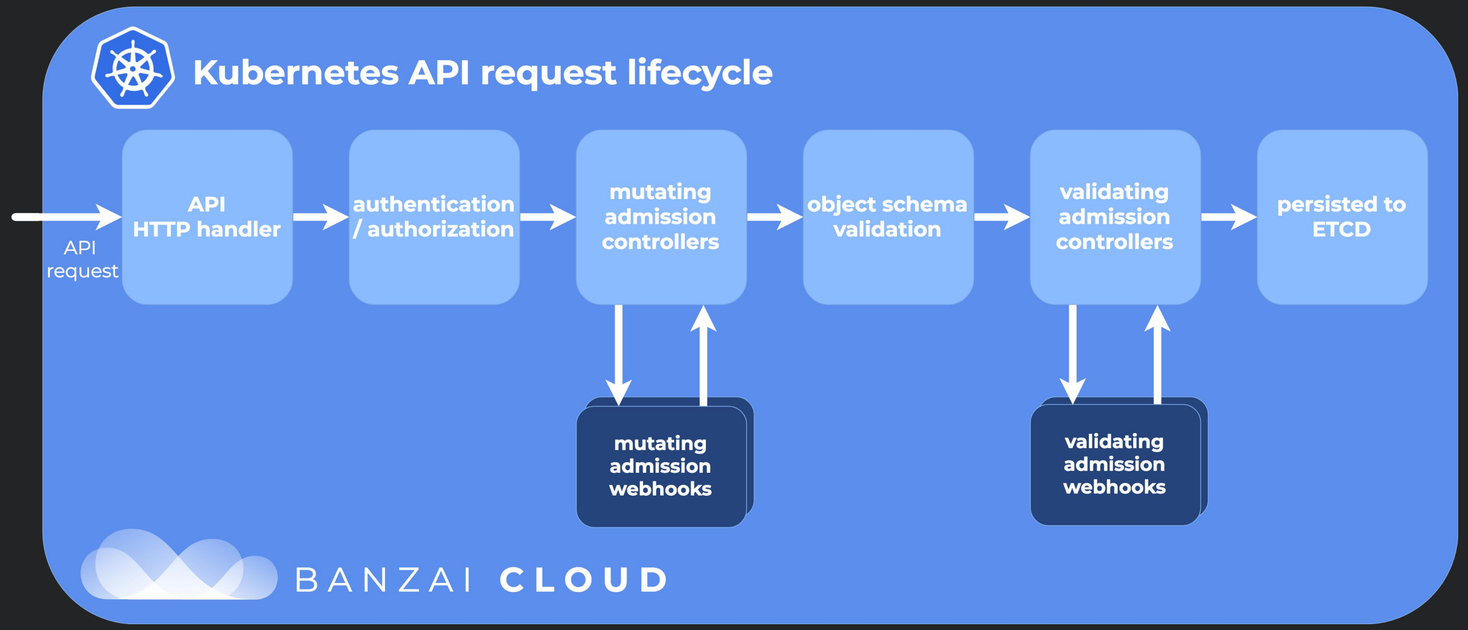
Kubernetes 安全机制

QINGTENG

准入控制（Admission Controll）

Kubernetes以webhook方式提供动态准入控制(Dynamic Admission Controll)，可以对所有api请求进行修改或者校验，包括
- mutating admission webhook允许对api请求进行修改。
- validating admission webhook不能修改请求，只能返回validate结果为true或false。

能力：检查RBAC, secret，配置，检查特权容器。必须容器化部署。
代表产品：Anchore Image Validator



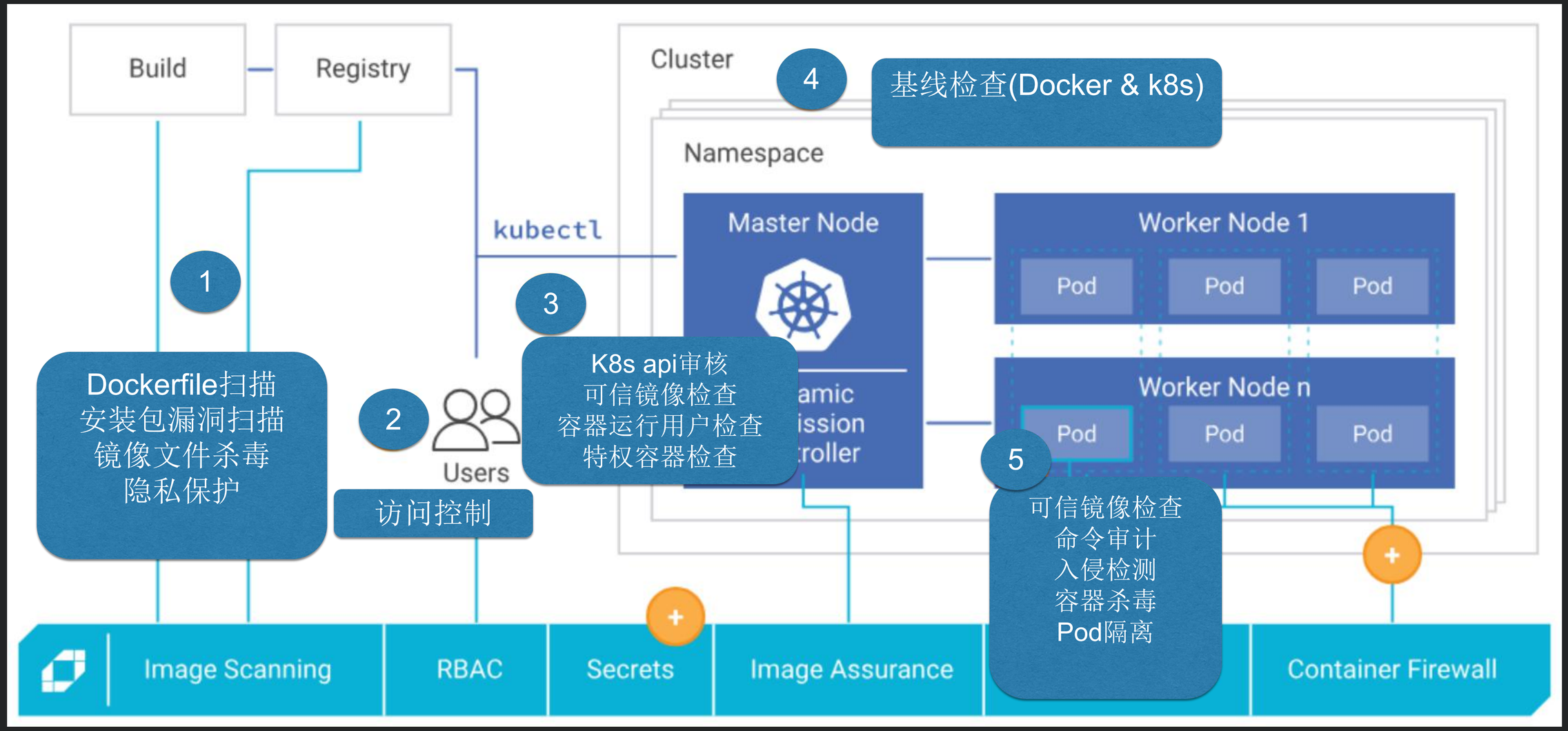
Pod隔离

k8s提供Network Policy机制，可以控制pod的网络联通性(Outbound&Inbound)。

能力：隔离异常 Pod
代表产品：Aqua

容器生命周期安全

QINGTENG



容器安全产品介绍

- Clair
- cAdvisor
- Sysdig
- Falco
- Anchore (Anchore Engine & Anchore Image Validator)

Clair

QINGTENG

CoreOS提供的镜像漏洞扫描工具。

- Harbor 集成（sh install.sh –with-clair）
- Quay 集成(Quay Security Scanner,提供Clair endpoint即可)

原理

- 在镜像所在机器上搭建一个http server，监听9279端口
- POST扫描请求到远程机器上的Clair
- Clair通过http下载层文件进行扫描

类似产品： Tenable

安装：

```
$ docker run --net=host -d -p 6060-6061:6060-6061 -v $PWD/clair_config:/config quay.io/coreos/clair-git:latest -config=/config/config.yaml
```

本机扫描：

```
$ analyze-local-images <Docker Image>
```

远程扫描：

```
$ analyze-local-images -endpoint "http://<CLAIR-IP-ADDRESS>:6060" -my-address "<MY-IP-ADDRESS>" <Docker Image>
```

cAdvisor

QINGTENG

google出的容器资源和性能监控工具。监控容器CPU、内存等信息。

原理

- 监控cgroup根目录(启动时要映射主机根目录)，获得容器启动(ContainerAdd)和退出(ContainerDelete)事件
- 通过docker api(docker stats)获取容器的资源(cpu,内存，磁盘，网络，io，进程)信息
- 通过映射主机端口，在cAdvisor容器内部提供dashboard

安装

```
docker run \
  --volume=/:/rootfs:ro \
  --volume=/var/run:/var/run:ro \
  --volume=/sys:/sys:ro \
  --volume=/var/lib/docker/:/var/lib/docker:ro \
  --volume=/dev/disk/:/dev/disk:ro \
  --publish=8080:8080 \
  --detach=true \
  --name=cadvisor \
  google/cadvisor:latest
```

Sysdig

QINGTENG

系统监控工具，主要监控系统调用。适用主机和容器，特权容器。

原理

- 使用Dynamic Kernel Module Support(DKMS)加载Sysdig-probe驱动，或ebpf(kernel version>=4.14)
- 驱动层捕获系统调用

安装

```
docker run -i -t --name sysdig --privileged
  -v /var/run/docker.sock:/host/var/run/docker.sock
  -v /dev:/host/dev
  -v /proc:/host/proc:ro
  -v /boot:/host/boot:ro
  -v /lib/modules:/host/lib/modules:ro
  -v /usr:/host/usr:ro
  sysdig/sysdig
```

输出示例

```
5352241 11:54:08.853532329 0 ssh-agent (13314) < stat res=0 path=/home/cizixs/.ssh
```

Falco

容器命令审计工具，容器化部署，特权容器。

原理

- 审计主机系统系统调用（事件来源：内核驱动sydig-probe/falco-probe，或ebpf(kernel version>=4.14)）
- 审计Kubernetes Pod/Service创建

安装

```
docker run -i -t --name falco --privileged \
  -v /var/run/docker.sock:/host/var/run/docker.sock \
  -v /dev:/host/dev \
  -v /proc:/host/proc:ro \
  -v /boot:/host/boot:ro \
  -v /lib/modules:/host/lib/modules:ro \
  -v /usr:/host/usr:ro \
  sysdig/falco
```

Falco规则示例

QINGTENG

```
- rule: Run shell untrusted
  desc: an attempt to spawn a shell below a non-shell application. Specific applications are monitored.
  condition: >
    spawned_process
    and shell_procs
    and proc.pname exists
    and protected_shell_spawner
    and not proc.pname in (shell_binaries, gitlab_binaries, cron_binaries,...)

- rule: Launch Privileged Container
  desc: Detect the initial process started in a privileged container. Exceptions are made for known trusted images.
  condition: >
    container_started and container
    and container.privileged=true
    and not trusted_containers
    and not user_trusted_containers
  output: Privileged container started (user=%user.name command=%proc.cmdline %container.info
image=%container.image.repository:%container.image.tag)
  priority: INFO
```

Anchor Engine

集成到Jenkins对镜像进行扫描。

- 提供Jenkins插件+Anchor Engine endpoint
- 功能：Dockerfile扫描，镜像漏洞扫描，隐私保护

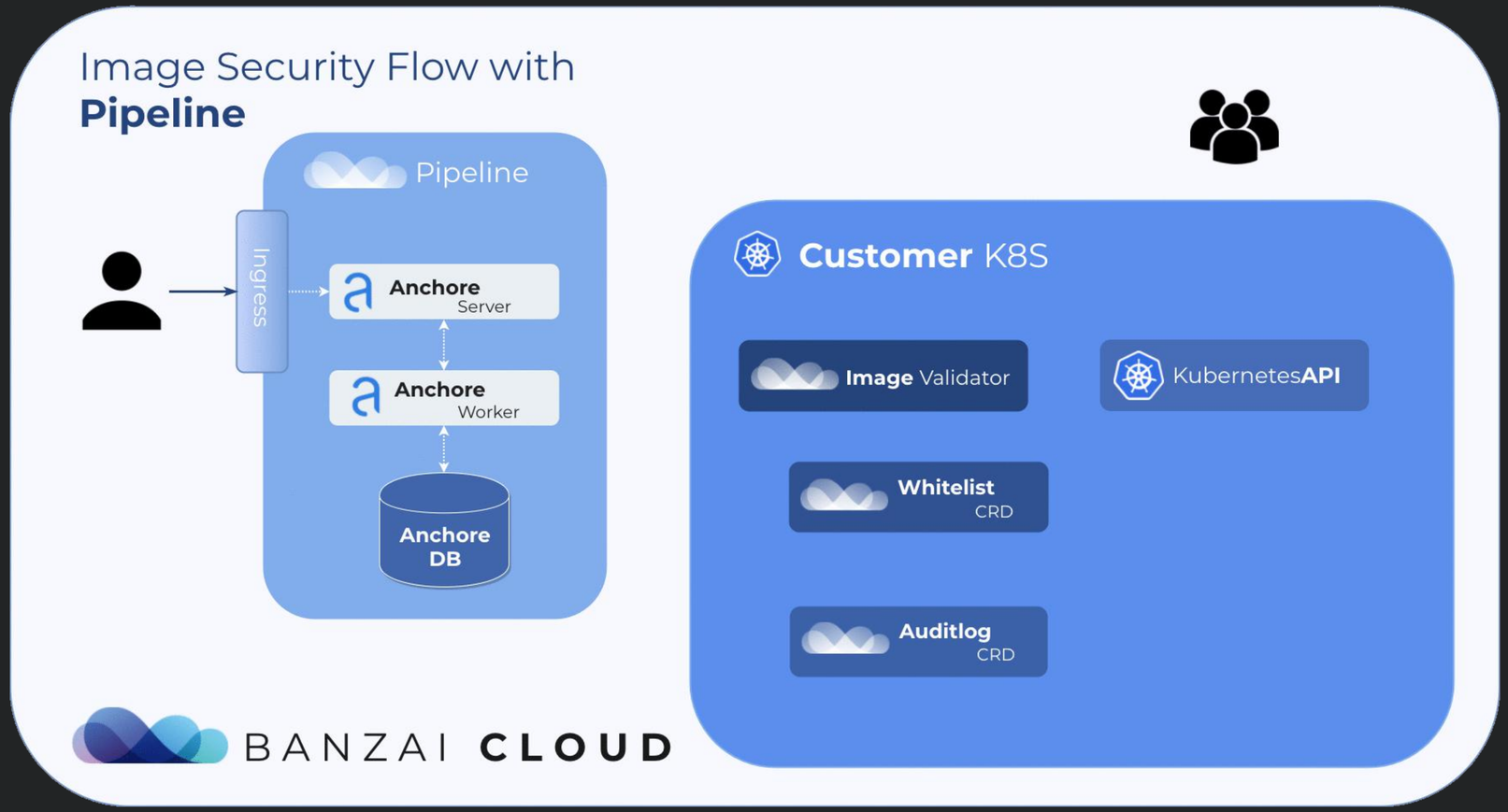
扫描能力

```
$ anchore-cli policy describe
+-----------------+-----------------------------------------------------------------+
| Gate            | Description                                                     |
+-----------------+-----------------------------------------------------------------+
| always          | Triggers that fire unconditionally if present in policy,        |
|                 | useful for things like testing and blacklisting.                |
+-----------------+-----------------------------------------------------------------+
| dockerfile      | Checks against the content of a dockerfile if provided, or      |
|                 | a guessed dockerfile based on docker layer history if the       |
|                 | dockerfile is not provided.                                     |
+-----------------+-----------------------------------------------------------------+
| files           | Checks against files in the analyzed image including file       |
|                 | content, file names, and filesystem attributes.                 |
+-----------------+-----------------------------------------------------------------+
| licenses        | License checks against found software licenses in the           |
|                 | container image                                                 |
+-----------------+-----------------------------------------------------------------+
| metadata        | Checks against image metadata, such as size, OS, distro,        |
|                 | architecture, etc.                                              |
+-----------------+-----------------------------------------------------------------+
| npms            | NPM Checks                                                      |
+-----------------+-----------------------------------------------------------------+
| packages        | Distro package checks                                           |
+-----------------+-----------------------------------------------------------------+
| passwd_file     | Content checks for /etc/passwd for things like usernames,       |
|                 | group ids, shells, or full entries.                             |
+-----------------+-----------------------------------------------------------------+
| ruby_gems       | Ruby Gem Checks                                                 |
+-----------------+-----------------------------------------------------------------+
| secret_scans    | Checks for secrets found in the image using configured          |
|                 | regexes found in the "secret_search" section of                 |
|                 | analyzer_config.yaml.                                           |
+-----------------+-----------------------------------------------------------------+
| vulnerabilities | CVE/Vulnerability checks.                                       |
+-----------------+-----------------------------------------------------------------+
```

Anchor Image Validator

QINGTENG

利用Kubernetes准入控制（Admission Controll）审核Kubernetes api的镜像操作。



QINGTENG

感谢观看

青藤云安全

北京市海淀区创业路8号群英科技园东门1号楼5层