## Agent

自制镜像

<http://git.yihecloud.com:88/PaaS/OpenBridge-Dockerfile/tree/master/docker/agent/1.9/>

测试环境镜像仓库

http://192.168.1.72:5000/agent:1.9

### @编译agent-DomeOS-new

#递归删除指定目录下的.git文件

find . -name .git | xargs rm –fr

#安装godep

go get github.com/tools/godep

#加入环境变量

export PATH=$GOPATH/bin:$PATH

#编译

godep go build .

### @domeos\agent\cron\reporter.go上报cpu,mem

E:\workspace\yh\OpenBridge-passos-proxy\open-faclon\src\github.com\

domeos\agent\cron\reporter.go

读取文件信息

/rootfs/proc/cpuinfo

/rootfs/proc/meminfo

### @domeos\agent\cron\reporter.go上报DockerId和NetworkRefId

containers := g.CurrentContainers()

//machineInfo, err := g.ContainerManager().GetMachineInfo()

if err != nil {

return

}

for \_, container := range containers {

network\_ref\_id, err1 := exec.Command("bash", "-c", "-f", "docker -H unix:///rootfs/var/run/docker.sock inspect -f '{{.HostConfig.NetworkMode}}' "+container+"|awk -F: '{print $2}'").Output()

if err1 != nil {

fmt.Println(err1.Error())

}

env, err2 := exec.Command("bash", "-c", "docker -H unix:///rootfs/var/run/docker.sock inspect -f '{{.Config.Env}}' "+container).Output()

if err2 != nil {

fmt.Println(err2.Error())

}

reg1 := regexp.MustCompile(`SERVICE\_ID=\w+`)

deploy\_id\_str := reg1.FindAllString(string(env), -1)

var deploy\_id string

if deploy\_id\_str != nil {

deploy\_id = strings.Split(deploy\_id\_str[0], "=")[1]

}

reg2 := regexp.MustCompile(`PODNAME=[\w|-]+`)

podname\_str := reg2.FindAllString(string(env), -1)

var podname string

if podname\_str != nil {

podname = strings.Split(podname\_str[0], "=")[1]

}

dreq := model.DockerReportRequest{

DockerId: container,

DeployId: deploy\_id,

NetworkRefId: string(network\_ref\_id),

Hostname: hostname,

Podname: podname,

}

var dresp model.SimpleRpcResponse

derr := g.HbsClient.Call("Agent.ReportDocker", dreq, &dresp)

if derr != nil || dresp.Code != 0 {

log.Println("call Agent.ReportDocker fail:", err, "Request:", dreq, "Response:", dresp)

}

log.Printf("agent version: 2.2, DockerId: %v , DeployId: %v , NetworkRefId: %v , Hostname: %v , Podname: %v",

container, deploy\_id\_str, string(network\_ref\_id), hostname, podname\_str)

}

### 查看容器cpu信息

E:\workspace\go\cadvisor\info\v1\test\datagen.go

E:\workspace\yh\OpenBridge-passos-proxy\open-faclon\src\agent-DomeOS-new\funcs\container.go

machineInfo, err := g.ContainerManager().GetMachineInfo()

### @docker login

如果无法更新，执行如下命令

$docker login -u admin -p admin123 docker.dev.yihecloud.com

### docker 执行脚本

vi /etc/resolv.conf

# Generated by NetworkManager

nameserver 192.168.1.4

#!/bin/bash

docker\_hub="docker.yihecloud.com" # docker仓库地址

monitor\_center\_ip="192.168.1.55" # 监控中心主机的IP

# 获取本机IP地址，支持动态和静态IP

IP=

host\_ips=(`ip addr show | grep inet | grep -v inet6 | grep brd | awk '{print $2}' | cut -f1 -d '/'`)

if [ "${host\_ips[0]}" == "" ]; then

echo "[ERROR] get ip address error"

exit 1

else

IP=${host\_ips[0]}

echo "[INFO] use host ip address: $IP"

fi

# 清理

docker rm -f agent

docker rmi -f $docker\_hub/agent:1.9

# 安装

docker run -d --restart=always -e HOSTNAME="\"$IP\"" \

-e TRANSFER\_ADDR="[\"$monitor\_center\_ip:8433\",\"$monitor\_center\_ip:8433\"]" \

-e TRANSFER\_INTERVAL="60" \

-e HEARTBEAT\_ENABLED="true" \

-e HEARTBEAT\_ADDR="\"$monitor\_center\_ip:6030\"" \

-v /:/rootfs:ro \

-v /var/run:/var/run:rw \

-v /sys:/sys:ro \

-v /var/lib/docker/:/var/lib/docker:ro \

-p 1988:1988 \

--name agent \

"$docker\_hub/agent:1.9"

# 查看结果

sleep 3;

docker logs -f agent

### agent安装192.168.1.55脚本

#!/bin/bash

docker\_hub="docker.yihecloud.com" # docker仓库地址

monitor\_center\_ip="192.168.1.55" # 监控中心主机的IP

# 获取本机IP地址，支持动态和静态IP

IP=

host\_ips=(`ip addr show | grep inet | grep -v inet6 | grep brd | awk '{print $2}' | cut -f1 -d '/'`)

if [ "${host\_ips[0]}" == "" ]; then

echo "[ERROR] get ip address error"

exit 1

else

IP=${host\_ips[0]}

echo "[INFO] use host ip address: $IP"

fi

# 清理

docker rm -f agent

docker rmi -f $docker\_hub/agent:1.9

# 安装

docker run -d --restart=always -e HOSTNAME="\"$IP\"" \

-e TRANSFER\_ADDR="[\"$monitor\_center\_ip:8433\",\"$monitor\_center\_ip:8433\"]" \

-e TRANSFER\_INTERVAL="60" \

-e HEARTBEAT\_ENABLED="true" \

-e HEARTBEAT\_ADDR="\"$monitor\_center\_ip:6030\"" \

-v /:/rootfs:ro \

-v /var/run:/var/run:rw \

-v /sys:/sys:ro \

-v /var/lib/docker/:/var/lib/docker:ro \

-p 1988:1988 \

--name agent \

"$docker\_hub/agent:1.9"

# 查看结果

sleep 3;

docker logs -f agent

### 安装脚本install-agent.sh 2.0

#!/bin/bash

# 安装openbridge-agent

# wangxinxiang@yihecloud.com

# 2016-08-08

OPTS=$(getopt -o s: --long registry: -- "$@")

if [ $? != 0 ]; then

echo "[ERROR] 参数错误！"

usage;

exit 1

fi

eval set -- "$OPTS"

registry="docker.dev.yihecloud.com:443"

version="2.0"

monitor\_ip=192.168.0.179

while true; do

case "$1" in

-s) monitor\_ip=$2; shift 2;;

--registry) registry=$2; shift 2;;

--) shift; break;;

esac

done

function check\_opt() {

arg="\$$1"

if [ "$(eval echo $arg)" = "" ]; then

echo "[ERROR] <$1> 参数缺失！"

usage;

exit 1

fi

}

function usage() {

echo "

Usage: $0

-s <monitor server ip> , eg: x.x.x.x

--registry <docker registry>, default: docker.yihecloud.com:443

"

}

# check options

check\_opt "registry"

check\_opt "monitor\_ip"

# 获取本机IP地址，支持动态和静态IP

IP=

host\_ips=(`ip addr show | grep inet | grep -v inet6 | grep brd | awk '{print $2}' | cut -f1 -d '/'`)

if [ "${host\_ips[0]}" == "" ]; then

echo "[ERROR] get ip address error"

exit 1

else

IP=${host\_ips[0]}

echo "[INFO] use host ip address: $IP"

fi

# 清理

docker rm -f agent

docker rmi -f $registry/agent:1.9

# run docker image

docker run -d --restart=always \

-e HOSTNAME="\"$IP\"" \

-e TRANSFER\_ADDR="[\"$monitor\_ip:8433\",\"$monitor\_ip:8433\"]" \

-e TRANSFER\_INTERVAL="60" \

-e HEARTBEAT\_ENABLED="true" \

-e HEARTBEAT\_ADDR="\"$monitor\_ip:6030\"" \

-v /:/rootfs:ro \

-v /var/run:/var/run:rw \

-v /sys:/sys:ro \

-v /var/lib/docker/:/var/lib/docker:ro \

-p 1988:1988 \

--name agent \

$registry/openbridge/agent:$version

# show status

docker ps |grep "agent"

sleep 3;

docker logs -f agent

### 部署脚本

yum install docker-io --enablerepo=epel

service docker start

更新docker连接

wget http://192.168.1.60/v2/install/install/centos7/docker.sh  
sh docker.sh --insecure\_registry=192.168.1.72:5000

~~export host\_ip=$(hostname --ip-address)~~

export host\_ip=192.168.1.71

export transfer\_ip=192.168.1.135

docker run -d --name agent \

--restart=always \

-e HOSTNAME="\"$host\_ip\"" \

-e TRANSFER\_ADDR="[\"$transfer\_ip:8433\",\"$transfer\_ip:8433\"]" \

-e TRANSFER\_INTERVAL="120" \

-v /:/rootfs:ro \

-v /var/run:/var/run:rw \

-v /sys:/sys:ro \

-v /var/lib/docker/:/var/lib/docker:ro \

-p 1988:1988 \

192.168.1.72:5000/agent:1.9

export host\_ip=192.168.1.71

export transfer\_ip=192.168.1.135

docker run -d --name agent \

--restart=always \

-e HOSTNAME="\"$host\_ip\"" \

-e TRANSFER\_ADDR="[\"$transfer\_ip:8433\",\"$transfer\_ip:8433\"]" \

-e TRANSFER\_INTERVAL="120" \

-v /:/rootfs:ro \

-v /var/run:/var/run:rw \

-v /sys:/sys:ro \

-v /var/lib/docker/:/var/lib/docker:ro \

-p 1988:1988 \

install/agent:1.9

### Cfg.json

{

"debug": true,

"hostname": "192.168.1.77",

"ip": "",

"plugin": {

"enabled": false,

"dir": "./plugin",

"git": "https://github.com/open-falcon/plugin.git",

"logs": "./logs"

},

"heartbeat": {

"enabled": false,

"addr": "127.0.0.1:6030",

"interval": 60,

"timeout": 1000

},

"transfer": {

"enabled": true,

"addrs": ["192.168.1.135:8433","192.168.1.135:8433"],

"interval": 30,

"timeout": 1000

},

"http": {

"enabled": true,

"listen": ":1988",

"backdoor": false

},

"collector": {

"ifacePrefix": ["eth", "em", "en"]

},

"ignore": {

"cpu.idle": true,

"cpu.steal": true,

"cpu.guest": true,

"net.if.in.packets": true,

"net.if.in.errors": true,

"net.if.in.dropped": true,

"net.if.in.fifo.errs": true,

"net.if.in.frame.errs": true,

"net.if.in.compressed": true,

"net.if.in.multicast": true,

"net.if.out.packets": true,

"net.if.out.errors": true,

"net.if.out.dropped": true,

"net.if.out.fifo.errs": true,

"net.if.out.collisions": true,

"net.if.out.carrier.errs": true,

"net.if.out.compressed": true,

"net.if.total.bytes": true,

"net.if.total.packets": true,

"net.if.total.errors": true,

"net.if.total.dropped": true,

"kernel.maxfiles": true,

"kernel.maxproc": true,

"kernel.files.allocated": true,

"kernel.files.left": true,

"load.1min": true,

"load.5min": true,

"load.15min": true,

"mem.memfree": true,

"mem.swaptotal": true,

"mem.swapused": true,

"mem.swapfree": true,

"mem.memfree.percent": true,

"mem.swapfree.percent": true,

"mem.swapused.percent": true,

"disk.io.read\_requests": true,

"disk.io.read\_merged": true,

"disk.io.read\_sectors": true,

"disk.io.msec\_read": true,

"disk.io.write\_requests": true,

"disk.io.write\_merged": true,

"disk.io.write\_sectors": true,

"disk.io.msec\_write": true,

"disk.io.ios\_in\_progress": true,

"disk.io.msec\_total": true,

"disk.io.msec\_weighted\_total": true,

"disk.io.avgrq\_sz": true,

"disk.io.avgqu-sz": true,

"disk.io.await": true,

"disk.io.svctm": true,

"disk.io.util": true,

"snmp.Udp.InCsumErrors": true,

"snmp.Udp.InDatagrams": true,

"snmp.Udp.InErrors": true,

"snmp.Udp.NoPorts": true,

"snmp.Udp.OutDatagrams": true,

"snmp.Udp.RcvbufErrors": true,

"snmp.Udp.SndbufErrors": true,

"df.bytes.free": true,

"df.bytes.free.percent": true,

"df.inodes.total": true,

"df.inodes.used": true,

"df.inodes.free": true,

"df.inodes.used.percent": true,

"df.inodes.free.percent": true,

"df.statistics.total": true,

"df.statistics.used": true,

"df.statistics.used.percent": true,

"TcpExt.ArpFilter": true,

"TcpExt.DelayedACKLocked": true,

"TcpExt.ListenDrops": true,

"TcpExt.ListenOverflows": true,

"TcpExt.LockDroppedIcmps": true,

"TcpExt.PruneCalled": true,

"TcpExt.TCPAbortFailed" : true,

"TcpExt.TCPAbortOnMemory": true,

"TcpExt.TCPAbortOnTimeout": true,

"TcpExt.TCPBacklogDrop": true,

"TcpExt.TCPDSACKUndo": true,

"TcpExt.TCPFastRetrans": true,

"TcpExt.TCPLossFailures": true,

"TcpExt.TCPLostRetransmit": true,

"TcpExt.TCPMemoryPressures": true,

"TcpExt.TCPMinTTLDrop": true,

"TcpExt.TCPPrequeueDropped": true,

"TcpExt.TCPSchedulerFailed": true,

"TcpExt.TCPSpuriousRTOs": true,

"TcpExt.TCPTSReorder": true,

"TcpExt.TCPTimeouts": true,

"TcpExt.TW": true,

"ss.closed": true,

"ss.estab": true,

"ss.orphaned": true,

"ss.slabinfo.timewait": true,

"ss.synrecv": true,

"ss.timewait": true,

"container.mem.working\_set": true,

"container.disk.io.read\_bytes": true,

"container.disk.io.write\_bytes": true,

"container.net.if.in.packets": true,

"container.net.if.in.errors": true,

"container.net.if.in.dropped": true,

"container.net.if.out.packets": true,

"container.net.if.out.errors": true,

"container.net.if.out.dropped": true

}

}

### 2@http/container.go获取容器基本信息

E:\workspace\yh\OpenBridge-passos-proxy\open-faclon\src\agent-DomeOS-new\http\container.go

http://192.168.0.179:1988/containers

### main.go设置startContainerMonitor cAdvisor

E:\workspace\yh\OpenBridge-passos-proxy\open-faclon\src\agent-DomeOS-new\main.go

### Cadvisor获取NetworkMode信息

E:\workspace\yh\OpenBridge-passos-proxy\open-faclon\src\agent-DomeOS-new\Godeps\\_workspace\src\github.com\docker\engine-api\types\types.go

NetworkMode

E:\workspace\yh\OpenBridge-passos-proxy\open-faclon\src\agent-DomeOS-new\Godeps\\_workspace\src\github.com\google\

cadvisor\container\docker\handler.go

// The network mode of the container

    networkMode dockercontainer.NetworkMode

### g\var.go启动容器监控startContainerMonitor

containerManager, err := manager.New(memoryStorage, sysFs, \*maxHousekeepingInterval, \*allowDynamicHousekeeping, ignoreMetrics.MetricSet)

**if** err != nil {

log.Fatalf("Failed to create a Container Manager: %s", err)

}

### g\var.go获取当前的容器

E:\workspace\yh\OpenBridge-passos-proxy\open-faclon\src\agent-DomeOS-new\g\var.go

**func** UpdateCurrentContainers() {

    reqParams := &info.ContainerInfoRequest{

        NumStats: 1,

    }

dockerContainers, err := ContainerManager().AllDockerContainers(reqParams)

**if** err != nil {

log.Println("Get docker containers error : %s", err.Error())

**return**;

}

containers := make([]string, 0)

**for** \_, container := **range** dockerContainers {

containers = append(containers, container.Id)

}

SetCurrentContainers(containers)

}

### agent push的数据

agent push的数据可以参考：<https://github.com/open-falcon/agent/tree/master/funcs>

reporter.go

req := model.AgentReportRequest{

            Hostname: hostname,

            IP: g.IP(),

            AgentVersion: g.VERSION,

            PluginVersion: g.GetCurrPluginVersion(),

        }

**var** resp model.SimpleRpcResponse

        err = g.HbsClient.Call("Agent.ReportStatus", req, &resp)