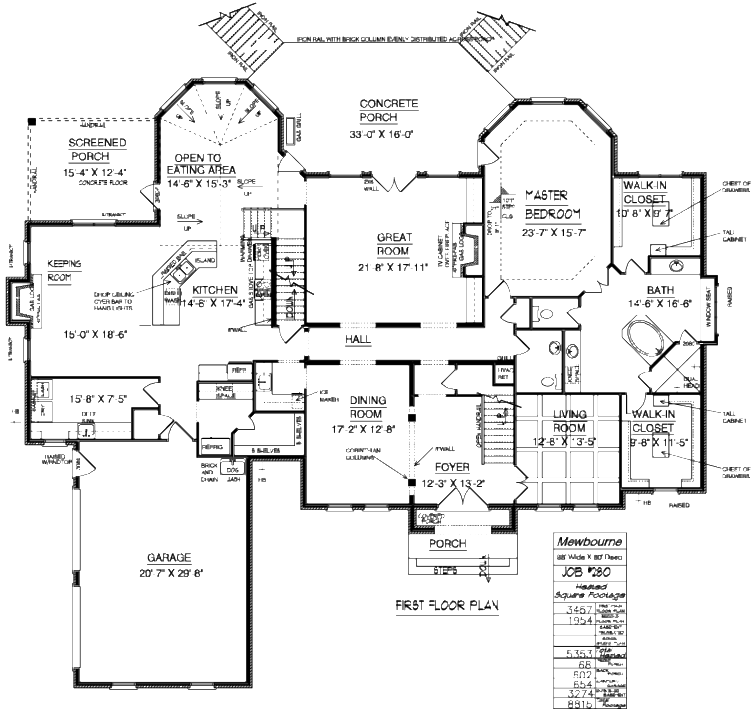

docker
for
DevOps
Engineers
2019
by
Siarhei Beliakou
part 3
LINUX

AGENDA

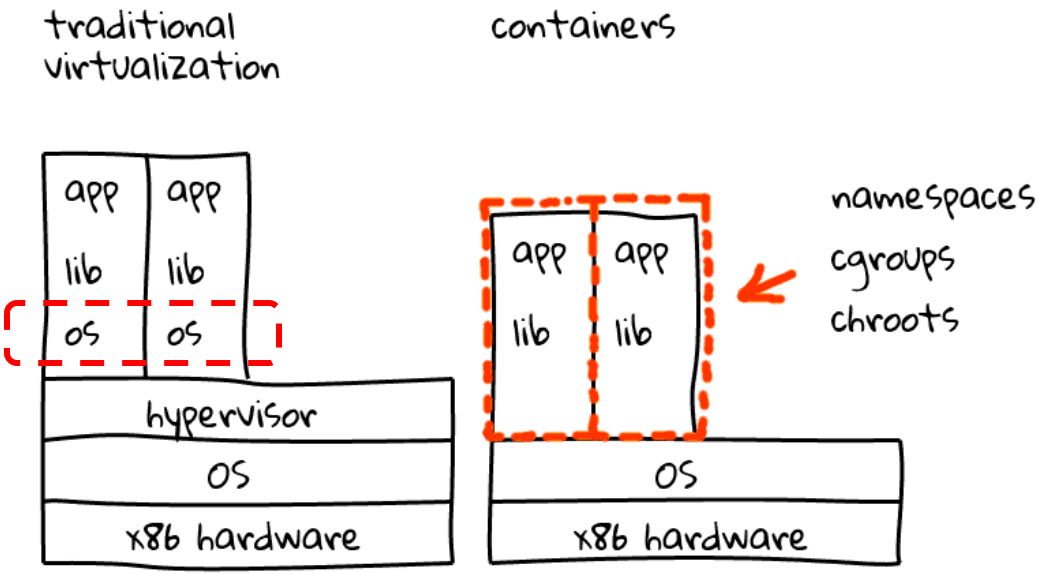
- Linux Kernel Namespaces and Control Groups
- Linux Process Capabilities
- Docker-In-Docker
- Remote API
- Docker Plugins
- Monitoring Containers
- Containers Security



Linux Kernel Namespaces and CGroups

- **Namespaces**
  - pid
  - net
  - uts
- **Cgroups**
  - Memory Constraints
  - CPU Constraints

What is a Container?



Each Container has:

- Own process space
- Own network interface
- Can run as root
- Can install packages
- Can run services
- etc

Why Containers run unlike VMs:

- It uses host's Kernel
- It doesn't need INIT system
- It doesn't need syslog, cron and so on

Cgroups and Namespaces

Cgroups = limits how much you can use

- Resource metering and limiting:
  - memory
  - CPU
  - block I/O
  - network
- Device node (/dev/*) access control

Namespaces = limits what you can see (and therefore use)

- Provide processes with their own view of the system
- Multiple namespaces:
  - pid – isolates the process ID number space
  - net – manages network devices
  - mnt – to see distinct single- directory hierarchies
  - uts – isolating hostnames
  - ipc – manages shared memory areas, message queues, and semaphores
- Each process is in one namespace of each type

Namespace PID

Run Container in Host PID Namespace:

```
# sleep 1000 &
[1] 19005
$ ps -ef | grep sleep | grep -v grep
root     19005     1  0 21:57 pts/0    00:00:00 sleep 1000
$ docker run --pid=host centos kill -9 19005
$ ps -ef | grep sleep
```

Run Container in another container PID Namespace:

```
# docker run -d --name=web-container nginx

# docker run --rm --pid=container:web-conatiner centos ps
  PID TTY          TIME CMD
    1 ?        00:00:00 nginx
   12 ?        00:00:00 ps
```

Namespace NET

```
# docker run -d --name nginx nginx
# docker run -d -it --name net-tools --net=container:nginx sbeliakou/net-tools
```

```
# docker exec nginx hostname -i
172.17.0.2
# docker exec net-tools hostname -i
172.17.0.2
```

```
# docker exec net-tools ps -ef
PID   USER     TIME   COMMAND
    1 root       0:00 /bin/bash
   11 root       0:00 ps -ef
```

```
# docker exec net-tools netstat -natpl
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address           Foreign Address         State       PID/Program name
tcp        0      0 0.0.0.0:80              0.0.0.0:*               LISTEN      -
```

```
# docker exec net-tools curl -sIL localhost | egrep "(HTTP|Server)"
HTTP/1.1 200 OK
Server: nginx/1.15.2
```

```
# curl -sIL 172.17.0.2 | egrep "(HTTP|Server)"
HTTP/1.1 200 OK
Server: nginx/1.15.2
```

Namespace UTS (System Name and Domain)

By default, a container runs with a UTS namespace (which defines the system name and domain) that is different from the UTS namespace of the host. To make a container use the same UTS namespace as the host, you can use the *--uts=host* option with the docker create and docker run commands from version 1.7.0 of Docker onward. This setting allows the container to track the UTS namespace of the host or to set the host name and domain from the container.

```
[root@docker-host ~]# hostname
docker-host
```

```
[root@docker-host ~]# docker run --rm busybox hostname
0feabf511a54
```

```
[root@docker-host ~]# docker run --rm --uts=host busybox hostname
docker-host
```

Cgroups: Memory Limits

```
# docker run -d centos sleep infinity
60614832f3ba25962c584d209a0ba1f99335c3c980563c72715ac14175fada5f
```

```
# docker stats --no-stream --format "table {{.Container}}\t{{.MemUsage}}" 60614832f3ba
CONTAINER           MEM USAGE / LIMIT
60614832f3ba        92KiB / 991.6MiB
```

```
# docker run -d -m 300M centos sleep infinity
b133cd44aa414645f94474ffdc49b7d31b7cccfcb14a122f8f0700bb0b7c80bd
```

```
# docker stats --no-stream --format "table {{.Container}}\t{{.MemUsage}}" b133cd44aa41
CONTAINER           MEM USAGE / LIMIT
b133cd44aa41        1016KiB / 300MiB
```

```
# docker run -d -m 300M --memory-reservation 100M centos sleep infinity
af6df26f67538433932f2ab9957afa2eaec586dedae5a99c837df2a5be6e2c18
```

```
# docker stats --no-stream --format "table {{.Container}}\t{{.MemUsage}}"
CONTAINER           MEM USAGE / LIMIT
af6df26f6753        968KiB / 300MiB
b133cd44aa41        1016KiB / 300MiB
60614832f3ba        92KiB / 991.6MiB
```

Check out documentation:

container memory constraints

docker stats reference

Cgroups: CPU Limits

CPU limits are based on shares. These shares are a weight between how much processing time one process should get compared to another. If a CPU is idle, then the process will use all the available resources. If a second process requires the CPU then the available CPU time will be shared based on the weighting.

`--cpu-shares`	-	CPU shares (relative weight).
`--cpu-period`	-	Limit CPU CFS (Completely Fair Scheduler) period.
`--cpu-quota`	-	Limit CPU CFS (Completely Fair Scheduler) quota.
`--cpuset-cpus`	-	CPUs in which to allow execution (0-3, 0,1).
`--cpuset-mems`	-	MEMs in which to allow execution (0-3, 0,1).

*The **--cpu-quota** option specifies the number of microseconds that a container has access to CPU resources during a period specified by **--cpu-period**. As the default value of **--cpu-period** is 100000, setting the value of **--cpu-quota** to 25000 limits a container to 25% of the CPU resources. By default, a container can use all available CPU resources, which corresponds to a **--cpu-quota** value of -1*

```
# docker run -it -d --cpu-quota=25000 --name cpu0.25 benhall/stress
```

```
# docker stats --no-stream --format "table {{.Name}}\t{{.CPUPerc}}" cpu0.25
NAME                CPU %
cpu0.25             25.03%
```

Linux Process Capabilities

- Capabilities Overview
- Enabling Capabilities
- SYS_TIME Example
- NET_ADMIN Example
- SYS_ADMIN Example
- Best Practices (Sources)

Linux Process Capabilities

Unix was designed to run two categories of processes :

- privileged processes, with the user id 0 which is the root user.
- unprivileged processes where the user id is different from zero.

While privileged processes bypass all kernel permission checks, all the other processes are subject to a permission check (the Unix-like system Kernel make the permission check based on the UID and the GID of the non root process.

Starting with Kernel 2.2, Linux divides the privileges into distinct units that we call capabilities. Capabilities can be independently enabled and disabled, which is a great security and permission management feature. A process can have CAP_NET_RAW capability while it is denied to use the CAP_NET_ADMIN capability. This is just an example explaining how capabilities works. You can find the complete list of the other capabilities in Linux man-page:

```
$ man capabilities
```

http://man7.org/linux/man-pages/man7/capabilities.7.html

Managing Capabilities on a Container

```
# docker run --help
  --cap-add: Add Linux capabilities
  --cap-drop: Drop Linux capabilities
  --privileged=false: Give extended privileges to this container
  --device=[]: Allows you to run devices inside the container without the privileged flag.
```

```
# docker run --privileged=true ...
```

List of Capabilities:
https://docs.docker.com/engine/reference/run/#runtime-privilege-and-linux-capabilities

Docker Security Article:
https://linux-audit.com/docker-security-best-practices-for-your-vessel-and-containers/

Capability: SYS_TIME

```
[root@localhost ~]# timedatectl | grep NTP
     NTP enabled: yes
NTP synchronized: yes
```

```
[root@localhost ~]# date
Tue  7 Aug 21:54:57 UTC 201
```

```
[vagrant@localhost ~]# docker run --rm centos date
Tue Aug  7 21:55:06 UTC 2018
```

```
[vagrant@localhost ~]# docker run --rm --cap-add=SYS_TIME centos date -s '+2 hours'
Tue Aug  7 23:55:15 UTC 2018
```

```
[vagrant@localhost ~]# date
Tue  7 Aug 23:55:17 UTC 2018
```

```
[root@localhost ~]# timedatectl | grep NTP
     NTP enabled: yes
NTP synchronized: no
```

Capability: NET_ADMIN

Perform various network-related operations:

- interface configuration;
- administration of IP firewall, masquerading, and accounting
- modify routing tables;
- bind to any address for transparent proxying;
- set type-of-service (TOS)
- clear driver statistics;
- set promiscuous mode;
- enabling multicasting;

```
$ docker run --cap-add=NET_ADMIN ubuntu:14.04 ip link add dummy0 type dummy
```

```
$ ip link | grep -A1 dummy
31: dummy0: <BROADCAST,NOARP> mtu 1500 qdisc noop state DOWN mode DEFAULT group default qlen 1000
    link/ether 7e:32:dd:ec:f1:22 brd ff:ff:ff:ff:ff:ff
```

```
docker run --cap-add=NET_ADMIN ubuntu:14.04 sh -c 'ip link eth0 down'
```

Capability: SYS_ADMIN

```
$ hostname
docker-host
```

```
$ docker run --rm busybox hostname
0feabf511a54
```

```
$ docker run --rm --uts=host busybox hostname
docker-host
```

```
$ docker run --rm --uts=host busybox hostname HOST
hostname: sethostname: Operation not permitted
```

```
$ docker run --rm --uts=host busybox hostname HOST
hostname: sethostname: Operation not permitted
```

```
$ docker run --rm --uts=host --cap-add=SYS_ADMIN busybox hostname HOST
$ hostname
HOST
```

Lean VM

```
$ docker run -d centos /usr/sbin/init
$ docker exec -it $(docker ps -ql) bash
[root@b81e02718292 /]# systemctl status
Failed to get D-Bus connection: Operation not permitted
```

```
$ docker run -d -v /sys/fs/cgroup centos /usr/sbin/init
$ docker exec -it $(docker ps -ql) bash
[root@2a3feeafc6fb /]# systemctl status
Failed to get D-Bus connection: Operation not permitted
```

```
$ docker run -d -v /sys/fs/cgroup --cap-add=SYS_ADMIN centos /usr/sbin/init
$ docker exec -it $(docker ps -ql) bash
[root@7033599262f5 /]# systemctl status
● 7033599262f5
    State: running
     Jobs: 0 queued
   Failed: 0 units
    Since: Mon 2018-10-15 18:39:54 UTC; 6s ago
   CGroup: /docker/7033599262f54477c13ccd4d2ebf3bb8af52eec5d41295c973b2a022b1de0ff9
           ├─ 1 /usr/sbin/init
           ├─31 bash
           ├─44 systemctl status
           ├─45 systemctl status
           └─system.slice
             └─  ...
```

Lean VM

CentOS.SSHD.Dockerfile

```
FROM centos

RUN yum install -y systemd openssh-server sudo && systemctl enable sshd
RUN useradd devops && echo devops | passwd devops --stdin

EXPOSE 22
ENTRYPOINT ["/usr/sbin/init"]
```

docker-compose.yml

```
version: "2.3"

services:
  centosvm:
    build:
      context: .
      dockerfile: CentOS.SSHD.Dockerfile
    volumes:
    - /sys/fs/cgroup
    cap_add:
    - SYS_ADMIN
```

Lean VM

```
$ docker-compose up -d
Creating network "leanvm_default" with the default driver
Creating leanvm_centosvm_1 ... done
```

```
$ docker-compose ps
      Name             Command       State   Ports
-------------------------------------------------
leanvm_centosvm_1   /usr/sbin/init   Up      22/tcp
```

```
$ docker-compose up -d --scale centosvm=5
Starting leanvm_centosvm_1 ... done
Creating leanvm_centosvm_2 ... done
Creating leanvm_centosvm_3 ... done
Creating leanvm_centosvm_4 ... done
Creating leanvm_centosvm_5 ... done
```

```
$ docker-compose ps
      Name             Command       State   Ports
-------------------------------------------------
leanvm_centosvm_1   /usr/sbin/init   Up      22/tcp
leanvm_centosvm_2   /usr/sbin/init   Up      22/tcp
leanvm_centosvm_3   /usr/sbin/init   Up      22/tcp
leanvm_centosvm_4   /usr/sbin/init   Up      22/tcp
leanvm_centosvm_5   /usr/sbin/init   Up      22/tcp
```

Lean VM

```
# docker inspect \
    -f '{{range .NetworkSettings.Networks}}{{.IPAddress}}{{end}}' \
    $(docker-compose ps -q)
172.24.0.2
172.24.0.5
172.24.0.6
172.24.0.4
172.24.0.3
```

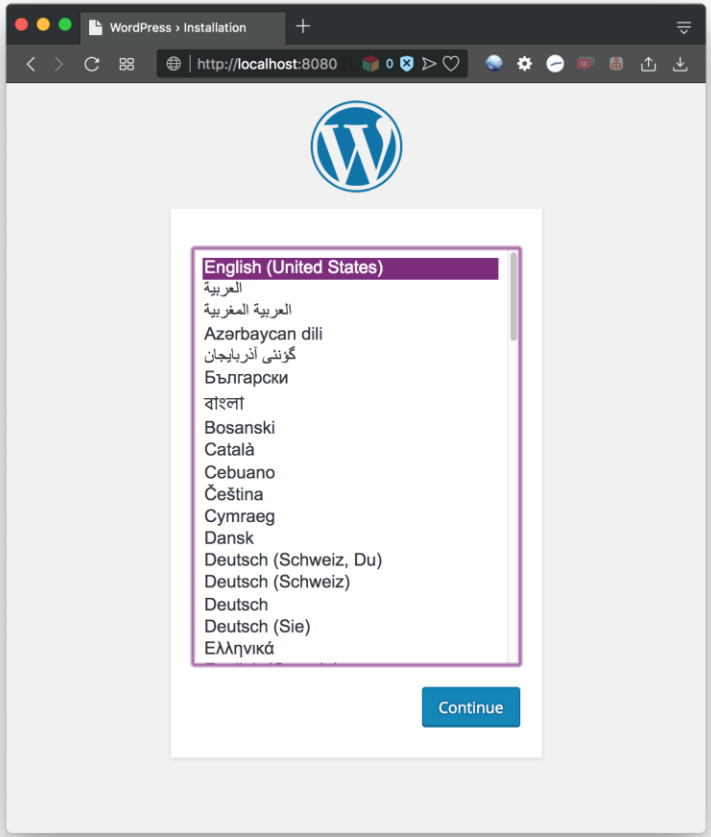
```
# ssh devops@172.25.0.4
The authenticity of host '172.25.0.4 (172.25.0.4)' can't be established.
ECDSA key fingerprint is SHA256:WRIJRv4kwjpx34t3HnHP2RdYsXVhd4j0EAALkw0IddQ.
ECDSA key fingerprint is MD5:42:87:a3:65:1b:b8:64:37:7a:9f:6f:b9:66:22:f3:f1.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '172.25.0.4' (ECDSA) to the list of known hosts.
devops@172.25.0.4's password:
[devops@83c5e3cd314e ~]$
```

WordPress Example

```
version: "3"

services:
  wordpress:
    image: endophage/wordpress:lean
    links:
        - mysql
    ports:
        - "8080:80"
    environment:
        - DB_PASSWORD=2671d40f658f595
    cap_drop:
        - ALL
    cap_add:
        - SETUID
        - SETGID
        - DAC_OVERRIDE
        - NET_BIND_SERVICE

  mysql:
    image: mariadb:10.1.10
    environment:
        - MYSQL_DATABASE=wordpress
        - MYSQL_ROOT_PASSWORD=2671d40f658f595
    ports:
        - "3306"
```

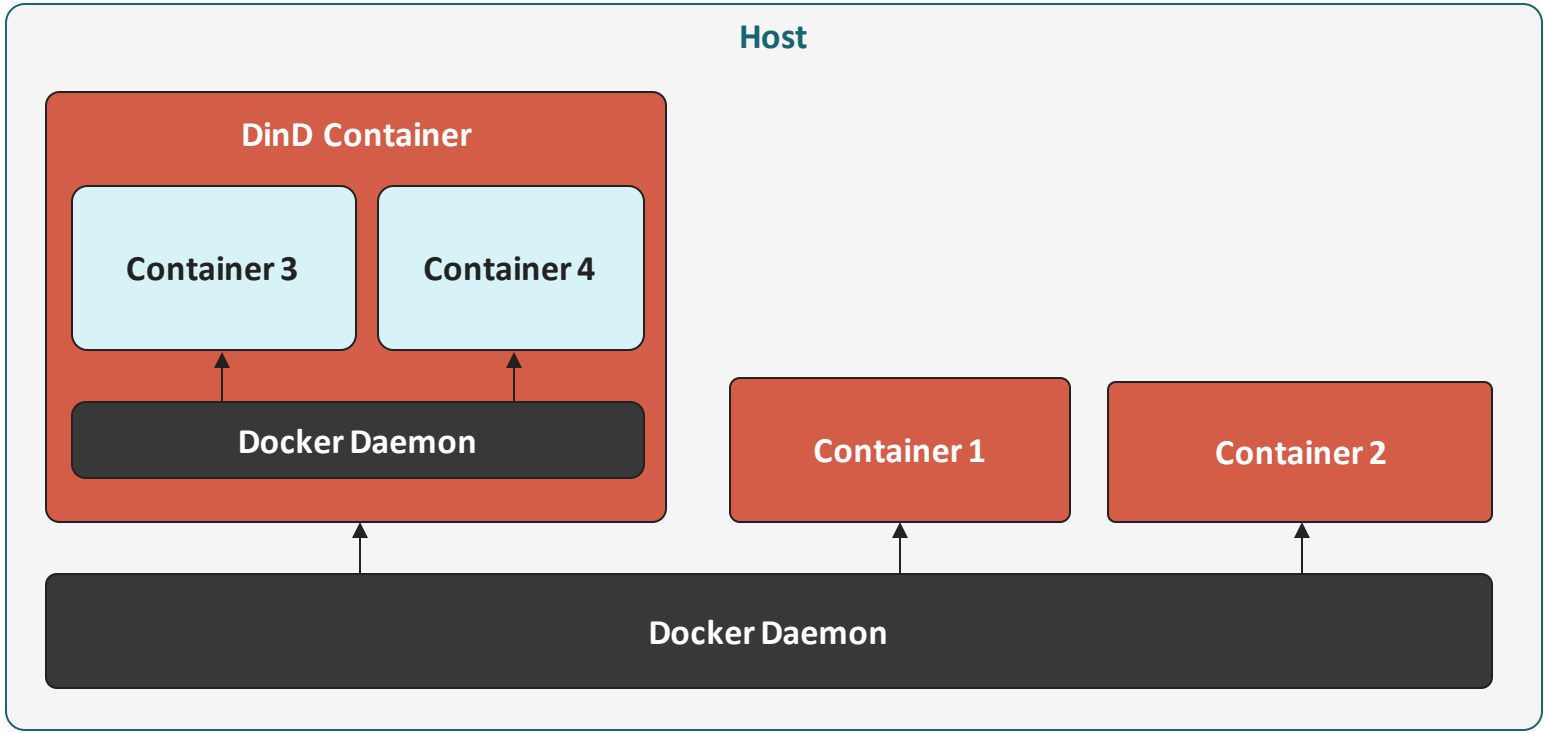


Docker-in-Docker

- How does this work?
- Do you really need this?

Docker in Docker (DinD)

How it looks:



Docker in Docker (DinD)

Run "dind" container:

```
# docker run --privileged -v /usr/local/bin -v /var/run --name dind -d docker:dind
```

Run "nginx" in "dind" container (from inside "dind"):

```
# docker exec dind docker run -d --name=nginx1 nginx
```

Run "nginx" in "dind" container (from outside "dind"):

```
# docker run --volumes-from dind -e PATH=/usr/local/bin busybox \
  docker run --name=nginx2 -d nginx
```

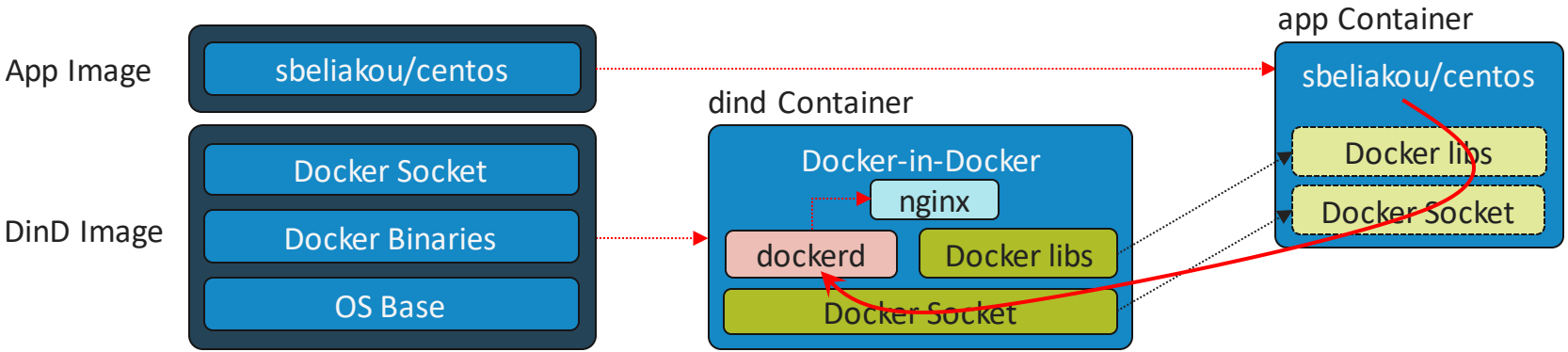
How goes?

```
# docker ps --format "table {{.Names}}\t{{.Status}}"
NAMES               STATUS
dind                Up 3 minutes
# docker exec -it dind docker ps --format "table {{.Names}}\t{{.Status}}"
NAMES               STATUS
nginx2              Up About a minute
nginx1              Up 3 minutes
```

Understanding DinD

```
$ docker run --name dind -d \
    --privileged \
    -v /usr/local/bin \
    -v /var/run \
    docker:dind
```

```
$ docker run --rm -ti --volumes-from=dind sbeliakou/centos bash
bash-4.2# docker run -d -P nginx
bash-4.2# docker ps
```



Data Saving Scenario

We need to create a volume for storing our dind's containers:

```
# docker volume create dind_data
```

Run our dind container with our volume:

```
# docker run --privileged \
  -v /usr/local/bin \
  -v /var/run \
  -v dind_data:/var/lib/docker \
  --name dind -d docker:dind
```

Let's start Nginx Container inside DinD:

```
# docker exec dind docker run -d --name=nginx --restart=always nginx
```

Stopping and Removing DinD:

```
# docker rm $(docker stop dind)
```

Start DinD again:

```
# docker run --privileged ... # command as above
```

Docker Remote API

- Configuring Remote Access
- Accessing Remote Docker Daemon
- Securing Remote Access

Configuring Docker Daemon for Remote Access

! systemd vs daemon.json

Configuring Docker to listen for connections using both the *systemd* unit file and the *daemon.json* file causes a conflict that prevents Docker from starting.

/etc/docker/daemon.json

```
{
  "hosts": [
    "unix:///var/run/docker.sock",
    "tcp://127.0.0.1:2375",
    "192.168.56.15"
  ]
}
```

More Configuration Options

/usr/lib/systemd/system/docker.service

```
...
[Service]
Type=notify
ExecStart=/usr/bin/dockerd -H unix://
ExecReload=/bin/kill -s HUP $MAINPID
TimeoutSec=0
RestartSec=2
Restart=always
...
```

```
$ systemctl daemon-reload
$ systemctl restart docker.service
```

```
# curl 192.168.56.15:2375/_ping
OK
```

Accessing Remote Docker Daemon

```
# export DOCKER_HOST=tcp://192.168.56.15:2375
# docker ps
...
```

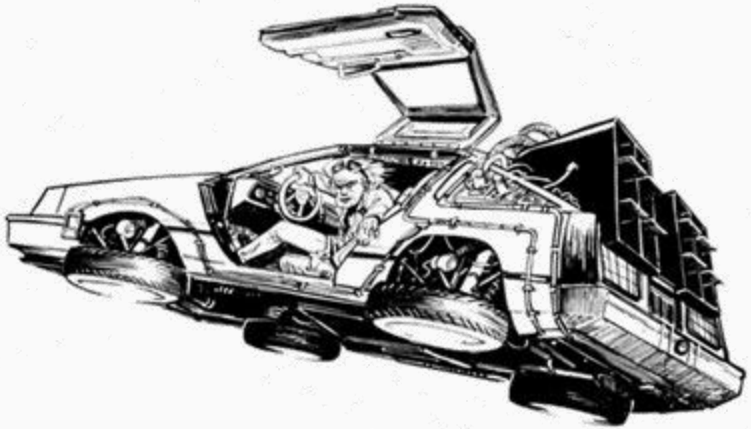
```
# docker -H tcp://192.168.56.15:2375 ps
...
```

`# curl 192.168.56.15:2375/info`	`# docker info`
`# curl 192.168.56.15:2375/images/json`	`# docker images`
`# curl 192.168.56.15:2375/containers/json`	`# docker ps`
`# curl 192.168.56.15:2375/containers/json?all=1`	`# docker ps -a`

Securing Remote Access

https://docs.docker.com/engine/security/https/
https://docs.docker.com/engine/extend/plugins_authorization/

1. Enable TLS (tlsverify, tlscacert)
2. Enable 2 way Authentication (with AuthZ plugins)



Docker Plugins

- Local-Persist
- SshFS

Docker Engine Plugins

You can extend the capabilities of the Docker Engine by loading third-party plugins.

Currently Docker supports following driver plugins:

- authorization
- volume
- Network

For example, let's install Local Persist Plugin:

```
$ curl -fsSL https://raw.githubusercontent.com/CWSpear/local-persist/master/scripts/install.sh | sudo bash
```

Creating "jenkins" Volume with "local-persist" driver:

```
$ docker volume create -d local-persist -o mountpoint=/data/jenkins --name=jenkins
```

Run Jenkins with the volume created above:

```
$ docker run -d -v jenkins:/var/jenkins_home jenkins/jenkins:2.146
```

Using Local-Persist Volume with Docker-Compose

```
version: '2'

services:
  jenkins:
    image: jenkins/jenkins:2.146
    ports:
      - 80:8080
    volumes:
      - jenkins:/var/jenkins_home

volumes:
  jenkins:
    driver: local-persist
    driver_opts:
      mountpoint: /data/jenkins
```

Docker volume plugin for sshFS

Install plugin with : https://github.com/vieux/docker-volume-sshfs

```
# docker plugin install vieux/sshfs
Plugin "vieux/sshfs" is requesting the following privileges:
 - network: [host]
 - mount: [/var/lib/docker/plugins/]
 - mount: []
 - device: [/dev/fuse]
 - capabilities: [CAP_SYS_ADMIN]
Do you grant the above permissions? [y/N] y
latest: Pulling from vieux/sshfs
52d435ada6a4: Download complete
Digest: sha256:1d3c3e42c12138da5ef7873b97f7f32cf99fb6edde75fa4f0bcf9ed277855811
Status: Downloaded newer image for vieux/sshfs:latest
Installed plugin vieux/sshfs
```

Check the Plugin Status:

```
$ docker plugin ls | grep sshfs
0c3c6b601473      vieux/sshfs:latest    sshFS plugin for Docker       true
```

Docker volume plugin for sshFS

Create the Volume:

```
# docker volume create -d vieux/sshfs \
    -o sshcmd=<user@host:path> \
    -o password=<password> \
    [-o port=<port>] [-o <any_sshfs_-o_option> ] \
  sshvolume sshvolume
```

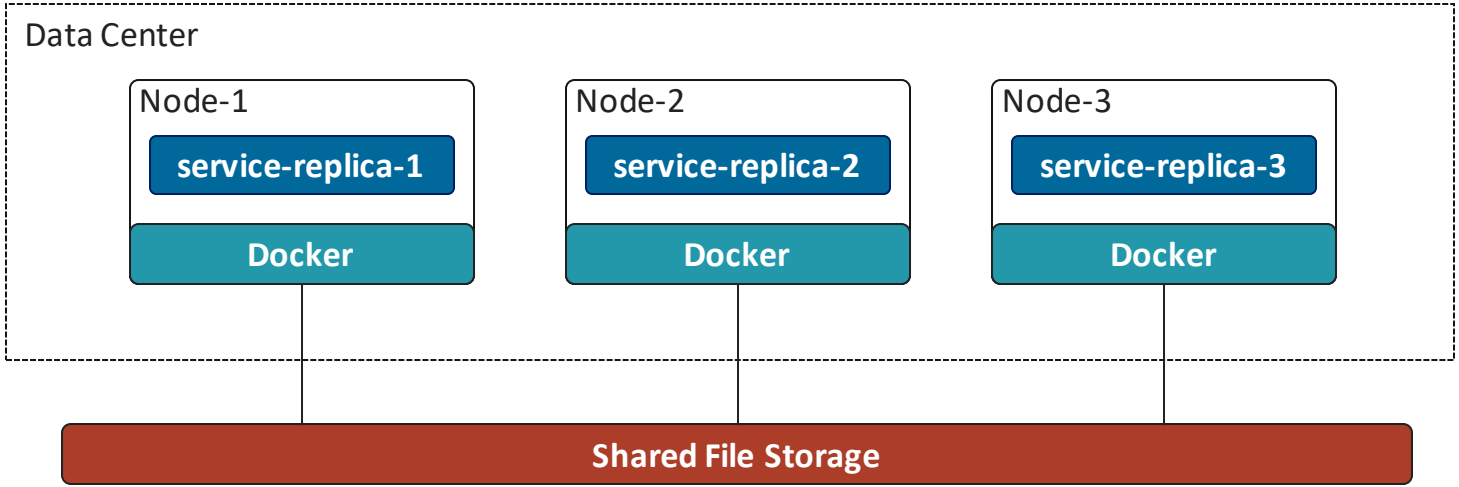
Check the volume :

```
# docker volume ls | grep sshvolume
vieux/sshfs       sshvolume
```

Use the volume :

```
# docker run -it -v sshvolume:<path_in_container> busybox ls <path>
```

When is It Useful?



NFS / AWS EBS or S3 / Azure File Storage

Using Volume Drivers in Clouds

https://docs.aws.amazon.com/AmazonECS/latest/developerguide/docker-volumes.html

https://docs.docker.com/docker-for-azure/persistent-data-volumes/

Example for Azure:

```
# docker plugin install \
       --alias cloudstor:azure \
       --grant-all-permissions docker4x/cloudstor:18.03.1-ce-azure1 \
       CLOUD_PLATFORM=AZURE \
       AZURE_STORAGE_ACCOUNT_KEY=$AZURE_STORAGE_ACCOUNT_KEY \
       AZURE_STORAGE_ACCOUNT=$AZURE_STORAGE_ACCOUNT
```

Rexray

Provider	Storage Platform	Docker
Amazon EC2	EBS	✓
	EFS	✓
	S3FS	✓
Ceph	RBD	✓
Dell EMC	Isilon	✓
	ScaleIO	✓
DigitalOcean	Block Storage	✓
FittedCloud	EBS Optimizer	✓
Google	GCE Persistent Disk	✓
Microsoft	Azure Unmanaged Disk	✓
OpenStack	Cinder	✓
VirtualBox	Virtual Media	✓

REX-Ray provides a vendor agnostic storage orchestration engine. The primary design goal is to provide persistent storage for Docker, Kubernetes, and Mesos.

```
docker plugin install \
  rexray/ebs \
  EBS_ACCESSKEY=access_key \
  EBS_SECRETKEY=secret_key
```

```
docker run -ti \
  --volume-driver=rexray/ebs \
  --volume test:/test \
  busybox
```

https://github.com/rexray/rexray

Monitoring Containers

- What you should look after
- Containers Monitoring with cAdvisor
- Collect Docker metrics with Prometheus

What You Should Look after

At least:

Image Count	-	The count of active and inactive images stored locally by Docker.
Image Size	-	The total disk usage of active and reclaimable images stored locally.
Container Count	-	The count of containers, segregated by state: created, running, paused, restarting, removing, exited and dead.
Container Size	-	The total size of disk used by containers, with used and reclaimable parts tracked separately.
Volume Count	-	The count of active and inactive volumes maintained by this Docker daemon.
Volume Size	-	The total size of disk used by volumes, with used and reclaimable parts tracked separately.
CPU Usage	-	The percentage of host CPU used by all currently running containers.
Memory Usage	-	The amount of host memory used by all currently running containers.
Network Bandwidth	-	The transmit and receive bandwidth of host network used by all currently running containers.
Disk Throughput	-	The disk read and write throughput (bytes/second) used by all currently running containers.
Process Count	-	The total number of processes running across all containers currently.

Monitoring Containers with cAdvisor

```
version: "3.6"

services:
  web:
    image: google/cadvisor:latest
    ports:
    - 0.0.0.0:9104:8080
    volumes:
    - /:/rootfs:ro
    - /var/run:/var/run:rw
    - /sys:/sys:ro
    - /var/lib/docker/:/var/lib/docker:ro
    - /dev/disk/:/dev/disk:ro
```

```
# docker-compose up -d
```

cAdvisor - /

Not Secure | 192.168.56.15:9104/containers/

Memory

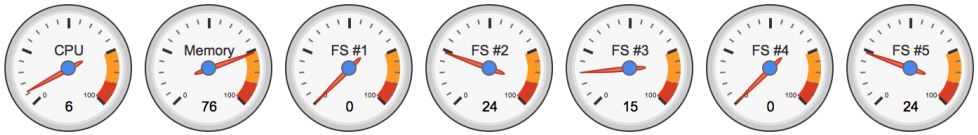
Reservation unlimited

Limit 487.58 *MB*

Swap Limit 2.00 *GB*

Usage

Overview



Processes

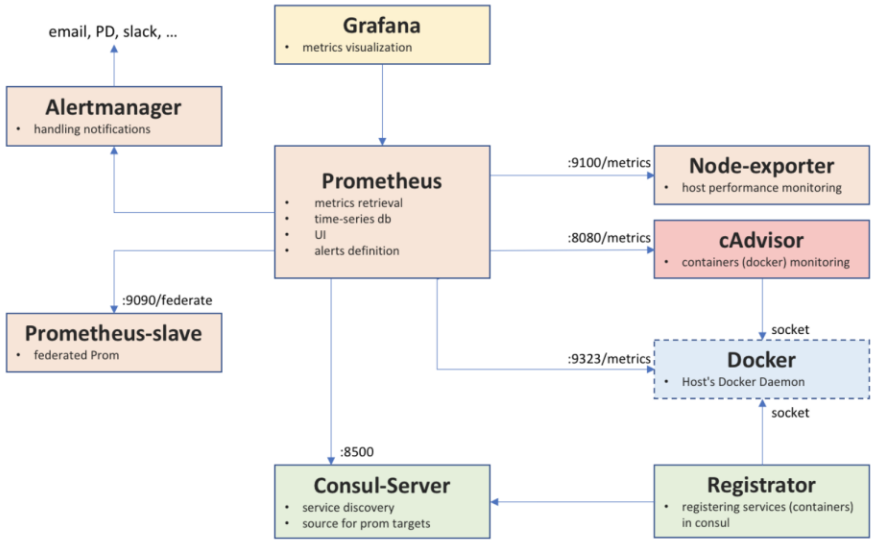
User	PID	PPID	Start Time	CPU %	MEM %	RSS	Virtual Size	Status	Running Time	Command	Container
root	9,694	9,677	18:59	5.00	13.70	67.09 MiB	443.49 MiB	Ssl	00:07:48	cadvisor	/docker/76c9637d77b036
root	1,003	1	10:09	1.10	9.40	46.07 MiB	1.46 GiB	Ssl	00:07:58	dockerd	/system.slice/docker
root	35	2	10:08	1.00	0.00	0.00 B	0.00 B	S	00:07:04	kswapd0	
root	1,267	1,003	10:09	0.30	2.50	12.64 MiB	1007.03 MiB	Ssl	00:02:23	docker-containe	/system.slice/docker
root	3	2	10:08	0.10	0.00	0.00 B	0.00 B	S	00:00:57	ksoftirqd/0	
root	9	2	10:08	0.10	0.00	0.00 B	0.00 B	S	00:01:13	rcu_sched	

Collect Docker metrics with Prometheus

/etc/docker/daemon.json

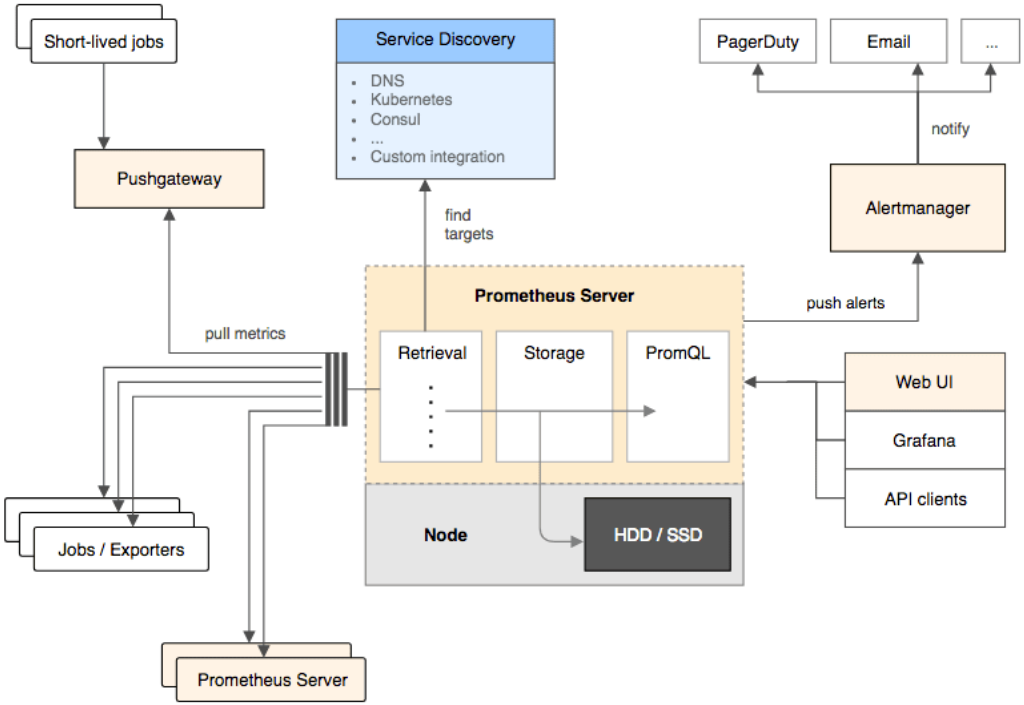
```
{
    "metrics-addr" : "0.0.0.0:9323",
    "experimental" : true
}
```

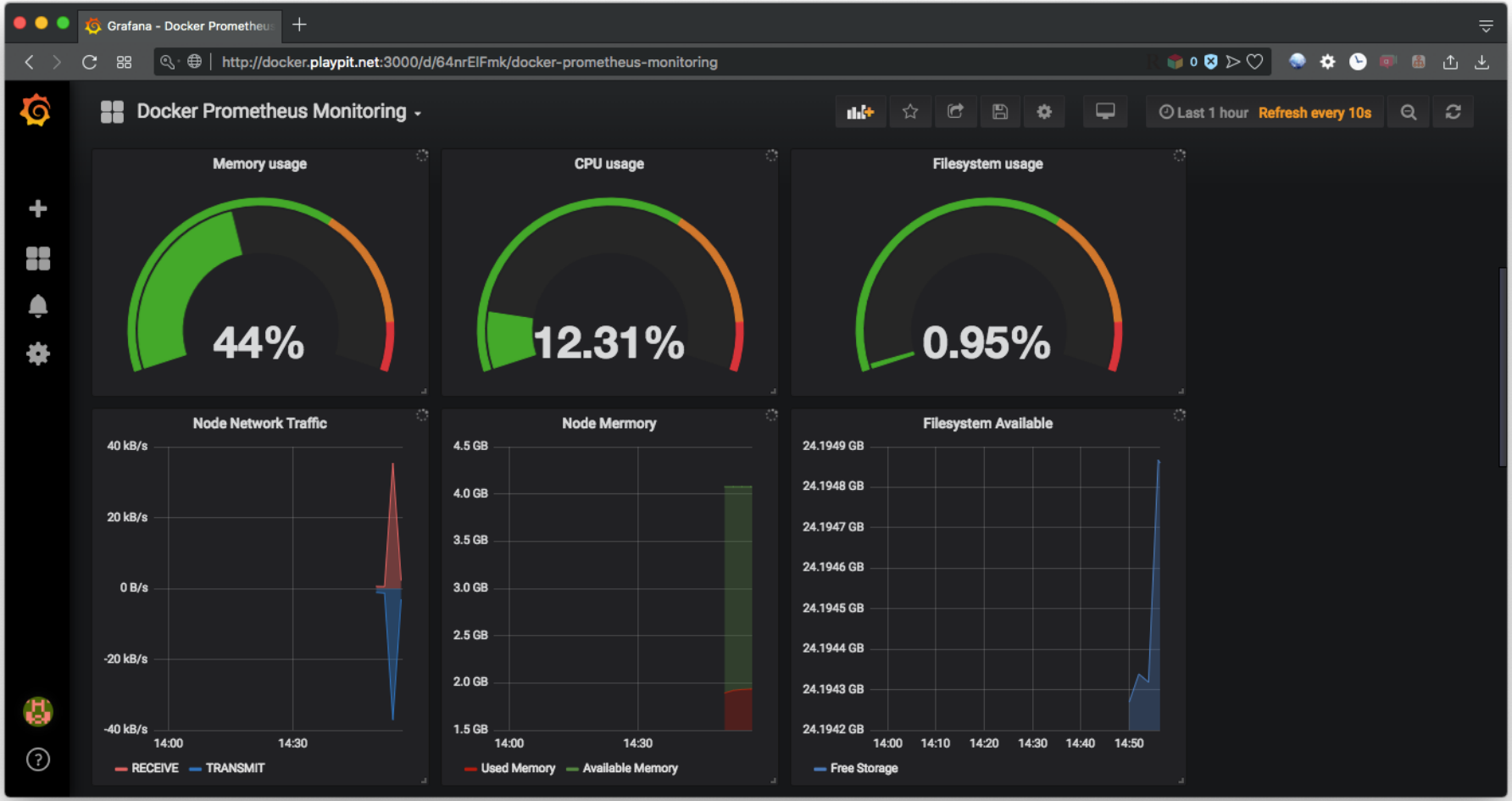
Detailed Example: monitoring/prometheus

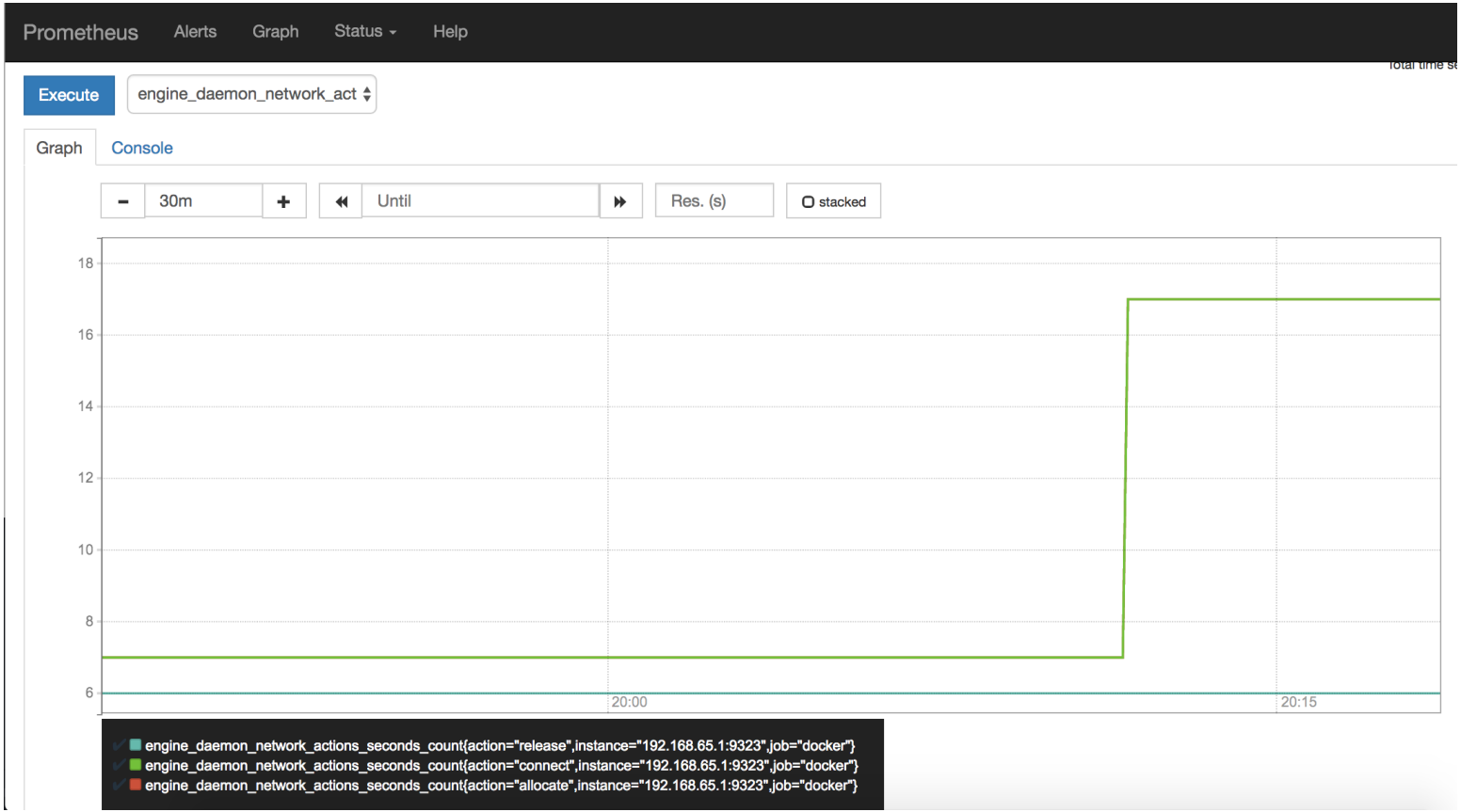


Prometheus Docker: https://docs.docker.com/config/thirdparty/prometheus/
Prometheus Project: https://prometheus.io/

Prometheus Architecture




Grafana - Docker Prometheus
http://docker.playpit.net:3000/d/64nrElFmk/docker-prometheus-monitoring
Docker Prometheus Monitoring
Last 1 hour
Refresh every 10s
Memory usage
44%
CPU usage
12.31%
Filesystem usage
0.95%
Node Network Traffic
40 kB/s
20 kB/s
0 B/s
-20 kB/s
-40 kB/s
14:00
14:30
RECEIVE
TRANSMIT
Node Mermory
4.5 GB
4.0 GB
3.5 GB
3.0 GB
2.5 GB
2.0 GB
1.5 GB
14:00
14:30
Used Memory
Available Memory
Filesystem Available
24.1949 GB
24.1948 GB
24.1947 GB
24.1946 GB
24.1945 GB
24.1944 GB
24.1943 GB
24.1942 GB
14:00
14:10
14:20
14:30
14:40
14:50
Free Storage


Prometheus
Alerts
Graph
Status
Help
Execute
engine_daemon_network_act
Graph
Console
30m
Until
Res. (s)
stacked
18
16
14
12
10
8
6
20:00
20:15
engine_daemon_network_actions_seconds_count{action="release",instance="192.168.65.1:9323",job="docker"}
engine_daemon_network_actions_seconds_count{action="connect",instance="192.168.65.1:9323",job="docker"}
engine_daemon_network_actions_seconds_count{action="allocate",instance="192.168.65.1:9323",job="docker"}

Useful Commands: Getting Info

Display system-wide information	`# docker system info`
Show docker disk usage	`# docker system df`
Show detailed information on space usage	`# docker system df -v`
Display total file sizes	`# docker ps -s`
Display resource usage statistics	`# docker stats --no-stream`
Show exited containers	`# docker ps -f status=exited`
Show unused images	`# docker images -f dangling=true`
Show unused volumes	`# docker volume ls -f dangling=true`

Useful Commands: Cleaning Up

```
# docker image prune
WARNING! This will remove all dangling images.
Are you sure you want to continue? [y/N]
you want to continue? [y/N]
```

```
# docker container prune
WARNING! This will remove all stopped containers.
Are you sure you want to continue? [y/N]
```

```
# docker volume prune
WARNING! This will remove all volumes not used by at least one container.
Are you sure you want to continue? [y/N]
```

```
# docker network prune
WARNING! This will remove all networks not used by at least one container.
Are you sure you want to continue? [y/N]
```

```
# docker system prune
WARNING! This will remove:
        - all stopped containers
        - all networks not used by at least one container
        - all dangling images
        - all build cache
Are you sure you want to continue? [y/N]
```

Containers Vulnerabilities

- Checking Containers

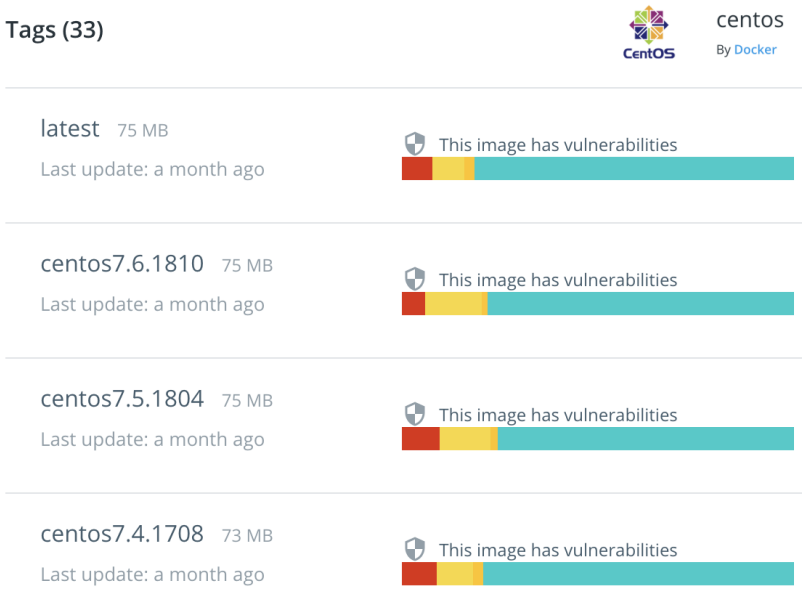
Docker Security

Docker vulnerabilities and threats to battle

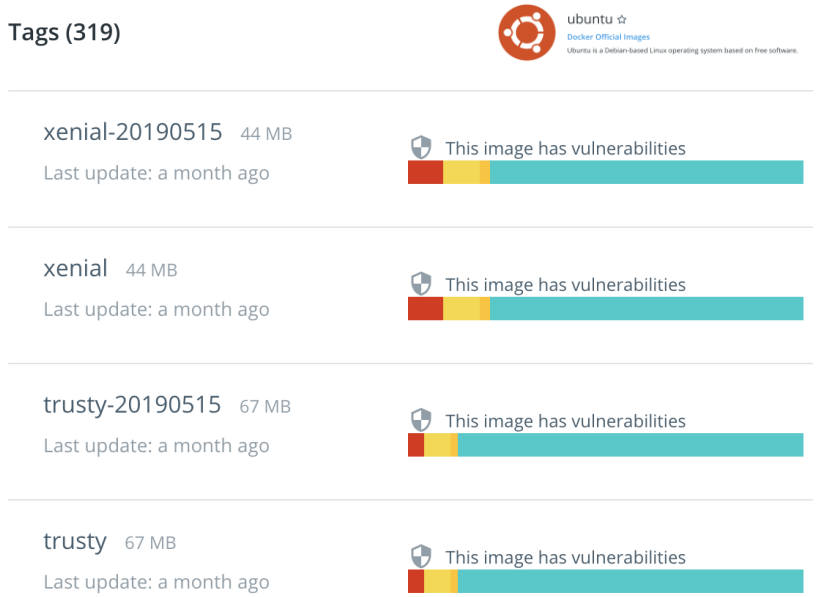
- Docker host and kernel security
- Docker container breakout
- Container image authenticity
- Container resource abuse
- Docker security vulnerabilities present in the static image
- Docker credentials and secrets
- Docker runtime security monitoring

Docker Image Vulnerabilities

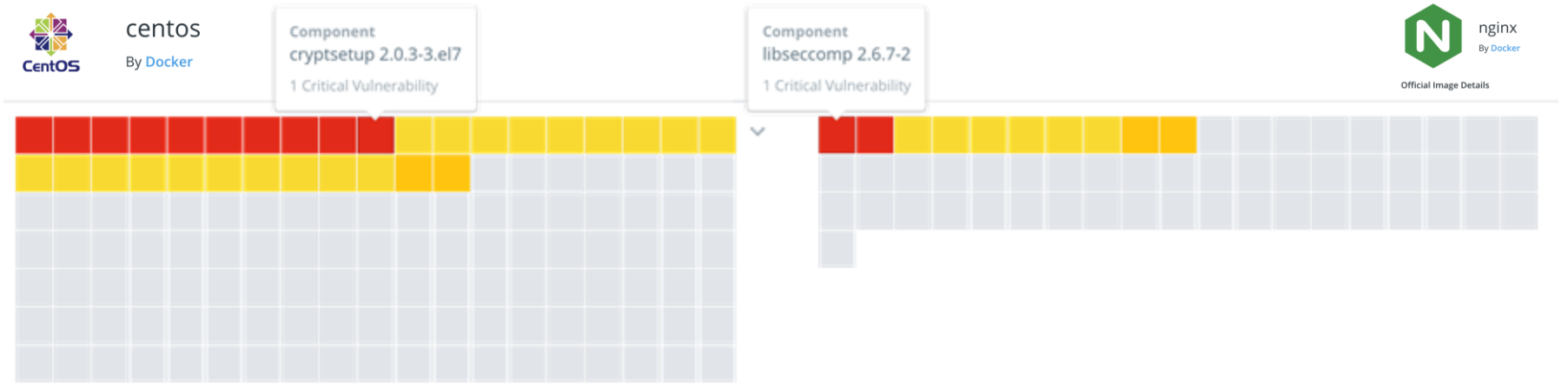
https://hub.docker.com/_/centos?tab=tags



https://hub.docker.com/_/ubuntu?tab=tags



Docker Image Vulnerabilities



Container Image Scanning Open-source Tools

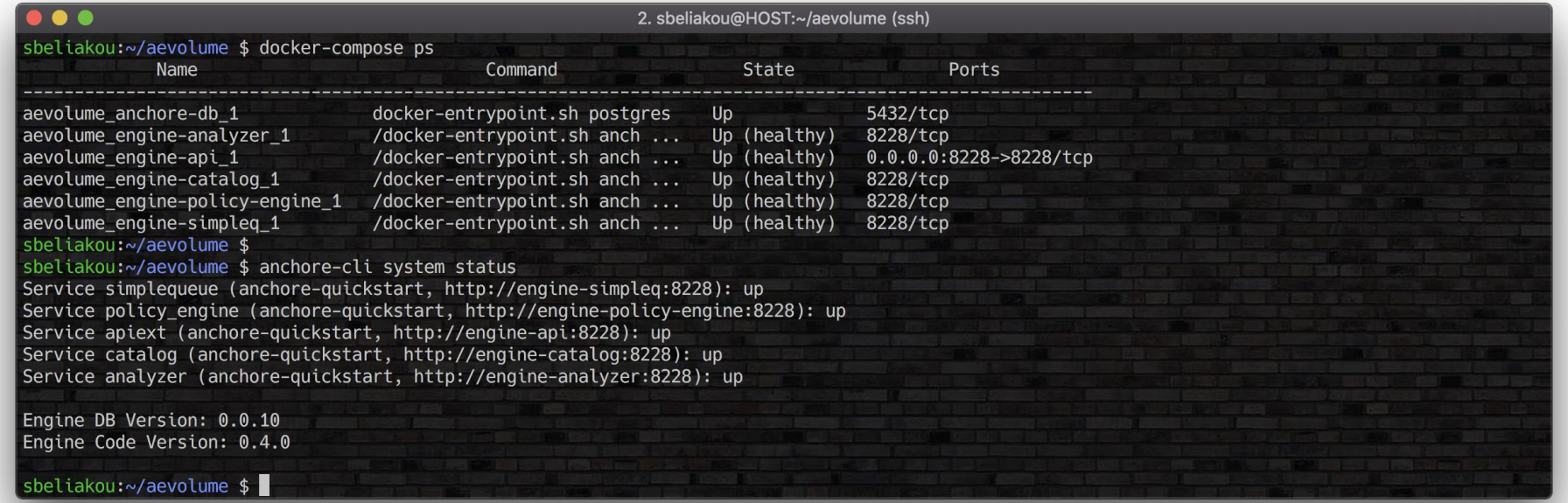
There are several Docker image scanning tools available, and some of the most popular include:

- **Anchore Engine:** Anchore Engine is an open source image scanning tool. Provides a centralized service for inspection, analysis and applies user-defined acceptance policies to allow automated validation and certification of container images.
- **CoreOS/Clair:** An open source project for the static analysis of vulnerabilities in application containers (currently including appc/Rkt and Docker).
- **Vuls.io:** Agent-less Linux vulnerability scanner based on information from NVD, OVAL, etc. It has some container image support, although is not a container specific tool.
- **OpenScap:** Suite of automated audit tools to examine the configuration and known vulnerabilities in your software, following the NIST-certified Security Content Automation Protocol (SCAP). Not container specific again, but does include some level of support.
- etc

https://sysdig.com/blog/20-docker-security-tools/

Anchore Engine

https://github.com/anchore/anchore-engine
https://github.com/anchore/anchore-engine#configuration
https://github.com/anchore/anchore-engine#running-anchore-engine-using-docker-compose
https://sysdig.com/blog/container-security-docker-image-scanning/



Anchore CLI

https://github.com/anchore/anchore-cli

```
$ yum install epel-release
$ yum install python-pip
$ pip install anchorecli
```

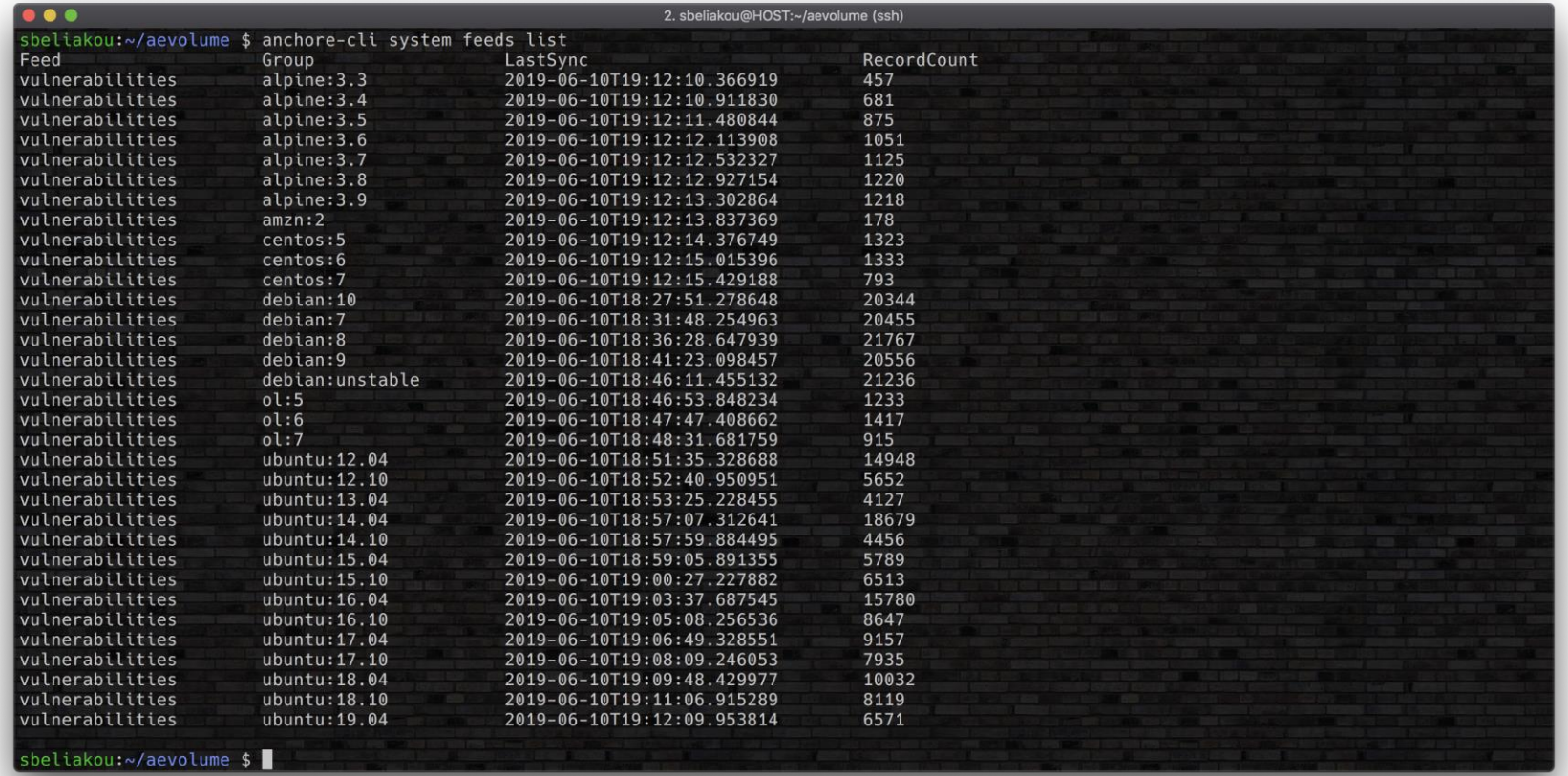
2. sbeliakou@HOST:~/aevolume (ssh)

```
sbeliakou:~/aevolume $ anchore-cli image add docker.io/library/ubuntu:19.04
Image Digest: sha256:887a7c706e88e4569494c9f470a53217a2705c9ddb3f8f75dc9062ece9e0bd36
Parent Digest: sha256:1f070d23781c4e0aa3841ae9d7494885283e199e61eddec8337cd1895f864d67
Analysis Status: not_analyzed
Image Type: docker
Image ID: 9b17fc7d68486409b6adf12e02fbf19b509714e5e7e2993e38d05ef47a29bfcc
Dockerfile Mode: None
Distro: None
Distro Version: None
Size: None
Architecture: None
Layer Count: None

Full Tag: docker.io/library/ubuntu:19.04


sbeliakou:~/aevolume $ anchore-cli image list
Full Tag                            Image Digest                                                            Analysis Status
docker.io/centos:7                  sha256:ca58fe458b8d94bc6e3072f1cfbd334855858e05e1fd633aa07cf7f82b048e66  not_analyzed
docker.io/jenkins/jenkins:latest    sha256:bb0dce6f9887619e01bbeff11b6f99a9470138f65ab570f7813d25029f43cb1e  analyzed
docker.io/jenkins/jenkins:lts       sha256:08bdd27b066e4ec6bfa1228876551e939df1aeaa9d878e6ddb2b183cd208dc2b  analyzing
docker.io/library/python:2.7        sha256:3079b0d54fd139626b117f01ce61fb3e84e37b73c5334d80b47cdd196a0e5036  analyzed
docker.io/library/python:3.5        sha256:8e4a6308dbdf20a6c46b79449d1eb7c191186c790233fdf247e5f40aea14438c  not_analyzed
docker.io/library/ubuntu:19.04      sha256:887a7c706e88e4569494c9f470a53217a2705c9ddb3f8f75dc9062ece9e0bd36  not_analyzed
sbeliakou:~/aevolume $
```

Anchore Vulnerabilities Feeds



2. sbeliakou@HOST:~/aevolume (ssh)

```
sbeliakou:~/aevolume $ anchore-cli image vuln docker.io/jenkins/jenkins:lts os
Vulnerability ID        Package                                 Severity        Fix             Vulnerability URL
CVE-2017-14062          libidn11-1.33-1                         High            None            https://security-tracker.debian.org/tracker/CVE-2017-14062
CVE-2017-17458          mercurial-4.0-1+deb9u1                  High            None            https://security-tracker.debian.org/tracker/CVE-2017-17458
CVE-2017-17458          mercurial-common-4.0-1+deb9u1           High            None            https://security-tracker.debian.org/tracker/CVE-2017-17458
CVE-2018-13347          mercurial-4.0-1+deb9u1                  High            None            https://security-tracker.debian.org/tracker/CVE-2018-13347
CVE-2018-13347          mercurial-common-4.0-1+deb9u1           High            None            https://security-tracker.debian.org/tracker/CVE-2018-13347
CVE-2019-8457           libsqlite3-0-3.16.2-5+deb9u1            High            None            https://security-tracker.debian.org/tracker/CVE-2019-8457
CVE-2018-16868          libgnutls30-3.5.8-5+deb9u4              Low             None            https://security-tracker.debian.org/tracker/CVE-2018-16868
CVE-2011-3389           libgnutls30-3.5.8-5+deb9u4              Medium          None            https://security-tracker.debian.org/tracker/CVE-2011-3389
CVE-2018-1000132        mercurial-4.0-1+deb9u1                  Medium          None            https://security-tracker.debian.org/tracker/CVE-2018-1000132
CVE-2018-1000132        mercurial-common-4.0-1+deb9u1           Medium          None            https://security-tracker.debian.org/tracker/CVE-2018-1000132
CVE-2018-12404          libnss3-2:3.26.2-1.1+deb9u1             Medium          None            https://security-tracker.debian.org/tracker/CVE-2018-12404
CVE-2018-13346          mercurial-4.0-1+deb9u1                  Medium          None            https://security-tracker.debian.org/tracker/CVE-2018-13346
```

2. sbeliakou@HOST:~/aevolume (ssh)

```
sbeliakou:~/aevolume $ anchore-cli image vuln docker.io/library/centos:latest os
Vulnerability ID        Package                         Severity        Fix                     Vulnerability URL
RHSA-2019:0679          libssh2-1.4.3-12.el7            High            0:1.4.3-12.el7_6.2      https://access.redhat.com/errata/RHSA-2019:0679
RHSA-2019:0710          python-2.7.5-76.el7             High            0:2.7.5-77.el7_6        https://access.redhat.com/errata/RHSA-2019:0710
RHSA-2019:0710          python-libs-2.7.5-76.el7        High            0:2.7.5-77.el7_6        https://access.redhat.com/errata/RHSA-2019:0710
RHSA-2019:1294          bind-license-9.9.4-73.el7_6     High            32:9.9.4-74.el7_6.1     https://access.redhat.com/errata/RHSA-2019:1294
RHSA-2019:0483          openssl-libs-1.0.2k-16.el7      Medium          1:1.0.2k-16.el7_6.1     https://access.redhat.com/errata/RHSA-2019:0483
sbeliakou:~/aevolume $
```

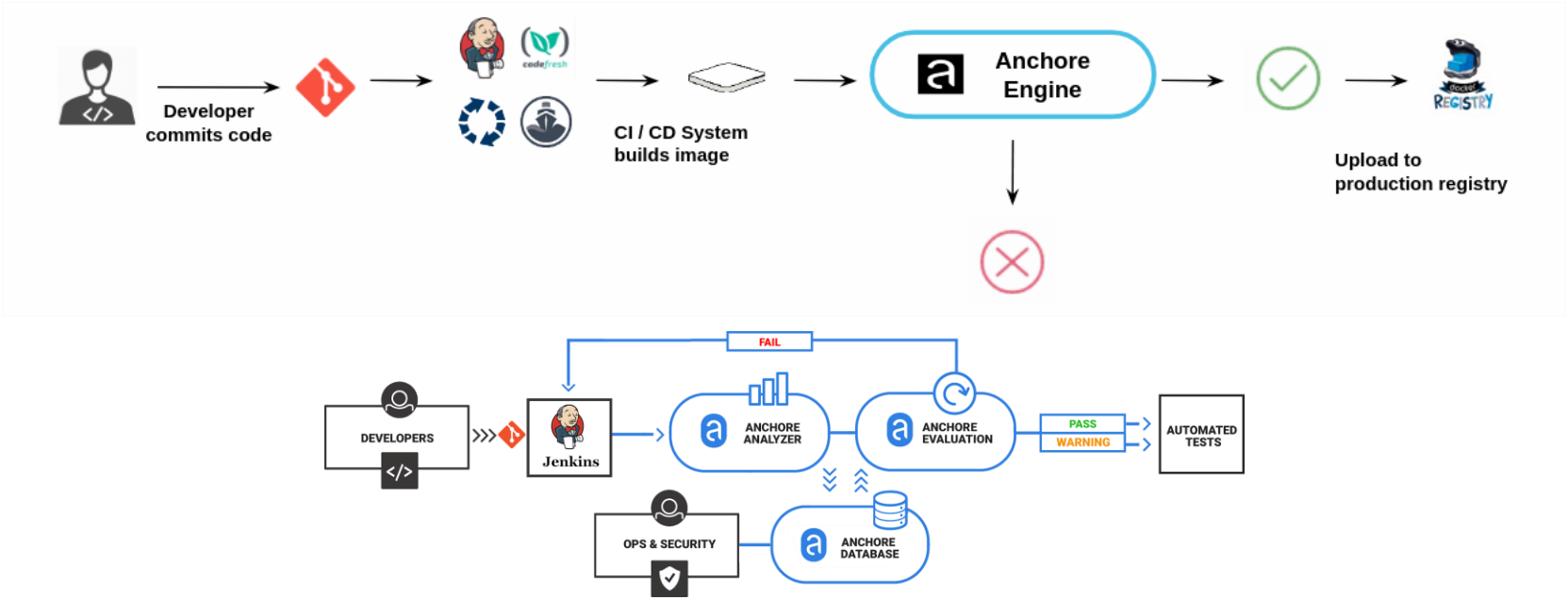
2. sbeliakou@HOST:~/aevolume (ssh)

```
sbeliakou:~/aevolume $ anchore-cli image vuln docker.io/library/ubuntu:18.04 os
Vulnerability ID        Package                         Severity        Fix             Vulnerability URL
CVE-2013-4235           login-1:4.5-1ubuntu2            Low             None            http://people.ubuntu.com/~ubuntu-security/cve/CVE-2013-4235
CVE-2013-4235           passwd-1:4.5-1ubuntu2           Low             None            http://people.ubuntu.com/~ubuntu-security/cve/CVE-2013-4235
CVE-2015-8985           libc-bin-2.27-3ubuntu1          Low             None            http://people.ubuntu.com/~ubuntu-security/cve/CVE-2015-8985
CVE-2015-8985           libc6-2.27-3ubuntu1             Low             None            http://people.ubuntu.com/~ubuntu-security/cve/CVE-2015-8985
CVE-2016-2781           coreutils-8.28-1ubuntu1         Low             None            http://people.ubuntu.com/~ubuntu-security/cve/CVE-2016-2781
CVE-2018-16868          libgnutls30-3.5.18-1ubuntu1     Low             None            http://people.ubuntu.com/~ubuntu-security/cve/CVE-2018-16868
CVE-2018-16869          libhogweed4-3.4-1               Low             None            http://people.ubuntu.com/~ubuntu-security/cve/CVE-2018-16869
CVE-2018-16869          libnettle6-3.4-1                Low             None            http://people.ubuntu.com/~ubuntu-security/cve/CVE-2018-16869
```

2. sbeliakou@HOST:~/aevolume (ssh)

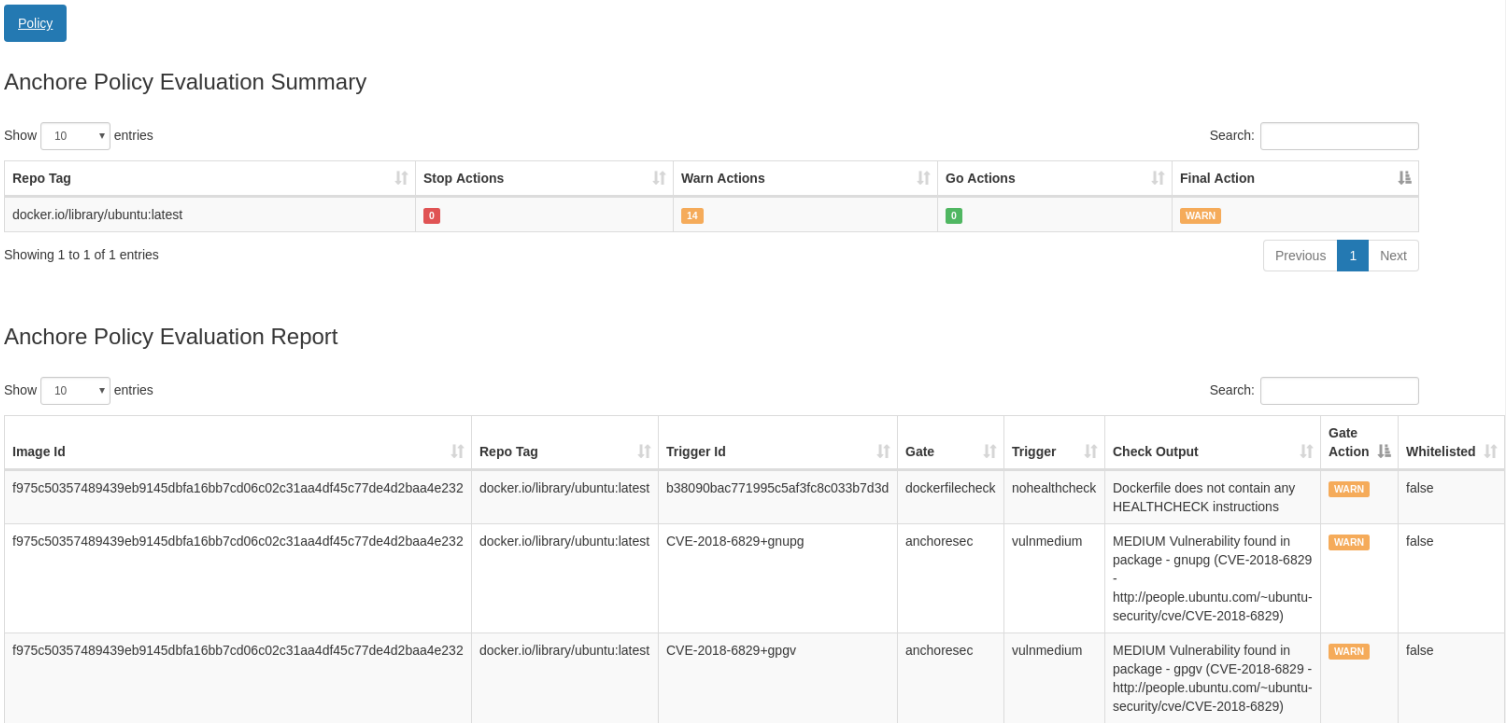
```
sbeliakou:~/aevolume $ anchore-cli image vuln docker.io/library/python:2.7 all | wc -l
1832
sbeliakou:~/aevolume $
```

Docker Scanner with Jenkins



https://wiki.jenkins.io/display/JENKINS/Anchore+Container+Image+Scanner+Plugin
https://medium.com/accelero/container-security-with-anchore-8785efa644a5

Docker Scanner with Jenkins



Policy

Anchore Policy Evaluation Summary

Show 10 entries Search:

Repo Tag	Stop Actions	Warn Actions	Go Actions	Final Action
docker.io/library/ubuntu:latest	0	14	0	WARN

Showing 1 to 1 of 1 entries Previous 1 Next

Anchore Policy Evaluation Report

Show 10 entries Search:

Image Id	Repo Tag	Trigger Id	Gate	Trigger	Check Output	Gate Action	Whitelisted
f975c50357489439eb9145dbfa16bb7cd06c02c31aa4df45c77de4d2baa4e232	docker.io/library/ubuntu:latest	b38090bac771995c5af3fc8c033b7d3d	dockerfilecheck	nohealthcheck	Dockerfile does not contain any HEALTHCHECK instructions	WARN	false
f975c50357489439eb9145dbfa16bb7cd06c02c31aa4df45c77de4d2baa4e232	docker.io/library/ubuntu:latest	CVE-2018-6829+gnupg	anchoresec	vulnmedium	MEDIUM Vulnerability found in package - gnupg (CVE-2018-6829 - http://people.ubuntu.com/~ubuntu-security/cve/CVE-2018-6829)	WARN	false
f975c50357489439eb9145dbfa16bb7cd06c02c31aa4df45c77de4d2baa4e232	docker.io/library/ubuntu:latest	CVE-2018-6829+gpgv	anchoresec	vulnmedium	MEDIUM Vulnerability found in package - gpgv (CVE-2018-6829 - http://people.ubuntu.com/~ubuntu-security/cve/CVE-2018-6829)	WARN	false

https://wiki.jenkins.io/display/JENKINS/Anchore+Container+Image+Scanner+Plugin

Anchore CI Demos

https://github.com/Btodhunter/ci-demos

Jenkins

```
pipeline{
    agent {
        docker {
            image 'docker:stable'
        }
    }
    environment {
        IMAGE_NAME = 'btodhunter/anchore-ci-demo'
        IMAGE_TAG = 'jenkins'
    }
    stages {
        stage('Build Image') {
            steps {
                sh 'docker build -t ${IMAGE_NAME}:ci .'
            }
        }
        stage('Scan') {
            steps {
                sh 'apk add bash curl'
                sh 'curl -s https://ci-tools.anchore.io/inline_scan-v0.3.3 | bash -s -- -d
            }
        }
        stage('Push Image') {
            steps {
                withDockerRegistry([credentialsId: "dockerhub-creds", url: ""]){
                    sh 'docker tag ${IMAGE_NAME}:ci ${IMAGE_NAME}:${IMAGE_TAG}'
                    sh 'docker push ${IMAGE_NAME}:${IMAGE_TAG}'
                }
            }
        }
    }
}
```

GitLab

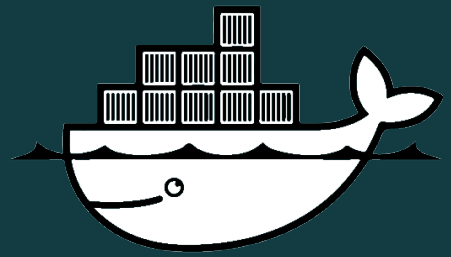
```
variables:
  IMAGE_NAME: ${CI_REGISTRY_IMAGE}/build:${CI_COMMIT_REF_SLUG}-${CI_COMMIT_SHA}

stages:
- build
- scan
- publish

container_build:
  stage: build
  image: docker:stable
  services:
  - docker:stable-dind

  variables:
    DOCKER_DRIVER: overlay2

  script:
  - echo "$CI_JOB_TOKEN" | docker login -u gitlab-ci-token --password-stdin "${CI_REGISTRY}"
  - docker build -t "$IMAGE_NAME" .
  - apk add bash curl
  - curl -s https://ci-tools.anchore.io/inline_scan-v0.3.3 | bash -s -- -r -t 500 "$IMAGE_NAME"
  - docker push "$IMAGE_NAME"
  - |
      echo "Parsing anchore reports."
      apk add jq
      bash <<'EOF'
      for f in anchore-reports/*; do
        if [[ "$f" =~ "content-os" ]]; then
          printf "\n%s\n" "The following OS packages are installed on ${IMAGE_NAME}:"
          jq '[.content | sort_by(.package) | .[] | {package: .package, version: .version}]' $f || true
        fi
        if [[ "$f" =~ "vuln" ]]; then
          printf "\n%s\n" "The following vulnerabilities were found on ${IMAGE_NAME}:"
          jq '[.vulnerabilities | group_by(.package) | .[] | {package: .[0].package, vuln: [.[].vuln]}]' $f || true
        fi
      done
      EOF
```



That's it for this training!

Thank you for your attention!

Siarhei Beliakou,

2019