**Docker Administration**

1)Running Docker on a specific Port

vi /etc/docker/daemon.json

{

"debug": true,

"hosts": ["tcp://10.0.10.4:2376"]

}

systemctl restart docker

docker -H 10.0.10.4:237 ps -a

2) Running Docker deamon in debug mode

{

"debug": true,

"hosts": ["tcp://10.0.10.4:2376"]

}

journalctl -r -o verbose -u docker.service

Read the logs

The daemon logs may help you diagnose problems. The logs may be saved in one of a few locations, depending on the operating system configuration and the logging subsystem used:

| Operating system | Location |
| --- | --- |
| RHEL, Oracle Linux | /var/log/messages |
| Debian | /var/log/daemon.log |
| Ubuntu 16.04+, CentOS | Use the command journalctl -u docker.service |
| Ubuntu 14.10- | /var/log/upstart/docker.log |
| macOS | ~/Library/Containers/com.docker.docker/Data/com.docker.driver.amd64-linux/console-ring |
| Windows | AppData\Local |

# 3) Collect Docker metrics with Prometheus

Add the below in the

#vi /etc/docker/daemon.json

{

"metrics-addr" : "<IP ADDR of Machine>:9323",

"experimental" : true

}

Create Prometheus Config file

#vi /tmp/prometheus.yml

# my global config

global:

scrape\_interval: 15s # Set the scrape interval to every 15 seconds. Default is every 1 minute.

evaluation\_interval: 15s # Evaluate rules every 15 seconds. The default is every 1 minute.

# scrape\_timeout is set to the global default (10s).

# Attach these labels to any time series or alerts when communicating with

# external systems (federation, remote storage, Alertmanager).

external\_labels:

monitor: 'codelab-monitor'

# Load rules once and periodically evaluate them according to the global 'evaluation\_interval'.

rule\_files:

# - "first.rules"

# - "second.rules"

# A scrape configuration containing exactly one endpoint to scrape:

# Here it's Prometheus itself.

scrape\_configs:

# The job name is added as a label `job=<job\_name>` to any timeseries scraped from this config.

- job\_name: 'prometheus'

# metrics\_path defaults to '/metrics'

# scheme defaults to 'http'.

static\_configs:

- targets: ['<IPADDR>:9090']

- job\_name: 'docker'

# metrics\_path defaults to '/metrics'

# scheme defaults to 'http'.

static\_configs:

- targets: ['<IPADDR>:9323']

Run the Prometheus docker

docker service create --replicas 1 --name my-prometheus \

--mount type=bind,source=/tmp/prometheus.yml,destination=/etc/prometheus/prometheus.yml \

--publish 9090:9090/tcp \

prom/prometheus

Forward localhost 9090 port to <IPADDR> 9090 port in Virtual box

Access Prometheus by using

<http://localhost:9090/targets>

Start Container automatically

4) To configure the restart policy for a container, use the --restart flag when using the docker run command. The value of the --restart flag can be any of the following:

| **Flag** | **Description** |
| --- | --- |
| no | Do not automatically restart the container. (the default) |
| on-failure | Restart the container if it exits due to an error, which manifests as a non-zero exit code. |
| unless-stopped | Restart the container unless it is explicitly stopped or Docker itself is stopped or restarted. |
| always | Always restart the container if it stops. |

The following example starts a Redis container and configures it to always restart unless it is explicitly stopped or Docker is restarted.

$ docker run -dit --restart unless-stopped redis

# 5) Keep containers alive during daemon downtime

#vi /etc/docker/daemon.json

{

"live-restore": true

}

#systemctl restart docker

#docker ps

#docker stop docker

#docker start docker

#docker ps

6) Setting Memory and CPU for a Docker

### Limit a container’s access to memory

Docker can enforce hard memory limits, which allow the container to use no more than a given amount of user or system memory, or soft limits, which allow the container to use as much memory as it needs unless certain conditions are met, such as when the kernel detects low memory or contention on the host machine. Some of these options have different effects when used alone or when more than one option is set.

Most of these options take a positive integer, followed by a suffix of b, k, m, g, to indicate bytes, kilobytes, megabytes, or gigabytes.

| **Option** | **Description** |
| --- | --- |
| -m or --memory= | The maximum amount of memory the container can use. If you set this option, the minimum allowed value is 4m (4 megabyte). |
| --memory-swap\* | The amount of memory this container is allowed to swap to disk. See [--memory-swap details](https://docs.docker.com/engine/admin/resource_constraints/#--memory-swap-details). |
| --memory-swappiness | By default, the host kernel can swap out a percentage of anonymous pages used by a container. You can set --memory-swappiness to a value between 0 and 100, to tune this percentage. See [--memory-swappiness details](https://docs.docker.com/engine/admin/resource_constraints/#--memory-swappiness-details). |
| --memory-reservation | Allows you to specify a soft limit smaller than --memory which is activated when Docker detects contention or low memory on the host machine. If you use --memory-reservation, it must be set lower than --memory in order for it to take precedence. Because it is a soft limit, it does not guarantee that the container will not exceed the limit. |
| --kernel-memory | The maximum amount of kernel memory the container can use. The minimum allowed value is 4m. Because kernel memory cannot be swapped out, a container which is starved of kernel memory may block host machine resources, which can have side effects on the host machine and on other containers. See [--kernel-memory details](https://docs.docker.com/engine/admin/resource_constraints/#--kernel-memory-details). |
| --oom-kill-disable | By default, if an out-of-memory (OOM) error occurs, the kernel kills processes in a container. To change this behavior, use the --oom-kill-disable option. Only disable the OOM killer on containers where you have also set the -m/--memory option. If the -m flag is not set, the host can run out of memory and the kernel may need to kill the host system’s processes to free memory. |

For more information about cgroups and memory in general, see the documentation for [Memory Resource Controller](https://www.kernel.org/doc/Documentation/cgroup-v1/memory.txt).

### --memory-swap details

--memory-swap is a modifier flag that only has meaning if --memory is also set. Using swap allows the container to write excess memory requirements to disk when the container has exhausted all the RAM that is available to it. There is a performance penalty for applications that swap memory to disk often.

Its setting can have complicated effects:

* If --memory-swap is set to a positive integer, then both --memory and --memory-swapmust be set. --memory-swap represents the total amount of memory and swap that can be used, and --memory controls the amount used by non-swap memory. So if --memory="300m" and --memory-swap="1g", the container can use 300m of memory and 700m (1g - 300m) swap.
* If --memory-swap is set to 0, the setting is ignored, and the value is treated as unset.
* If --memory-swap is set to the same value as --memory, and --memory is set to a positive integer, **the container will not have access to swap**. See [Prevent a container from using swap](https://docs.docker.com/engine/admin/resource_constraints/#prevent-a-container-from-using-swap).
* If --memory-swap is unset, and --memory is set, the container can use twice as much swap as the --memory setting, if the host container has swap memory configured. For instance, if --memory="300m" and --memory-swap is not set, the container can use 300m of memory and 600m of swap.
* If --memory-swap is explicitly set to -1, the container is allowed to use unlimited swap, up to the amount available on the host system.

#### **PREVENT A CONTAINER FROM USING SWAP**

If --memory and --memory-swap are set to the same value, this will prevent containers from using any swap. This is because --memory-swap is the amount of combined memory and swap that can be used, while --memory is only the amount of physical memory that can be used.

### --memory-swappiness details

* A value of 0 turns off anonymous page swapping.
* A value of 100 sets all anonymous pages as swappable.
* By default, if you do not set --memory-swappiness, the value is inherited from the host machine.

### --kernel-memory details

Kernel memory limits are expressed in terms of the overall memory allocated to a container. Consider the following scenarios:

* **Unlimited memory, unlimited kernel memory**: This is the default behavior.
* **Unlimited memory, limited kernel memory**: This is appropriate when the amount of memory needed by all cgroups is greater than the amount of memory that actually exists on the host machine. You can configure the kernel memory to never go over what is available on the host machine, and containers which need more memory need to wait for it.
* **Limited memory, umlimited kernel memory**: The overall memory is limited, but the kernel memory is not.
* **Limited memory, limited kernel memory**: Limiting both user and kernel memory can be useful for debugging memory-related problems. If a container is using an unexpected amount of either type of memory, it will run out of memory without affecting other containers or the host machine. Within this setting, if the kernel memory limit is lower than the user memory limit, running out of kernel memory will cause the container to experience an OOM error. If the kernel memory limit is higher than the user memory limit, the kernel limit will not cause the container to experience an OOM.

When you turn on any kernel memory limits, the host machine tracks “high water mark” statistics on a per-process basis, so you can track which processes (in this case, containers) are using excess memory. This can be seen per process by viewing /proc/<PID>/status on the host machine.

## **CPU**

By default, each container’s access to the host machine’s CPU cycles is unlimited. You can set various constraints to limit a given container’s access to the host machine’s CPU cycles.

### Configure the default CFS scheduler

The CFS is the Linux kernel CPU scheduler for normal Linux processes. Several runtime flags allow you to configure the amount of access to CPU resources your container has. When you use these settings, Docker modifies the settings for the container’s cgroup on the host machine.

| **Option** | **Description** |
| --- | --- |
| --cpus=<value> | Specify how much of the available CPU resources a container can use. For instance, if the host machine has two CPUs and you set --cpus="1.5", the container will be guaranteed to be able to access at most one and a half of the CPUs. This is the equivalent of setting --cpu-period="100000" and --cpu-quota="150000". Available in Docker 1.13 and higher. |
| --cpu-period=<value> | Specify the CPU CFS scheduler period, which is used alongside--cpu-quota. Defaults to 100 micro-seconds. Most users do not change this from the default. If you use Docker 1.13 or higher, use --cpus instead. |
| --cpu-quota=<value> | Impose a CPU CFS quota on the container. The number of microseconds per --cpu-period that the container is guaranteed CPU access. In other words, cpu-quota / cpu-period. If you use Docker 1.13 or higher, use --cpus instead. |
| --cpuset-cpus | Limit the specific CPUs or cores a container can use. A comma-separated list or hyphen-separated range of CPUs a container can use, if you have more than one CPU. The first CPU is numbered 0. A valid value might be 0-3 (to use the first, second, third, and fourth CPU) or 1,3 (to use the second and fourth CPU). |
| --cpu-shares | Set this flag to a value greater or less than the default of 1024 to increase or reduce the container’s weight, and give it access to a greater or lesser proportion of the host machine’s CPU cycles. This is only enforced when CPU cycles are constrained. When plenty of CPU cycles are available, all containers use as much CPU as they need. In that way, this is a soft limit. --cpu-shares does not prevent containers from being scheduled in swarm mode. It prioritizes container CPU resources for the available CPU cycles. It does not guarantee or reserve any specific CPU access. |

If you have 1 CPU, each of the following commands will guarantee the container at most 50% of the CPU every second.

**Docker 1.13 and higher**:

docker run -it --cpus=".5" ubuntu /bin/bash

### 7) Runtime directory

### vi /etc/docker/daemon.json

{

"graph": "/mnt/docker-data",

}

### #systemctl restart docker

### 8) HTTP/HTTPS proxy

The Docker daemon uses the HTTP\_PROXY, HTTPS\_PROXY, and NO\_PROXY environmental variables in its start-up environment to configure HTTP or HTTPS proxy behavior. You cannot configure

these environment variables using the daemon.json file.

This example overrides the default docker.service file.

If you are behind an HTTP or HTTPS proxy server, for example in corporate settings, you will need to add this configuration in the Docker systemd service file.

1. Create a systemd drop-in directory for the docker service:
2. $ mkdir -p /etc/systemd/system/docker.service.d
3. Create a file called /etc/systemd/system/docker.service.d/http-proxy.conf that adds the HTTP\_PROXY environment variable:
4. [Service]
5. Environment="HTTP\_PROXY=http://proxy.example.com:80/"

Or, if you are behind an HTTPS proxy server, create a file called/etc/systemd/system/docker.service.d/https-proxy.conf that adds the HTTPS\_PROXYenvironment variable:

[Service]

Environment="HTTPS\_PROXY=https://proxy.example.com:443/"

1. If you have internal Docker registries that you need to contact without proxying you can specify them via the NO\_PROXY environment variable:
2. Environment="HTTP\_PROXY=http://proxy.example.com:80/" "NO\_PROXY=localhost,127.0.0.1,docker-registry.somecorporation.com"

Or, if you are behind an HTTPS proxy server:

Environment="HTTPS\_PROXY=https://proxy.example.com:443/" "NO\_PROXY=localhost,127.0.0.1,docker-registry.somecorporation.com"

1. Flush changes:
2. $ sudo systemctl daemon-reload
3. Restart Docker:
4. $ sudo systemctl restart docker
5. Verify that the configuration has been loaded:
6. $ systemctl show --property=Environment docker
7. Environment=HTTP\_PROXY=http://proxy.example.com:80/

Or, if you are behind an HTTPS proxy server:

$ systemctl show --property=Environment docker

Environment=HTTPS\_PROXY=https://proxy.example.com:443/