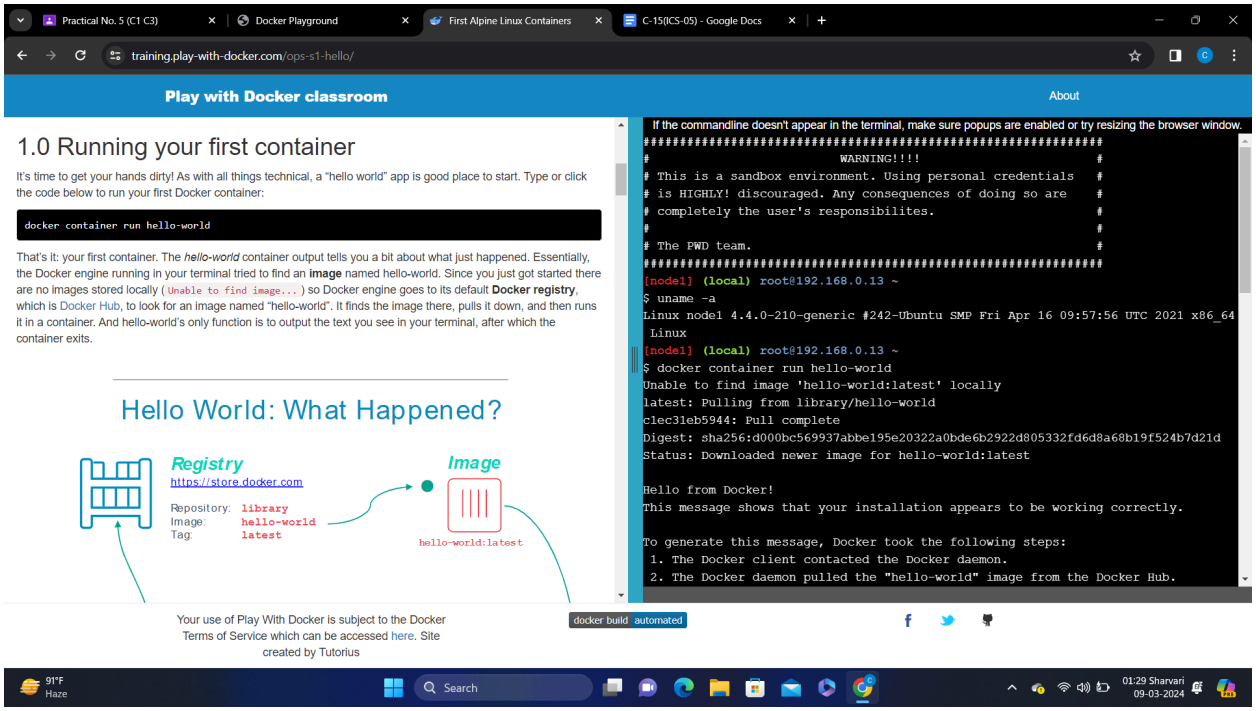
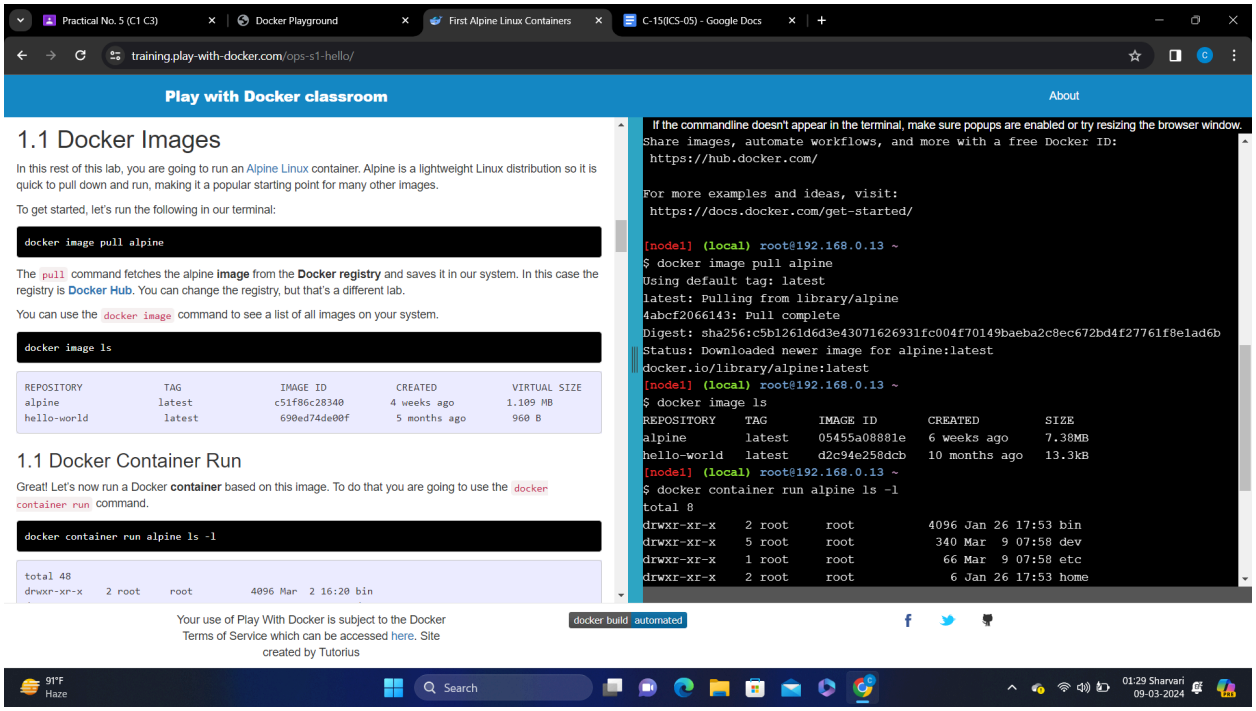
Implement container management with Kubernetes/Docker





Practical No. 5 (C1 C3) | Docker Playground | First Alpine Linux Containers | C-15(ICS-05) - Google Docs

training.play-with-docker.com/ops-s1-hello/

Play with Docker classroom About

1.1 Docker Container Run

Great! Let's now run a Docker **container** based on this image. To do that you are going to use the `docker container run` command.

```
docker container run alpine ls -l
```

```
total 48
drwxr-xr-x    2 root     root          4096 Mar  2 16:20 bin
drwxr-xr-x    5 root     root           360 Mar 18 09:47 dev
drwxr-xr-x   13 root     root          4096 Mar 18 09:47 etc
drwxr-xr-x    2 root     root          4096 Mar  2 16:20 home
drwxr-xr-x    5 root     root          4096 Mar  2 16:20 lib
......
......
```

While the output of the `ls` command may not be all that exciting, behind the scenes quite a few things just took place. When you call `run`, the Docker client finds the image (alpine in this case), creates the container and then runs a command in that container. When you run `docker container run alpine`, you provided a command (`ls -l`), so Docker executed this command inside the container for which you saw the directory listing. After the `ls` command finished, the container shut down.

---

docker run Details

If the commandline doesn't appear in the terminal, make sure popups are enabled or try resizing the browser window.

```
$ docker image ls
REPOSITORY    TAG       IMAGE ID       CREATED         SIZE
alpine        latest    05455a08881e   6 weeks ago     7.38MB
hello-world   latest    d2c94e258dcb   10 months ago   13.3kB
[node1] (local) root@192.168.0.13 ~
$ docker container run alpine ls -l
total 8
drwxr-xr-x    2 root     root          4096 Jan 26 17:53 bin
drwxr-xr-x    5 root     root           340 Mar  9 07:58 dev
drwxr-xr-x    1 root     root            66 Mar  9 07:58 etc
drwxr-xr-x    2 root     root             6 Jan 26 17:53 home
drwxr-xr-x    7 root     root           242 Jan 26 17:53 lib
drwxr-xr-x    5 root     root            44 Jan 26 17:53 media
drwxr-xr-x    2 root     root             6 Jan 26 17:53 mnt
drwxr-xr-x    2 root     root             6 Jan 26 17:53 opt
dr-xr-xr-x  873 root     root             0 Mar  9 07:58 proc
drwx------    2 root     root             6 Jan 26 17:53 root
drwxr-xr-x    2 root     root             6 Jan 26 17:53 run
drwxr-xr-x    2 root     root          4096 Jan 26 17:53 sbin
drwxr-xr-x    2 root     root             6 Jan 26 17:53 srv
dr-xr-xr-x   13 root     root             0 Mar  9 02:37 sys
drwxrwxrwt    2 root     root             6 Jan 26 17:53 tmp
drwxr-xr-x    7 root     root            66 Jan 26 17:53 usr
drwxr-xr-x   12 root     root           137 Jan 26 17:53 var
[node1] (local) root@192.168.0.13 ~
$
```

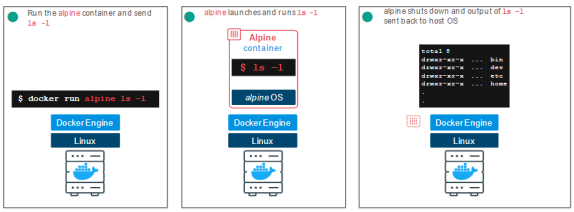

docker build automated

Practical No. 5 (C1 C3) | Docker Playground | First Alpine Linux Containers | C-15(ICS-05) - Google Docs

training.play-with-docker.com/ops-s1-hello/

Play with Docker classroom About

docker run Details



If the commandline doesn't appear in the terminal, make sure popups are enabled or try resizing the browser window.

```
hello-world   latest    d2c94e258dcb   10 months ago   13.3kB
[node1] (local) root@192.168.0.13 ~
$ docker container run alpine ls -l
total 8
drwxr-xr-x    2 root     root          4096 Jan 26 17:53 bin
drwxr-xr-x    5 root     root           340 Mar  9 07:58 dev
drwxr-xr-x    1 root     root            66 Mar  9 07:58 etc
drwxr-xr-x    2 root     root             6 Jan 26 17:53 home
drwxr-xr-x    7 root     root           242 Jan 26 17:53 lib
drwxr-xr-x    5 root     root            44 Jan 26 17:53 media
drwxr-xr-x    2 root     root             6 Jan 26 17:53 mnt
drwxr-xr-x    2 root     root             6 Jan 26 17:53 opt
dr-xr-xr-x  873 root     root             0 Mar  9 07:58 proc
drwx------    2 root     root             6 Jan 26 17:53 root
drwxr-xr-x    2 root     root             6 Jan 26 17:53 run
drwxr-xr-x    2 root     root          4096 Jan 26 17:53 sbin
drwxr-xr-x    2 root     root             6 Jan 26 17:53 srv
dr-xr-xr-x   13 root     root             0 Mar  9 02:37 sys
drwxrwxrwt    2 root     root             6 Jan 26 17:53 tmp
drwxr-xr-x    7 root     root            66 Jan 26 17:53 usr
drwxr-xr-x   12 root     root           137 Jan 26 17:53 var
[node1] (local) root@192.168.0.13 ~
$ docker container run alpine echo "hello from alpine"
hello from alpine
[node1] (local) root@192.168.0.13 ~
$
```

The fact that the container exited after running our command is important, as you will start to see. Let's try something more exciting. Type in the following:

```
docker container run alpine echo "hello from alpine"
```

And you should get the following output:

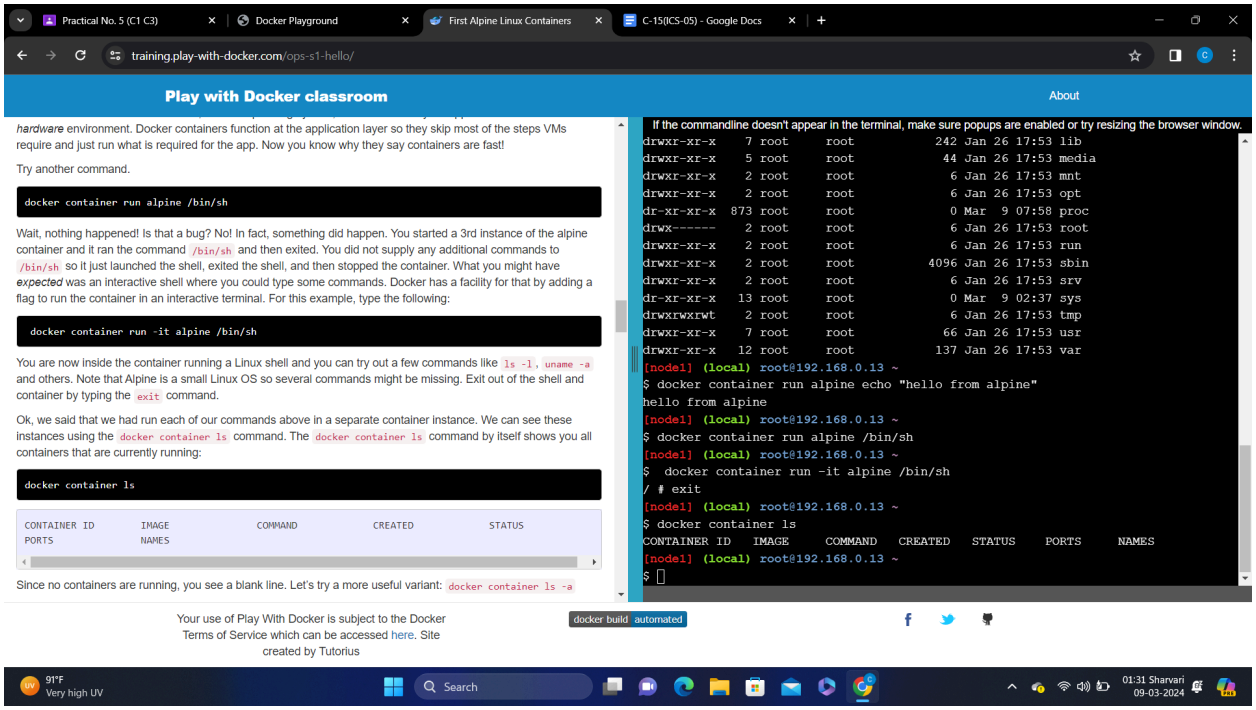
```
hello from alpine
```

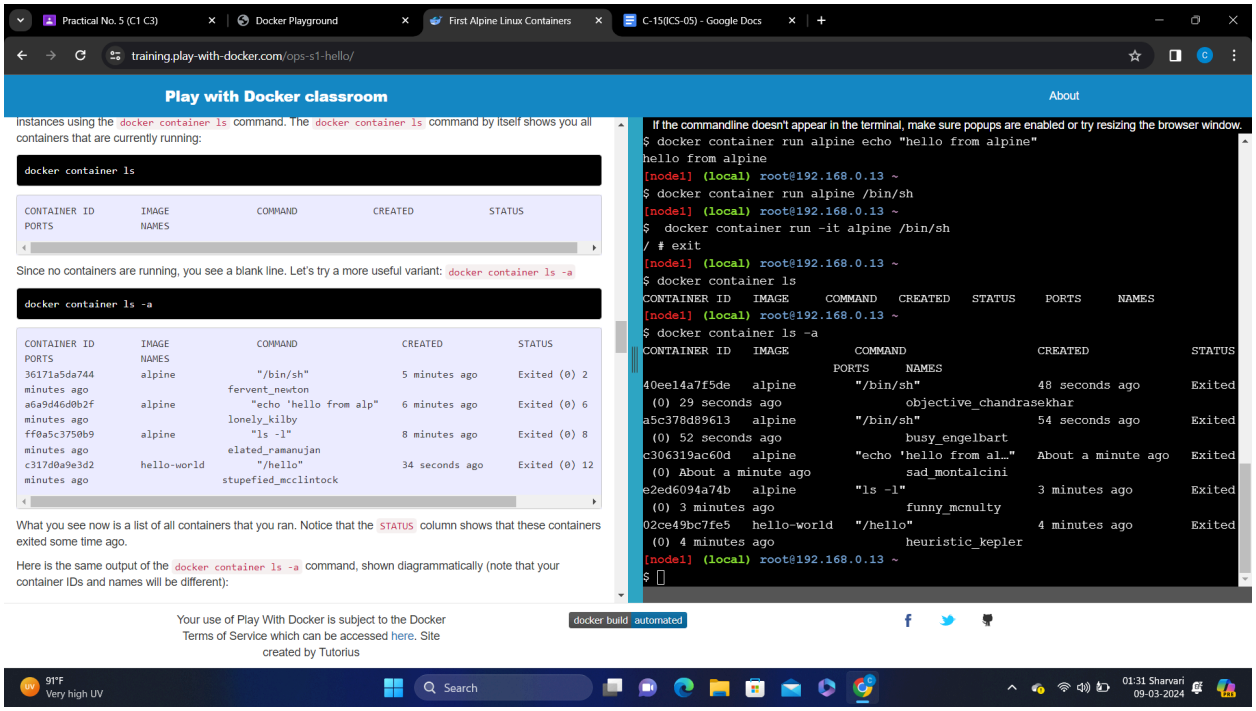
In this case, the Docker client dutifully ran the `echo` command inside our alpine container and then exited. If you noticed, all of that happened pretty quickly and again our container exited. As you will see in a few more steps, the `echo` command ran in a separate container instance. Imagine booting up a virtual machine (VM), running a

Your use of Play With Docker is subject to the Docker Terms of Service which can be accessed here. Site created by Tutorius

docker build automated

USD/JPY -0.65% | Search | 01:30 Sharvari 09-03-2024


Practical No. 5 (C1 C3)
Docker Playground
First Alpine Linux Containers
C-15(ICS-05) - Google Docs
training.play-with-docker.com/ops-s1-hello/
Play with Docker classroom
About
hardware environment. Docker containers function at the application layer so they skip most of the steps VMs require and just run what is required for the app. Now you know why they say containers are fast!
Try another command.
docker container run alpine /bin/sh
Wait, nothing happened! Is that a bug? No! In fact, something did happen. You started a 3rd instance of the alpine container and it ran the command /bin/sh and then exited. You did not supply any additional commands to /bin/sh so it just launched the shell, exited the shell, and then stopped the container. What you might have expected was an interactive shell where you could type some commands. Docker has a facility for that by adding a flag to run the container in an interactive terminal. For this example, type the following:
docker container run -it alpine /bin/sh
You are now inside the container running a Linux shell and you can try out a few commands like ls -l, uname -a and others. Note that Alpine is a small Linux OS so several commands might be missing. Exit out of the shell and container by typing the exit command.
Ok, we said that we had run each of our commands above in a separate container instance. We can see these instances using the docker container ls command. The docker container ls command by itself shows you all containers that are currently running:
docker container ls
CONTAINER ID IMAGE COMMAND CREATED STATUS
PORTS NAMES
Since no containers are running, you see a blank line. Let's try a more useful variant: docker container ls -a
If the commandline doesn't appear in the terminal, make sure popups are enabled or try resizing the browser window.
drwxr-xr-x 7 root root 242 Jan 26 17:53 lib
drwxr-xr-x 5 root root 44 Jan 26 17:53 media
drwxr-xr-x 2 root root 6 Jan 26 17:53 mnt
drwxr-xr-x 2 root root 6 Jan 26 17:53 opt
dr-xr-xr-x 873 root root 0 Mar 9 07:58 proc
drwx------ 2 root root 6 Jan 26 17:53 root
drwxr-xr-x 2 root root 6 Jan 26 17:53 run
drwxr-xr-x 2 root root 4096 Jan 26 17:53 sbin
drwxr-xr-x 2 root root 6 Jan 26 17:53 srv
dr-xr-xr-x 13 root root 0 Mar 9 02:37 sys
drwxrwxrwt 2 root root 6 Jan 26 17:53 tmp
drwxr-xr-x 7 root root 66 Jan 26 17:53 usr
drwxr-xr-x 12 root root 137 Jan 26 17:53 var
[node1] (local) root@192.168.0.13 ~
$ docker container run alpine echo "hello from alpine"
hello from alpine
[node1] (local) root@192.168.0.13 ~
$ docker container run alpine /bin/sh
[node1] (local) root@192.168.0.13 ~
$ docker container run -it alpine /bin/sh
/ # exit
[node1] (local) root@192.168.0.13 ~
$ docker container ls
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
[node1] (local) root@192.168.0.13 ~
$
Your use of Play With Docker is subject to the Docker Terms of Service which can be accessed here. Site created by Tutorius
docker build automated
91°F Very high UV
Search
01:31 Sharvari 09-03-2024


Practical No. 5 (C1 C3)
Docker Playground
First Alpine Linux Containers
C-15(ICS-05) - Google Docs
training.play-with-docker.com/ops-s1-hello/
Play with Docker classroom
About
instances using the docker container ls command. The docker container ls command by itself shows you all containers that are currently running:
docker container ls
CONTAINER ID IMAGE COMMAND CREATED STATUS
PORTS NAMES
Since no containers are running, you see a blank line. Let's try a more useful variant: docker container ls -a
docker container ls -a
CONTAINER ID IMAGE COMMAND CREATED STATUS
PORTS NAMES
36171a5da744 alpine "/bin/sh" 5 minutes ago Exited (0) 2 minutes ago fervent_newton
a6a9d46d0b2f alpine "echo 'hello from alp" 6 minutes ago Exited (0) 6 minutes ago lonely_kilby
ff0a5c3750b9 alpine "ls -l" 8 minutes ago Exited (0) 8 minutes ago elated_ramanujan
c317d0a9e3d2 hello-world "/hello" 34 seconds ago Exited (0) 12 minutes ago stupefied_mcclintock
What you see now is a list of all containers that you ran. Notice that the STATUS column shows that these containers exited some time ago.
Here is the same output of the docker container ls -a command, shown diagrammatically (note that your container IDs and names will be different):
$ docker container run alpine echo "hello from alpine"
hello from alpine
[node1] (local) root@192.168.0.13 ~
$ docker container run alpine /bin/sh
[node1] (local) root@192.168.0.13 ~
$ docker container run -it alpine /bin/sh
/ # exit
[node1] (local) root@192.168.0.13 ~
$ docker container ls
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
[node1] (local) root@192.168.0.13 ~
$ docker container ls -a
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
40ee14a7f5de alpine "/bin/sh" 48 seconds ago Exited (0) 29 seconds ago objective_chandrasekhar
a5c378d89613 alpine "/bin/sh" 54 seconds ago Exited (0) 52 seconds ago busy_engelbart
c306319ac60d alpine "echo 'hello from al…" About a minute ago Exited (0) About a minute ago sad_montalcini
e2ed6094a74b alpine "ls -l" 3 minutes ago Exited (0) 3 minutes ago funny_mcnulty
02ce49bc7fe5 hello-world "/hello" 4 minutes ago Exited (0) 4 minutes ago heuristic_kepler
[node1] (local) root@192.168.0.13 ~
$
Your use of Play With Docker is subject to the Docker Terms of Service which can be accessed here. Site created by Tutorius
docker build automated
91°F Very high UV
Search
01:31 Sharvari 09-03-2024

Practical No. 5 (C1 C3) | Docker Playground | First Alpine Linux Containers | C-15(ICS-05) - Google Docs

training.play-with-docker.com/ops-s1-hello/

Play with Docker classroom About

1.2 Container Isolation

In the steps above we ran several commands via container instances with the help of `docker container run`. The `docker container ls -a` command showed us that there were several containers listed. Why are there so many containers listed if they are all from the *alpine* image?

This is a critical security concept in the world of Docker containers! Even though each `docker container run` command used the same alpine ***image***, each execution was a separate, isolated ***container***. Each container has a separate filesystem and runs in a different namespace; by default a container has no way of interacting with other containers, even those from the same image. Let's try another exercise to learn more about isolation.

```
docker container run -it alpine /bin/ash
```

The `/bin/ash` is another type of shell available in the alpine image. Once the container launches and you are at the container's command prompt type the following commands:

```
echo "hello world" > hello.txt

ls
```

The first `echo` command creates a file called "hello.txt" with the words "hello world" inside it. The second command gives you a directory listing of the files and should show your newly created "hello.txt" file. Now type `exit` to leave this container.

To show how isolation works, run the following:

```
docker container run alpine ls
```

It is the same `ls` command we used inside the container's interactive ash shell, but this time, did you notice that your "hello.txt" file is missing? That's isolation! Your command ran in a new and separate *instance*, even though it is based on the same *image*. The 2nd instance has no way of interacting with the 1st instance because the Docker

If the commandline doesn't appear in the terminal, make sure popups are enabled or try resizing the browser window.

```
[node1] (local) root@192.168.0.13 ~
$ docker container run -it alpine /bin/ash
/ # echo "hello world" > hello.txt
/ # ls
bin         hello.txt   media       proc        sbin        tmp
dev         home        mnt         root        srv         usr
etc         lib         opt         run         sys         var
/ # docker container run alpine ls
/bin/ash: docker: not found
/ # exit
[node1] (local) root@192.168.0.13 ~
$ docker container run alpine ls
bin
dev
etc
home
lib
media
mnt
opt
proc
root
run
sbin
srv
sys
```


Practical No. 5 (C1 C3) | Docker Playground | First Alpine Linux Containers | C-15(ICS-05) - Google Docs

training.play-with-docker.com/ops-s1-hello/

Play with Docker classroom About

users take their changes and move them up to production using this basic concept and the built-in capabilities of Docker Enteprise. We will explore more of that in later exercises.

Right now, the obvious question is "how do I get back to the container that has my 'hello.txt' file?"

Once again run the

```
docker container ls -a
```

command again and you should see output similar to the following:

```
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS
PORTS               NAMES
36171a5da744        alpine              "ls"                     2 minutes ago       Exited (0) 2
minutes ago                       distracted_bhaskara
3030c9c91e12        alpine              "/bin/ash"               5 minutes ago       Exited (0) 2
minutes ago                       fervent_newton
a6a9d46d0b2f        alpine              "echo 'hello from alp"   6 minutes ago       Exited (0) 6
minutes ago                       lonely_kilby
ff0a5c3750b9        alpine              "ls -l"                  8 minutes ago       Exited (0) 8
minutes ago                       elated_ramanujan
c317d0a9e3d2        hello-world         "/hello"                 34 seconds ago      Exited (0) 12
minutes ago                     stupefied_mcclintock
```

Graphically this is what happened on our Docker Engine:

If the commandline doesn't appear in the terminal, make sure popups are enabled or try resizing the browser window.

```
sbin
srv
sys
tmp
usr
var
[node1] (local) root@192.168.0.13 ~
$ docker container ls -a
CONTAINER ID   IMAGE         COMMAND                  CREATED          STATUS
                        PORTS     NAMES
426b2bf7b936   alpine        "ls"                     25 seconds ago   Exited (0)
 23 seconds ago                   hopeful_greider
82a6be230238   alpine        "/bin/ash"               2 minutes ago    Exited (12
7) 27 seconds ago                 vigorous_antonelli
40ee14a7f5de   alpine        "/bin/sh"                3 minutes ago    Exited (0)
 3 minutes ago                    objective_chandrasekhar
a5c378d89613   alpine        "/bin/sh"                3 minutes ago    Exited (0)
 3 minutes ago                    busy_engelbart
c306319ac60d   alpine        "echo 'hello from al…"   4 minutes ago    Exited (0)
 4 minutes ago                    sad_montalcini
e2ed6094a74b   alpine        "ls -l"                  5 minutes ago    Exited (0)
 5 minutes ago                    funny_mcnulty
02ce49bc7fe5   hello-world   "/hello"                 7 minutes ago    Exited (0)
 7 minutes ago                    heuristic_kepler
[node1] (local) root@192.168.0.13 ~
$
```

Practical No. 5 (C1 C3) | Docker Playground | First Alpine Linux Containers | C-15(ICS-05) - Google Docs

training.play-with-docker.com/ops-s1-hello/

Play with Docker classroom About

The container in which we created the "hello.txt" file is the same one where we used the `/bin/ash` shell, which we can see listed in the "COMMAND" column. The *Container ID* number from the first column uniquely identifies that particular container instance. In the sample output above the container ID is `3030c9c91e12`. We can use a slightly different command to tell Docker to run this specific container instance. Try typing:

```
docker container start <container ID>
```

- **Pro tip:** Instead of using the full container ID you can use just the first few characters, as long as they are enough to uniquely ID a container. So we could simply use "3030" to identify the container instance in the example above, since no other containers in this list start with these characters.

Now use the `docker container ls` command again to list the running containers.

```
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS
PORTS               NAMES
3030c9c91e12        alpine              "/bin/ash"          2 minutes ago       Up 14 second
s                   distracted_bhaskara
```

Notice this time that our container instance is still running. We used the ash shell this time so the rather than simply exiting the way /bin/sh did earlier, ash waits for a command. We can send a command in to the container to run by using the `exec` command, as follows:

```
docker container exec <container ID> ls
```

This time we get a directory listing and it shows our "hello.txt" file because we used the container instance where we created that file.

If the commandline doesn't appear in the terminal, make sure popups are enabled or try resizing the browser window.

```
[node1] (local) root@192.168.0.13 ~
$ docker container start <82a6be230238>
bash: syntax error near unexpected token `newline'
[node1] (local) root@192.168.0.13 ~
$ docker container start 82a6be230238
82a6be230238
[node1] (local) root@192.168.0.13 ~
$ docker container ls
CONTAINER ID   IMAGE     COMMAND      CREATED         STATUS         PORTS     NA
MES
82a6be230238   alpine    "/bin/ash"   8 minutes ago   Up 9 seconds             vi
gorous_antonelli
[node1] (local) root@192.168.0.13 ~
$ docker container exec 82a6be230238 ls
bin
dev
etc
hello.txt
home
lib
media
mnt
opt
proc
root
run
```


Practical No. 5 (C1 C3) | Docker Playground | First Alpine Linux Containers | C-15(ICS-05) - Google Docs

training.play-with-docker.com/ops-s1-hello/

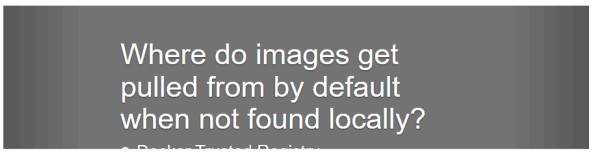
Play with Docker classroom About

create larger, more complex images in a simple, automated manner.

1.3 Terminology

In the last section, you saw a lot of Docker-specific jargon which might be confusing to some. So before you go further, let's clarify some terminology that is used frequently in the Docker ecosystem.

- *Images* - The file system and configuration of our application which are used to create containers. To find out more about a Docker image, run `docker image inspect alpine`. In the demo above, you used the `docker image pull` command to download the **alpine** image. When you executed the command `docker container run hello-world`, it also did a `docker image pull` behind the scenes to download the **hello-world** image.
- *Containers* - Running instances of Docker images — containers run the actual applications. A container includes an application and all of its dependencies. It shares the kernel with other containers, and runs as an isolated process in user space on the host OS. You created a container using `docker run` which you did using the alpine image that you downloaded. A list of running containers can be seen using the `docker container ls` command.
- *Docker daemon* - The background service running on the host that manages building, running and distributing Docker containers.
- *Docker client* - The command line tool that allows the user to interact with the Docker daemon.
- *Docker Hub* - Store is, among other things, a registry of Docker images. You can think of the registry as a directory of all available Docker images. You'll be using this later in this tutorial.



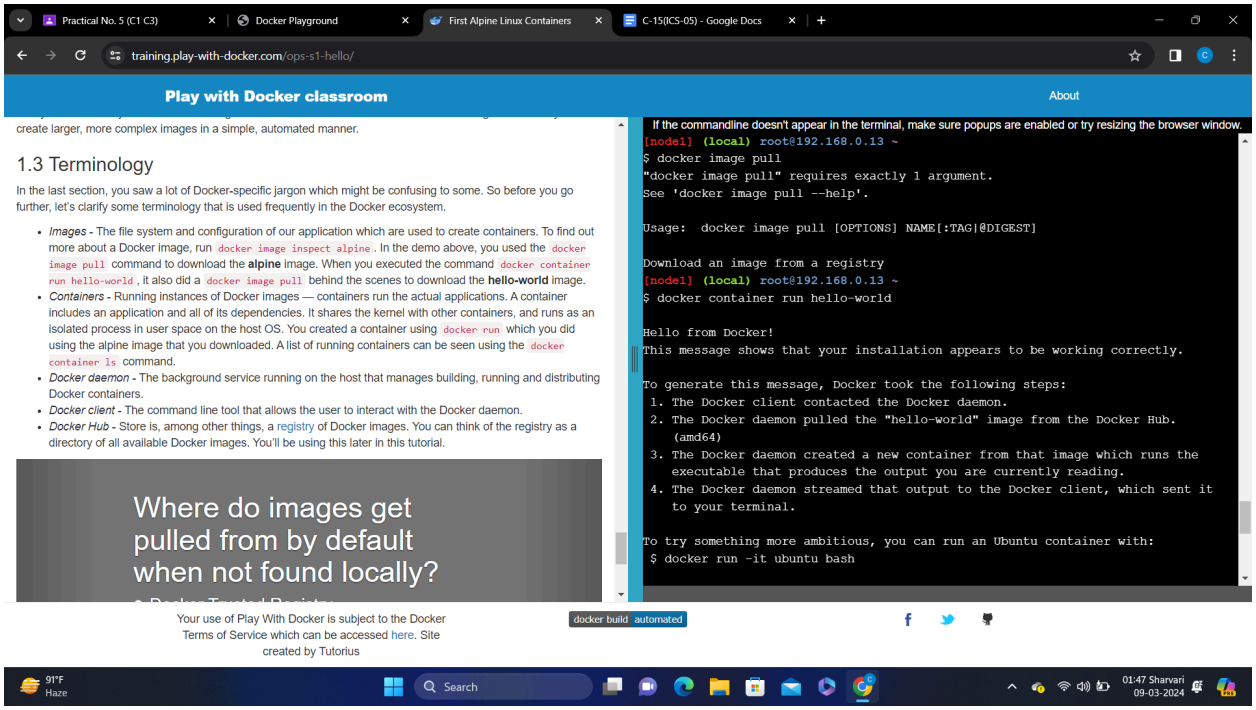
If the commandline doesn't appear in the terminal, make sure popups are enabled or try resizing the browser window.

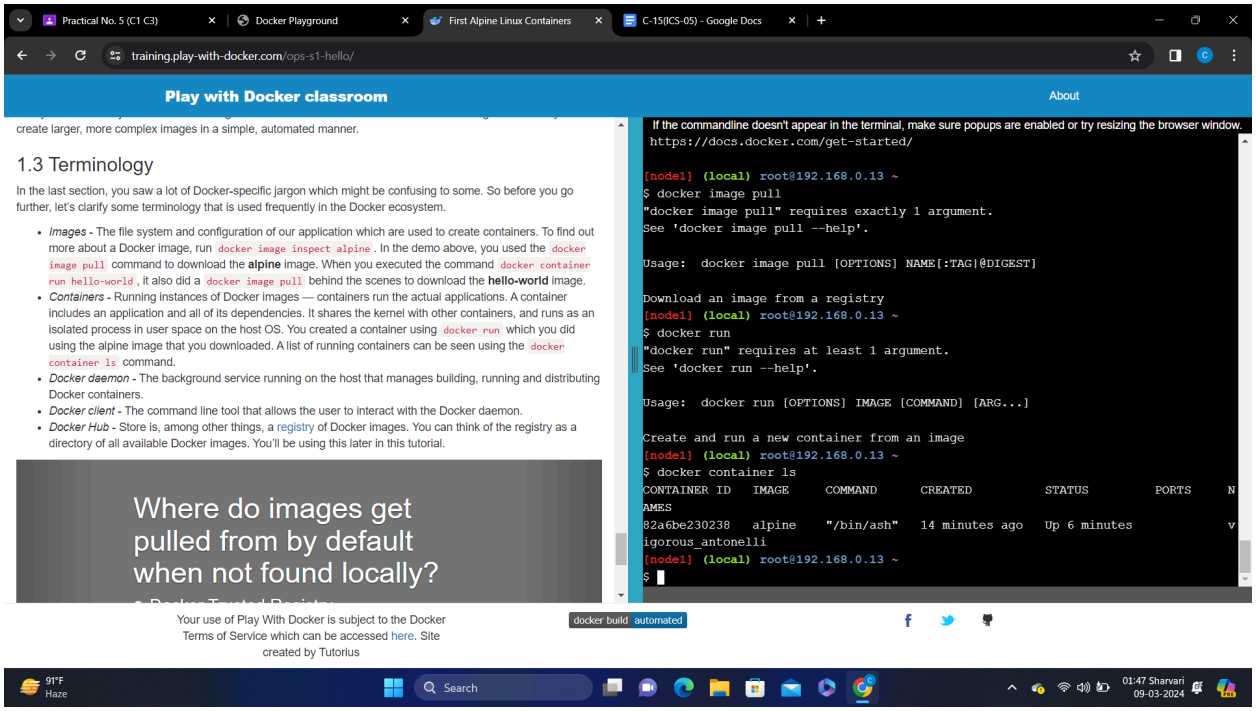
```
[node1] (local) root@192.168.0.13 ~
$ docker image inspect alpine
[
    {
        "Id": "sha256:05455a08881ea9cf0e752bc48e61bbd71a34c029bb13df01e40e3e70e0d
007bd",
        "RepoTags": [
            "alpine:latest"
        ],
        "RepoDigests": [
            "alpine@sha256:c5b1261d6d3e43071626931fc004f70149baeba2c8ec672bd4f277
61f8e1ad6b"
        ],
        "Parent": "",
        "Comment": "",
        "Created": "2024-01-27T00:30:48.743965523Z",
        "Container": "4189cbc534955765760c227f328ec1cdd52e8550681c2bf9f8f990b27b6
44f9c",
        "ContainerConfig": {
            "Hostname": "4189cbc53495",
            "Domainname": "",
            "User": "",
            "AttachStdin": false,
            "AttachStdout": false,
            "AttachStderr": false,
            "Tty": false,
```

Your use of Play With Docker is subject to the Docker Terms of Service which can be accessed here. Site created by Tutorius

docker build automated

91°F Haze | Search | 01:47 Sharvari 09-03-2024


Practical No. 5 (C1 C3)
Docker Playground
First Alpine Linux Containers
C-15(ICS-05) - Google Docs
training.play-with-docker.com/ops-s1-hello/
Play with Docker classroom
About
create larger, more complex images in a simple, automated manner.
1.3 Terminology
In the last section, you saw a lot of Docker-specific jargon which might be confusing to some. So before you go further, let's clarify some terminology that is used frequently in the Docker ecosystem.
Images - The file system and configuration of our application which are used to create containers. To find out more about a Docker image, run docker image inspect alpine. In the demo above, you used the docker image pull command to download the alpine image. When you executed the command docker container run hello-world, it also did a docker image pull behind the scenes to download the hello-world image.
Containers - Running instances of Docker images — containers run the actual applications. A container includes an application and all of its dependencies. It shares the kernel with other containers, and runs as an isolated process in user space on the host OS. You created a container using docker run which you did using the alpine image that you downloaded. A list of running containers can be seen using the docker container ls command.
Docker daemon - The background service running on the host that manages building, running and distributing Docker containers.
Docker client - The command line tool that allows the user to interact with the Docker daemon.
Docker Hub - Store is, among other things, a registry of Docker images. You can think of the registry as a directory of all available Docker images. You'll be using this later in this tutorial.
Where do images get pulled from by default when not found locally?
If the commandline doesn't appear in the terminal, make sure popups are enabled or try resizing the browser window.
[node1] (local) root@192.168.0.13 ~
$ docker image pull
"docker image pull" requires exactly 1 argument.
See 'docker image pull --help'.
Usage: docker image pull [OPTIONS] NAME[:TAG|@DIGEST]
Download an image from a registry
[node1] (local) root@192.168.0.13 ~
$ docker container run hello-world
Hello from Docker!
This message shows that your installation appears to be working correctly.
To generate this message, Docker took the following steps:
1. The Docker client contacted the Docker daemon.
2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
(amd64)
3. The Docker daemon created a new container from that image which runs the executable that produces the output you are currently reading.
4. The Docker daemon streamed that output to the Docker client, which sent it to your terminal.
To try something more ambitious, you can run an Ubuntu container with:
$ docker run -it ubuntu bash
Your use of Play With Docker is subject to the Docker Terms of Service which can be accessed here. Site created by Tutorius
docker build automated
91°F Haze
Search
01:47 Sharvari 09-03-2024


Practical No. 5 (C1 C3)
Docker Playground
First Alpine Linux Containers
C-15(ICS-05) - Google Docs
training.play-with-docker.com/ops-s1-hello/
Play with Docker classroom
About
create larger, more complex images in a simple, automated manner.
1.3 Terminology
In the last section, you saw a lot of Docker-specific jargon which might be confusing to some. So before you go further, let's clarify some terminology that is used frequently in the Docker ecosystem.
Images - The file system and configuration of our application which are used to create containers. To find out more about a Docker image, run docker image inspect alpine. In the demo above, you used the docker image pull command to download the alpine image. When you executed the command docker container run hello-world, it also did a docker image pull behind the scenes to download the hello-world image.
Containers - Running instances of Docker images — containers run the actual applications. A container includes an application and all of its dependencies. It shares the kernel with other containers, and runs as an isolated process in user space on the host OS. You created a container using docker run which you did using the alpine image that you downloaded. A list of running containers can be seen using the docker container ls command.
Docker daemon - The background service running on the host that manages building, running and distributing Docker containers.
Docker client - The command line tool that allows the user to interact with the Docker daemon.
Docker Hub - Store is, among other things, a registry of Docker images. You can think of the registry as a directory of all available Docker images. You'll be using this later in this tutorial.
Where do images get pulled from by default when not found locally?
If the commandline doesn't appear in the terminal, make sure popups are enabled or try resizing the browser window.
https://docs.docker.com/get-started/
[node1] (local) root@192.168.0.13 ~
$ docker image pull
"docker image pull" requires exactly 1 argument.
See 'docker image pull --help'.
Usage: docker image pull [OPTIONS] NAME[:TAG|@DIGEST]
Download an image from a registry
[node1] (local) root@192.168.0.13 ~
$ docker run
"docker run" requires at least 1 argument.
See 'docker run --help'.
Usage: docker run [OPTIONS] IMAGE [COMMAND] [ARG...]
Create and run a new container from an image
[node1] (local) root@192.168.0.13 ~
$ docker container ls
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS N
AMES
82a6be230238 alpine "/bin/ash" 14 minutes ago Up 6 minutes v
igorous_antonelli
[node1] (local) root@192.168.0.13 ~
$
Your use of Play With Docker is subject to the Docker Terms of Service which can be accessed here. Site created by Tutorius
docker build automated
91°F Haze
Search
01:47 Sharvari 09-03-2024