

# DevOps Katas

Hands-On DevOps

Dave Swersky

# **DevOps Katas**

Hands-On DevOps

Dave Swersky

© 2017 Dave Swersky

# Contents

<b>Chapter 5: Docker Katas</b> . . . . .	<b>1</b>
Docker Kata 1: The Basic Commands . . . . .	1
Step 1: Running Your First Container . . . . .	1
Step 2: Listing Containers . . . . .	3
Step 3: Listing Images . . . . .	4
Step 4: Running a Named Container . . . . .	5
Step 5: Run a Container in Interactive Mode . . . . .	7
Step 6: Remove all Containers and Images . . . . .	10
Docker Kata 4: Running a Web Server in a Container . . . . .	12
Step 1: Run a Web Server . . . . .	12
Step 2: Run a Second Webserver on a Different Port . . . . .	15
Docker Kata 6: Creating Docker Images . . . . .	17
Step 1: Creating an Image From a Modified Container . . . . .	17
Step 2: Create an Image with a Dockerfile . . . . .	24
<b>Docker Kata Practice Page</b> . . . . .	<b>31</b>
Kata 1: Basic Commands . . . . .	31
Kata 4: Running a Web Server in a Container . . . . .	33
Kata 6: Creating Docker Images . . . . .	34



# Chapter 5: Docker Katas

## Docker Kata 1: The Basic Commands

Let's get started with the first Docker kata. Each kata will be broken into steps. Below each command is a sample of the output that you should see when you execute the command. Each step includes, after the commands, a breakdown of the commands and a summary of what they do.

Docker Kata 1 will introduce you to the basics: working with *containers* and *images*.

Open the Ubuntu GNOME Terminal from the launcher:



GNOME Terminal

## Step 1: Running Your First Container

**RUN:**

```
docker container run hello-world
```

```

devops@DevOpsKatas:~$ docker container run hello-world
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
78445dd45222: Pull complete
Digest: sha256:c5515758d4c5e1e838e9cd307f6c6a0d620b5e07e6f927b07d05f6d12a1ac8d7
Status: Downloaded newer image for hello-world:latest

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.
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

Share images, automate workflows, and more with a free Docker ID:
https://cloud.docker.com/

For more examples and ideas, visit:
https://docs.docker.com/engine/userguide/

devops@DevOpsKatas:~$ █

```

### Your first container

Parameter	Description
<code>docker container</code>	The <i>parent command</i> . Parent commands act on Docker objects such as containers, images, networks, etc.
<code>run</code>	runs an <i>image</i> , creating an instance of a <i>container</i> .
<code>hello-world</code>	<code>hello-world</code> is an <i>image</i> . The <code>docker run hello-world</code> command in Step 1 ran the <code>hello-world</code> image, and an instance of a <i>container</i> was created.

## Command Summary

This first command runs a container from an image.

A Docker *image* is a blueprint for a *container*. Containers are running instances of an image. The difference between an image and a container is the same as the difference between an executable (such as Notepad.exe,) and the running program. Just as multiple instances of Notepad (and most other programs) can run side-by-side, so can multiple containers be run at the same time, from the same image.

Note the first line of the output:

```
Unable to find image 'hello-world:latest' locally
```

Docker will first check the local image repository for the image indicated in the command. If the image is not found, it will then check **Docker Hub**. Docker Hub is a public repository of images maintained by Docker Inc. The `hello-world` image is a test image stored on Docker Hub <http://hub.docker.com>.

Image definitions include a “start” command. The start command runs a process inside the container when an image is run (we’ll work with that in later katas.) The start command of the `hello-world` image just echoes the text you see in the Output window.

## Step 2: Listing Containers

**RUN:**

```
docker container ls
```

```
devops@DevOpsKatas:~$ docker container ls
CONTAINER ID   IMAGE     COMMAND                  CREATED        STATUS        PORTS        NAMES
```

### List Running Containers

Parameter	Description
<code>docker container</code>	The parent command.
<code>ls</code>	Lists running containers.

## Command Summary

The `docker container ls` command lists containers. The first execution, however, returns an empty list. We ran the `hello-world` container in the first step, so why is the list empty?

The list is empty because `docker container ls` only lists *running* containers. Containers run as long as their start process runs. If that process exits, the container exits. The start command for the `hello-world` image echoes the output from the first step, then exits immediately. The `docker container ls` command returns an empty list because no containers are running.

**RUN:**

```
docker container ls -a
```

```
devops@DevOpsKatas:~$ docker container ls -a
CONTAINER ID   IMAGE          COMMAND                  CREATED        STATUS        PORTS          NAMES
4cd0fb9acbia   hello-world    "/hello"                About an hour ago    Exited (0)    About an hour ago    silly_brattain
devops@DevOpsKatas:~$
```

#### List All Containers

Parameter	Description
<code>docker container</code>	The parent command.
<code>ls</code>	Lists running containers.
<code>-a</code>	Lists <i>all</i> containers.

## Command Summary

The second command includes the `-a` parameter, which lists *all* images, including those that have started and exited.

## Step 3: Listing Images

### RUN:

```
docker image ls
```

```
devops@DevOpsKatas:~$ docker image ls
REPOSITORY    TAG       IMAGE ID       CREATED        SIZE
hello-world   latest    48b5124b2768   2 months ago   1.84 kB
devops@DevOpsKatas:~$
```

#### List Images

Parameter	Description
<code>docker image</code>	The parent command.
<code>ls</code>	Lists all images.

## Command Summary

This command lists all images in the local image repository. That will include all images you've downloaded from Docker Hub, and any you've created yourself (we'll be doing that!)



## Step 4: Running a Named Container

### **RUN:**

```
docker container run --name my_container hello-world
```

```
devops@DevOpsKatas:~$ docker container run --name my_container 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.
```

#### Named Container

Parameter	Description
docker container	The parent command.
run	Runs a container.
--name	Assigns a name to a container.
my_container	The name assigned to the container.
hello-world	The name of the image to run.

## Command Summary

When running a container, you do not have to specify a name. The Docker Engine will assign a random one. However, naming containers makes them easier to keep track of when running many at one time. The name assigned with the `--name` parameter will appear in the container list. That name can then be used to refer to the container in other commands, such as `inspect`.

### **RUN:**

```
docker container ls -a
```

```
devops@DevOpsKatas:~$ docker container ls -a
CONTAINER ID   IMAGE          COMMAND                  CREATED        STATUS              PORTS          NAMES
d8e39fe6830d   hello-world    "/hello"                2 minutes ago   Exited (0) 2 minutes ago           my_container
4cd0fb9acb1a   hello-world    "/hello"                About an hour ago   Exited (0) About an hour ago           silly_brattain
devops@DevOpsKatas:~$
```

#### List All Containers

Parameter	Description
<code>docker container</code>	The parent command.
<code>ls</code>	Lists containers.
<code>-a</code>	Lists <i>all</i> containers, both running and exited.

## Command Summary

Lists all containers, including running and exited containers.

### **RUN:**

```
docker container inspect my_container
```

```
devops@DevOpsKatas:~$ docker container inspect my_container
[
  {
    "Id": "d8e39fe6830dab4d5f4ac35bc78e8869b246705a58dbba24296f94f1fd278c30",
    "Created": "2017-03-19T19:17:39.276519532Z",
    "Path": "/hello",
    "Args": [],
    "State": {
      "Status": "exited",
      "Running": false,
      "Paused": false,
      "Restarting": false,
      "OOMKilled": false,
      "Dead": false,
      "Pid": 0,
      "ExitCode": 0,
      "Error": "",
      "StartedAt": "2017-03-19T19:17:39.587859806Z",
      "FinishedAt": "2017-03-19T19:17:39.610658653Z"
    },
    "Image": "sha256:48b5124b2768d2b917edcb640435044a97967015485e812545546cbed5cf0233",
    "ResolvConfPath": "/var/lib/docker/containers/d8e39fe6830dab4d5f4ac35bc78e8869b246705a58dbba24296f94f1fd278c30/resolv.conf",
    "HostnamePath": "/var/lib/docker/containers/d8e39fe6830dab4d5f4ac35bc78e8869b246705a58dbba24296f94f1fd278c30/hostname",
    "HostsPath": "/var/lib/docker/containers/d8e39fe6830dab4d5f4ac35bc78e8869b246705a58dbba24296f94f1fd278c30/hosts",
    "LogPath": "/var/lib/docker/containers/d8e39fe6830dab4d5f4ac35bc78e8869b246705a58dbba24296f94f1fd278c30/json.log",
    "Name": "/my_container",
    "RestartCount": 0,
    "ContainerIDFile": "/var/lib/docker/containers/d8e39fe6830dab4d5f4ac35bc78e8869b246705a58dbba24296f94f1fd278c30/container-id-file"
  }
]
```

### Inspect a Container

Parameter	Description
<code>docker container</code>	The parent command.
<code>inspect</code>	Returns detailed information about an object.
<code>my_container</code>	The name of the container to inspect.

## Command Summary

The `inspect` command works on several object types, including containers. When executed on a container object, `inspect` returns JSON-formatted data that describes the container.

You can also use a container ID to refer to a container when executing a command. The ID is a long string of characters. The first twelve of those characters is listed in the ID column:

```
devops@DevOpsKatas:~$ docker container ls -a
```

CONTAINER ID	IMAGE	COMMAND	CREATED
c91a3a7c8134	hello-world	<code>"/hello"</code>	39 seconds ago
42eddbc93b81	hello-world	<code>"/hello"</code>	37 minutes ago

You can use the first three or more characters to refer to a container (and other objects with IDs):

```
devops@DevOpsKatas:~$ docker inspect 42e
```

```
[
  {
    "Id": "42eddbc93b81ca7c61c9abb385451d673e4e08293086966fa836cf6e048a145",
    "Created": "2017-03-09T02:02:48.196355507Z",
    "Path": "/hello",
    "Args": [],
    ...
  }
]
```

## Step 5: Run a Container in Interactive Mode

**RUN:**

```
docker container run -it ubuntu bash
```

```
devops@DevOpsKatas:~$ docker container run -it ubuntu bash
Unable to find image 'ubuntu:latest' locally
latest: Pulling from library/ubuntu
d54efb8db41d: Pull complete
f8b845f45a87: Pull complete
e8db7bf7c39f: Pull complete
9654c40e9079: Pull complete
6d9ef359eaaa: Pull complete
Digest: sha256:dd7808d8792c9841d0b460122f1acf0a2dd1f56404f8d1e56298048885e45535
Status: Downloaded newer image for ubuntu:latest
root@b0d82f66a618:/#
```

**Run Container Interactively**

Parameter	Description
<code>docker container</code>	The parent command.
<code>run</code>	Runs a container.
<code>-it</code>	represents two separate parameters: <code>-i</code> runs the container <i>interactively</i> . <code>-t</code> allocates a “psuedo-tty.” This is a terminal-like interface between two processes. Single-character parameters can be combined, prepended by a single dash. Together, the <code>-it</code> parameters allow interactive connection to a container.
<code>ubuntu</code>	An image based on the Ubuntu operating system.
<code>bash</code>	The parameter after the image name ( <code>ubuntu</code> ) is a command to run within the container. In this case, the command is <code>bash</code> .

## Command Summary

Running a container interactively allows a user to connect to the container in a “live” fashion, using a text console interface. This is similar to using SSH (a secure terminal interface) to connect to a remote server. A container that is run interactively behaves much like any other remote server. Administrators can view files in the filesystem, run programs, and install components to the container. This mechanism is useful for debugging, testing, and defining new images.

If you look closely at the prompt in the `bash` shell console, you’ll see this:

```
root@e8fedb60e8e6: /#
```

This is similar to the prompt you see in the *DevOps Katas* Learning VM terminal window:

```
devops@DevOpsKatas: ~$
```

The first part before the `@` sign is the logged-on user. You connect as the `root` user to the container, which is the administrative ‘SuperUser’ of a Unix OS. The second part after the `@` sign is the computer name. The name of the container is its Container ID:

```
devops@DevOpsKatas:~$ docker container ls -a
CONTAINER ID        IMAGE               COMMAND             CREATED
STATUS
e8fedb60e8e6       ubuntu             "bash"             2 days ago
Exited (130) 21 minutes ago
```

**RUN:**

```
ls
```

```
root@b0d82f66a618:/# ls
bin boot dev etc home lib lib64 media mnt opt proc root run sbin srv sys tmp usr var
root@b0d82f66a618:/#
```

List Container Files

Command	Description
ls	Lists the files and directories in the container.

Command Summary

The `ls` command Lists the files in the container filesystem.

**RUN:**

```
ls /bin
```

```
root@b0d82f66a618:/# ls /bin
bash dd false journalctl mknod pidof sed systemd
cat df fgrep kill mktemp ps sh systemd-ask-password
chgrp dir findmnt ln more sh.distrib systemd-escape
chmod dmesg grep login mount rbash sleep systemd-inhibit
chown dnsdomainname gunzip loginctl mountpoint readlink stty systemd-machine-id-se
cp domainname gzexe ls mv rm su systemd-notify
dash echo gzip lsblk networkctl rmdir sync systemd-tmpfiles
date egrep hostname mkdir nisdomainname run-parts systemctl systemd-tty-ask-passw
root@b0d82f66a618:/#
```

List Bin Directory

Command	Description
ls /bin	Lists the files and directories in /bin directory of the container. The files displayed here are Linux commands and programs.

Command Summary

The `ls /bin` command Lists the files in the `bin` folder.

**RUN:**

```
exit
```

Command	Description
<code>exit</code>	Closes the pseudo-tty connection. This also terminates the <code>bash</code> process started when the container was run, so the container exits as well.

**Command Summary**

The `exit` command exits the Bash shell running in the container.

**Step 6: Remove all Containers and Images****RUN:**

```
docker container rm $(docker container ls -a -q)
```

```
devops@DevOpsKatas:~$ docker container rm $(docker container ls -a -q)
b0d82f66a618
d8e39fe6830d
4cd0fb9acb1a
devops@DevOpsKatas:~$
```

**Remove All Containers**

Parameter	Description
<code>docker container</code>	The parent command.
<code>rm</code>	Removes an object type- in this case a <code>docker container</code> .
<code>\$(docker container ls -a -q)</code>	<p>This is a Unix shell <i>command substitution</i>. The output of this “subcommand” will be “fed” to the <code>docker container rm</code> command.</p> <p><code>ls</code> lists containers.  <code>-a</code> lists all containers.  <code>-q</code> is “quiet mode,” which returns only container IDs.</p> <p>The entire command runs <code>docker container rm</code> once for each output of <code>docker container ls -a -q</code>. The effect is that all containers on the system are removed.</p>

## Command Summary

This is a cleanup step that will be executed between some katas and steps to clear all images and containers. These would *not* be a commonly run in a real production environment, given the potential to destroy work.

### ***RUN:***

```
docker container ls -a
```

```
devops@DevOpsKatas:~$ docker container ls -a
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS              PORTS              NAMES
```

### List All Containers

Parameter	Description
docker container	The parent command.
ls	Lists containers.
-a	Lists all containers.

## Command Summary

The listing is empty because all containers, running and exited, have been removed.

### ***RUN:***

```
docker image rm $(docker image ls -aq)
```

```
devops@DevOpsKatas:~$ docker image rm $(docker image ls -aq)
Untagged: ubuntu:latest
Untagged: ubuntu@sha256:dd7808d8792c9841d0b460122f1acf0a2dd1f56404f8d1e56298048885e45535
Deleted: sha256:0ef2e08ed3fabfc44002ccb846c4f2416a2135affc3ce39538834059606f32dd
Deleted: sha256:0d58a35162057295d273c5fb8b7e26124a31588cdadad125f4bce63b638dddb5
Deleted: sha256:cb7f997e049c07cdd872b8354052c808499937645f6164912c4126015df036cc
Deleted: sha256:fcba4581c4f016b2e9761f8f69239433e1e123d6f5234ca9c30c33eba698487cc
Deleted: sha256:b53cd3273b78f7f9e7059231fe0a7ed52e0f8e3657363eb015c61b2a6942af87
Deleted: sha256:745f5be9952c1a22dd4225ed6c8d7b760fe0d3583efd52f91992463b53f7aea3
Untagged: hello-world:latest
Untagged: hello-world@sha256:c5515758d4c5e1e838e9cd307f6c6a0d620b5e07e6f927b07d05f6d12a1ac8d7
Deleted: sha256:48b5124b2768d2b917edcb640435044a97967015485e812545546cbcd5cf0233
Deleted: sha256:98c944e98de8d35097100ff70a31083ec57704be0991a92c51700465e4544d08
devops@DevOpsKatas:~$
```

### Remove All Images

Parameter	Description
<code>image</code>	The parent command.
<code>ls</code>	Lists images.
<code>\$(docker image ls -aq)</code>	A command substitution that outputs the IDs of every <code>image</code> in the local image repository. Note that the <code>-aq</code> parameters can be combined, similar to <code>-it</code> .

## Command Summary

This command removes all images from the local computer.

## Docker Kata 4: Running a Web Server in a Container

Containers are useful as components of complex information systems. Web servers are, of course, an important component of any web-based application. This kata will demonstrate how containers can be used to run a web server.

### Step 1: Run a Web Server

#### ***RUN:***

```
docker container stop $(docker container ls -q)
docker container rm $(docker container ls -aq)
```



```
devops@DevOpsKatas:~$ docker container stop $(docker container ls -q)
a6c9adcde337
9bcaf7e1b93b
c0700f445990
devops@DevOpsKatas:~$ docker container rm $(docker container ls -aq)
a6c9adcde337
9bcaf7e1b93b
c0700f445990
934a76f20b08
499abb802fc5
devops@DevOpsKatas:~$
```

### Stop and Remove Containers

Parameter	Description
<code>docker container stop \$(docker container ls -q)</code>	Stop and remove all containers.
<code>docker container rm \$(docker container ls -aq)</code>	

## Command Summary

These commands stop and remove all containers.

### **RUN:**

```
docker container run -d -p 80:80 --name webserver nginx
```

```
devops@DevOpsKatas:~$ docker container run -d -p 80:80 --name webserver nginx
89600ae614bd127d526fa06c0e6f9a4e2543408d7a329d0716a0e862eb4ff075
devops@DevOpsKatas:~$
```

### Run Web Server

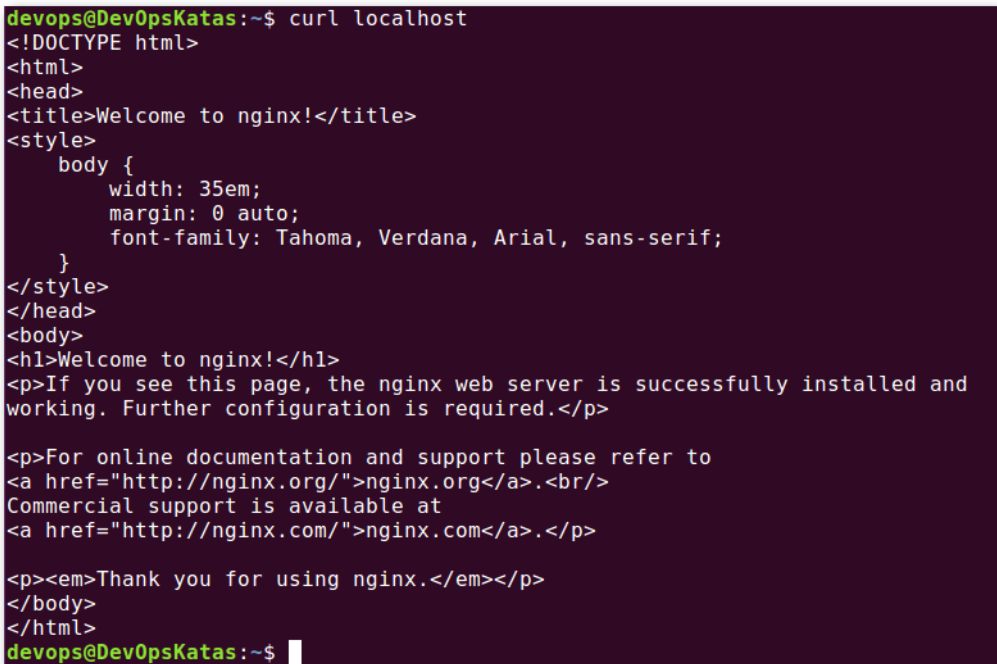
Parameter	Description
<code>docker container</code>	The parent command.
<code>run</code>	Runs a container.
<code>-d</code>	Runs a container in disconnected mode.
<code>-p</code>	Maps a TCP <i>port</i> from the host to the container.
<code>80:80</code>	“Publishes” a port from the container to the host. The format is <i>host port:container port</i> . Requests sent to the specified host port are forwarded to the container port.
<code>--name</code>	Assigns a name to a container.
<code>webserver</code>	The name assigned to the container.
<code>nginx</code>	The name of the image to run.

## Command Summary

Previous katas have used the NGINX container as a demonstration of a container that runs in disconnected mode. This kata shows an NGINX container doing what it is designed to do: run an HTTP server. The `-p` parameter “publishes” a container port to a host port.

### **RUN:**

```
curl localhost
```



```
devops@DevOpsKatas:~$ curl localhost
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
  body {
    width: 35em;
    margin: 0 auto;
    font-family: Tahoma, Verdana, Arial, sans-serif;
  }
</style>
</head>
<body>
<h1>Welcome to nginx!</h1>
<p>If you see this page, the nginx web server is successfully installed and
working. Further configuration is required.</p>

<p>For online documentation and support please refer to
<a href="http://nginx.org/">nginx.org</a>.<br/>
Commercial support is available at
<a href="http://nginx.com/">nginx.com</a>.</p>

<p><em>Thank you for using nginx.</em></p>
</body>
</html>
devops@DevOpsKatas:~$
```

### NGINX Welcome Page

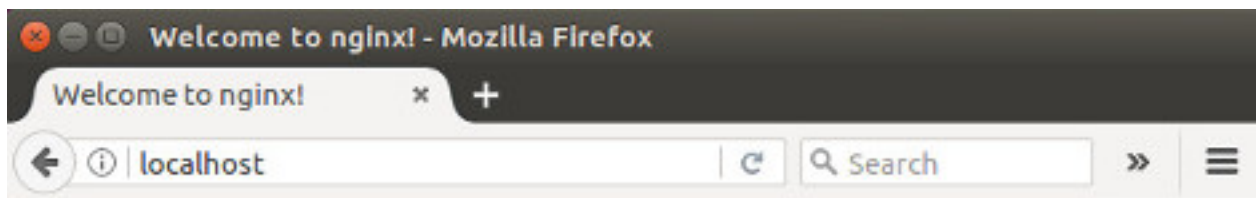
Parameter/Command	Description
<code>curl localhost</code>	<p><code>curl</code> is a command-line program that sends HTTP requests.</p> <p><code>localhost</code> is an alias for the local network adapter. This command sends a request to the local machine on port 80, which is the default HTTP port.</p>

## Command Summary

The `curl` program is a simple command-line HTTP client. This command uses `curl` to issue a request to the NGINX server from the command line. The final step shows that Firefox can be used to view the page in a web browser.

**RUN:**

Open Firefox and go to `http://localhost`



# Welcome to nginx!

If you see this page, the nginx web server is successfully installed and working. Further configuration is required.

For online documentation and support please refer to [nginx.org](http://nginx.org).  
Commercial support is available at [nginx.com](http://nginx.com).

*Thank you for using nginx.*

Firefox

## Command Summary

The last step demonstrates that the NGINX server can also be accessed by a web browser.

## Step 2: Run a Second Webserver on a Different Port

**RUN:**

```
docker container run -d -p 81:80 --name webserver2 nginx
```

```
devops@DevOpsKatas:~$ docker container run -d -p 81:80 --name webserver2 nginx
97e157830467355249cb3e3b5e5984573a8d06604abf787202103baae08adf27
devops@DevOpsKatas:~$
```

### HTTP Server on A Different Host Port

Parameter	Description
<code>docker container</code>	The parent command.
<code>run</code>	Runs a container.
<code>-d</code>	Runs a container in disconnected mode.
<code>-p</code>	Maps a TCP <i>port</i> from the host to the container.
<code>81:80</code>	“Publishes” a port from the container to the host. The format is <i>host port:container port</i> . Requests sent to the specified host port are forwarded to the container port.
<code>--name</code>	Assigns a name to a container.
<code>webserver2</code>	The name assigned to the container.
<code>nginx</code>	The name of the image to run.

## Command Summary

This command is identical to command in the first step, with one exception. This command uses host port **81** instead of 80. Both NGINX containers are now running, one mapped to host port 80, the other to host port 81. You should be able to use `curl`, alternating between ports 80 and 81, to view either web server. Port publishing, or mapping, can be used for a variety of use cases. Running web servers side-by-side can support load balancing, service versioning, and testing.

### **RUN:**

```
curl localhost:81
```

```
devops@DevOpsKatas:~$ curl localhost:81
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
  body {
    width: 35em;
    margin: 0 auto;
    font-family: Tahoma, Verdana, Arial, sans-serif;
  }
}
```

### Second Server Welcome Page

Parameter/Command	Description
<code>curl localhost:81</code>	<code>curl</code> is a command-line program that sends HTTP requests to an endpoint. <code>localhost</code> is an alias for the local network adapter. This command specifies port 81 (instead of the default port 80.)

## Command Summary

This command demonstrates that the second NGINX container is indeed running and mapped to host port 81. You can also use Firefox to view the page by visiting `localhost:81`.

## Docker Kata 6: Creating Docker Images

The previous katas have used images provided on Docker Hub. Most of those images are created and managed by software vendors.

This kata will demonstrate how you can define and build your own images, for your own use or to distribute to others. There are two ways to create an image:

- Commit changes to a modified container
- Build an image from a Dockerfile

## Step 1: Creating an Image From a Modified Container

First, stop and remove all containers.

**RUN:**

```
mkdir dockerimage
cd dockerimage
cp ../index.html .
ls
```

```
devops@DevOpsKatas:~$ mkdir dockerimage
devops@DevOpsKatas:~$ cd dockerimage
devops@DevOpsKatas:~/dockerimage$ cp ../index.html .
devops@DevOpsKatas:~/dockerimage$ ls
index.html
devops@DevOpsKatas:~/dockerimage$
```

Create Image Directory

Parameter/Command	Description
<code>mkdir dockerimage</code> <code>cd dockerimage</code>	Creates, and then changes to, a new directory called <code>dockerimage</code> .
<code>cp ../index.html .</code> <code>ls</code>	Copies the <code>index.html</code> file from the parent directory to the <code>dockerimage</code> directory. Lists the files in the <code>dockerimage</code> directory. The <code>index.html</code> file should be listed.

## Command Summary

These commands create a new directory called `dockerimage` and copy a sample `index.html` file into that directory.

### ***RUN:***

```
docker container run --name web -d -p 80:80 nginx
```

```
devops@DevOpsKatas:~/dockerimage$ docker container run --name web -d -p 80:80 nginx
f718e6610ad2d879006452d19a046561003ebe30fc05263d1cd7cec615ebc80c
devops@DevOpsKatas:~/dockerimage$
```

### Run NGINX Container

Parameter	Description
<code>docker container</code>	The parent command.
<code>run</code>	Runs a container.
<code>--name</code>	Assigns a name to a container.
<code>web</code>	The name to assign to the container.
<code>-d</code>	Runs a container in disconnected mode.
<code>-p</code>	Publishes a port from the container to the host.
<code>80:80</code>	Publishes port 80 on the container to port 80 on the host.
<code>nginx</code>	The name of the image to run.

## Command Summary

This command starts an NGINX container, using the same command from Kata 4.

### ***RUN:***

```
docker container cp index.html web:/usr/share/nginx/html/index.html
```

```
devops@DevOpsKatas:~/dockerimage$ docker container cp index.html web:/usr/share/nginx/html/index.html
devops@DevOpsKatas:~/dockerimage$
```

#### Copy File Into Container

Parameter	Description
<code>docker container</code>	The parent command.
<code>cp</code>	The <code>cp</code> command copies a file to a container from the host, or vice versa.
<code>index.html</code>	The name of the file to copy into the container.
<code>web:/usr/share/nginx/html/index.html</code>	<p>This is the image and the path. The name of the container to which the file should be copied is before the colon.</p> <p>The path inside the container, to which the file should be copied, follows the colon. The effect is that the <code>index.html</code> file is copied into the container.</p>

## Command Summary

The `cp` command copies files between containers and their hosts. This step copied the `index.html` file from the host to the container.

#### ***RUN:***

```
curl localhost
```

```
devops@DevOpsKatas:~/dockerimage$ curl localhost
<!DOCTYPE html>
<html>
<head>
<title>Welcome to DevOps Katas!</title>
<style>
  body {
    width: 35em;
    margin: 0 auto;
    font-family: Tahoma, Verdana, Arial, sans-serif;
  }
</style>
</head>
<body>
<h1>Welcome to DevOpsKatas!</h1>
<p>If you see this page, You have sucessfully executed this part of your Docker Image kata.</p>
<p><em>Thank you for practicing your DevOps Katas!</em></p>
</body>
</html>
devops@DevOpsKatas:~/dockerimage$
```

#### View Custom Page

The `curl` command issues a request to the local system via HTTP. You can also use Firefox to view this page. Note that welcome page has been modified. The `index.html` file copied in the previous step overwrote the default NGINX welcome page.

#### **RUN:**

```
docker container ls
```

```
devops@DevOpsKatas:~/dockerimage$ docker container ls
CONTAINER ID   IMAGE     COMMAND                  CREATED        STATUS        PORTS                               NAMES
f718e6610ad2   nginx    "nginx -g 'daemon ...'"  5 minutes ago Up 5 minutes   0.0.0.0:80->80/tcp, 443/tcp        web
devops@DevOpsKatas:~/dockerimage$
```

#### List Containers

Parameter	Description
<code>docker container</code>	The parent command.
<code>ls</code>	Lists running containers when combined with the <code>docker container</code> parent command.

## Command Summary

This step listed all running containers. The NGINX container we ran in an earlier step is running.

#### **RUN:**



```
docker container commit web kataimage_nginx
```

```
devops@DevOpsKatas:~/dockerimage$ docker container commit web kataimage_nginx
sha256:94c2d42358395bf0dac3fd4f2d3f99966ca24c8ec4b76512cb00af890c3994d6
devops@DevOpsKatas:~/dockerimage$
```

### List Containers

Parameter	Description
<code>docker container</code>	The parent command.
<code>commit</code>	The <code>commit</code> command stops a running container, then creates a new image from that container. Changes to the filesystem of the container are persisted in the new image.
<code>web</code>	The name of the container from which to derive a new image.
<code>kataimage_nginx</code>	The name assigned to the new image.

## Command Summary

The `commit` command is used to commit the changes to a container, thus creating a new image.

Recall in Kata 3 when we learned about filesystem changes in a container. When we copied the `index.html` file from the host to the NGINX container, we modified the *container*, but not the *image* from which that container was started. The `commit` command creates a new image, called `kataimage_nginx`, that includes that change.

### RUN:

```
docker image ls
```

```
devops@DevOpsKatas:~/dockerimage$ docker image ls
REPOSITORY          TAG             IMAGE ID        CREATED         SIZE
kataimage_nginx     latest          94c2d4235839   About a minute ago  182 MB
nginx                latest          6b914bbcb89e   2 weeks ago     182 MB
ubuntu              latest          0ef2e08ed3fa   2 weeks ago     130 MB
devops@DevOpsKatas:~/dockerimage$
```

### List Images

Parameter	Description
<code>docker image</code>	The parent command.
<code>ls</code>	Lists images when combined with the <code>docker image</code> parent command. Note that the <code>kataimage_nginx</code> image is listed.

## Command Summary

This command lists all images. The `kataimage_nginx` image we created previously should be listed.

### ***RUN:***

```
docker container run -d -p 81:80 kataimage_nginx
```

```
devops@DevOpsKatas:~/dockerimage$ docker container run -d -p 81:80 kataimage_nginx
ae9ff22274908e4acaa01bd3909afd5ffad60d4b96af582e7c22b94df1b43557
devops@DevOpsKatas:~/dockerimage$
```

### Run the New Image

Parameter	Description
<code>docker container</code>	The parent command.
<code>run</code>	Runs a new container.
<code>-d</code>	Runs a container in disconnected mode.
<code>-p</code>	Publishes a container port to a host port.
<code>81:80</code>	Publishes container port 80 to host port 81.
<code>kataimage_nginx</code>	The name of the image to run.

## Command Summary

This command runs a second NGINX container, this time using the `kataimage_nginx` image we created in the previous step.

### ***RUN:***

```
curl localhost:81
```

```
devops@DevOpsKatas:~/dockerimage$ curl localhost
<!DOCTYPE html>
<html>
<head>
<title>Welcome to DevOps Katas!</title>
<style>
  body {
    width: 35em;
    margin: 0 auto;
    font-family: Tahoma, Verdana, Arial, sans-serif;
  }
</style>
</head>
<body>
<h1>Welcome to DevOpsKatas!</h1>
<p>If you see this page, You have sucessfully executed this part of your Docker Image kata.</p>
<p><em>Thank you for practicing your DevOps Katas!</em></p>
</body>
</html>
devops@DevOpsKatas:~/dockerimage$
```

#### View Custom Page in New Image

Open Firefox and go to <http://localhost:81> to see this in a web browser.

## Command Summary

The `curl` command issues a request to the local system via HTTP. You can also use Firefox (visit <http://localhost:81>) to view this page. Note that welcome page is the customized *DevOps Katas* welcome page. The `kataimage_nginx` image includes the modified `index.html` file.

## Step 2: Create an Image with a Dockerfile

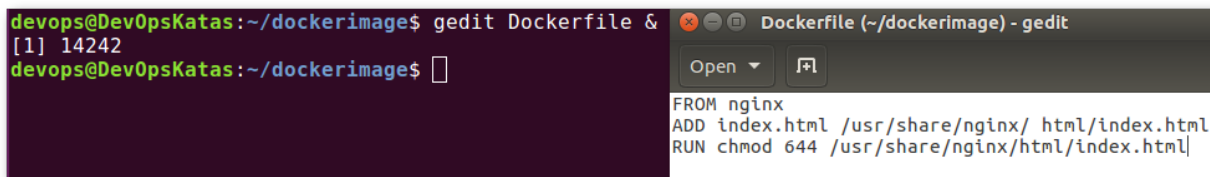
First, stop and remove all containers.

**RUN:**

```
gedit Dockerfile &
```

Enter (or copy/paste) the following text into the editor:

```
FROM nginx
ADD index.html /usr/share/nginx/html/index.html
RUN chmod 644 /usr/share/nginx/html/index.html
```



### Dockerfile

Parameter/Command	Description
<code>gedit index.html &amp;</code>	<code>gedit</code> is a text editor. This command opens the <code>index.html</code> file in the editor. The <code>&amp;</code> ampersand starts the editor in a new process (this allows a user to continue using the terminal window.)
<code>FROM nginx</code>	The first line in the Dockerfile is always <code>FROM</code> . This indicates the <i>base image</i> that will be used to create the image defined by a Dockerfile. This Dockerfile specifies the NGINX image as the base.
<code>ADD index.html /usr/share/nginx/html/index.html</code>	The <code>ADD</code> directive adds a file to the image when it is built. This can be a local file or a URL. <code>ADD</code> is followed by the path to the source file, then the path to which the file should be copied in the new image.
<code>RUN chmod 644 /usr/share/nginx/html/index.html</code>	The <code>RUN</code> directive runs a command within the container. This command uses the <code>chmod</code> Linux command to change the permissions of the <code>index.html</code> file copied into the image. This permissions change allows the NGINX server to read the <code>index.html</code> file.

## Command Summary

This step demonstrates the second method to create Docker images: a Dockerfile. The result is the same as the previous step, however, the `commit` method is an *imperative* method for creating images. That is good for experimentation and testing, however, a *declarative* method is better for building containers as part of a software delivery process.

A Dockerfile uses a declarative syntax to describe the contents of a container:

The `FROM` indicates the *base image* from which to create a new image. Any existing image may be used as the basis for any new image. This step uses the NGINX image as its base image. The build process starts a temporary container from the base image defined in the `FROM` directive, then executes the following directives, creating the image.

The `ADD` command adds a file to the image. The source can be a file on the local file system, or a URL to a file on the web. This step adds the custom `index.html` file (the same file as the one in the previous step) to the image.

`RUN` entries execute commands within the image as it is built. This command updates the permissions on the `index.html` file, which allows the NGINX HTTP server to read the file.

This Dockerfile builds a new image from the NGINX image. That image is available on Docker Hub. Here is the Dockerfile for the version of the NGINX image:

```
FROM debian:jessie
```

```
MAINTAINER NGINX Docker Maintainers "docker-maint@nginx.com"
```

```
ENV NGINX_VERSION 1.11.10-1~jessie
```

```
RUN apt-key adv --keyserver hkp://pgp.mit.edu:80 --recv-keys > 573BFD6B3D8FBC641\
079A6ABABF5BD827BD9BF62 \
```

```
&& echo "deb http://nginx.org/packages/mainline/debian/ jessie nginx" >> > /etc
/aptsources.list \
```

```
&& apt-get update \
```

```
&& apt-get install --no-install-recommends --no-install-suggests -y \
```

```
ca-certificates \
```

```
nginx=${NGINX_VERSION} \
```

```
nginx-module-xslt \
```

```
nginx-module-geoip \
```

```
nginx-module-image-filter \
```

```
nginx-module-perl \
```

```
nginx-module-njs \
```

```
gettext-base \
```

```
&& rm -rf /var/lib/apt/lists/*
```

```
# forward request and error logs to docker log collector
```

```

RUN ln -sf /dev/stdout /var/log/nginx/access.log \
    && ln -sf /dev/stderr /var/log/nginx/error.log

EXPOSE 80 443

CMD ["nginx", "-g", "daemon off;"]

```

This Dockerfile includes some of the same directives that are in the Dockerfile for `kataimage_nginx`, and some that are not.

The `FROM` directive indicates the base image of the official NGINX image. The official NGINX image is based on the *jessie* version of the official *Debian Linux* container. Public, official images are typically “layered” in this fashion. Base images start at the “bottom” or “first” layer with official OS distributions. Additional layers add software to the base image, creating a specialized container made for a specific purpose, such as a web server.

The `MAINTAINER` directive is deprecated in the current version of Docker. It has been replaced by the `AUTHOR` directive, which

The `ENV` directive creates environment variables in the target image.

The following `RUN` command combines multiple installations and filesystem changes in a single command. The double-ampersand `&&` may be used in Linux to execute multiple commands in one line. This technique is important when defining an image.

Docker uses a specialized *layered* filesystem to define images [TODO: REF]. When an image is created using `FROM`, a new layer is added to the filesystem that defines the image. The new “top” layer represents all the differences (installed programs, added files) between the base image and the new image. This layering system keeps images small by, storing only the differences between layers. Keeping images as small as possible makes image deployment faster, and reduces the need for storage.

[TODO: IMAGE]

Each `RUN` command creates a new file system layer. If each of the commands in the NGINX Dockerfile were run in separate `RUN` directives, a new layer would be created for each execution. This would create many new layers, resulting in a large image. Running all the commands in a single execution creates just one new layer. This Dockerfile uses the double-ampersand `&&` command concatenation to execute all commands in a single `RUN` directive.

The `EXPOSE` directive indicates to Docker that the container will listen on the port numbers specified. This is not the same as publishing the ports. The `-p` parameter is still necessary when running a container. It is possible to use the `-P` (capital ‘P’) parameter to publish all ports identified with `EXPOSE`. The `-P` parameter will publish all exposed ports to the same port on the host. Given this NGINX Dockerfile, these two commands have the same effect:

```

docker container run -d -p 80:80 -p 443:443 nginx
docker container run -d -P nginx

```

The last directive in the NGINX Dockerfile is `CMD`:

```
CMD ["nginx", "-g", "daemon off;"]
```

There can be only one CMD directive in a Dockerfile (if there is more than one, the last one defined is the only one that will run.)

The CMD directive is one of several methods to define the “start” command of the container. This method defines the program to run and its parameters:

```
CMD ["nginx", "-g", "daemon off;"]
```

Parameter/Command	Description
nginx	The NGINX HTTP server program.
-g	Indicates that the following parameters are global configuration directives. This changes the configuration for this execution of the NGINX server.
daemon off	Configures the NGINX server process to run in the foreground as the primary process. This is recommended for running NGINX in a container.

#### **RUN:**

```
docker image build -t katadockerfile_image .
```

```
devops@DevOpsKatas:~/dockerimages$ docker image build -t katadockerfile_image .
Sending build context to Docker daemon 3.072 kB
Step 1/3 : FROM nginx
--> 6b914bbcb89e
Step 2/3 : ADD index.html /usr/share/nginx/html/index.html
--> a6414ceae1d
Removing intermediate container b344a77f92f3
Step 3/3 : RUN chmod 644 /usr/share/nginx/html/index.html
--> Running in b5f219b22276
--> ec7daf672ddb
Removing intermediate container b5f219b22276
Successfully built ec7daf672ddb
devops@DevOpsKatas:~/dockerimages$
```

#### **Build Image**

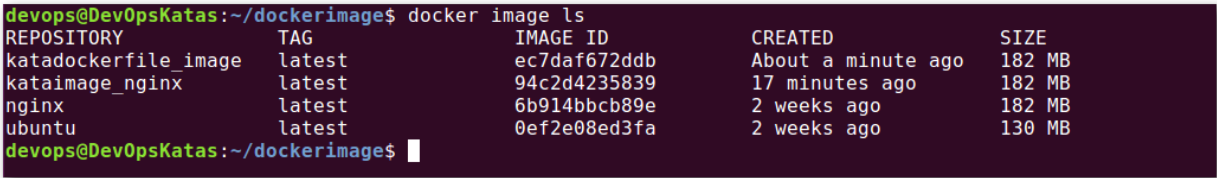
Parameter	Description
docker image	The parent command.
build	Builds a new image from a Dockerfile when combined with the docker image parent command.
-t	Assigns a name, and if desired, a tag to the new image.

Parameter	Description
katadockerfile_image .	The name of the new image. The period after the name indicates the path to the Dockerfile. In Linux, a single period indicates the current directory.

Command Summary

The `build` subcommand creates a new Docker image from a Dockerfile.

```
docker image ls
```



List Images

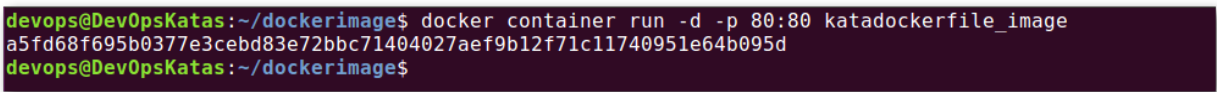
Parameter	Description
docker image	The parent command.
ls	Lists images when combined with the <code>docker image</code> parent command.

Command Summary

This command lists all images. The `katadockerfile_image` image should be listed.

**RUN:**

```
docker container run -d -p 80:80 katadockerfile_image
```



Run Built Image



Parameter	Description
<code>docker container</code>	The parent command.
<code>run</code>	Runs a container.
<code>-d</code>	Runs a container in disconnected mode.
<code>-p</code>	Publishes a container port to a host port.
<code>80:80</code>	Publishes container port 80 to host port 80.
<code>katadockerfile_image</code>	The name of the image to run.

## Command Summary

This command runs the image that was created in the previous step using `docker image build`.

### **RUN:**

```
curl localhost
```

```
devops@DevOpsKatas:~/dockerimage$ curl localhost
<!DOCTYPE html>
<html>
<head>
<title>Welcome to DevOps Katas!</title>
<style>
  body {
    width: 35em;
    margin: 0 auto;
    font-family: Tahoma, Verdana, Arial, sans-serif;
  }
</style>
</head>
<body>
<h1>Welcome to DevOpsKatas!</h1>
<p>If you see this page, You have successfully executed this part of your Docker Image kata.</p>

<p><em>Thank you for practicing your DevOps Katas!</em></p>
</body>
</html>
devops@DevOpsKatas:~/dockerimage$
```

[View Custom Page](#)

## Command Summary

This command returns the response from the NGINX container using `curl`. Open Firefox and go to `http://localhost` to see the page in a web browser. As expected, the result is the same as the previous kata step. This method, however uses a declarative method. The Dockerfile used to create this image can be added to source control, and used as part of an automated build process.



# Docker Kata Practice Page

## Kata 1: Basic Commands

### Step 1

```
docker container run hello-world
```

### Step 2

```
docker container ls
```

```
docker container ls -a
```

### Step 3

```
docker image ls
```

### Step 4

```
docker container run --name my_container hello-world
```

```
docker container ls -a
```

```
docker container inspect my_container
```

## Step 5

```
docker container run -it ubuntu bash
ls
ls /bin
exit
```

## Step 6

```
docker container rm $(docker container ls -a -q)
```

```
docker container ls -a
```

```
docker image rm $(docker image ls -aq)
```

## Kata 4: Running a Web Server in a Container

### Step 1

```
docker container stop $(docker container ls -q)
```

```
docker container rm $(docker container ls -aq)
```

```
docker container run -d -p 80:80 --name webserver nginx
```

```
curl localhost
```

Open Firefox and go to <http://localhost>

### Step 2

```
docker container run -d -p 81:80 --name webserver2 nginx
```

```
curl localhost:81
```

```
Open Firefox and go to http://localhost:81
```

## Kata 6: Creating Docker Images

### Step 1

```
mkdir dockerimage  
cd dockerimage  
cp ../index.html .  
ls
```

```
docker container run --name web -d -p 80:80 nginx
```

```
docker container cp index.html web:/usr/share/nginx/html/index.html
```

```
curl localhost
```

```
docker container ls
```

```
docker container commit web kataimage_nginx
```

```
docker image ls
```

```
docker container run -d -p 81:80 kataimage_nginx
```

```
curl localhost:81
```

## Step 2

Stop and remove all containers.

```
gedit Dockerfile &
```

Enter (or copy/paste) the following text into the editor:

```
FROM nginx ADD index.html /usr/share/nginx/html/index.html  
RUN chmod 644 /usr/share/nginx/html/index.html
```

```
docker image build -t katadockerfile_image .
```

```
docker container run -d -p 80:80 katadockerfile_image
```

```
curl localhost
```

Open Firefox and go to <http://localhost>