Examples using the Docker Engine SDKs and Docker API

- Run a container
- Run a container in the background
- List and manage containers
- Stop all running containers
- Print the logs of a specific container
- List all images
- Pull an image
- Pull an image with authentication
- Commit a container

After you install Docker, you can install the Go or Python SDK and also try out the Docker Engine API.

Each of these examples show how to perform a given Docker operation using the Go and Python SDKs and the HTTP API using `curl`.

Run a container

This first example shows how to run a container using the Docker API. On the command line, you would use the `docker run` command, but this is just as easy to do from your own apps too.

This is the equivalent of typing `docker run alpine echo hello world` at the command prompt:

{{< tabs group="lang" >}} {{< tab name="Go" >}}

```
package main


import (
```

```
	"context"
	"io"
	"log"
	"os"

	"github.com/moby/moby/api/pkg/stdcopy"
	"github.com/moby/moby/api/types/container"
	"github.com/moby/moby/client"
)

func main() {
	ctx := context.Background()
	apiClient, err := client.New(client.FromEnv, client.WithUserAgent("my-application/1.0.0"))
	if err != nil {
		log.Fatal(err)
	}
	defer apiClient.Close()

	reader, err := apiClient.ImagePull(ctx, "docker.io/library/alpine", client.ImagePullOptions{})
	if err != nil {
		log.Fatal(err)
	}

	defer reader.Close()
	// cli.ImagePull is asynchronous.
	// The reader needs to be read completely for the pull operation to complete.
	// If stdout is not required, consider using io.Discard instead of os.Stdout.
	io.Copy(os.Stdout, reader)

	resp, err := apiClient.ContainerCreate(ctx, client.ContainerCreateOptions{
		Config: &container.Config{
			Cmd: []string{"echo", "hello world"},
			Tty: false,
		},
		Image: "alpine",
	})
	if err != nil {
		log.Fatal(err)
	}
```

```
	if _, err := apiClient.ContainerStart(ctx, resp.ID, client.ContainerStartOptions{}); err !=
nil {
		log.Fatal(err)
	}

	wait := apiClient.ContainerWait(ctx, resp.ID, client.ContainerWaitOptions{})
	select {
	case err := <-wait.Error:
		if err != nil {
			log.Fatal(err)
		}
	case <-wait.Result:
	}

	out, err := apiClient.ContainerLogs(ctx, resp.ID, client.ContainerLogsOptions{ShowStdout:
true})
	if err != nil {
		log.Fatal(err)
	}

	stdcopy.StdCopy(os.Stdout, os.Stderr, out)
}
```

{{< /tab >}} {{< tab name="Python" >}}

```
import docker
client = docker.from_env()
print(client.containers.run("alpine", ["echo", "hello", "world"]))
```

{{< /tab >}} {{< tab name="HTTP" >}}

```
$ curl --unix-socket /var/run/docker.sock -H "Content-Type: application/json" \
  -d '{"Image": "alpine", "Cmd": ["echo", "hello world"]}' \
  -X POST http://localhost/v{{% param "latest_engine_api_version" %}}/containers/create
{"Id":"1c6594faf5","Warnings":null}

$ curl --unix-socket /var/run/docker.sock -X POST http://localhost/v{{% param
"latest_engine_api_version" %}}/containers/1c6594faf5/start

$ curl --unix-socket /var/run/docker.sock -X POST http://localhost/v{{% param
```

```
"latest_engine_api_version" %}}/containers/1c6594faf5/wait
{"StatusCode":0}

$ curl --unix-socket /var/run/docker.sock "http://localhost/v{{% param
"latest_engine_api_version" %}}/containers/1c6594faf5/logs?stdout=1"
hello world
```

When using cURL to connect over a Unix socket, the hostname isn't important. The previous examples use `localhost`, but any hostname would work.

> [!IMPORTANT]
>
> The previous examples assume you're using cURL 7.50.0 or above. Older versions of cURL used a non-standard URL notation when using a socket connection.
>
> If you're' using an older version of cURL, use `http:/<API version>/` instead, for example: `http:/v{{% param "latest_engine_api_version" %}}/containers/1c6594faf5/start`.

{{< /tab >}} {{< /tabs >}}

Run a container in the background

You can also run containers in the background, the equivalent of typing `docker run -d bfirsh/reticulate-splines`:

{{< tabs group="lang" >}} {{< tab name="Go" >}}

```
package main

import (
□"context"
□"fmt"
□"io"
□"log"
□"os"

□"github.com/moby/moby/client"
```

```
)

func main() {
	ctx := context.Background()
	apiClient, err := client.New(client.FromEnv, client.WithUserAgent("my-application/1.0.0"))
	if err != nil {
		log.Fatal(err)
	}
	defer apiClient.Close()

	imageName := "bfirsh/reticulate-splines"

	out, err := apiClient.ImagePull(ctx, imageName, client.ImagePullOptions{})
	if err != nil {
		log.Fatal(err)
	}
	defer out.Close()

	io.Copy(os.Stdout, out)

	resp, err := apiClient.ContainerCreate(ctx, client.ContainerCreateOptions{
		Image: imageName,
	})
	if err != nil {
		log.Fatal(err)
	}

	if _, err := apiClient.ContainerStart(ctx, resp.ID, client.ContainerStartOptions{}); err !=
nil {
		log.Fatal(err)
	}

	fmt.Println(resp.ID)
}
```

{{< /tab >}} {{< tab name="Python" >}}

```
import docker
client = docker.from_env()
container = client.containers.run("bfirsh/reticulate-splines", detach=True)
```

```
print(container.id)
```

{{< /tab >}} {{< tab name="HTTP" >}}

```
$ curl --unix-socket /var/run/docker.sock -H "Content-Type: application/json" \
  -d '{"Image": "bfirsh/reticulate-splines"}' \
  -X POST http://localhost/v{{% param "latest_engine_api_version" %}}/containers/create
{"Id":"1c6594faf5","Warnings":null}

$ curl --unix-socket /var/run/docker.sock -X POST http://localhost/v{{% param
"latest_engine_api_version" %}}/containers/1c6594faf5/start
```

{{< /tab >}} {{< /tabs >}}

List and manage containers

You can use the API to list containers that are running, just like using `docker ps`:

{{< tabs group="lang" >}} {{< tab name="Go" >}}

```
package main

import (
	"context"
	"fmt"
	"log"

	"github.com/moby/moby/client"
)

func main() {
	ctx := context.Background()
	apiClient, err := client.New(client.FromEnv, client.WithUserAgent("my-application/1.0.0"))
	if err != nil {
		log.Fatal(err)
	}
	defer apiClient.Close()

	containers, err := apiClient.ContainerList(ctx, client.ContainerListOptions{})
```

```
	if err != nil {
		log.Fatal(err)
	}

	for _, container := range containers.Items {
		fmt.Println(container.ID)
	}
}
```

{{< /tab >}} {{< tab name="Python" >}}

```
import docker
client = docker.from_env()
for container in client.containers.list():
  print(container.id)
```

{{< /tab >}} {{< tab name="HTTP" >}}

```
$ curl --unix-socket /var/run/docker.sock http://localhost/v{{% param
"latest_engine_api_version" %}}/containers/json
[{
  "Id":"ae63e8b89a26f01f6b4b2c9a7817c31a1b6196acf560f66586fbc8809ffcd772",
  "Names":["/tender_wing"],
  "Image":"bfirsh/reticulate-splines",
  ...
}]
```

{{< /tab >}} {{< /tabs >}}

Stop all running containers

Now that you know what containers exist, you can perform operations on them. This example stops all running containers.

> [!NOTE]
>
> Don't run this on a production server. Also, if you're' using swarm services, the containers stop, but Docker creates new ones to keep the service running in its

> configured state.

{{< tabs group="lang" >}} {{< tab name="Go" >}}

```
package main

import (
	"context"
	"fmt"
	"log"

	"github.com/moby/moby/client"
)

func main() {
	ctx := context.Background()
	apiClient, err := client.New(client.FromEnv, client.WithUserAgent("my-application/1.0.0"))
	if err != nil {
		log.Fatal(err)
	}
	defer apiClient.Close()

	containers, err := apiClient.ContainerList(ctx, client.ContainerListOptions{})
	if err != nil {
		log.Fatal(err)
	}

	for _, container := range containers.Items {
		fmt.Print("Stopping container ", container.ID[:10], "... ")
		noWaitTimeout := 0 // to not wait for the container to exit gracefully
		if _, err := apiClient.ContainerStop(ctx, container.ID, client.ContainerStopOptions{Timeout:
&noWaitTimeout}); err != nil {
			log.Fatal(err)
		}
		fmt.Println("Success")
	}
}
```

{{< /tab >}} {{< tab name="Python" >}}

```
import docker
client = docker.from_env()
for container in client.containers.list():
  container.stop()
```

{{< /tab >}} {{< tab name="HTTP" >}}

```
$ curl --unix-socket /var/run/docker.sock http://localhost/v{{% param
"latest_engine_api_version" %}}/containers/json
[{
  "Id":"ae63e8b89a26f01f6b4b2c9a7817c31a1b6196acf560f66586fbc8809ffcd772",
  "Names":["/tender_wing"],
  "Image":"bfirsh/reticulate-splines",
  ...
}]

$ curl --unix-socket /var/run/docker.sock \
  -X POST http://localhost/v{{% param "latest_engine_api_version"
%}}/containers/ae63e8b89a26/stop
```

{{< /tab >}} {{< /tabs >}}

Print the logs of a specific container

You can also perform actions on individual containers. This example prints the logs of a container given its ID. You need to modify the code before running it to change the hard-coded ID of the container to print the logs for.

{{< tabs group="lang" >}} {{< tab name="Go" >}}

```
package main

import (
"context"
"io"
"log"
"os"
```

```
	"github.com/moby/moby/client"
)

func main() {
	ctx := context.Background()
	apiClient, err := client.New(client.FromEnv, client.WithUserAgent("my-application/1.0.0"))
	if err != nil {
		log.Fatal(err)
	}
	defer apiClient.Close()

	options := client.ContainerLogsOptions{ShowStdout: true}
	// Replace this ID with a container that really exists
	out, err := apiClient.ContainerLogs(ctx, "f1064a8a4c82", options)
	if err != nil {
		log.Fatal(err)
	}

	io.Copy(os.Stdout, out)
}
```

{{< /tab >}} {{< tab name="Python" >}}

```
import docker
client = docker.from_env()
container = client.containers.get('f1064a8a4c82')
print(container.logs())
```

{{< /tab >}} {{< tab name="HTTP" >}}

```
$ curl --unix-socket /var/run/docker.sock "http://localhost/v{{% param
"latest_engine_api_version" %}}/containers/ca5f55cdb/logs?stdout=1"
Reticulating spline 1...
Reticulating spline 2...
Reticulating spline 3...
Reticulating spline 4...
Reticulating spline 5...
```

{{< /tab >}} {{< /tabs >}}

List all images

List the images on your Engine, similar to `docker image ls`:

{{< tabs group="lang" >}} {{< tab name="Go" >}}

```
package main

import (
	"context"
	"fmt"
	"log"

	"github.com/moby/moby/client"
)

func main() {
	ctx := context.Background()
	apiClient, err := client.New(client.FromEnv, client.WithUserAgent("my-application/1.0.0"))
	if err != nil {
		log.Fatal(err)
	}
	defer apiClient.Close()

	images, err := apiClient.ImageList(ctx, client.ImageListOptions{})
	if err != nil {
		log.Fatal(err)
	}

	for _, image := range images.Items {
		fmt.Println(image.ID)
	}
}
```

{{< /tab >}} {{< tab name="Python" >}}

```
import docker
client = docker.from_env()
for image in client.images.list():
```

```
  print(image.id)
```

{{< /tab >}} {{< tab name="HTTP" >}}

```
$ curl --unix-socket /var/run/docker.sock http://localhost/v{{% param
"latest_engine_api_version" %}}/images/json
[{
  "Id":"sha256:31d9a31e1dd803470c5a151b8919ef1988ac3efd44281ac59d43ad623f275dcd",
  "ParentId":"sha256:ee4603260daafe1a8c2f3b78fd760922918ab2441cbb2853ed5c439e59c52f96",
  ...
}]
```

{{< /tab >}} {{< /tabs >}}

Pull an image

Pull an image, like `docker pull`:

{{< tabs group="lang" >}} {{< tab name="Go" >}}

```
package main

import (
	"context"
	"io"
	"log"
	"os"

	"github.com/moby/moby/client"
)

func main() {
	ctx := context.Background()
	apiClient, err := client.New(client.FromEnv, client.WithUserAgent("my-application/1.0.0"))
	if err != nil {
		log.Fatal(err)
	}
	defer apiClient.Close()
```

```
	out, err := apiClient.ImagePull(ctx, "alpine", client.ImagePullOptions{})
	if err != nil {
		log.Fatal(err)
	}
	defer out.Close()

	io.Copy(os.Stdout, out)
}
```

{{< /tab >}} {{< tab name="Python" >}}

```
import docker
client = docker.from_env()
image = client.images.pull("alpine")
print(image.id)
```

{{< /tab >}} {{< tab name="HTTP" >}}

```
$ curl --unix-socket /var/run/docker.sock \
  -X POST "http://localhost/v{{% param "latest_engine_api_version"
%}}/images/create?fromImage=alpine"
{"status":"Pulling from library/alpine","id":"3.1"}
{"status":"Pulling fs layer","progressDetail":{},"id":"8f13703509f7"}
{"status":"Downloading","progressDetail":{"current":32768,"total":2244027},"progress":"[\u003e
                                                    ] 32.77 kB/2.244 MB","id":"8f13703509f7"}
...
```

{{< /tab >}} {{< /tabs >}}

Pull an image with authentication

Pull an image, like `docker pull`, with authentication:

> [!NOTE]
>
> Credentials are sent in the clear. Docker's official registries use HTTPS. Private registries should also be configured to use HTTPS.

{{< tabs group="lang" >}} {{< tab name="Go" >}}

```
package main

import (
	"context"
	"io"
	"log"
	"os"

	"github.com/moby/moby/api/pkg/authconfig"
	"github.com/moby/moby/api/types/registry"
	"github.com/moby/moby/client"
)

func main() {
	ctx := context.Background()
	apiClient, err := client.New(client.FromEnv, client.WithUserAgent("my-application/1.0.0"))
	if err != nil {
		log.Fatal(err)
	}
	defer apiClient.Close()

	authStr, err := authconfig.Encode(registry.AuthConfig{
		Username: "username",
		Password: "password",
	})
	if err != nil {
		log.Fatal(err)
	}

	out, err := apiClient.ImagePull(ctx, "alpine", client.ImagePullOptions{RegistryAuth: authStr})
	if err != nil {
		log.Fatal(err)
	}
	defer out.Close()

	io.Copy(os.Stdout, out)
}
```

{{< /tab >}} {{< tab name="Python" >}}

The Python SDK retrieves authentication information from the credentials store file and integrates with credential helpers. It's possible to override these credentials, but that's out of scope for this example guide. After using `docker login`, the Python SDK uses these credentials automatically.

```
import docker
client = docker.from_env()
image = client.images.pull("alpine")
print(image.id)
```

{{< /tab >}} {{< tab name="HTTP" >}}

This example leaves the credentials in your shell's history, so consider this a naive implementation. The credentials are passed as a Base-64-encoded JSON structure.

```
$ JSON=$(echo '{"username": "string", "password": "string", "serveraddress": "string"}' |
base64)

$ curl --unix-socket /var/run/docker.sock \
  -H "Content-Type: application/tar"
  -X POST "http://localhost/v{{% param "latest_engine_api_version"
%}}/images/create?fromImage=alpine"
  -H "X-Registry-Auth"
  -d "$JSON"
{"status":"Pulling from library/alpine","id":"3.1"}
{"status":"Pulling fs layer","progressDetail":{},"id":"8f13703509f7"}
{"status":"Downloading","progressDetail":{"current":32768,"total":2244027},"progress":"[\u003e
                                                  ] 32.77 kB/2.244 MB","id":"8f13703509f7"}
...
```

{{< /tab >}} {{< /tabs >}}

Commit a container

Commit a container to create an image from its contents:

{{< tabs group="lang" >}} {{< tab name="Go" >}}

```
package main

import (
	"context"
	"fmt"
	"log"

	"github.com/moby/moby/api/types/container"
	"github.com/moby/moby/client"
)

func main() {
	ctx := context.Background()
	apiClient, err := client.New(client.FromEnv, client.WithUserAgent("my-application/1.0.0"))
	if err != nil {
		log.Fatal(err)
	}
	defer apiClient.Close()

	createResp, err := apiClient.ContainerCreate(ctx, client.ContainerCreateOptions{
		Config: &container.Config{
			Cmd: []string{"touch", "/helloworld"},
		},
		Image: "alpine",
	})
	if err != nil {
		log.Fatal(err)
	}

	if _, err := apiClient.ContainerStart(ctx, createResp.ID, client.ContainerStartOptions{}); err
!= nil {
		log.Fatal(err)
	}

	wait := apiClient.ContainerWait(ctx, createResp.ID, client.ContainerWaitOptions{})
	select {
	case err := <-wait.Error:
		if err != nil {
			log.Fatal(err)
		}
```

```
	case <-wait.Result:
	}

	commitResp, err := apiClient.ContainerCommit(ctx, createResp.ID,
	client.ContainerCommitOptions{Reference: "helloworld"})
	if err != nil {
		log.Fatal(err)
	}

	fmt.Println(commitResp.ID)
	}
```

{{< /tab >}} {{< tab name="Python" >}}

```
import docker
client = docker.from_env()
container = client.containers.run("alpine", ["touch", "/helloworld"], detach=True)
container.wait()
image = container.commit("helloworld")
print(image.id)
```

{{< /tab >}} {{< tab name="HTTP" >}}

```
$ docker run -d alpine touch /helloworld
0888269a9d584f0fa8fc96b3c0d8d57969ceea3a64acf47cd34eebb4744dbc52
$ curl --unix-socket /var/run/docker.sock\
  -X POST "http://localhost/v{{% param "latest_engine_api_version"
%}}/commit?container=0888269a9d&repo=helloworld"
{"Id":"sha256:6c86a5cd4b87f2771648ce619e319f3e508394b5bfc2cdbd2d60f59d52acda6c"}
```

{{< /tab >}} {{< /tabs >}}

□□□ #2
□ □□□ □□□ 2026-10-31 19:40:22 UTC
□ □□□ □□□ 2026-10-31 19:40:24 UTC