debezium

Using Docker with Debezium

Debezium uses Docker in many ways, including within our tutorials, to run databases we test against, and to run the tools that build our website. There are lots of good summaries and tutorials you can follow to learn more about Docker, but we wanted to capture some tips and techniques for getting the most out of Docker when it comes to Debezium.

Docker on Linux, Windows, and OS X

Docker uses Linux containers, and therefore currently runs natively only on Linux and runs on OS X and Windows using those platform's virtualization mechanisms. Make sure that Docker is installed using the latest documentation and tools for Linux, OS X, and Windows. You can either manually start Docker or configure it to run automatically on startup. In either case, the *Docker host* is your local machine.

If you're using these installations, there is no need to read the rest of the document.

Docker Toolbox on Windows and OS X

The older approach for running Docker on Windows and OS X is more complicated, and it requires using Docker Toolbox and its Docker Machine to run the Docker host in a *virtual machine*. Make sure you follow the instructions to install or upgrade Docker Toolbox. Docker Machine supports running multiple virtual machines, but from this point on it is assumed that you are going to run the virtual machine named "default".

The rest of this document explains how to use Docker Machine.

Start Docker

First, check the status of the "default" machine:

```bash
$ docker-machine status default
```

This will return "Running" if it is running, "Stopped" if the "default" machine is not running, or "Host does not exist" if you specify the name of a machine that is not known. If the machine is not running, then start it using:

```bash
$ docker-machine start default
```

This also configures your terminal with several environment variables so any Docker commands you run on your host computer will know how to communicate with the Docker daemon running in the virtual machine.

However, whenever you create a *new terminal*, you will need to run the following command to configure it with the environment variables:

```bash
$ eval $(docker-machine env default)
```

TIP

As of Docker Machine 0.6, you can leave off the name of the machine of many commands. For example:

```bash
$ eval $(docker-machine env)
```

The Docker host

An important concept when working with Docker is the *Docker host*. In order to communicate with software running in a Docker container, one or more of the container's exposed ports must be mapped to ports on the Docker host. Other software can then communicate with the container by using one of those Docker host ports.

If you're running Docker on Linux, then the Docker daemon is running on your machine and your machine is the Docker host, and all mapped ports are on your machine. So for example, if you were to run a web server in a Docker container and map port 80 in the container to port 80 on the Docker host (e.g., your machine), then you can point your browser to `localhost:80`, and the Docker daemon will forward the request to the container and return the response.

Unfortunately, things are not as straightforward when running Docker on Windows or OS X. When you use Docker Machine, the Docker daemon runs in the virtual machine, which means the *virtual machine* is the Docker host. So, if you were to run a web server in a Docker container and map port 80 in the container to port 80 on the Docker host (e.g., the virtual machine), then you must point your browser to the *virtual machine's address* in order to hit the web server — pointing your browser to `localhost:80' will not work.

The following command will display information about the virtual machine named "default":

```bash
$ docker-machine env default
```

and will output something like this:

```bash
export DOCKER_TLS_VERIFY="1"
export DOCKER_HOST="tcp://192.168.99.100:2376"
export DOCKER_CERT_PATH="/Users/jsmith/.docker/machine/machines/default"
export DOCKER_MACHINE_NAME="default"
# Run this command to configure your shell:
# eval $(docker-machine env)
```

The `DOCKER_HOST` environment variable includes the IP address of the virtual machine, which is `192.168.99.100` in our example output. This means that you can point a browser to `http://192.168.99.100:80` to hit the web server running in a container that maps the web server's port to port 80 on the virtual machine. Using `localhost:80` will not work.

Be aware that this address may change whenver you start up the named virtual machine using Docker Machine.

Port forwarding

Rather than pointing your apps or browser to the the specific IP address of the Docker host virtual machine (on Windows and OS X), you can use Docker Machine or Boot2Docker to monitor a specific port on your local machine and forward all requests on that port to the Docker host. For Docker Machine, start a new terminal and run the following commands to forward port 4242:

```bash
$ eval $(docker-machine env)
$ docker-machine ssh default -vnNTL *:4242:$(docker-machine ip):4242
```

or, for Boot2Docker:

```bash
$ boot2docker shellinit
$ boot2docker ssh -vnNTL *:4242:$(boot2docker ip 2>/dev/null):4242
```

Adjust the port number as necessary. Use CTRL-C to stop this port forwarding process when you're finished testing.

Troubleshooting Docker

If you're using Docker Machine, you may sometimes get the following error when running one of the Docker commands:

```
Cannot connect to the Docker daemon. Is the docker daemon running on this host?
```

This means that the commands running in your terminal cannot communicate with the Docker virtual machine (i.e., the Docker daemon), either because it is not running or because the required environment variables in the terminal are not set properly. So first, verify that your Docker machine is indeed running:

```bash
$ docker-machine status default
```

Start the machine if needed, but it if is already running then the error almost certainly means that terminal was not configured to use the machine. Use the following command to do configure the terminal:

```bash
$ eval $(docker-machine env default)
```

This should resolve the error so that you can run the Docker command.