CloudShare Docker-Machine Plugin

Plugin Information
View CloudShare Docker-Machine on the plugin site for more information.

The current version of this plugin may not be safe to use. Please review the following warnings before use:

- CloudShare Docker-Machine Plugin stores credentials in plain text

Execute Docker commands on dedicated docker-machines on CloudShare, instead of running on the Jenkins host itself.

Motivation

By using a dedicated docker-machine for each Jenkins project you enjoy the following benefits:

- Each project gets its own dedicated VM that does the actual building & running of docker images. In other words, you get easy parallelization without using Jenkins slaves.
- You can execute docker-compose based tests without worrying about conflicting published ports.
- Since any docker-compose based test runs in isolation on its own VM, you can easily SSH into it and debug a failed test, if needed, without worrying about disturbing/pausing Jenkins itself.
- No need to worry about docker container/image/volume accumulation and cleanup. The VMs are disposable, and your Jenkins host won't get clogged up with obsolete docker files.
- CloudShare VMs automatically get suspended after there's no more docker activity. You don't need to worry about shutting down slaves when they're not needed to cut costs.
- If your Jenkins actually run in a container, you won't need to mount the docker daemon's socket as a volume and you won't need to run Jenkins as a priviliged container, which is a security concern.

Setup

Requirements

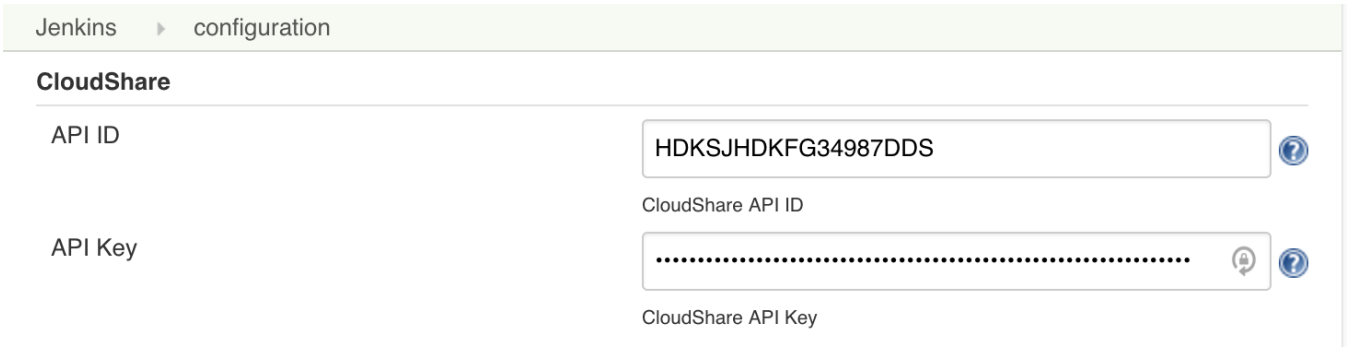
- Docker (strictly speaking, you just need the binaries, the daemon does not have to be running).
- Docker-Machine.
- CloudShare docker-machine driver (must be installed and in Jenkins' **PATH)**.

If you are using Jenkins slaves, make sure the above is installed on all of them, as well as the master node.

Install this plugin through the Jenkins Plugin Manager.

Configure

Once the plugin is installed, enter your CloudShare API key & ID in the global configuration page ("Configure System").



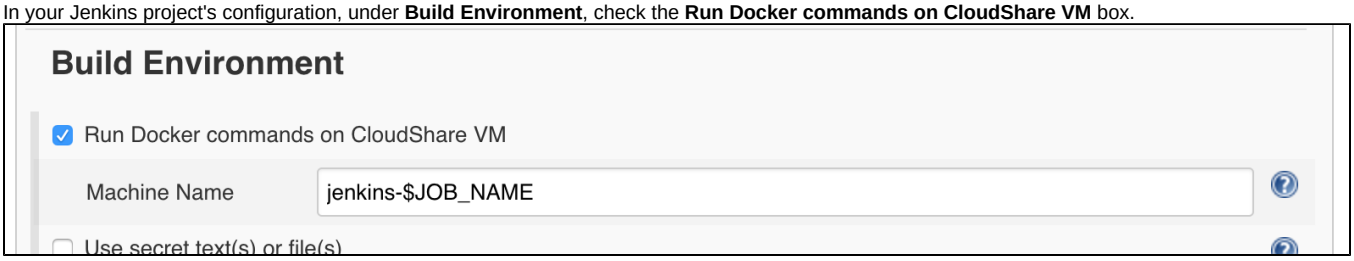
You can obtain the API key in your details page on CloudShare.

Usage

You can enable CloudShare docker-machines for your builds both in classic projects (under **Build Environment**) and as a pipeline step.

Build Environment

In your Jenkins project's configuration, under **Build Environment**, check the **Run Docker commands on CloudShare VM** box.



You can leave the default machine name template as is.

Now every build step that invokes docker (build, run, docker-compose, etc.) will run against a remote CloudShare docker-machine automatically.

Pipeline Step

Another way of achieving the same effect is with the **cloudshareDockerMachine** DSL step.

For example, in this pipeline script:

```
node {
  stage('build') {
    git 'https://github.com/cloudshare/express-ws-chat.git'
    cloudshareDockerMachine {
      sh 'docker-compose -p ${JOB_NAME} build'
    }
  }
}
```

The above **docker-compose** command will run against a dedicated CloudShare docker-machine, and not on the Jenkins host itself.

If you want to modify the name of the CloudShare environment that's created for the project, you can specify:

```
cloudShareDockerMachine(name: 'my-environment') {
// docker stuff
}
```

Outside the scope of the **cloudshareDockerMachine** step any docker command would run against the local Docker daemon.

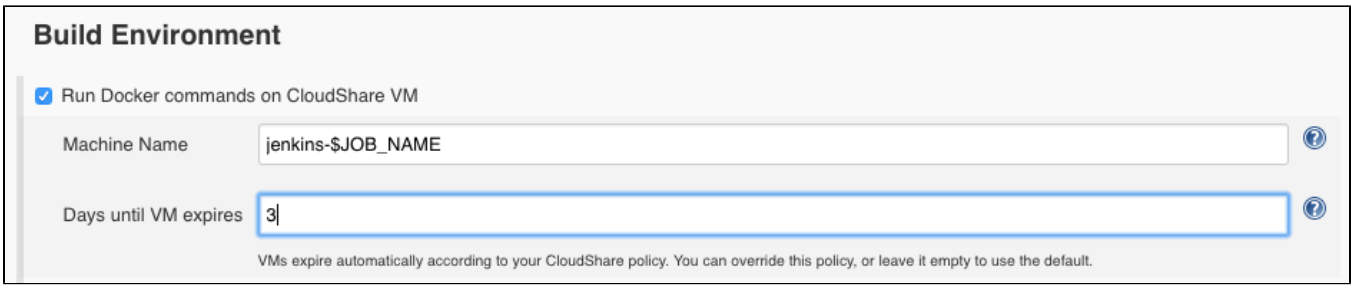
A note about concurrent builds

The default docker-machine name is **jenkins-$JOB_NAME**, which means there will be one CloudShare environment per Jenkins job. If you have enabled **Execute concurrent builds if necessary** in your Jenkins job, be aware that concurrent builds will run on the same CloudShare VM. If this is a problem for you (e.g. you are publishing host ports during the build), consider changing the docker-machine name to something like **jenknis-JOB_NAME-EXECUTOR_NUMBER**, to achieve complete isolation. However, this has the disadvantage of using multiple environments to build the same job, thus incurring more Docker layer cache misses and slower builds. Another solution would be to avoid publishing container ports to the host, for example by using named networks in Docker-Compose, where the project name contains the build number.

Cleanup

A note about concurrent builds
The default docker-machine name is `jenkins-$JOB_NAME`, which means there will be one CloudShare environment per Jenkins job. If you have enabled `Execute concurrent builds if necessary` in your Jenkins job, be aware that concurrent builds will run on the same CloudShare VM. If this is a problem for you (e.g. you are publishing host ports during the build), consider changing the docker-machine name to something like `jenknis-$JOB_NAME-$EXECUTOR_NUMBER`, to achieve complete isolation. However, this has the disadvantage of using multiple environments to uild the same job, thus incurring more Docker layer cache misses and slower builds. Another solution would be to avoid publishing container ports to the host, for example by using named networks in Docker-Compose, where the project name contains the build number.
When a dedicated environment is created per job, you might find yourself accumulating unused environments over time. However, CloudShare environments have a default policy that governs how long they persist. By default environments will be deleted after 14 days. So environment accumulation is not a huge concern, after all.

You can override the default policy for docker-machine environments, if you wish. For example, if you want VMs to hang around for at most 3 days, instead of 14, you can configure the expiry in the project settings (or using the pipeline syntax).


Build Environment
Run Docker commands on CloudShare VM
Machine Name
jenkins-$JOB_NAME
Days until VM expires
3
VMs expire automatically according to your CloudShare policy. You can override this policy, or leave it empty to use the default.