

---

## Skydock - Automagic Service Discovery for Docker

build unknown

### NOTICE

Docker supports DNS based service discovery now. You should use the Docker implementation instead of this project. Skydock was built at a time when Docker did not support DNS discovery or auto registration. I'll keep the repo up for past years and as reference for others but don't use it if you have a recent version of Docker.

Skydock monitors docker events when containers start, stop, die, kill, etc and inserts records into a dynamic DNS server skydns. This allows standard DNS queries for services running inside docker containers. Because lets face it, if you have to modify your application code to work with other service discovery solutions you might as well just give up. DNS just works and it works well.

Also you cannot be expected to modify application code that you don't own. Passing service urls via the cli or in static config files (nginx) will not be possible if your service discovery solution requires a client library just to fetch an IP.

Skydns is a very small and simple server that does DNS for discovery very well. The authors and contributors to skydns helped a lot to make this project possible. Skydns exposes a very simple REST API to add, update, and remove services.

**The Details** When you start a container with docker an event is sent to skydock via the events endpoint. Skydock will then inspect the container's information and add an entry to skydns. We setup skydns to bind it's nameserver to the **docker0** bridge so that it is available to containers. For DNS queries that are not part of the domain registered with skydns for service discovery, skydns will forward the query to an authoritative nameserver. Skydns will return A, AAAA, and SRV records for registered services.

When designing skydock I made the assumption that when in the context of service discovery a client does not care about a specific instance of a service running inside a container. The client cares about the service and in the context of docker a service is defined by an image. We have many images; redis, postgres, our frontend applications, queues, and workers. The URL scheme is designed using this assumption.

**Parts of the URL** \* Domain (domain name to resolve DNS requests for) \* Environment (context of what type of service is running dev, production, qa, uat) \* Service (the actual service name derived from the image name minus the repository crosbymichael/redis -> redis) \* Instance (container's name

---

representing the actual instance of a service) \* Region (currently not used but will be your docker host, digitalocean, ec2; this will be used for multihost)

A typical query will look like this if your domain is `crosbymichael.com` and environment is `production`:

```
1 curl webapp.production.crosbymichael.com
```

The query above will return the IP for a container running the image `webapp`. If we want a specific instance we can prepend the container name to the query. If our `webapp` container's name was `webapp1` we could do this to get the specific container.

```
1 curl webapp1.webapp.production.crosbymichael.com
```

Very simple and easy and no client code was harmed in this demonstration. Wildcard queries are also supported.

```
1 dig @172.17.42.1 "webapp.*.crosbymichael.com"
```

**Setup** So what type of hacks do you have to do to get this running? Nothing, everything runs inside docker containers. You are just two `docker run`s away from bliss.

Ok, I lied. You have to make one change to your docker daemon. You need to run the daemon with the `-dns` flag so that docker injects a specific nameserver into the `/etc/resolv.conf` of each container that is run. First get the IP address of the `docker0` bridge. We will need to know the IP of the bridge so that we can tell skydns to bind it's nameserver to that IP. For this example we will use the ip `172.17.42.1` as the `docker0` bridge.

```
1 # start your daemon with the -dns flag, figure it out...
2 docker -d --bip=172.17.42.1/16 --dns=172.17.42.1 # + what other
  settings you use
```

**Note:** You can also pass the `-dns` flag to individual containers so that the DNS options only apply to specific containers and not everything started by the daemon. But what fun is that?

Now we need to start skydns before our other containers are run or else they will not be able to resolve DNS queries.

```
1 docker pull crosbymichael/skydns
2 docker run -d -p 172.17.42.1:53:53/udp --name skydns crosbymichael/
  skydns -nameserver 8.8.8.8:53 -domain docker
```

We add the name `skydns` to the container and we use `-p` and tell docker to bind skydns port 53/udp to the `docker0` bridge's IP. We give skydns a nameserver of `8.8.8.8:53`. This nameserver is used to

---

forward queries that are not for service discovery to a real nameserver. If you don't know 8.8.8.8 is google's public DNS address.

Next is the `-domain` flag. This is the domain that you want skydns to resolve DNS queries for. In this example I am running docker on my local development machine so I am using the domain name `docker`. Any requests for services `*.docker` will be resolved by skydns for service discovery, all other requests will be forwarded to 8.8.8.8.

Now that skydns is running we can start skydock to bridge the gap between docker and skydns.

```
1 docker pull crosbymichael/skydock
2 docker run -d -v /var/run/docker.sock:/docker.sock --name skydock
  crosbymichael/skydock -ttl 30 -environment dev -s /docker.sock -
  domain docker -name skydns
```

This one is a little more involved but the parts are still simple. First we give it the name of skydock and we bind docker's unix socket into the container. I'm guessing for most, you do not want to service the docker API on a tcp port for containers to reach. If we bind the unix socket into this container we don't have to worry about other containers accessing the API, only skydock. We also add a link for skydns so that skydock knows where to make requests to insert new records. We are pre DNS discovery at this point.

Now we have a few settings to assign to skydock. First is the TTL value that you want all services to have when skydock adds them to DNS. I'm using a TTL value 30 seconds but you can set it higher or lower if needed. Skydock will also start a heartbeat for the service after it is added. You can use the `-beat` flag to set this default interval in seconds for the heartbeat or skydock will set the heartbeat interval to  $TTL - (TTL/4)$ . I know, too complicated.

Next is the `-environment` flag which is the second part of your DNS queries. I set this to `dev` because it is running on my local machine. `-s` is the final option and it just tells skydock where to find docker's unix socket so that it can make requests to docker's API.

Now you're done. Just start containers and use intuitive urls to discover your services. Here is a small example starting a redis server and connecting the redis-cli to that instance of the service. Because it's DNS you can specify the urls on `docker run`.

```
1 # run an instance of redis
2 docker run -d --name redis1 crosbymichael/redis
3 03582c0de0ebb10665678d6ed530ae98bebd7d63dad5e7fb1cd53ffb1f85d91d
4
5 # run the cli and connect to our new instance
6 docker run -t -i crosbymichael/redis-cli -h redis1.redis.dev.docker
7
8 redis.dev.docker:6379> set name koye
9 OK
10 redis.dev.docker:6379> get name
```

---

```
11 "koye"
12 redis.dev.docker:6379>
```

That is, the `redis1` named `crosvmichael/redis` container was available under the hostname `redis1.redis.dev.docker`.

```
1 dig @172.17.42.1 +short redis1.redis.dev.docker
2 172.17.0.4
```

If you were to run additional `crosvmichael/redis` containers, they would all be available under the `redis.dev.docker` hostname.

```
1 docker run -d --name redis2 crosvmichael/redis
2 docker run -d --name redis3 crosvmichael/redis
3
4 dig @172.17.42.1 +short redis.dev.docker
5 172.17.0.4
6 172.17.0.5
7 172.17.0.6
```

**Plugin support** I just added plugin support via `otto` to allow users to write plugins in javascript. Currently only one function uses plugins and that is `createService(container)`. This function takes a container's configuration and converts it into a DNS service. The current functionality is implementing in this javascript function:

```
1 function createService(container) {
2     return {
3         Port: 80,
4         Environment: defaultEnvironment,
5         TTL: defaultTTL,
6         Service: cleanImageName(container.Image),
7         Instance: removeSlash(container.Name),
8         Host: container.NetworkSettings.IPAddress
9     };
10 }
```

Your function must be called `createservice` which takes one object, the container, and must return a service with the fields shown above. In your plugin you have access to the following global variables and functions.

```
1 var defaultEnvironment = "string - the environment from the -
   environment flag";
2 var defaultTTL = 30; // int - the ttl value from the -ttl flag
3
4 function cleanImageName(string) string // cleans the repo and tags of
   the passed parameter returning the result
```

---

```
5 function removeSlash(string) string // removes all / from the passed
   parameter returning the result
```

And that is it. Just add a `createservice` function to a `.js` file then use the `-plugins` flag to enable your new plugin. Plugins are loaded at start so changes made to the functions during the life of skydock are not reflected, you have to restart ( done for performance ).

```
1 docker run -d -v /var/run/docker.sock:/docker.sock -v /myplugins.js:/
  myplugins.js --name skydock --link skydns:skydns crosbymichael/
  skydock -s /docker.sock -domain docker -plugins /myplugins.js
```

Feel free to submit your plugins to this repo under the `plugins/` directory.

## TODO/ROADMAP

- Multihost support
- Handle multiple ports via SRV records

## Bugs

- Please report all skydock bugs on this repository
- Report all skydns bugs here

**License - MIT** Copyright (c) 2014 Michael Crosby. michael@crosbymichael.com

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the “Software”), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED “AS IS”, WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.