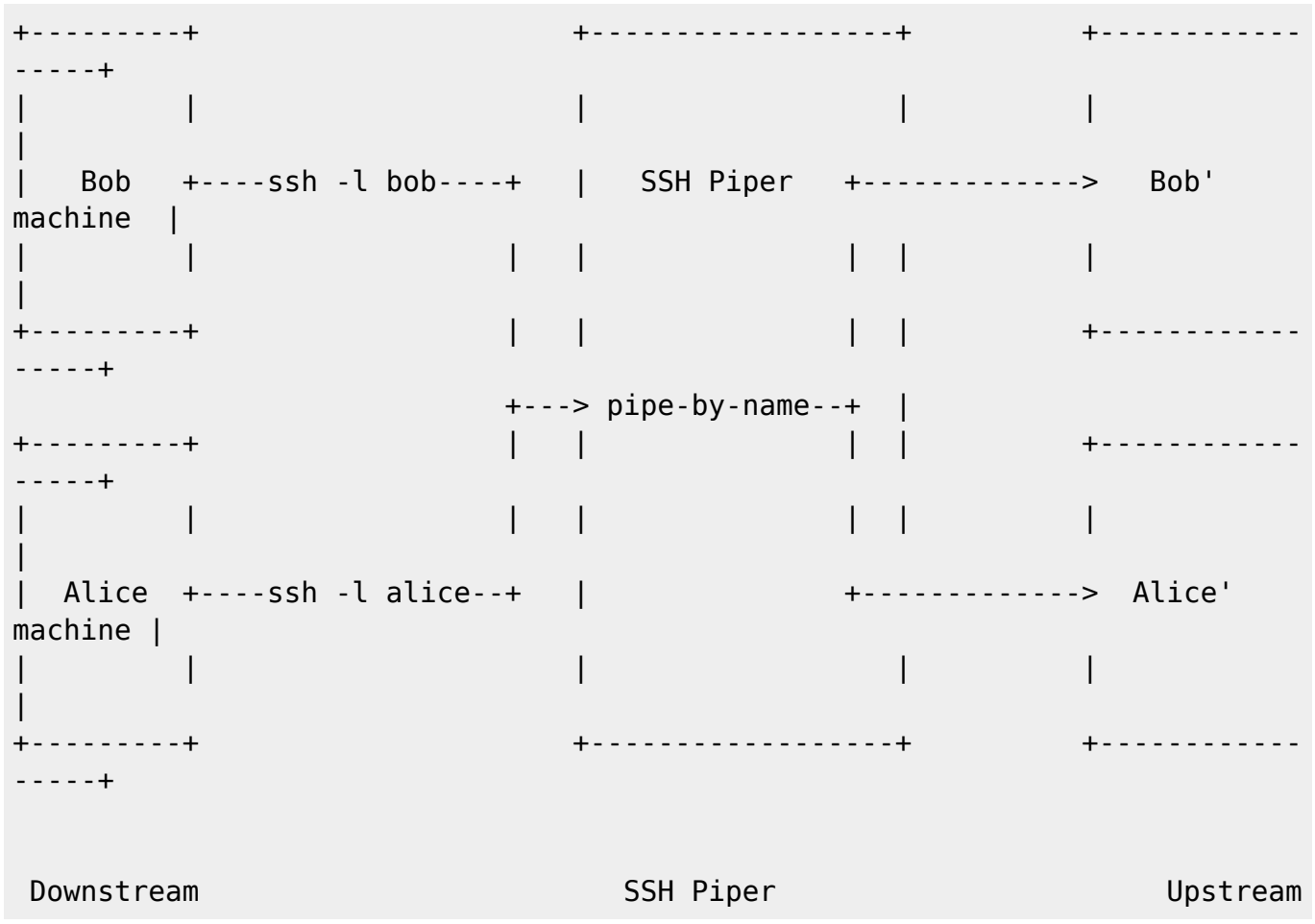
SSH connection routing based on Username - sshpiper

I recently stumbled over a cool tool to allow to route an ssh connection to different servers based on the username that is used to log-in.

the only method to use multiple ssh servers behind a single IP address is usually to assign custom ports to each of those ssh servers and then use those ports instead of port 22 to acces the server. but with SSHpiper we can do something else: we can assign user names to the different servers. Here is the scenario from their own description:



so when bob connects to port 22 of the sshpiper server, his connection is automatically routed to his own server and if alice connects with her username, her connection is terminated by her own server. both bob and alice connect to the same IP/hostname and the same port 22 but with their respective users.

since SSH Piper is a man-in-the-middle it also allows to log ssh sessions so that they can be audited later on in case that's needed.

I think this sounds like a neat solution for various use cases. Like for example in a situation, where each user receives his own instance of a docker container to ssh into some CLI tool to do stuff with. Or another usage scenario might be a situation where you have servers behind a firewall which are managed by different admins and rather than forwarding each server's port 22 to a different public

port, you want to make sure, that only specified users are allowed to access those servers from the outside. for example, to make sure, that if one of the server admins chooses to add a user “test” with password “1234” to their server, his machine is not immediately hacked.

Downside

ther is one quite significant downside hower.. In order to read your username and redirect your connection to another server, SSHpiper needs to act as a **man-in-the-middle** between your server and your client. This means, you client can't authenticate directly on the server with a priveate-public key pair. instead, your client needs to have its public key stored on the SSHpiper server and the SSHpiper server then needs a new private key for which the public key is stored on the Upstream target host in its `authorized_keys`. This ultimately means, that the SSHpiper server holds the necessary authentication keys for all the configured users on all their upstream servers. So in case someone gets a hold of that SSHPiper server, the attacker would automatically get access to all the upstream servers as well. The second problem with that is, that it is not really transparent for the user.. one needs to implement a way that each user can somehow upload their public key to the SSHpiper server and to download the SSHpiper's public key to add it to their upstreams `authorized_keys` file.

password authentication is transparent, but who uses passwords for ssh? However, if passwor auth is your way of doing business, then SSHpiper can even be used to add additional authorization methods not supported by openssh to your session. so thre's another advantage.

Try it

i quickly played around with it.. the simplest way is to run the docker image they provide.. here is how i ran it:

```
mkdir -p ~/piper/log && docker run --rm -d -p 1111:2222 -v
~/piper:/var/sshpiper -v ~/piper/log:/var/log --name piper -e
SSHPIPERD_LOG_PATH="/var/log/sshpiper.log" farmer1992/sshpiperd
```

this will save a log file to `~/piper/log/sshpiperd` so you can track what it's doing

it will also use the `workdir` plugin to store the various upstream configs, so all your users and their config is saved to `~/piper/<username`

now simply use

```
docker exec piper /sshpiperd pipe add -n <username> --upstream-username
<upstream user> -u <hostname> -p <port>
```

to add a new user

to enable key authentication, you can create a new private key using `ssh-keygen` and then save it to `~/piper/<username>/id_rsa` and add the public key to the `authorized_keys` file on the

target upstream server. This key pair won't be used however, until you create a `~/piper/<username>/authorized_keys` file and add the client's public key to it. Only then will piper use key authentication against the upstream host as well. Otherwise it would let anybody in without a password and that would be really bad!

important the private key `id_rsa` as well as the `authorized_keys` file must belong to the piper-user (in case of docker that's `root` and they must not allow group or public access to those files, so macke sure you `chmod 600` those files.

Besides the `workdir` backend, there are also other backends for the upstream config. namely lots of database backents which include `mysql` etc. This would simplify for e

read the manual!

the SSHpiper GitHub page holds alot of information, but it is incomplete. It is a good idea to read the manpage which you can get from your docker container like so:

```
docker exec piper /sshpiperd manpage | man -l -
```