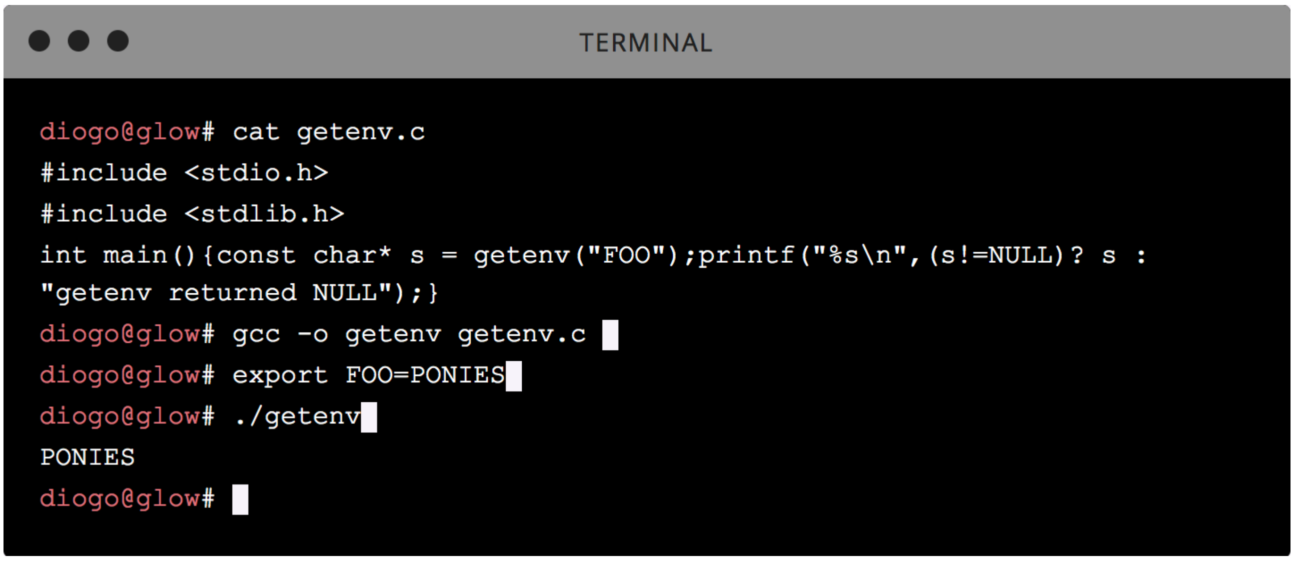
27 MARCH 2017 / DOCKER

Why you shouldn't use ENV variables for secret data

The twelve-factor app manifesto recommends that you pass application configs as ENV variables. However, if your application requires a password, SSH private key, TLS Certificate, or any other kind of sensitive data, you shouldn't pass it alongside your configs.



```
diogo@glow# cat getenv.c
#include <stdio.h>
#include <stdlib.h>
int main(){const char* s = getenv("FOO");printf("%s\n",(s!=NULL)? s :
"getenv returned NULL");}
diogo@glow# gcc -o getenv getenv.c
diogo@glow# export FOO=PONIES
diogo@glow# ./getenv
PONIES
diogo@glow#
```

When you store your secret keys in an environment variable, you are prone to accidentally exposing them—exactly what we want to avoid. Here are a few reasons why ENV variables are bad for secrets:

- Given that the environment is implicitly available to the process, it's hard, if not impossible, to track access and how the contents get exposed (`ps -eww <PID>`).
- It's common to have applications grab the whole environment and print it out for debugging or error reporting. So many secrets get leaked to PagerDuty that they have a well-greased internal process to

scrub them from their infrastructure.

- Environment variables are passed down to child processes, which allows for unintended access. This breaks the principle of least privilege. Imagine that as part of your application, you call to a third-party tool to perform some action—all of a sudden that third-party tool has access to your environment, and god knows what it will do with it.
- When applications crash, it's common for them to store the environment variables in log-files for later debugging. This means plain-text secrets on disk.
- Putting secrets in ENV variables quickly turns into tribal knowledge. New engineers who are not aware of the sensitive nature of specific environment variables will not handle them appropriately/with care (filtering them to sub-processes, etc).

Overall, secrets in ENV variables break the principle of least surprise, are a bad practice, and will lead to the eventual leak of secrets.

If not env variables then what?

At a previous job I helped solve this problem with a really elegant solution: Keywhiz. At Docker we went a step further and built a similar solution directly into Docker itself. If you're using swarm, you now have a trivial way to manage your secrets securely:

```
openssl rand -base64 32 | docker secret create secure-secret -
```

And that's it. You can now use your secret:

```
docker service create --secret="secure-secret" redis:alpine
```

And make your application read the secret contents from the in-memory tmps that gets created under `/run/secrets/secure-secret`:

```
# cat /run/secrets/secure-secret
eHwX8kV8sFt/y30WASgz8kimnKhUkCrt07XMrmewYr8=
```

By integrating secrets into Docker, we were able to deliver a solution for the secrets management that follows these principles:

- Secrets are always encrypted, both in transit and at rest.
- Secrets are difficult to unintentionally leak when consumed by the final application.
- Secrets access adheres to the principle of least-privilege.

Having an easy-to-use, secure-by-default secret distribution mechanism is exactly what developers and ops need to solve the secrets management problem once and for all.