Dynamically provisioning app secrets during container run time

Imran Shaikh
Lead Systems Engineer
YP

2015 USENIX Container Management Summit (UCMS '15)

Assumptions

- Know about Docker and its ecosystem
- Knows about orchestration tools like Mesos or Kubernetes or docker compose etc.
- Probably are exploring running containerized workload
- Or are probably running it but well aware of its limitations

VMs,
Static
provisioning



- You were made to convince that you need 100x servers to run this product
- Vendors have sold you beefed up hardware with unnecessary functionalities
- And you haven't used that functionalities until that become EOL'ed
- You were sanctioned budget for procurement. You use it or loose it.
- But you are nice person
- You don't feel good inside. You feel this has to stop. So, you are looking at container technologies

Fig1: Evolution 1 (Parry, 2012).

Why are we here?

- One of the main things while running containerized workload is how do you deal with secrets
- Secrets are important. More important how to properly secure them in containerized envt. Some of the secrets that you may need are:
  - Database credentials
  - API tokens
  - TLS certificates/keys
  - GPG keys

Baking secrets

- First thing you will do when you need secrets for your app: bake into your image
- Put under .some file, chmod 400, chown root. Thinking that every thing is fine.
- It is the prevalent anti-pattern in security

Baking secrets

- Issues?
- When it is published to any registry, anyone can pull that image and the secrets would be at their disposal.
- None of Github or Dockerhub or your repository is designed to securely store or distribute secrets
- Updating secrets is a tedious job. Updating all the images.
- This could still be ok if you have few number of images, but consider you tie in CI/CI pipeline to your image build process. Now you are managing tons of images
- Accounting for certificate expiration becomes difficult
- Old, EOL/EOS or decommissioned hardware can cause secrets leak

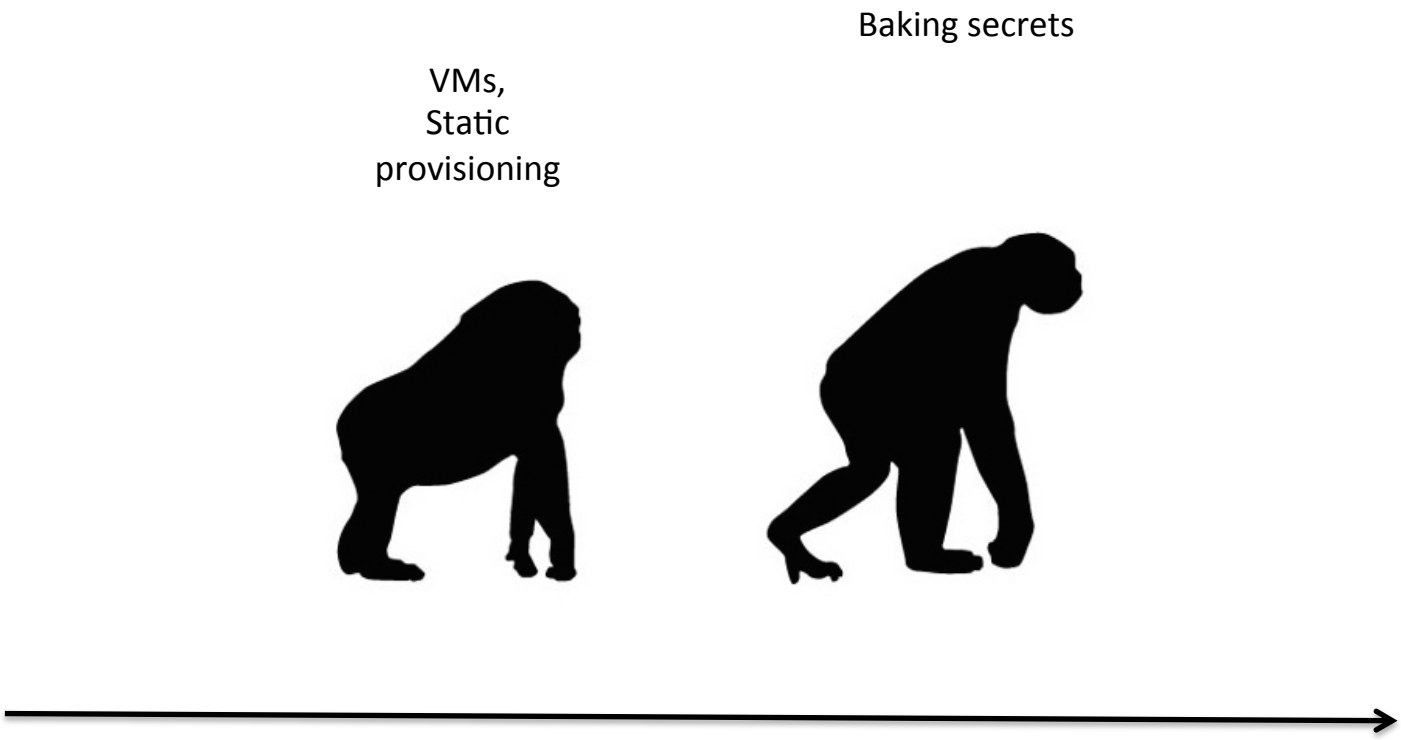


Fig1: Evolution 2 (Parry, 2012).

Environment variables

- Most common way to pass secrets to the applications (more than 90% of people do it)
- Most used because “12 factor app” guidelines suggest it (web apps or cloud apps)
- Easiest but procrastinating issues
  - docker run –it –e “DBUSER=dbuser” –e “DBPASSWD=dbpasswd” myimage /bin/bash

Environment variables

- Issues? (Stijn, 2015; Monica, 2015).
  - Kept in intermediate layers of image and can be easily viewed using “docker inspect”
  - Accessible by all the processes in the image. Can be easily leaked
  - Shared with any linked container
  - Incredibly common having the app grab the whole envt., print it out or even send it part of error report or pager duty.
  - Env. variables are passed down to child processes. Imagine that you call third party tool to perform some action, all of a sudden that third party has access to your envt.
  - Very common for the apps that crashes to store env. variables in log files for debugging.

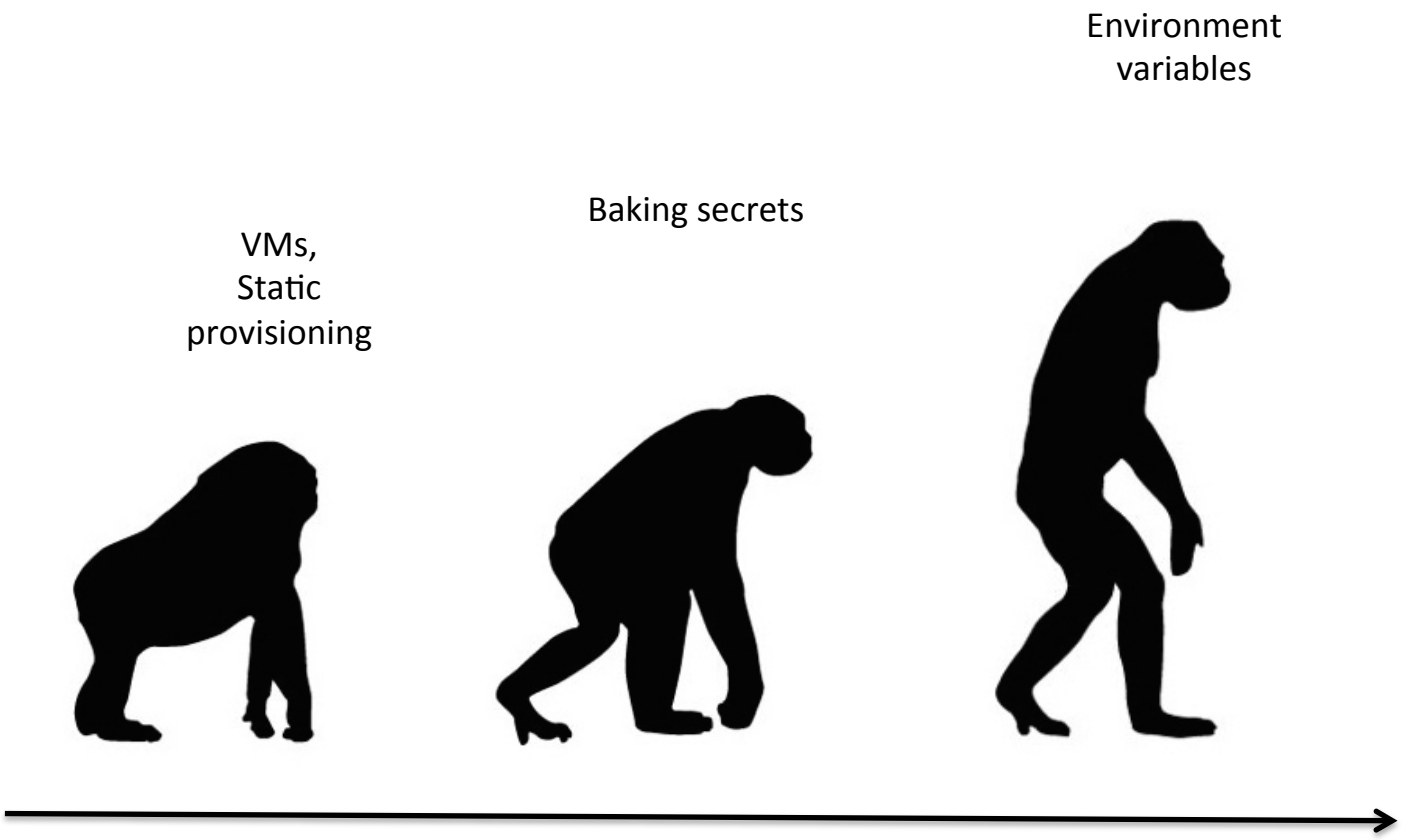


Fig1: Evolution 3 (Parry, 2012).

Volume Mounts

- This is again as straight forward as passing ENV variables.
- You put your secrets in some directory structure on your docker hosts.
- That directory structure can be on local file system, NFS or DFS like CEPH.
- You then mount the right directory inside the container for that particular app
  - docker run –i –t –v /mnt/app1/secrets:/secrets myimage /bin/bash

Volume Mounts

- Issues?
  - Bad design putting all the secrets for all the images on a single machine.
  - Secrets are unencrypted, in plain text

Secrets encryption

- Some people are paranoid about keeping their secrets in plain text
- And they are more paranoid to put that same image with plaintext secrets to private/public docker registries.
- So, they encrypt the secrets using public key & elliptic curve cryptography using tools like EJSON from Shopify and others.
- To decrypt, private keys are hosted on the docker hosts. Production machines are locked down.
- At least, with this way your image is safe from snooping.

Secrets encryption

```
# mkdir -p /opt/ejson/keys

# ejson keygen -w
761d855033dad06a3c5d340a2e4c081059a15c980dd45b7c675112e100ea4340

# vi test.ejson
{
    "_public_key": "761d855033dad06a3c5d340a2e4c081059a15c980dd45b7c675112e100ea4340",
    "database_passwd": "123456"
}


# ejson encrypt test.ejson
# cat test.ejson
{
  "_public_key": "761d855033dad06a3c5d340a2e4c081059a15c980dd45b7c675112e100ea4340",
  "database_passwd": "EJ[1:WGtIJ2Knvu13xZIbE3Nc8CoqEzserRPgsyuDpBuKEAI=:4q/FZZnwOXRfoeQv2
}

# ejson decrypt test.ejson
# cat test.ejson
{
    "_public_key": "761d855033dad06a3c5d340a2e4c081059a15c980dd45b7c675112e100ea4340",
    "database_passwd": "123456"
}
```

Secrets encryption

- Issues?
  - To update secrets, you need to create new images.
  - Solution is static
  - You can still see which private keys are used to decrypt using “docker inspect”
  - Private keys again has to be hosted on the docker host either through local file system, NFS or DFS.

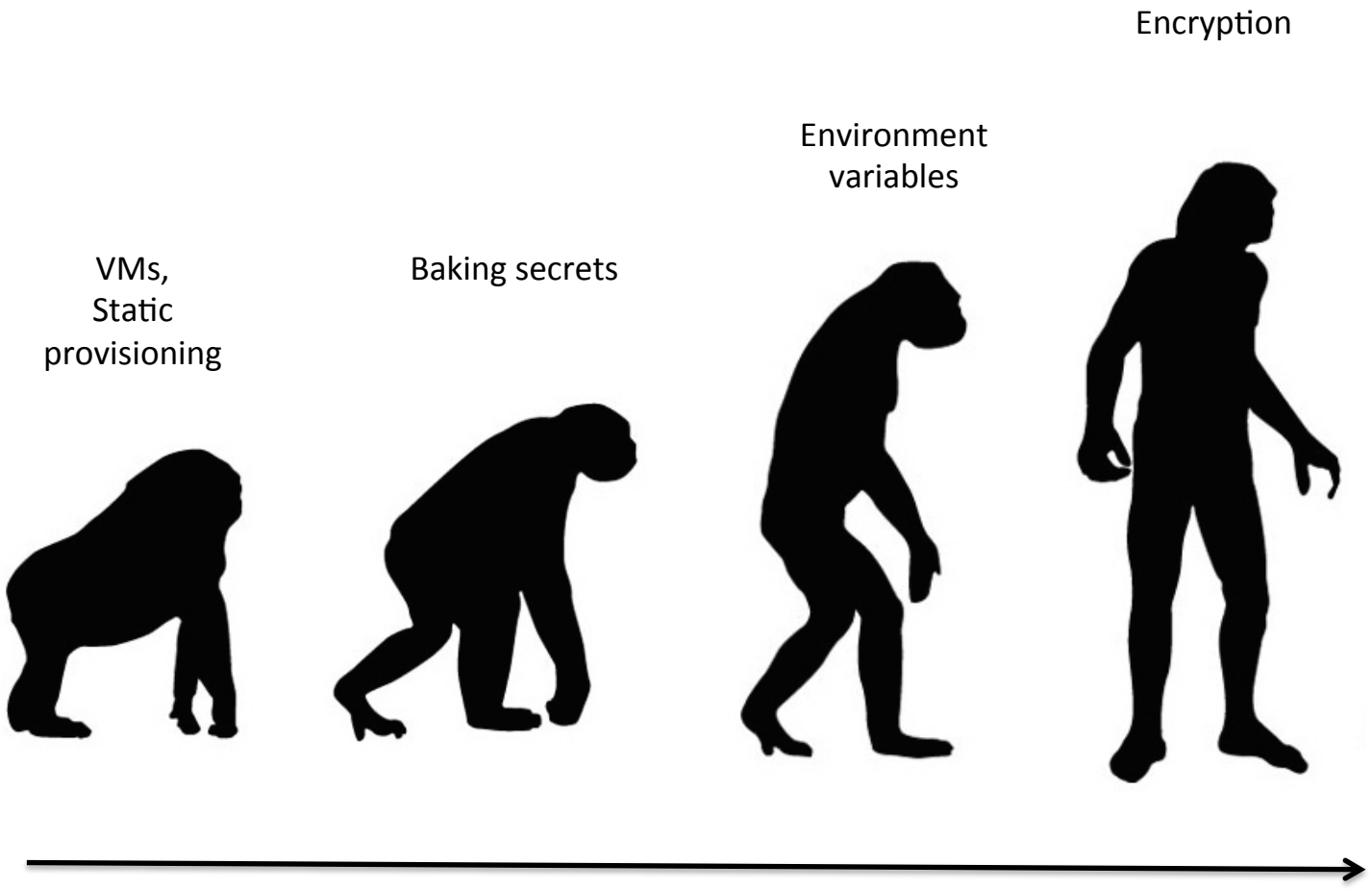
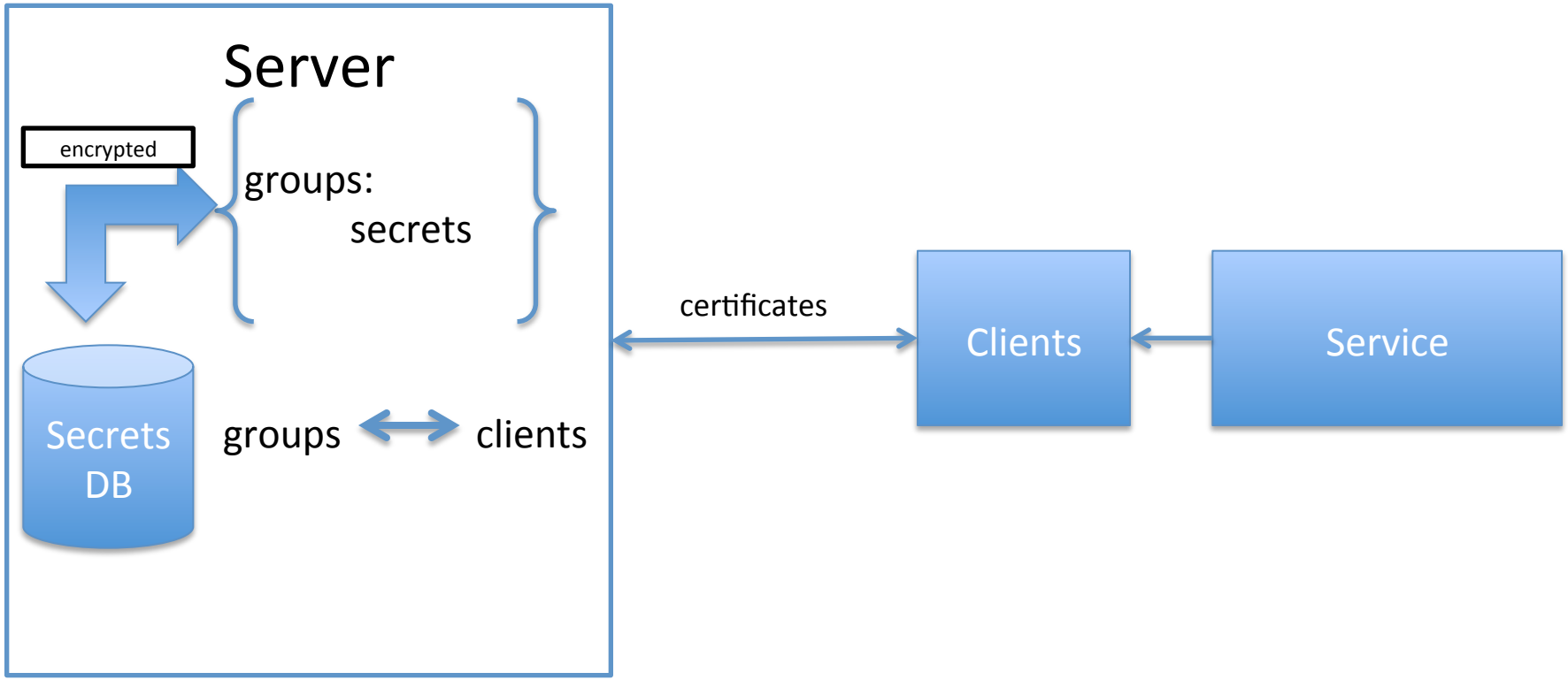


Fig1: Evolution 4 (Parry, 2012).

Secrets store

- There are secrets management and distribution services like: HashiCorp's Vault, Square's Keywhiz and Sneaker (for AWS)
  - Helps you generate and distribute secrets for services
  - Secrets are centrally managed
  - There is auditability with the secret access
  - API based
  - Mostly reliable

Secrets store



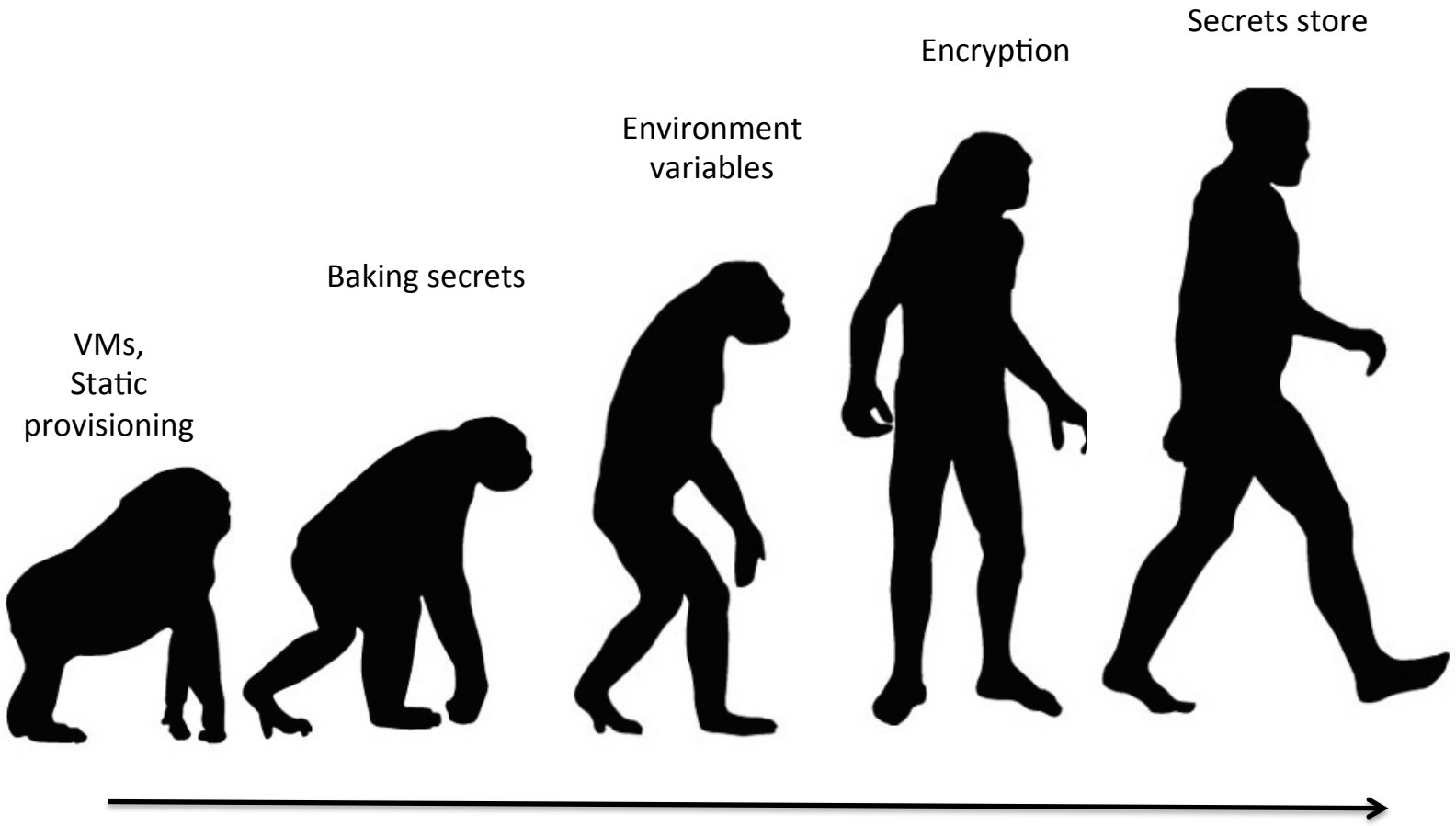


Fig1: Evolution 5 (Parry, 2012).

Build time Secrets

- So far we have talked about run time secrets. But you also need secrets during build time. For ex: In your Dockerfile, you need:
  - To hit some private repo to pull dependency
  - Need private keys for remote SSH connection to pull stuff
  - Build envt. is behind PROXY than deploy envt.
  - Tons of other use cases

HTTP Server serving private keys

docker run –p
172.17.42.1:14242:3000 -v **~/.ssh**:/
vault/.ssh dockito/vault




```
FROM node:0.10.38

# installs Dockito Vault ONVAULT utility
# https://github.com/dockito/vault
RUN apt-get update -y && \
    apt-get install -y curl && \
    curl -L https://raw.githubusercontent.com/dockito/vault/master/ONVAULT \
    > /usr/local/bin/ONVAULT && \
    chmod +x /usr/local/bin/ONVAULT

RUN mkdir -p /usr/src/app
WORKDIR /usr/src/app

COPY package.json /usr/src/app/
RUN ONVAULT npm install --unsafe-perm
COPY . /usr/src/app

CMD [ "npm", "start" ]
```

Vault from Dockito

Build time ENV variables

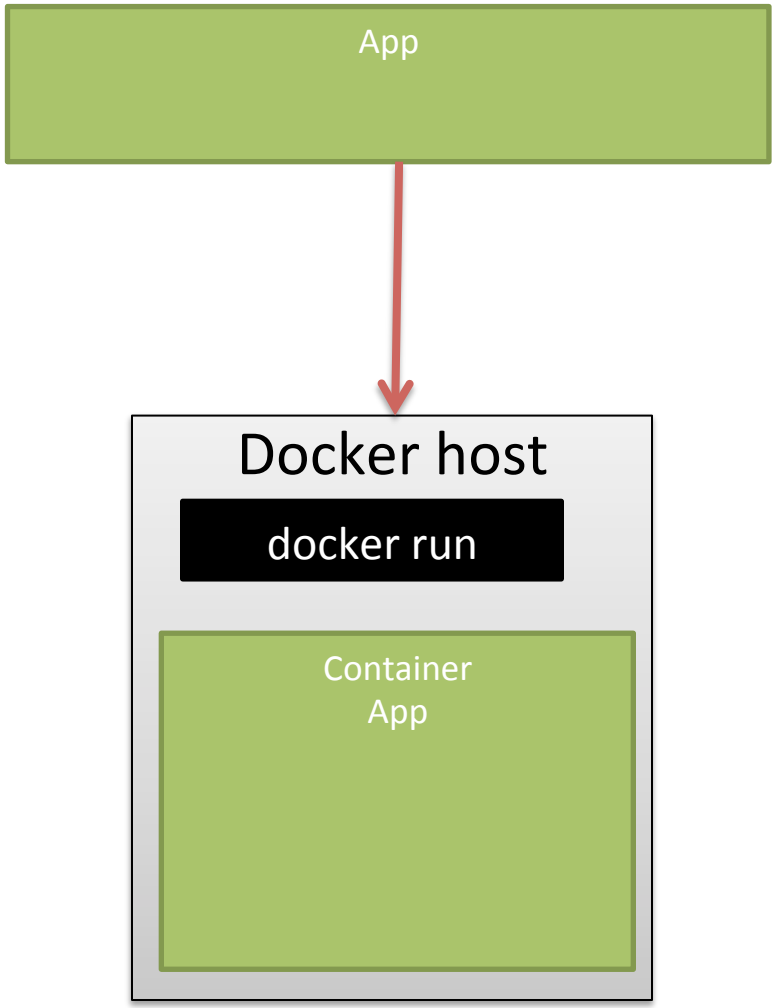
- What are docker layers?
  - Every image consists of series of layers.
  - Docker make use of union file system to combine these layers into single image.
  - One of the reason why docker is lightweight is because of these layers. For example: if you change only version number, then a new layer gets built. It is then slapped over the existing layer.
  - Instead of rebuilding or replacing the whole image, only that layer is added or updated.
- People use ENV variables in Dockerfile to store their secrets

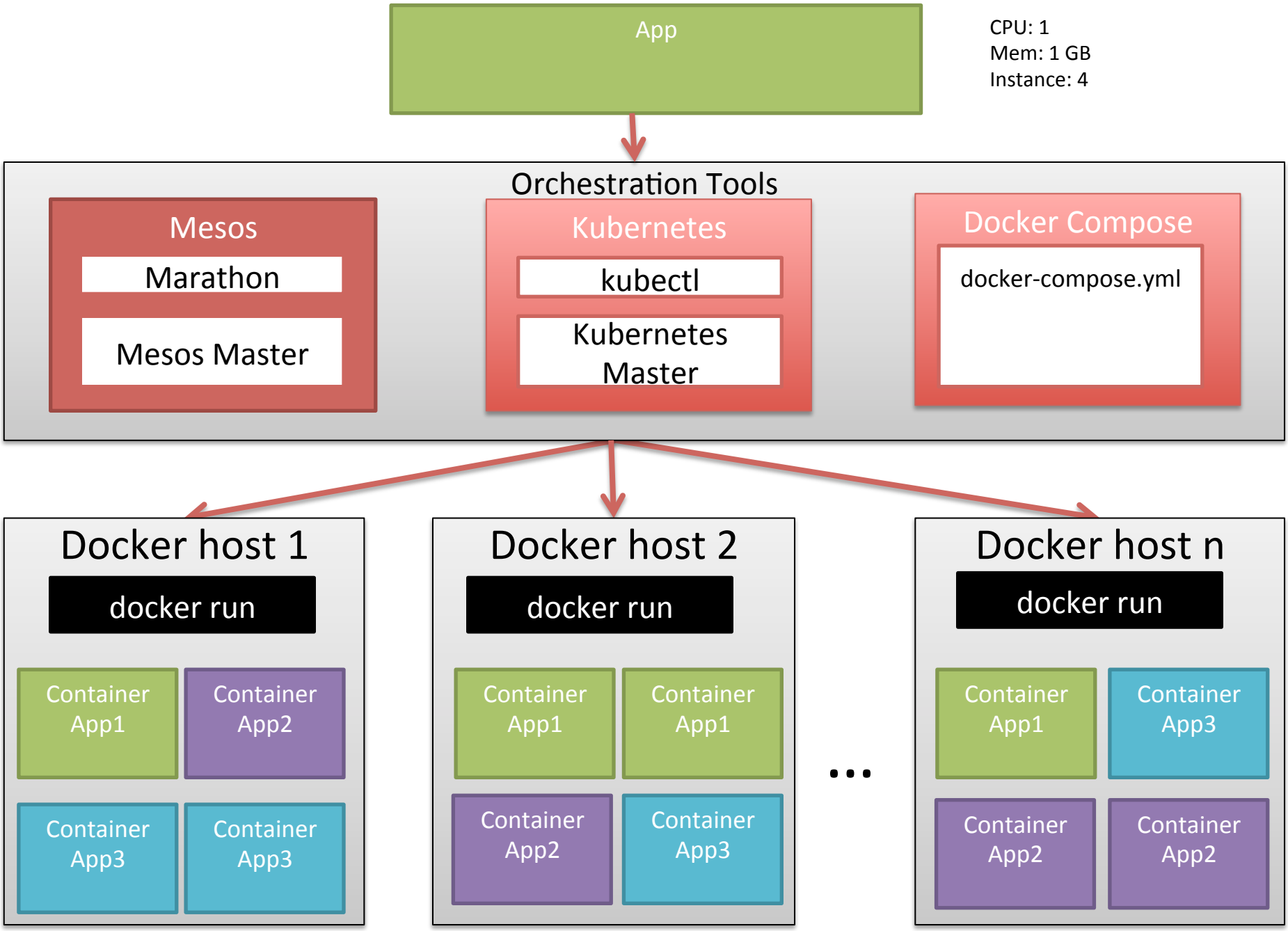
Build time ENV variables

- Issues?
  - Problem is that ENV variable are preserved in Docker intermediate layers
  - So, people use tools like "docker-squash" to flatten all the layers into one final image.
  - But even then the ENV are still preserved in Docker build cache

Lots of PRs

- Proposals: (Stijn, 2015).
  - Add private files support #5836
  - Add secret store #6075
  - Continuation of the docker secret storage feature #6697
  - The Docker Vault" #10310
  - Provide roadmap / design for officially handling secrets. Make injecting secrets pluggable, so that they use existing offerings in this area, for example: Vault, Keywiz, Sneaker


App
Docker host
docker run
Container
App


App
CPU: 1
Mem: 1 GB
Instance: 4
Orchestration Tools
Mesos
Marathon
Mesos Master
Kubernetes
kubectl
Kubernetes Master
Docker Compose
docker-compose.yml
Docker host 1
docker run
Container App1
Container App2
Container App3
Container App3
Docker host 2
docker run
Container App1
Container App1
Container App2
Container App3
...
Docker host n
docker run
Container App1
Container App3
Container App2
Container App2

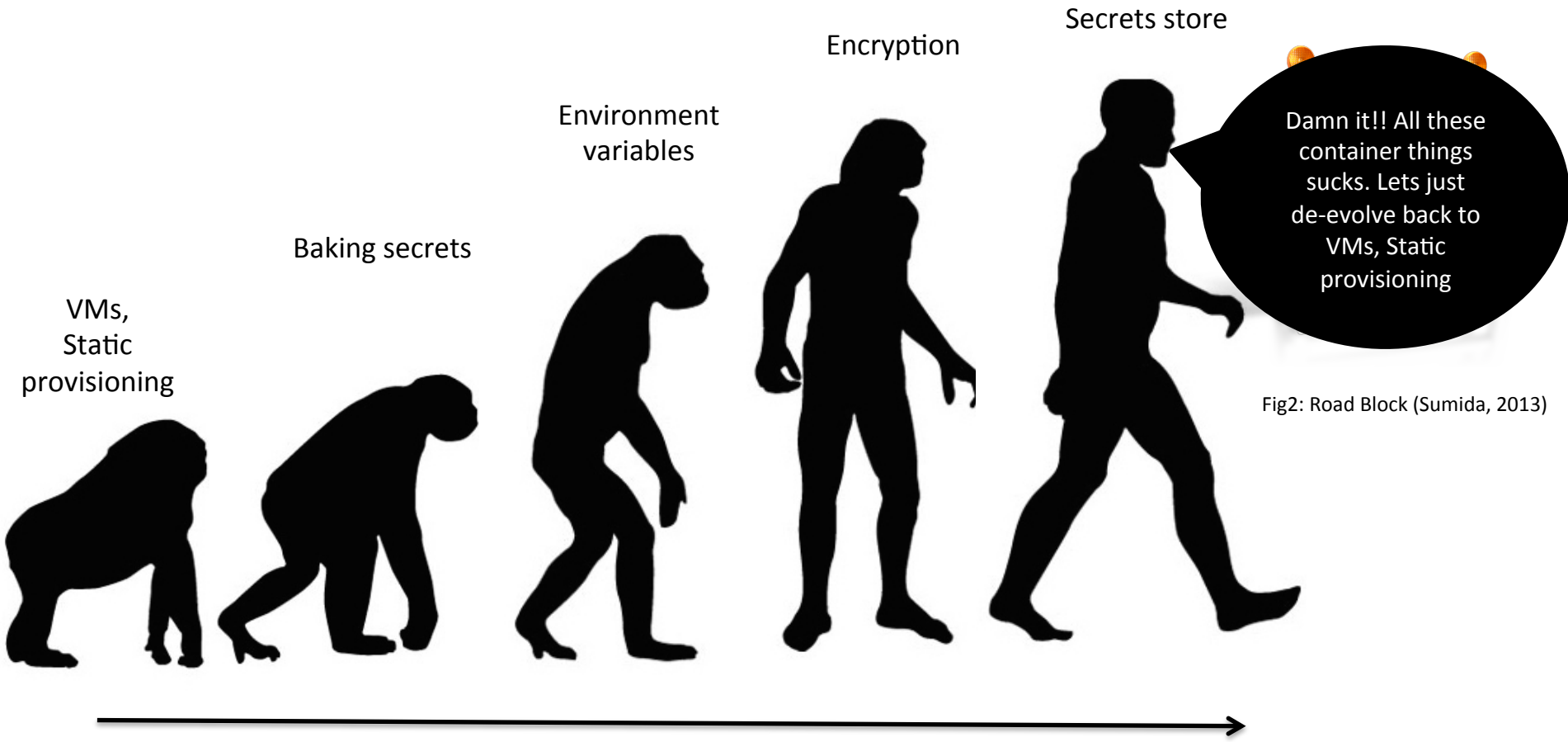
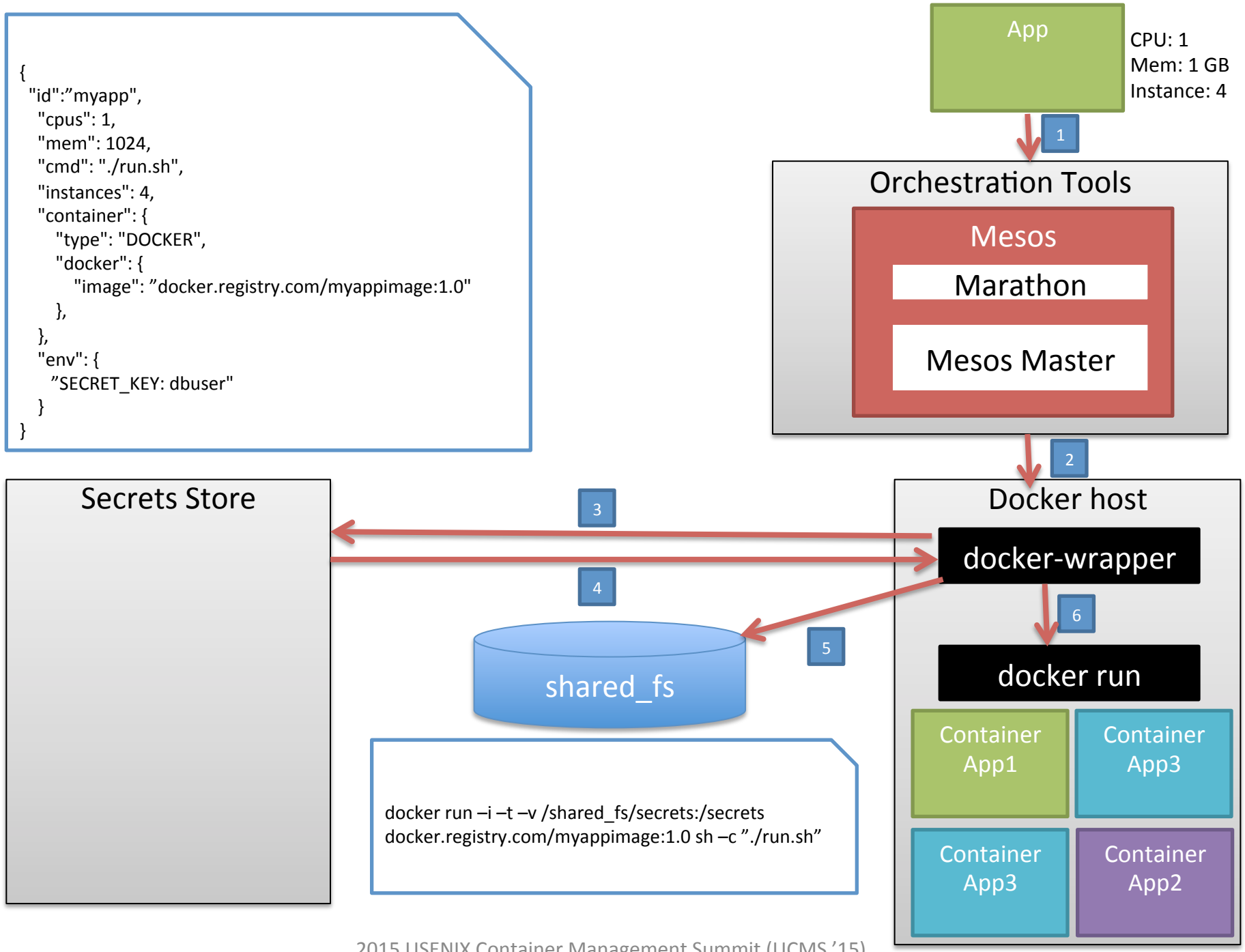
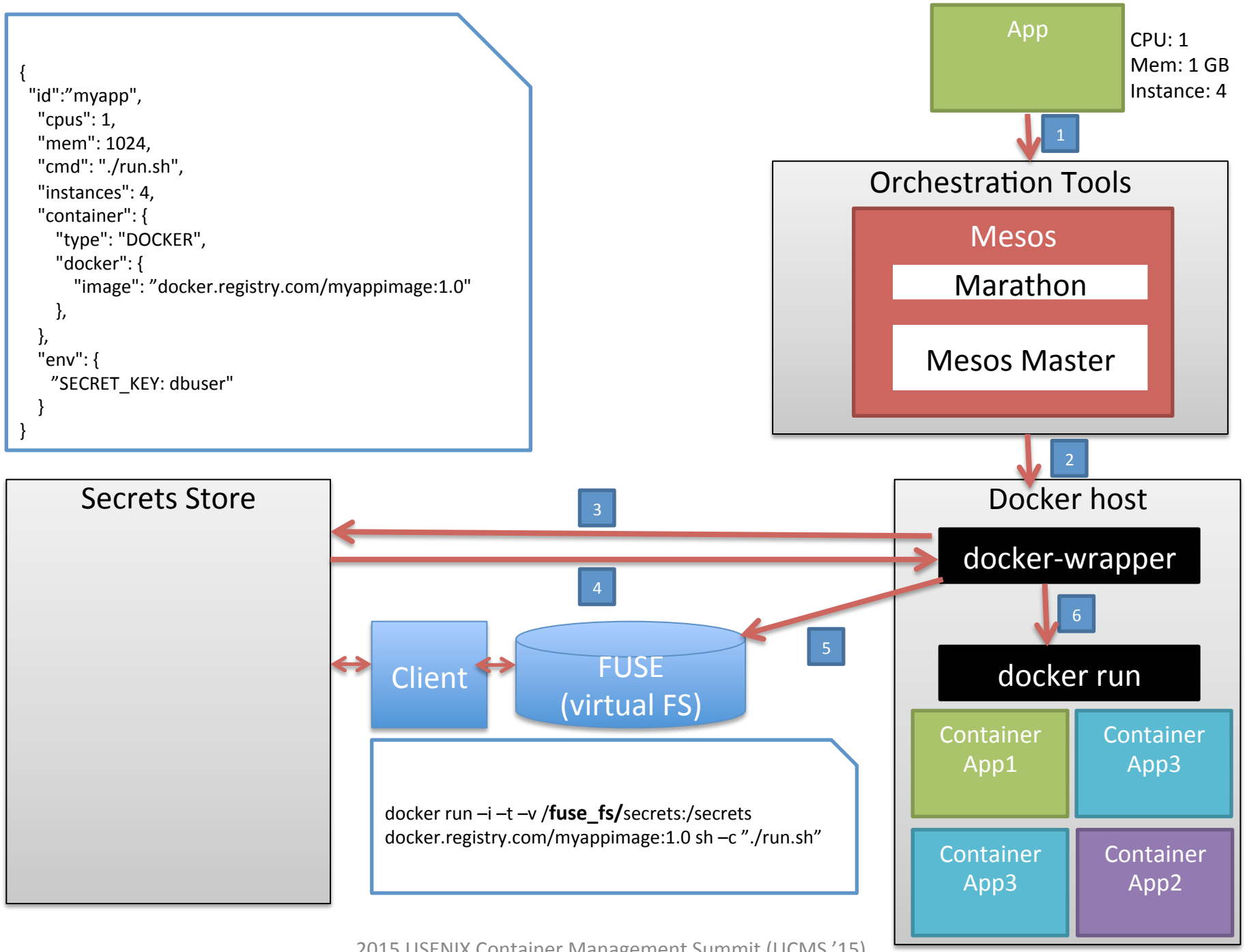


Fig2: Road Block (Sumida, 2013)

Fig1: Secrets Evolution (Parry, 2012).


{
"id":"myapp",
"cpus": 1,
"mem": 1024,
"cmd": "./run.sh",
"instances": 4,
"container": {
"type": "DOCKER",
"docker": {
"image": "docker.registry.com/myappimage:1.0"
},
},
"env": {
"SECRET_KEY: dbuser"
}
}
App
CPU: 1
Mem: 1 GB
Instance: 4
1
Orchestration Tools
Mesos
Marathon
Mesos Master
2
Secrets Store
3
4
Docker host
docker-wrapper
6
docker run
5
shared_fs
Container App1
Container App3
Container App3
Container App2
docker run –i –t –v /shared_fs/secrets:/secrets
docker.registry.com/myappimage:1.0 sh –c "./run.sh"


{
"id":"myapp",
"cpus": 1,
"mem": 1024,
"cmd": "./run.sh",
"instances": 4,
"container": {
"type": "DOCKER",
"docker": {
"image": "docker.registry.com/myappimage:1.0"
},
},
"env": {
"SECRET_KEY: dbuser"
}
}
App
CPU: 1
Mem: 1 GB
Instance: 4
1
Orchestration Tools
Mesos
Marathon
Mesos Master
2
Secrets Store
3
4
Docker host
docker-wrapper
6
docker run
Container App1
Container App3
Container App3
Container App2
Client
FUSE (virtual FS)
5
docker run –i –t –v /fuse_fs/secrets:/secrets
docker.registry.com/myappimage:1.0 sh –c "./run.sh"

Docker plugins

- To extend functionality of docker, people have been proposing docker to add extension. Finally, they came out with docker plugins.
- Presently they support “voldriver” plugin and “network” plugin
- There are few community plugins already available using “voldriver”. One of them being keywhiz for secret stores.
- People can use that but our problem is solving at orchestration level.

Limitations

- It is brittle. Any changes to docker CLI will break our wrapper.
- We are assuming our orchestration tool “ Mesos” will continue to use CLI
  - If they decide to use APIs, then we will put HTTP proxy for Docker API (something like Powerstrip from ClusterHQ)

Conclusion

- We need a standard and a secure way to pass secrets to the running containers in the ephemeral environment
- Maybe docker-wrapper is close to that. We just need to add FUSE support to that
- This can be used with any orchestration tool of your choice
- So, start by grabbing our “docker-wrapper”. It’s already modularize
- Github: https://github.com/yp-engineering/docker-wrapper/

What are we doing at YP Engineering?

- Writing bunch of Mesos supported technologies:
  - Centralized logging
  - Dynamically provisioning Application Secrets to containers
  - Persistent storage
  - Distributed monitoring & alerting
  - Metrics
  - Self-serve
  - QoS on orchestration systems

- Open source contribution:
  www.github.com/yp-engineering

- You can reach us at:
  IRC: chat.freenode.net (#yp-engineering)
  Twitter: @imranshaikh
  Email: ishaikh@yp.com, ishaikh@gmail.com

- Any questions?
  We have a big contingent of a team here.

REFERENCE LIST

- Parry, Wynne. (2012). File:human-evolution.jpg.[Image file]. Retrieved from: http://www.livescience.com/images/i/000/025/831/original/human-evolution.jpg?1332952687
- Sumida, Lynn. (2013). File: roadblcok.png.[Image file]. Retrieved from: http://foodfreedomnow.ca/wp-content/uploads/2013/01/roadblcok.png
- Stijn, Sebastiaan. (2015). Secrets: write-up best practices, do's and don'ts, roadmap #13490. Retrieved from: https://github.com/docker/docker/issues/13490
- Monica, Diogo. (2015). Proposal: Add support for build-time environment variables to the 'build' API #9176. Retrieved from: https://github.com/docker/docker/pull/9176#issuecomment-99542089
- Docker: http://www.docker.com
- Mesos: http://mesos.apache.org
- Kubernetes: http://kubernetes.io
- Docker compose: https://docs.docker.com/compose/
- Ejson: https://github.com/Shopify/ejson
- Vault: https://www.vaultproject.io/
- Keywhiz: http://square.github.io/keywhiz/
- Sneaker: https://github.com/codahale/sneaker
- Dockito vault: https://github.com/dockito/vault
- docker-squash: https://github.com/jwilder/docker-squash
- Powerstrip: https://github.com/ClusterHQ/powerstrip