Making OpenLambda Production Ready

Tyler Caraza-Harter

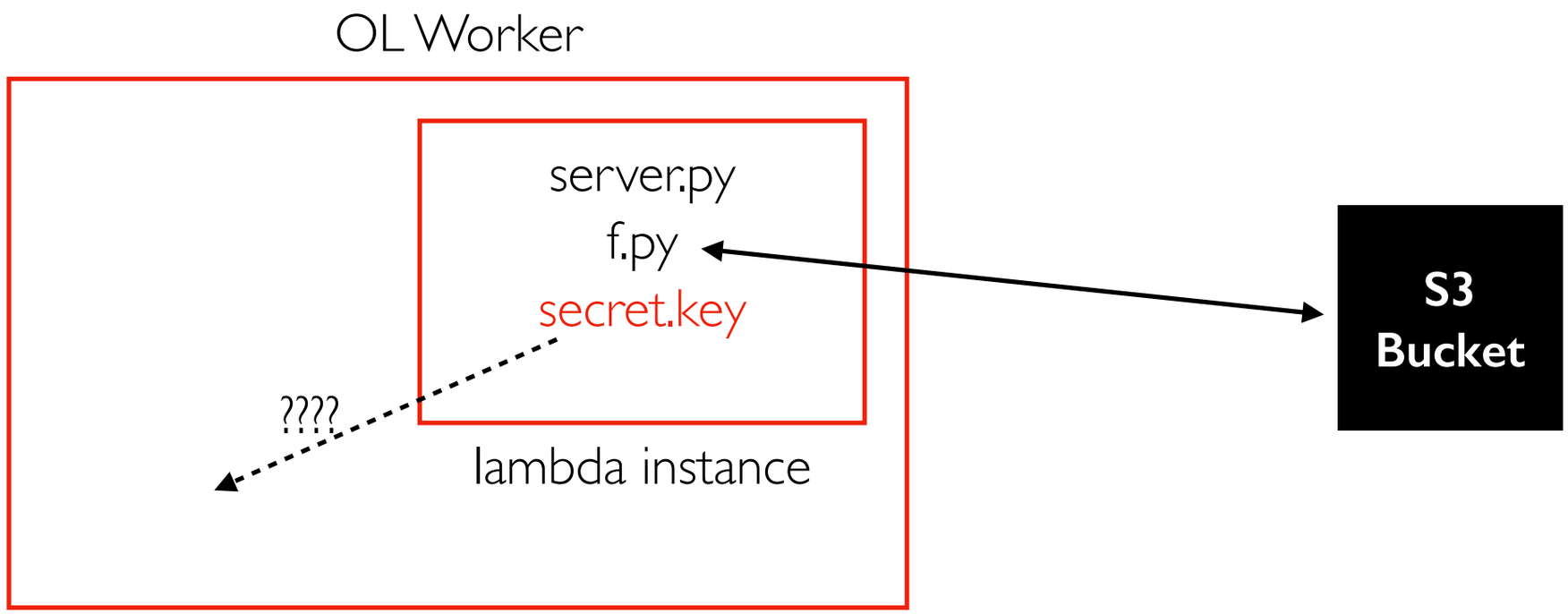
Outline

Security

Resource Management

Usability

Secret/Credential Management

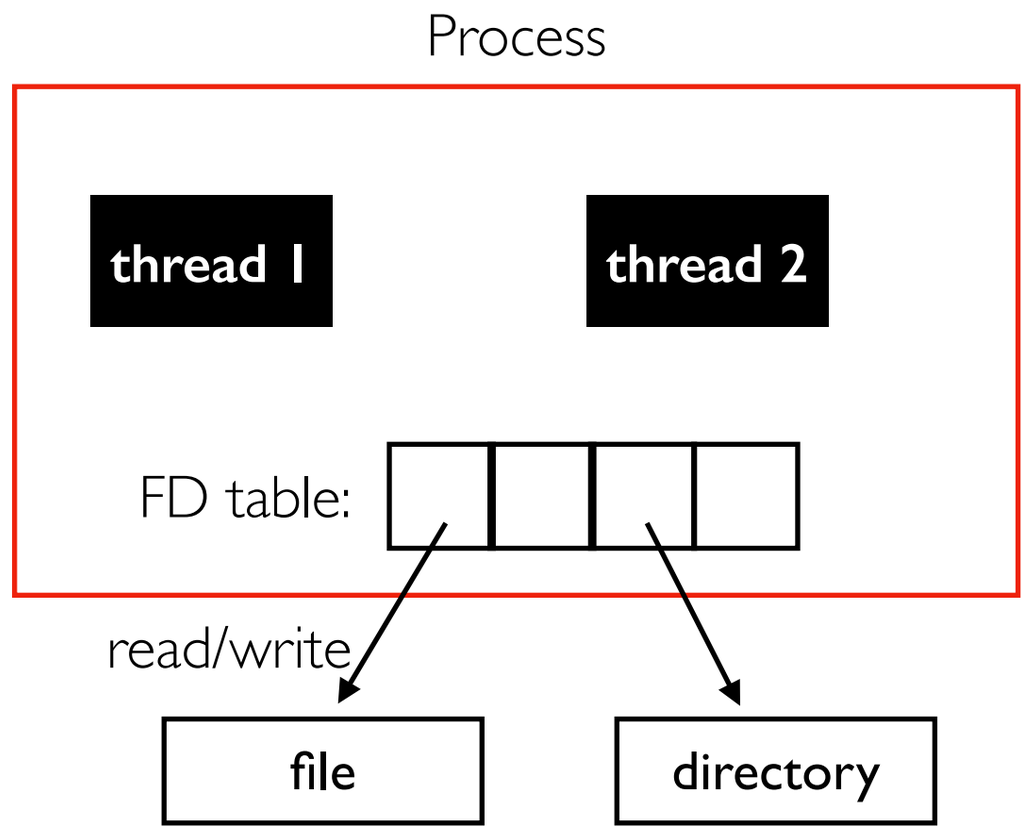


Is there a way to limit risk by moving secrets outside of instances while still giving access?

Something like how AWS Lambda's are given IAM roles that entail access to specific resources?

General challenge: in building an open source serverless platform, how many other services must you partially build to match behavior? (e.g., IAM, API Gateway, ...)

Zygote Suitability Analysis

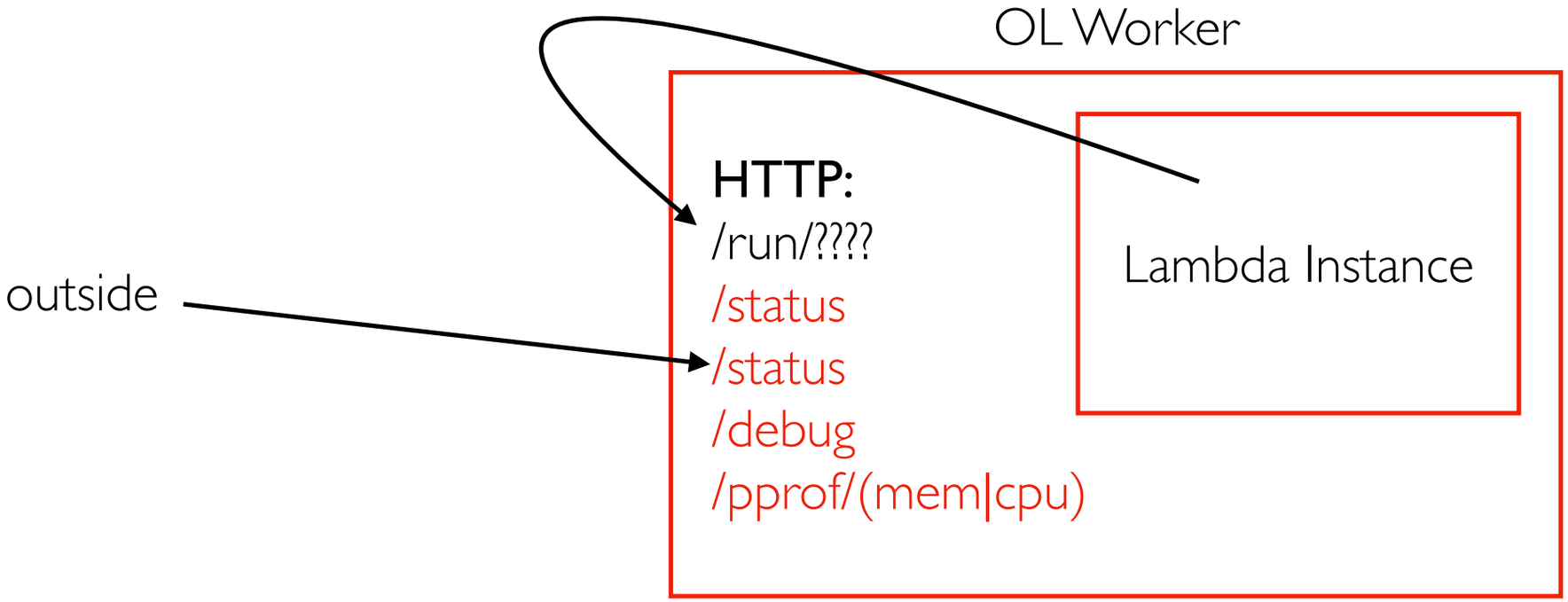


This process would be a bad Zygote, for three reasons:

1. multiple threads (functionality)
2. R/W file (security: can change data seen by siblings)
3. open directory (security: can escape chroot)

Can we automate detecting packages that turn processes into unsuitable zygotes? And make sure those packages are never in a zygote tree?

HTTP Endpoints



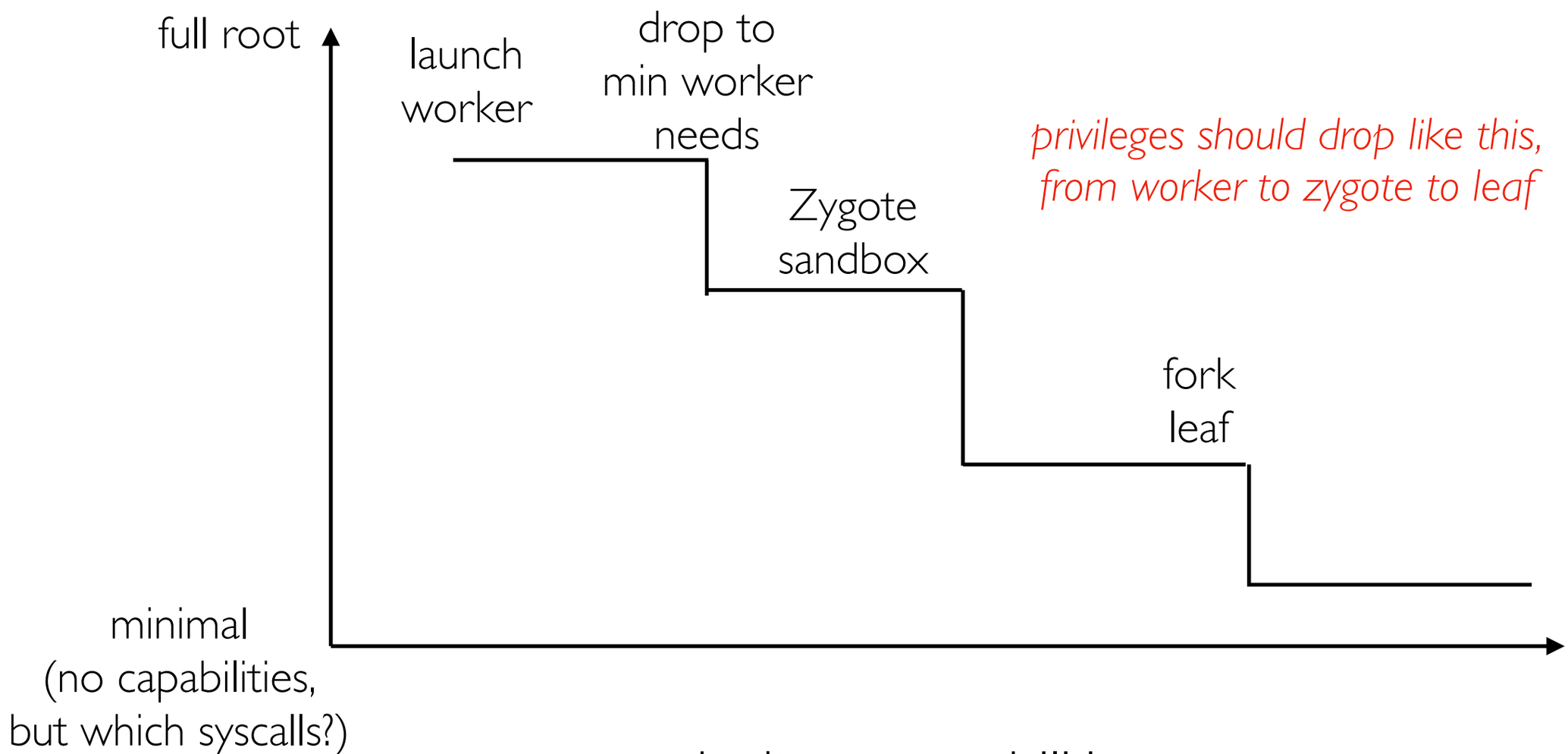
Outsiders should only be able to hit /run/????

Should a Lambda instance be able to hit anything?

Should admin-related endpoints be a UNIX sock file?

Should worker machines have two network interfaces on separate networks? One for general/admin. One for all the lambda instances to share.

seccomp + capabilities



privileges should drop like this, from worker to zygote to leaf

- we don't use capabilities now
- we DO drop seccomp for Sandboxes (Zygotes and leafs)
- we could drop leaves further (Zygotes need to interact with namespaces, but leaves don't)

Outline

Security

Resource Management

Usability

Resources Per Function

Resourses that get allocated when a function is run the first time

1. disk space for lambda code
2. struct in a big map (lambda name => lambda struct)
3. goroutine per function (~8 KB)
4. lambda instances
    A. goroutine
    B. Sandboxes (e.g., SOCK containers)

When functions stop being used, we cleanup 4 (lambda instances). We never cleanup 1-3. It's not nearly as resource intense, but it's an issue with very little RAM (or when we run long enough).

Reduce Goroutine Usage

Resourses that get allocated when a function is run the first time

1. disk space for lambda code
2. struct in a big map (lambda name => lambda struct)
3. goroutine per function (~8 KB)
4. lambda instances
    A. goroutine
    B. Sandboxes (e.g., SOCK containers)

use locks instead

When functions stop being used, we cleanup 4 (lambda instances). We never cleanup 1-3. It's not nearly as resource intense, but it's an issue with very little RAM (or when we run long enough).

Evict State for Long Unused Functions

Resourses that get allocated when a function is run the first time

1. disk space for lambda code
2. struct in a big map (lambda name => lambda struct)
3. goroutine per function (~8 KB)
4. lambda instances
    A. goroutine
    B. Sandboxes (e.g., SOCK containers)

evict

When functions stop being used, we cleanup 4 (lambda instances). We never cleanup 1-3. It's not nearly as resource intense, but it's an issue with very little RAM (or when we run long enough).

Limit Disk Consumption

Resourses that get allocated when a function is run the first time

1. disk space for lambda code *use FS quotas (per user?) to limit code/scratch space*
2. struct in a big map (lambda name => lambda struct)
3. goroutine per function (~8 KB)
4. lambda instances
    A. goroutine
    B. Sandboxes (e.g., SOCK containers)

When functions stop being used, we cleanup 4 (lambda instances). We never cleanup 1-3. It's not nearly as resource intense, but it's an issue with very little RAM (or when we run long enough).

https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/9/html/managing_file_systems/limiting-storage-space-usage-on-ext4-with-quotas_managing-file-systems

Package State

We never evict PyPI packages once installed.

Maybe it's OK for well-provisioned machines?

- could just about host all of PyPI locally
- support multiple versions, so it's not like old pkg versions are harmful.

Accounting

We can limit CPU/RAM/processes per invocations.

More accounting needed:

- how many resources can a function/user cumulatively consume?
- how to report back cumulative usage?

All if this gets even more complicated in cluster deployments (if invocations of the same function are happening on multiple machines concurrently, can you avoid going over a limit without excessive coordination?).

Outline

Security

Resource Management

Usability

cgroup Perf Anomolies

Problem: certain interfaces seemed syncronous (so we assumed that), but have started seeing rare asyncronous behavior.

Solution: retry (but how long?). Need:

- use event interface and wait as long as necessary
- report perf anomolies (instead of panic'ing)
- investigate anomolies: why can you almost always immediately delete a cgroup after trying to kill all the processes, but sometimes it takes >500ms?

Config Files

Lamda .py files currently have comments like these:

ol-install: parso,jedi,idna,chardet,certifi,requests

ol-import: parso,jedi,idna,chardet,certifi,requests,urllib3

We should have an ol.yaml file (perhaps in addition? Still nice to support self-contained single-file lambdas).

cgroup Config

Should expose basically everything cgroup2 offers as a config.

Config per lambda and/or invocation.

Advanced PyPI Requirements

Example METADATA from PyPI Package

```
Requires-Dist: charset-normalizer (<4,>=2)
Requires-Dist: idna (<4,>=2.5)
Requires-Dist: urllib3 (<3,>=1.21.1)
Requires-Dist: certifi (>=2017.4.17)
Provides-Extra: security
Provides-Extra: socks
Requires-Dist: PySocks (!=1.5.7,>=1.5.6) ; extra == 'socks'
Provides-Extra: use_chardet_on_py3
Requires-Dist: chardet (<6,>=3.0.2) ; extra == 'use_chardet_on_py3'
```

Needs

- correctly handle advaced pkg=>pkg dependencies: range versions and extras
- advanced lambda=>pkg depedencies: extras
  (but probably not ranges to avoid non-deterministic behavior?)
- make it easier for lambda function to "freeze" indirect dependency versions (like "pip3 freeze" command)

Function Outputs

Outputs

- return: HTTP response
- stdout: worker.out logs
- stderr: worker.out logs

```
98              cmd.Env = []string{} // for security, DO NOT expose host env to guest
99              cmd.ExtraFiles = cgFiles
100
101             // TODO: route this to a file
102             cmd.Stdout = os.Stdout
103             cmd.Stderr = os.Stderr
104
105             if err := cmd.Start(); err != nil {
106                     return err
107             }
108
109             // Runtimes will fork off a new container,
110             // so this won't block long
111             return cmd.Wait()
```

`freshProc() in src/worker/sandbox/sock.go`

Expose this back to the caller? Function owner? (no "owner" concept currently).

Why did my function die?

Reasons

- bug in code
- exceeded memory limit
- syscall blocked by seccop
- too many processes/threads

It's VERY hard to know what the Linux kernel suddenly killed your process based on cgroup configs.

Need to (a) probe more into what hints we can get from cgroups and (b) surface this to developers.

Possible interface?
$ curl -X POST localhost:5000/run/buggy -d '"'"'
request id 1234 failed
$ curl localhost:5000/debug/buggy/1234/error
The lambda tried to call `mount`, a blocked syscall
$ curl localhost:5000/debug/buggy/1234/stdout
... output here ...