Programming for Hostile Environments

Our adversary: bare metal infrastructure

packet

June 2018

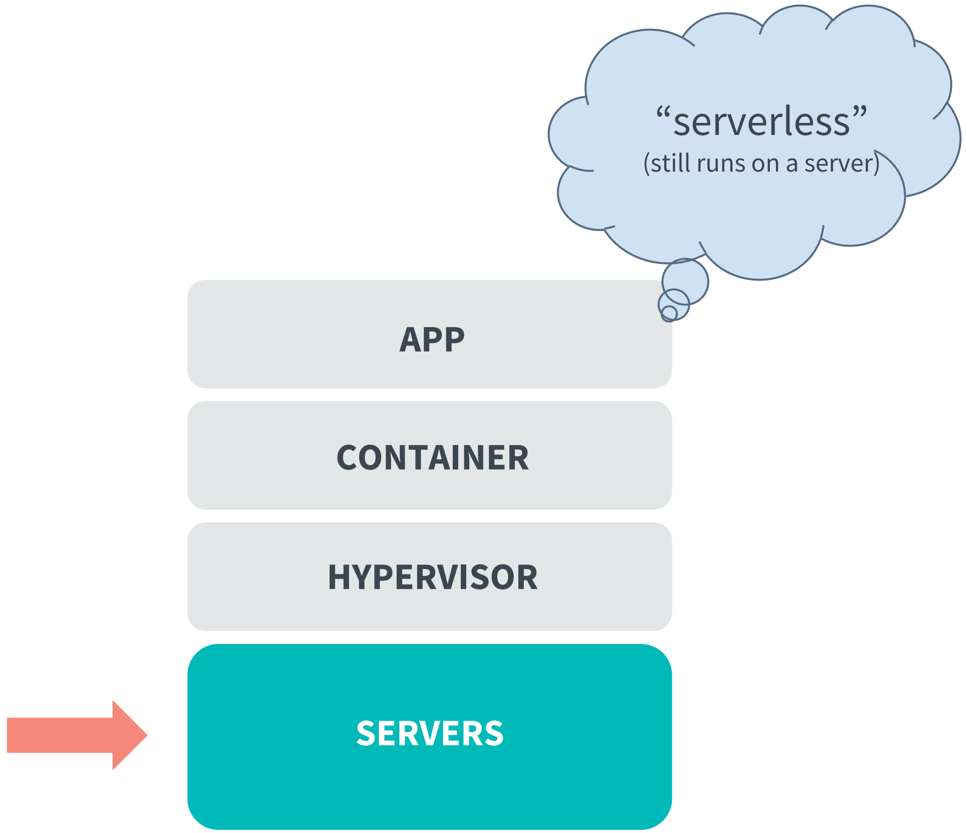
About Me *Nathan Goulding, SVP Engineering*

- ~15 years frontline engineer for infrastructure/cloud and media companies
- Currently lead engineering team at Packet
- me = n+3

What Packet Does

We automate bare metal, physical infrastructure

- Founded in 2014 by infrastructure geeks
- Over 15,000 users
- x86 and ARM CPU architectures
- 16 locations around the world
- 20 supported operating systems
- 50,000 installs per month



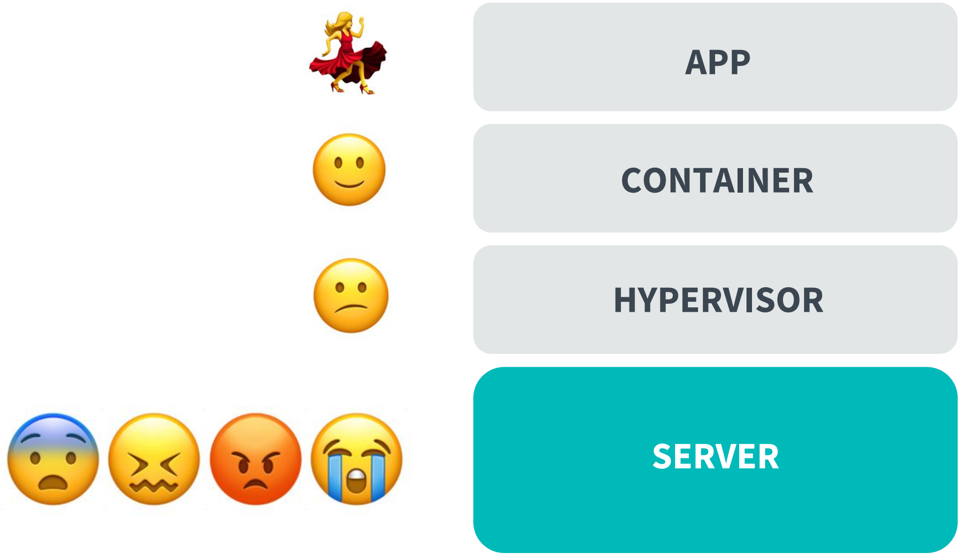
“DRIVERLESS”

Programming for Hostile Environments

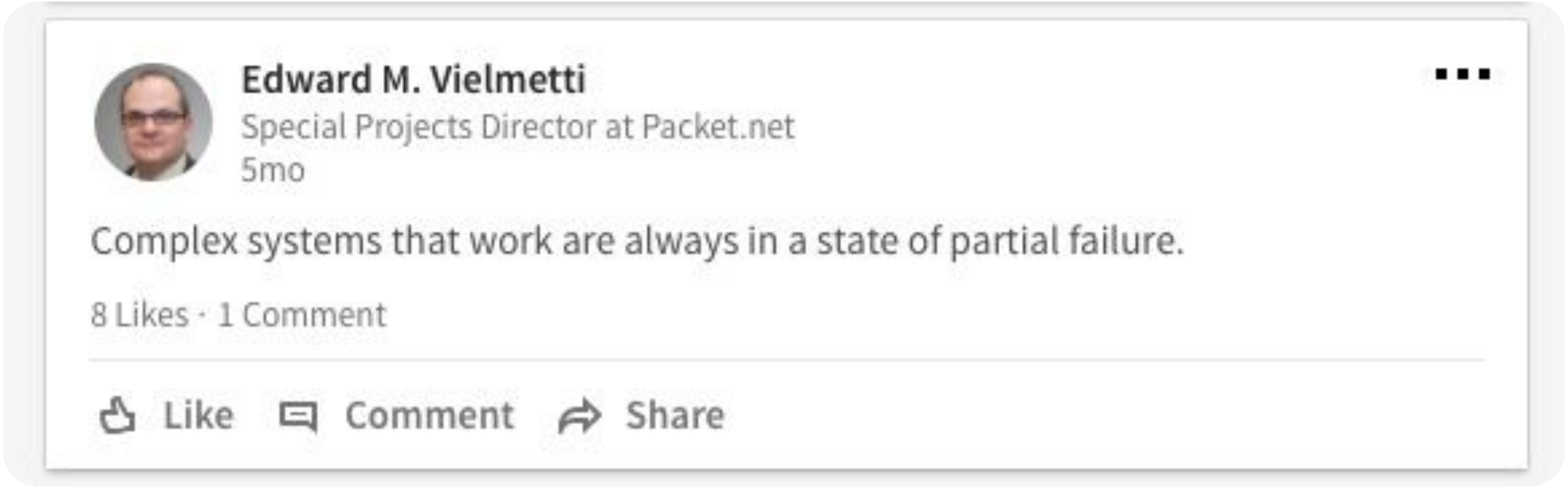
Topics we'll cover:

- Transitioning from monolith (ruby) to microservices (golang)
- Turning antipatterns into patterns
- Applied best practices
- Goals we set for ourselves
- Ephemeral nanoservices

Hostility of the Environment



The Problem, Abstract



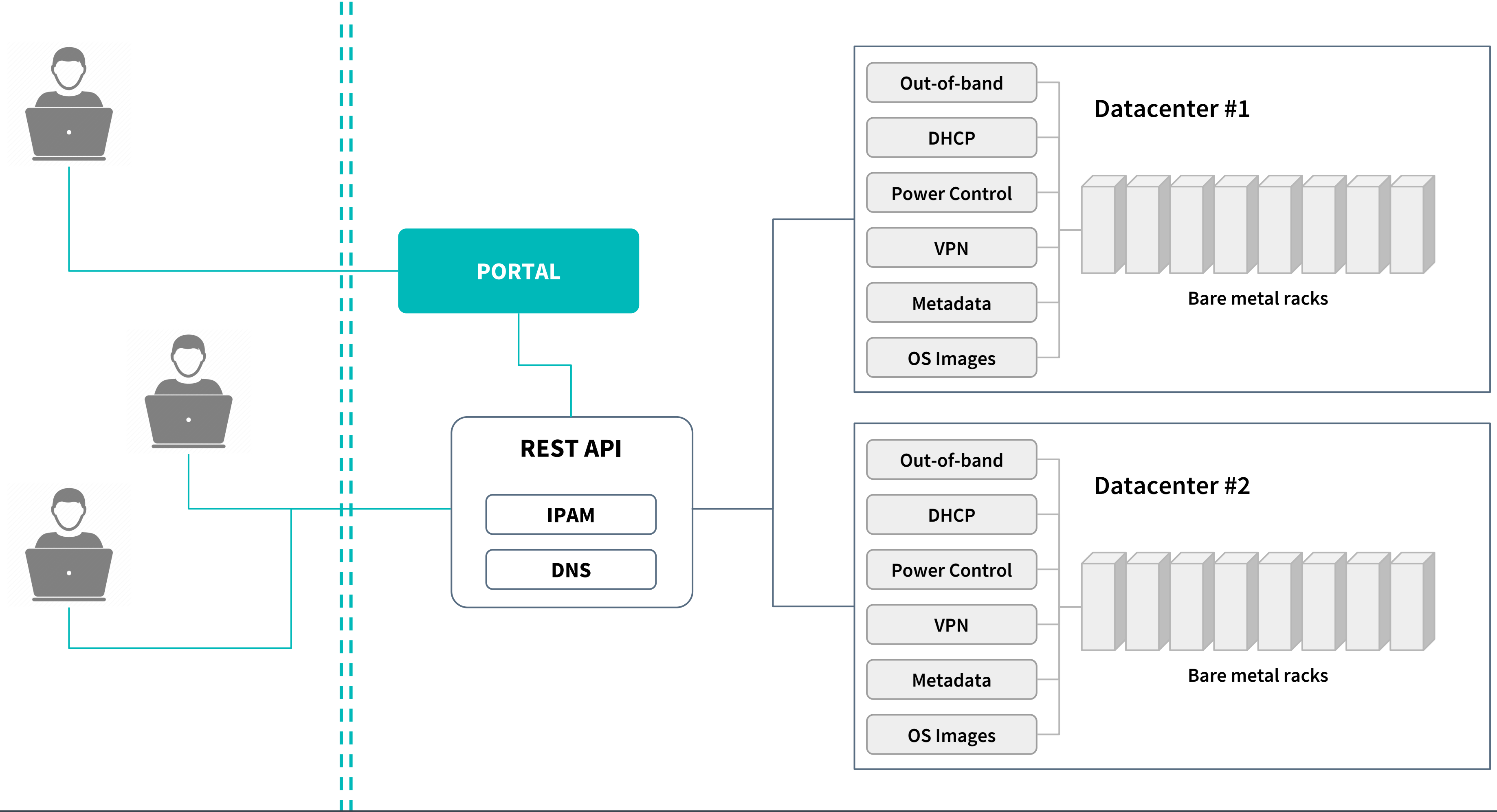
Edward M. Vielmetti

Special Projects Director at Packet.net

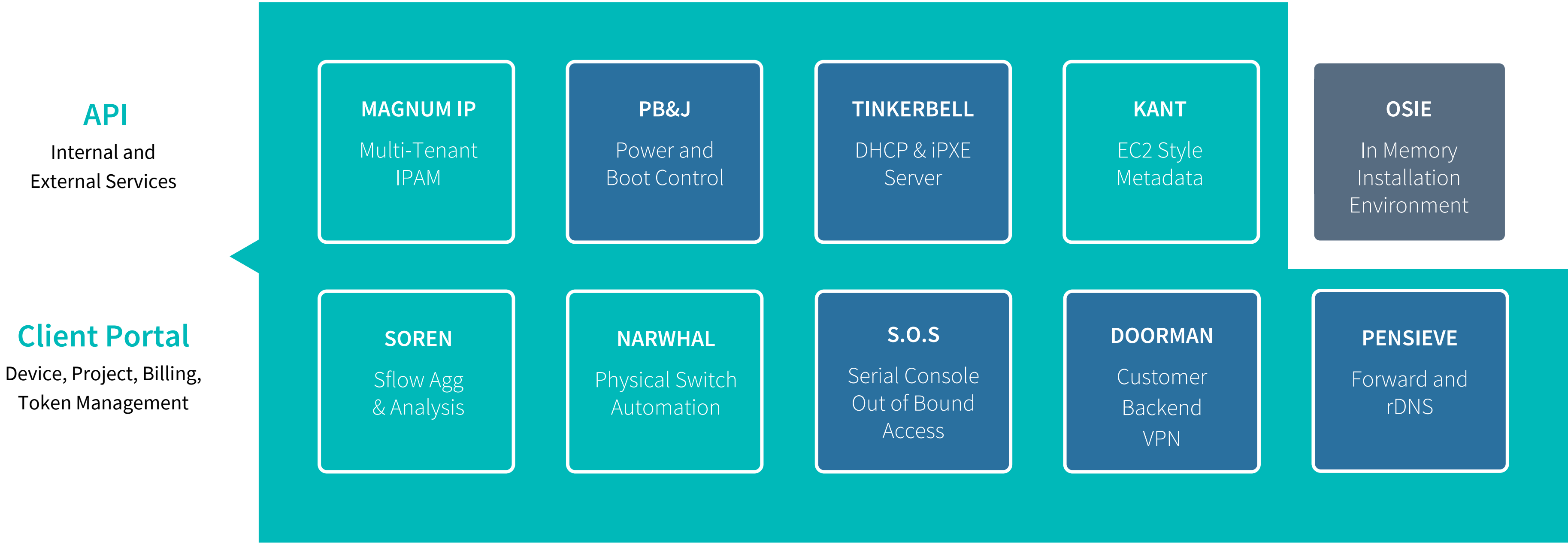
5mo

Complex systems that work are always in a state of partial failure.

8 Likes · 1 Comment


PORTAL
REST API
IPAM
DNS
Out-of-band
DHCP
Power Control
VPN
Metadata
OS Images
Datacenter #1
Bare metal racks
Out-of-band
DHCP
Power Control
VPN
Metadata
OS Images
Datacenter #2
Bare metal racks

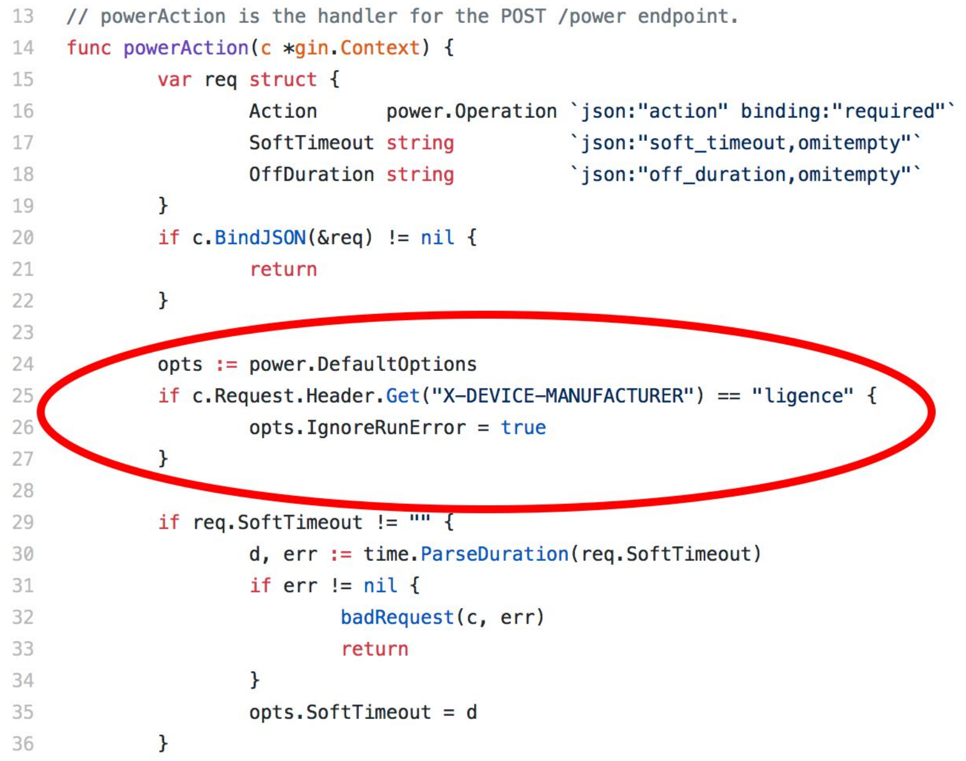
From monolith to microservices



Moving to golang

- Compiled
- Static typing
- Very little “magic”
- The best of prior programming languages minus the cruft

An emerging pattern

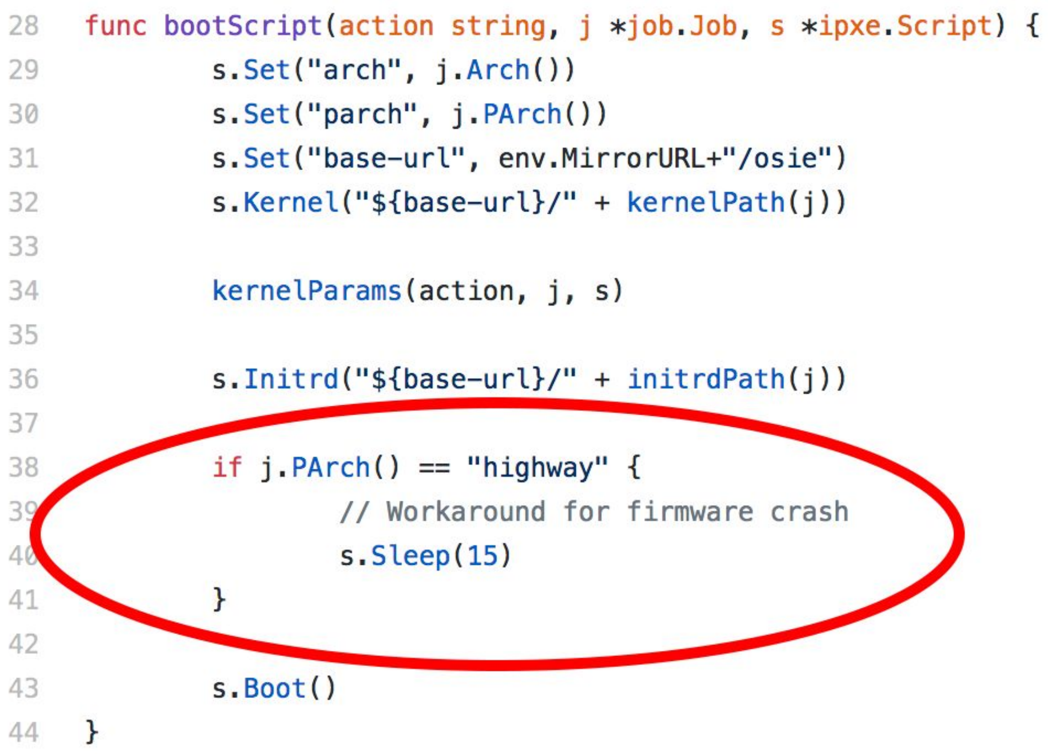


```go
// powerAction is the handler for the POST /power endpoint.
func powerAction(c *gin.Context) {
	var req struct {
		Action      power.Operation `json:"action" binding:"required"`
		SoftTimeout string          `json:"soft_timeout,omitempty"`
		OffDuration string          `json:"off_duration,omitempty"`
	}
	if c.BindJSON(&req) != nil {
		return
	}

	opts := power.DefaultOptions
	if c.Request.Header.Get("X-DEVICE-MANUFACTURER") == "ligence" {
		opts.IgnoreRunError = true
	}

	if req.SoftTimeout != "" {
		d, err := time.ParseDuration(req.SoftTimeout)
		if err != nil {
			badRequest(c, err)
			return
		}
		opts.SoftTimeout = d
	}
```

An emerging pattern



```go
func bootScript(action string, j *job.Job, s *ipxe.Script) {
	s.Set("arch", j.Arch())
	s.Set("parch", j.PArch())
	s.Set("base-url", env.MirrorURL+"/osie")
	s.Kernel("${base-url}/" + kernelPath(j))

	kernelParams(action, j, s)

	s.Initrd("${base-url}/" + initrdPath(j))

	if j.PArch() == "highway" {
		// Workaround for firmware crash
		s.Sleep(15)
	}

	s.Boot()
}
```

Best Practices, in Practice

#1 - gRPC for communication / rpc

#2 - Get your data as close to where you need it as quickly as possible

#3 - Don’t hide code you don’t like

#1 gRPC for communication / rpc

- Handles backoff / retry
- Straightforward service definition for request / response
- Streaming data and authentication via SSL
- Paradigm for dealing with message format changes

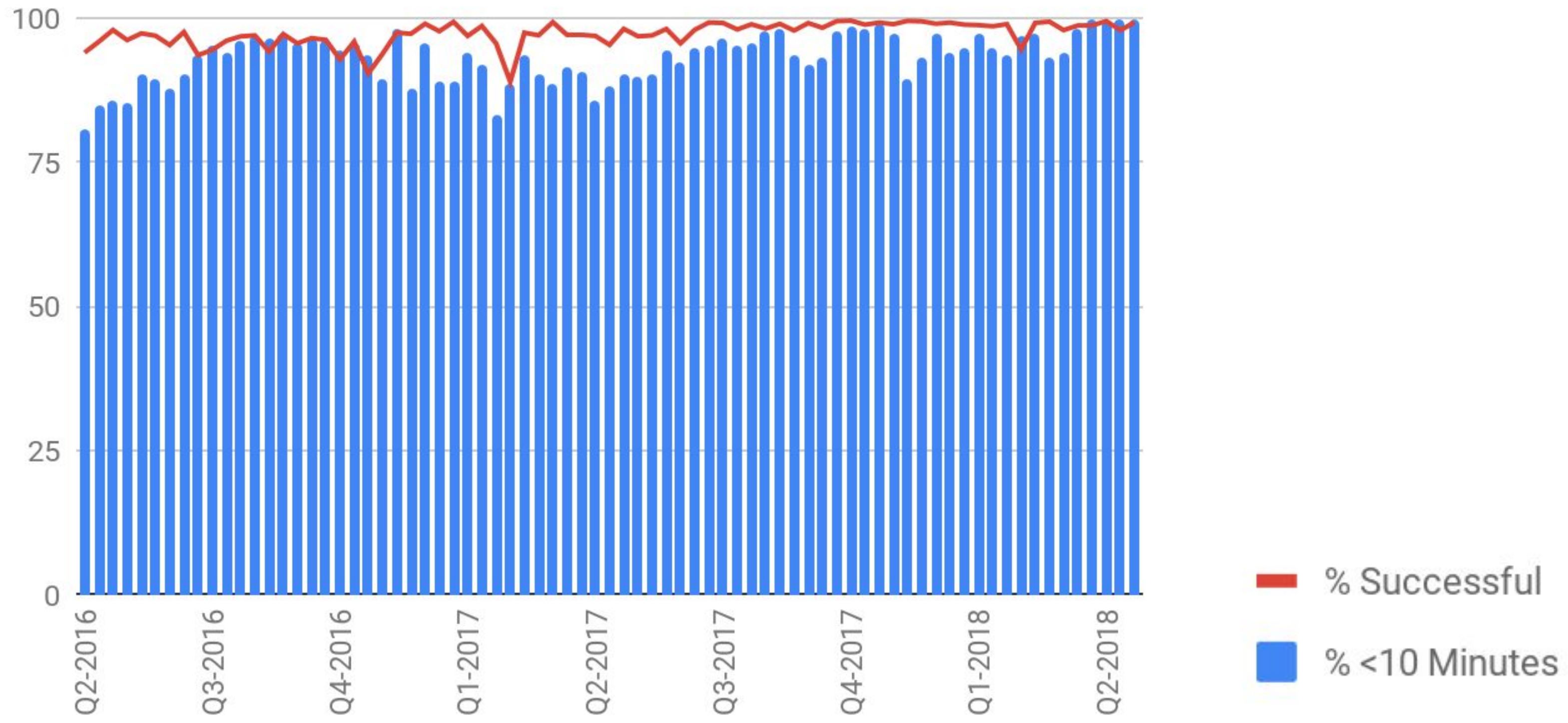
#2 Get data close to where it needs to be, quickly

- The network is unreliable, the network is unreliable, the network is unreliable
- Speed up access times + experience for everyone
- Be careful of “I’ll just request it (remotely) whenever I need it”

#3 Don’t hide code you don’t like

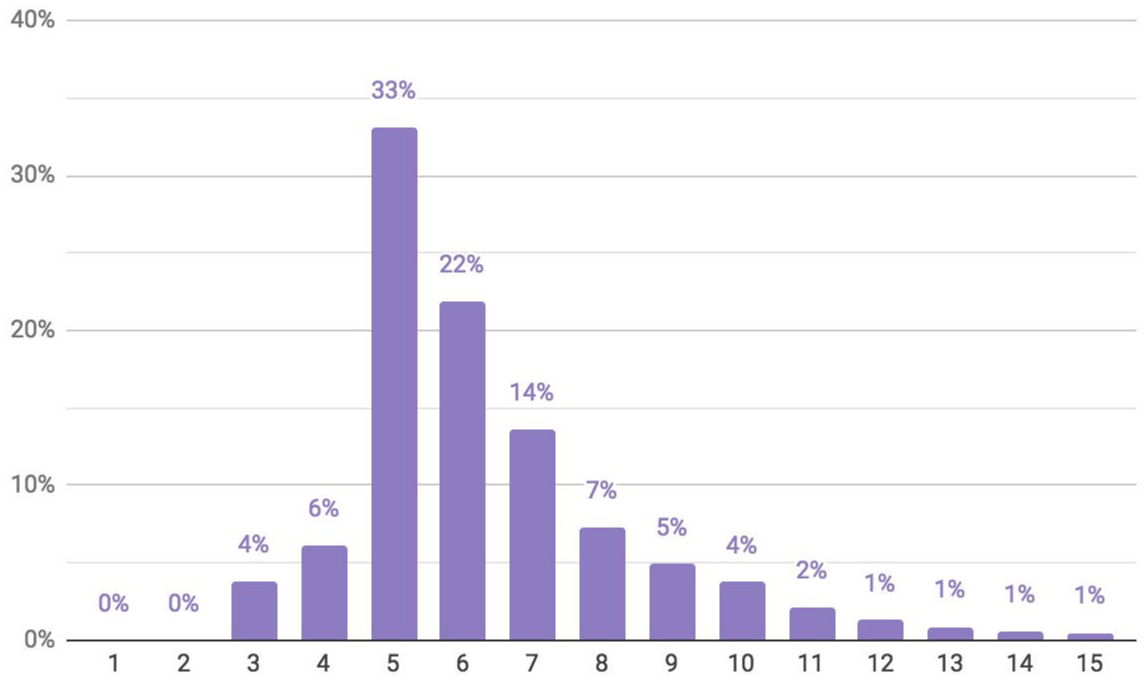
- Don’t use interfaces / providers to hide code you wish didn’t exist
- Use drivers / implementations where it counts

Why Does it Matter?

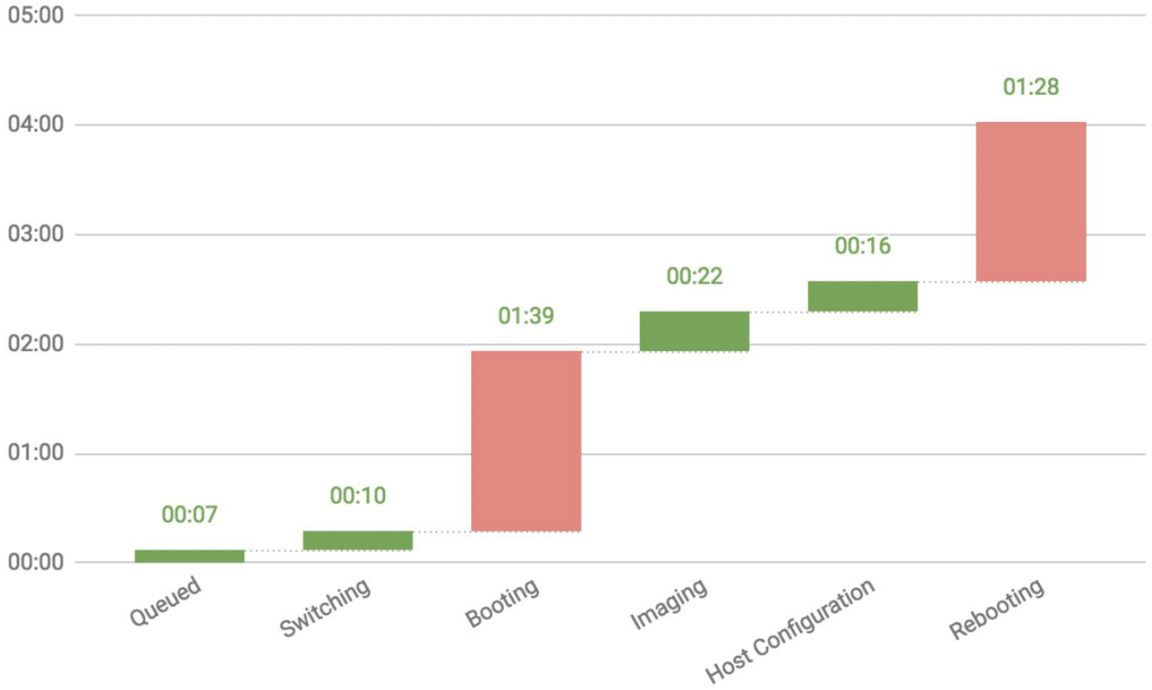


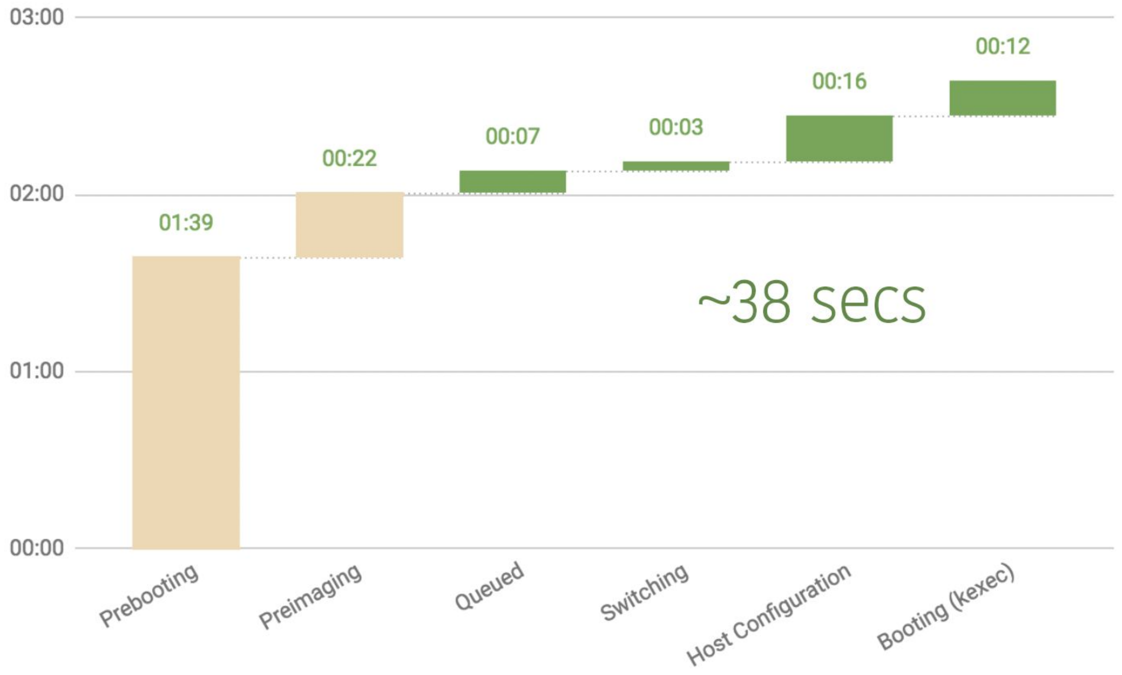
Goal #1: Can we provision in under 60 seconds?

Provisioning Timing Distribution



Provisioning Timeline




03:00
02:00
01:00
00:00
01:39
00:22
00:07
00:03
00:16
00:12
Prebooting
Preimaging
Queued
Switching
Host Configuration
Booting (kexec)

~38 secs

Ephemeral Nanoservices

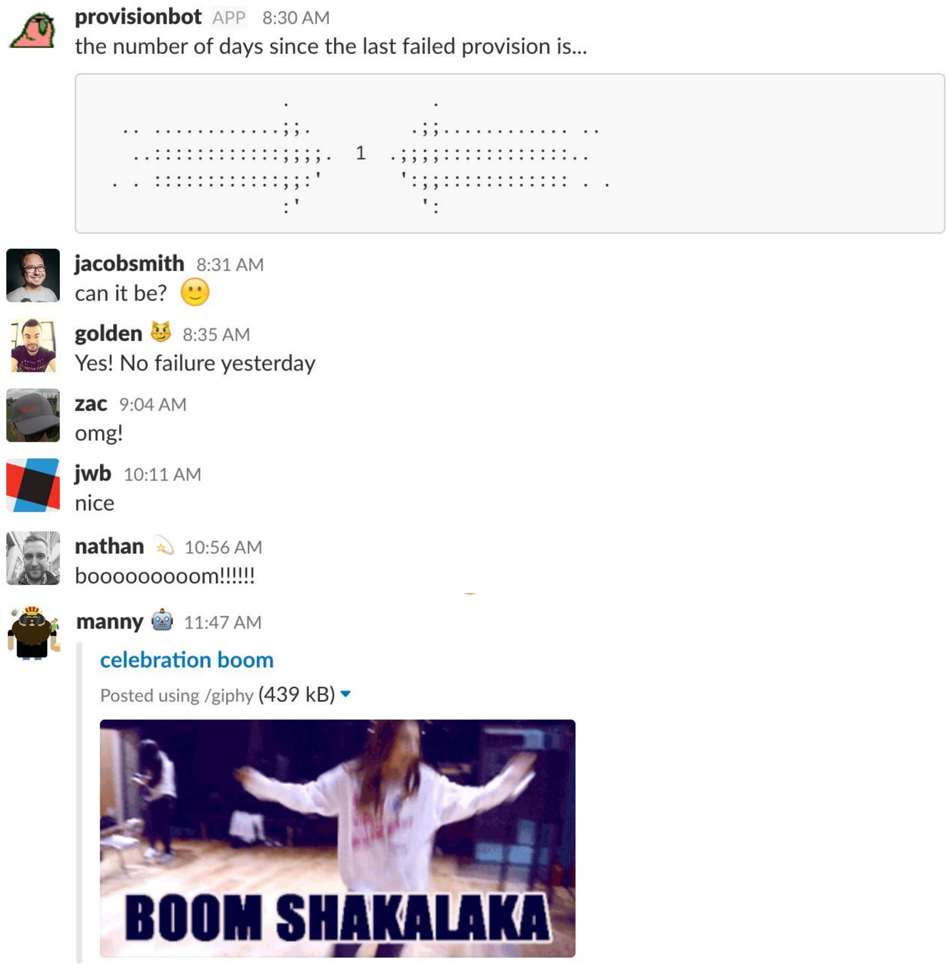
	Function	Job	Nanoservice	Microservice	Monolith
Ephemeral	✓	✓	✓	✗	✗
Encapsulated	✓	✓	✓	✓	✗
Logging	?	✓	✓	✓	✓
Complex tasks	✗	✓	✓	✓	✓
Monitored	✗	✗	✓	✓	✓

Nanoservice Use Cases

- Services that have complex tasks or functionality to perform, and...
- Need to communicate with other services, and...
- Need to be kept up and running, but...
- Will never be used past their "life"

Analogy: an ephemeral nanoservice is an "instantiation" of a microservice

Goal #2: Can we go a full day without a single provisioning failure?



provisionbot APP 8:30 AM
the number of days since the last failed provision is...

```
                .             .
 .. ............;;.         .;;............ ..
  ..::::::::::::;;;;.  1  .;;;;::::::::::::..
. . ::::::::::::;;:'       ':;;:::::::::::: . .
                :'           ':
```

jacobsmith 8:31 AM
can it be? 🙂

golden 😼 8:35 AM
Yes! No failure yesterday

zac 9:04 AM
omg!

jwb 10:11 AM
nice

nathan 💫 10:56 AM
boooooooom!!!!!!

manny 🤖 11:47 AM
celebration boom
Posted using /giphy (439 kB)

What’s next?

#1 - Flexible workflows via directed graphs

#2 - Distributed tracing for service logs

Q&A

(we're hiring)

packet