# Effective Kubernetes Development

Turbocharge Your Dev Loop

Philip Lombardi
@TheBigLombowski

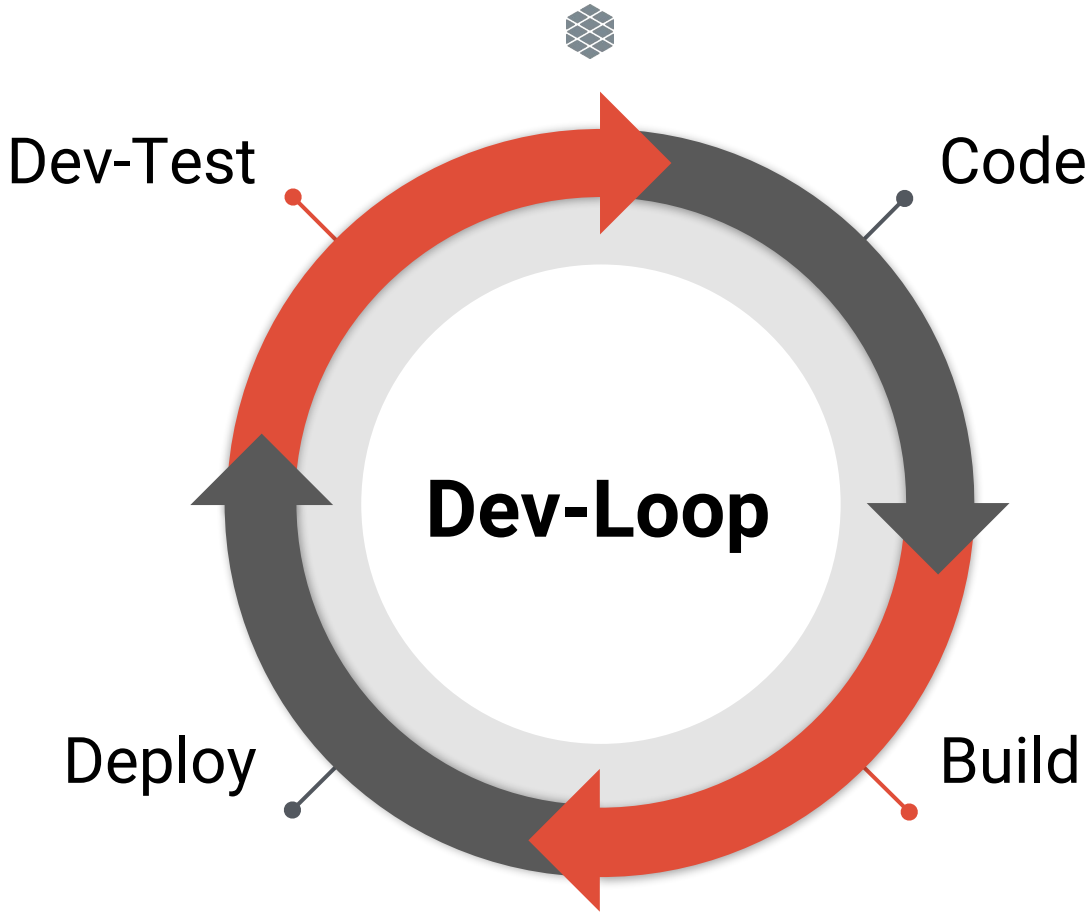
# This Talk

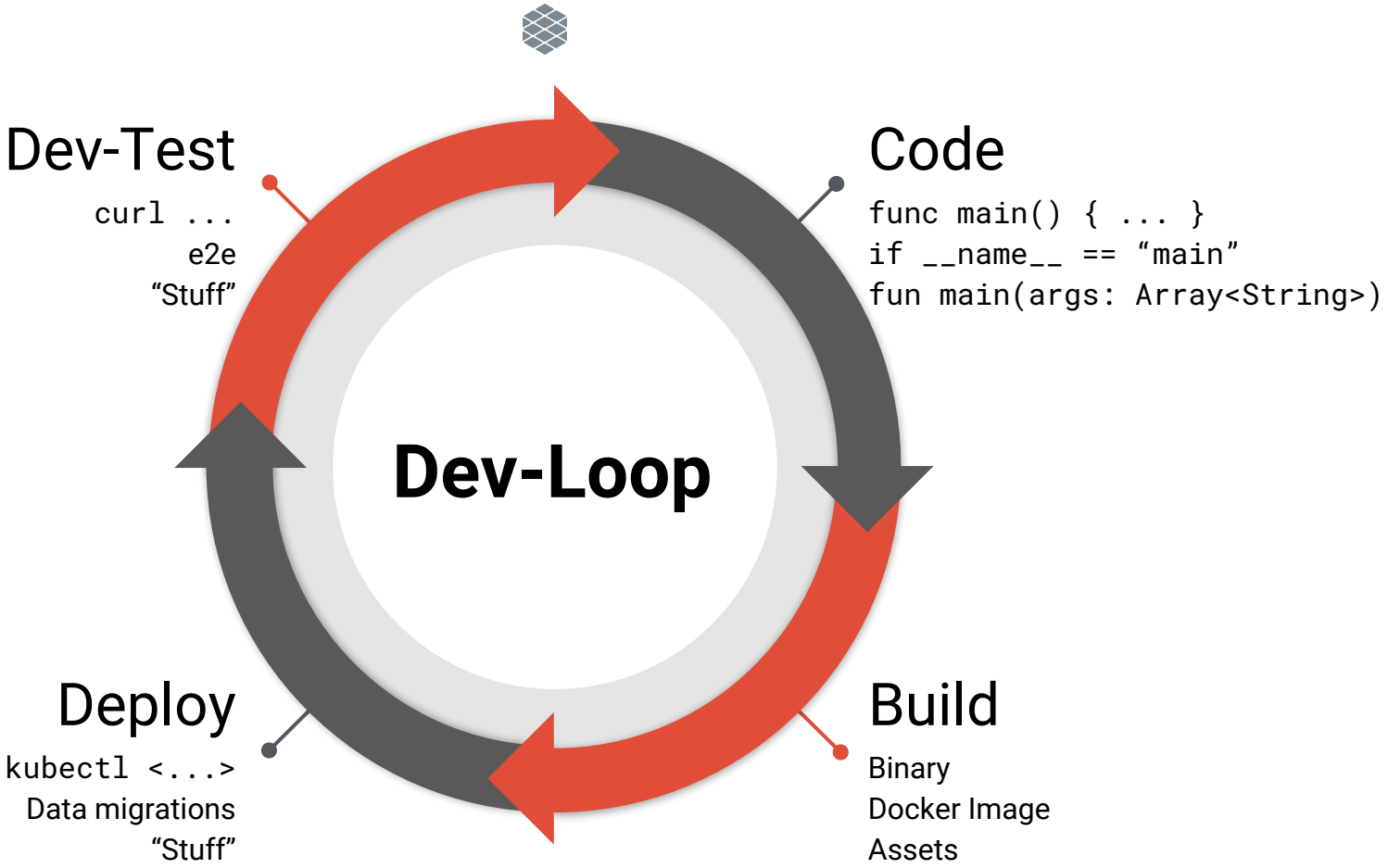
- Share the Cloud Native workflow we have evolved over three years

- A journey about how we got where we are along with some broad stroke technical and communication insights

# $(whoami)

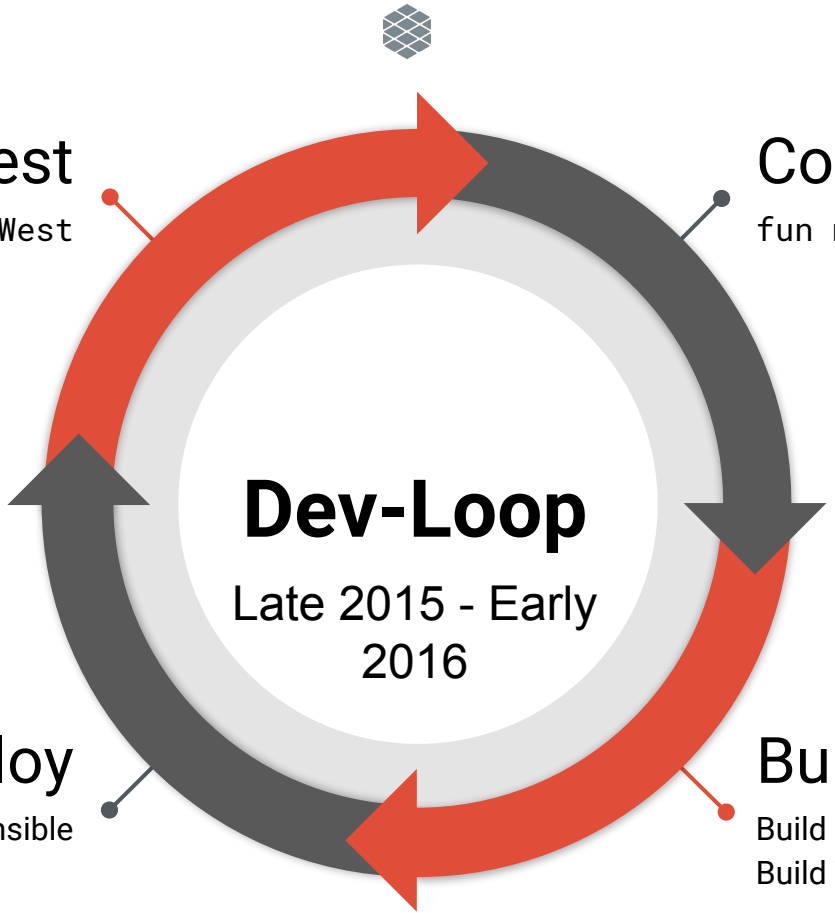
- Internal dev tooling and infrastructure
- Occasionally hack on product
- Very impatient... on a mission to make Cloud Native development less cumbersome


Dev-Test
Code
Dev-Loop
Deploy
Build


Dev-Test
curl ...
e2e
“Stuff”
Code
func main() { ... }
if __name__ == “main”
fun main(args: Array<String>)
Dev-Loop
Deploy
kubectl <...>
Data migrations
“Stuff”
Build
Binary
Docker Image
Assets

# Late 2015

- Not on Kubernetes yet (kube 1.0 brand new)
- Bootstrapping our backend... need to ship fast and ship often
- Dev loop is very primitive...


Dev-Test
Wild West
Code
fun main(args: Array<String>)
Dev-Loop
Late 2015 - Early 2016
Deploy
Mix of: Shell, Terraform and Ansible
Build
Build + Package Binary
Build VM image

# Problems 1.0

1. VM images take *forever* to create (10 min)
2. Build fails... start over again (10 mins x Failures)
3. Developers hate dealing with Packer as part of their dev loop.
4. Shell + Terraform + Ansible stuff is pretty janky too and causes problems.

# Solutions 1.0

1. Ignore all the speed complaints!

2. Slap a fresh coat of paint on packer by hiding it in Makefiles and Gradle scripts.


MISSION ACCOMPLISHED

# Reality...

Me


CTO + Coworkers

# Lessons 1.0

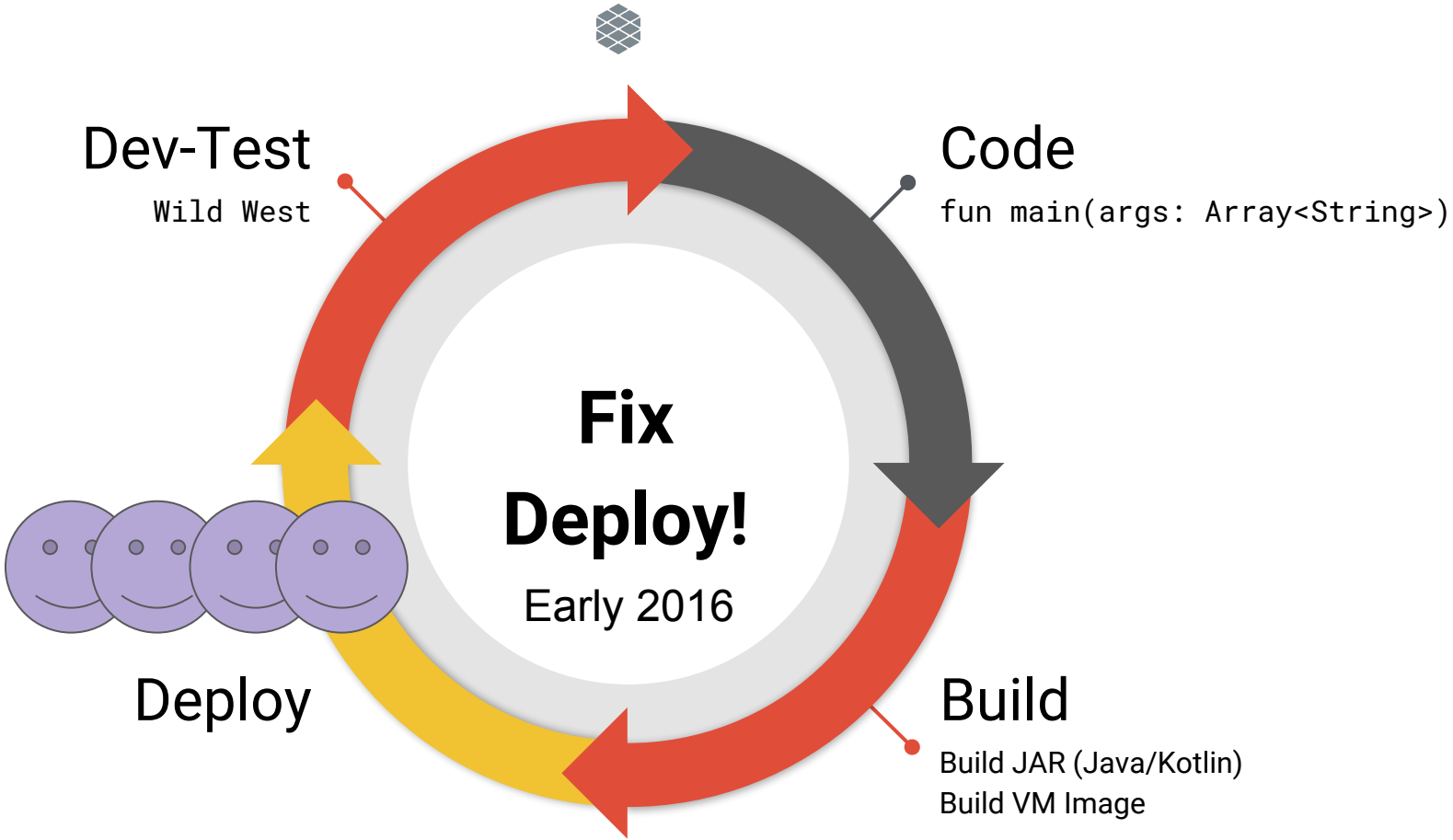
1. I shouldn’t ignore my developers!

2. I need to listen and understand their problems better…

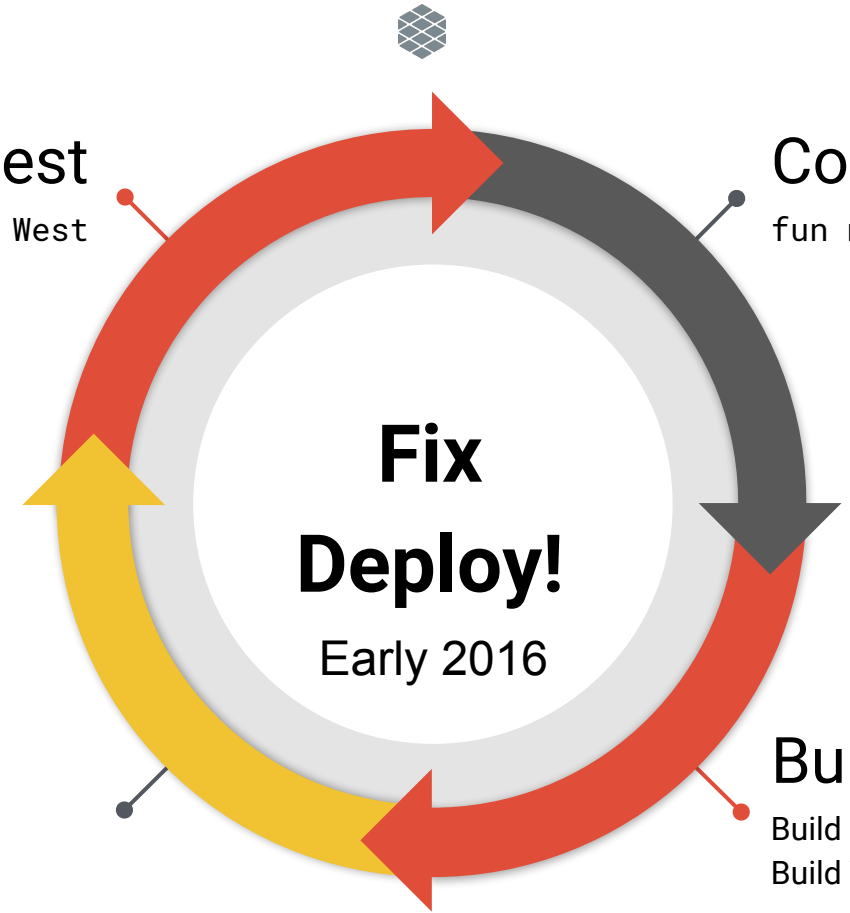
# Problems 1.1 and 1.2

**Problem**: Our devs do not always know what their problem is...

**Problem**: I listened to our developers and we fixed the wrong problem


Dev-Test
Wild West
Code
fun main(args: Array<String>)
Fix
Deploy!
Early 2016
Deploy
Build
Build JAR (Java/Kotlin)
Build VM Image

- Automate Workflow!
- Fancy Custom Tools!



# swap.py

- Does all the fancy orchestration to accomplish an automated **blue-green** deployment.
- Solved some pain… but it also created lots of new pain and still didn’t address the *real* pain.
- We’re smart people though so we can fix this!

# Solution 2.0: Doubledown!


# Expected Outcome... :D

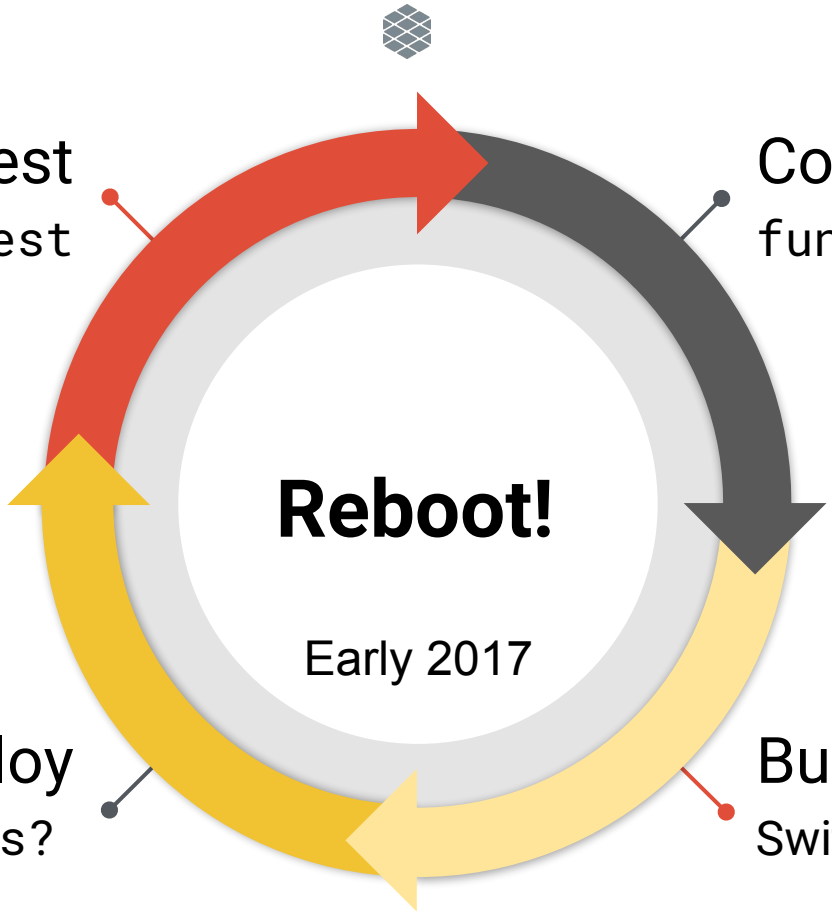
# … Actual Reaction D:

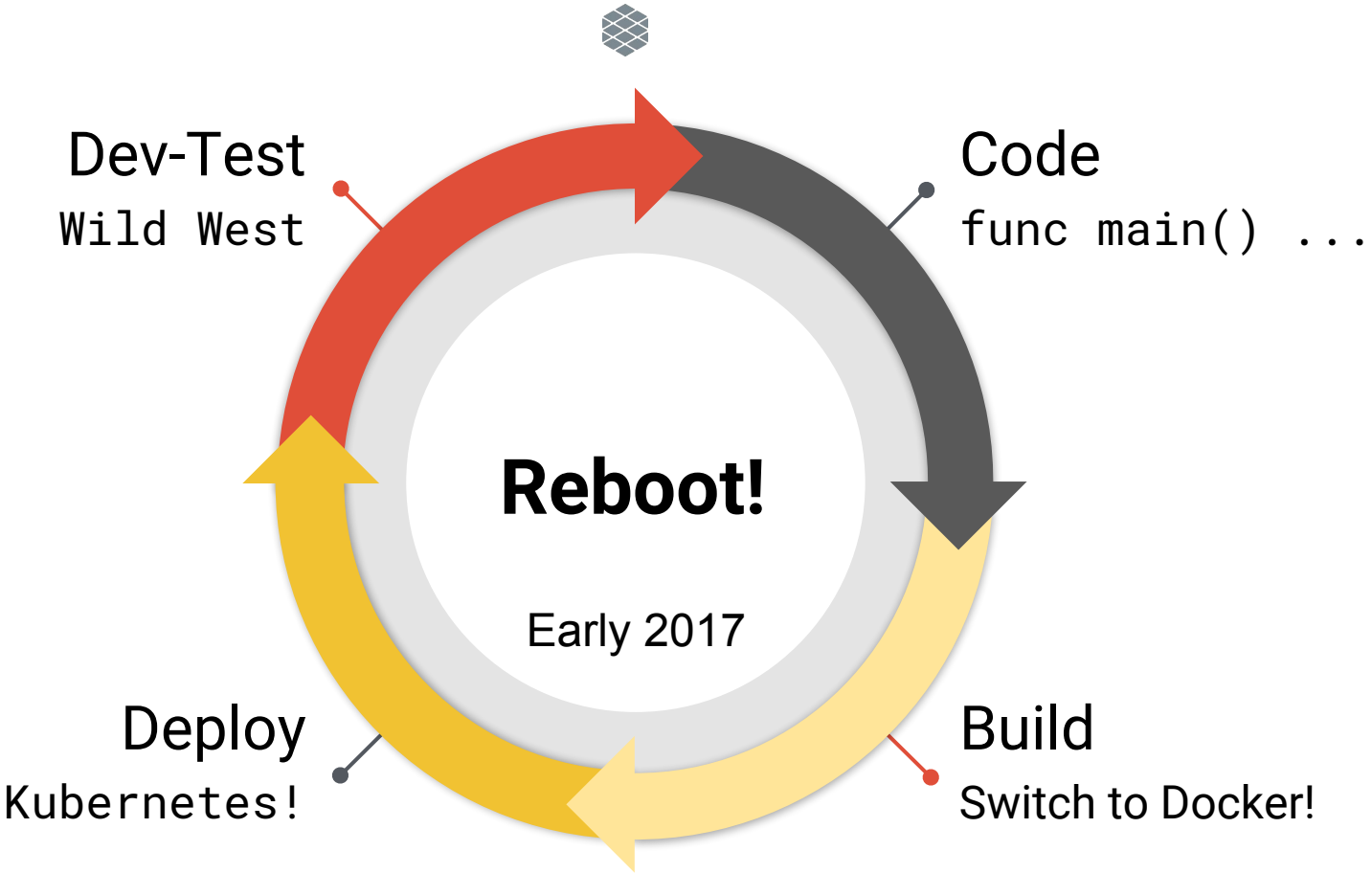
# Lessons 2.0

- Thankful my bosses are patient people ;)
- We need to do a better job identifying the *real* pain.
- Don’t get too technology crazy.

# Solution 3.0

- Scrap it all(!)
- Re-evaluate where things were slow
- Finally we all realized that the inability to **build** quickly was killing us as well.

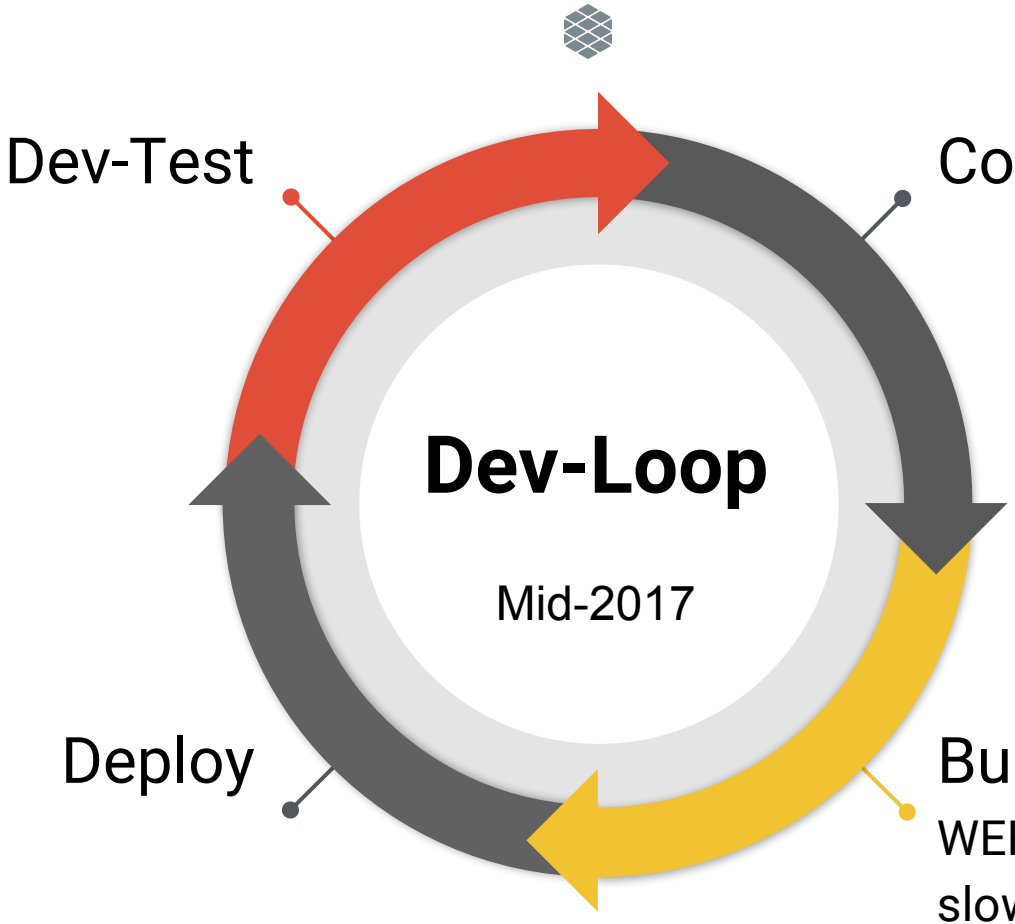

Dev-Test
Wild West
Code
func main() ...
Reboot!
Early 2017
Deploy
Swarm? Kubernetes?
Mesos? Options!
Build
Switch to Docker!


Dev-Test
Wild West
Code
func main() ...
Reboot!
Early 2017
Deploy
Kubernetes!
Build
Switch to Docker!

# Good News!

- Scrapping + Rethink a success!

- Devs Happy! Velocity increased!


Dev-Test
Code
Dev-Loop
Mid-2017
Deploy
Build
WEH! This is *still* too slow!11!!

# They’re kind of Right…

- Compile code (30s+)
- Assembly binary / package (30s+)
- Build Container Image (15-30s+)
- Push Container Image  (15-30s+)

# 120 seconds*

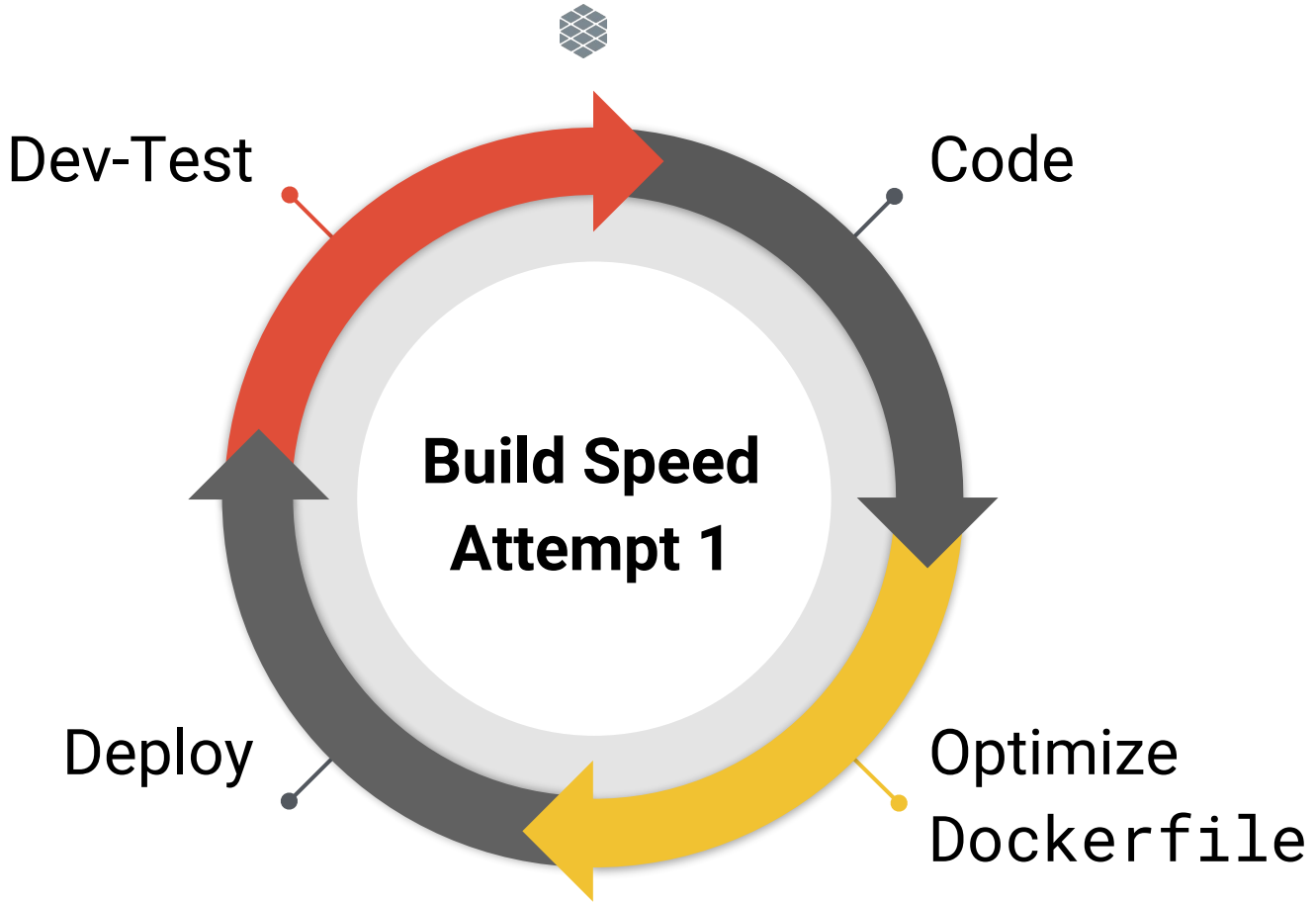
* Not scientific… possibly higher, but most likely not lower

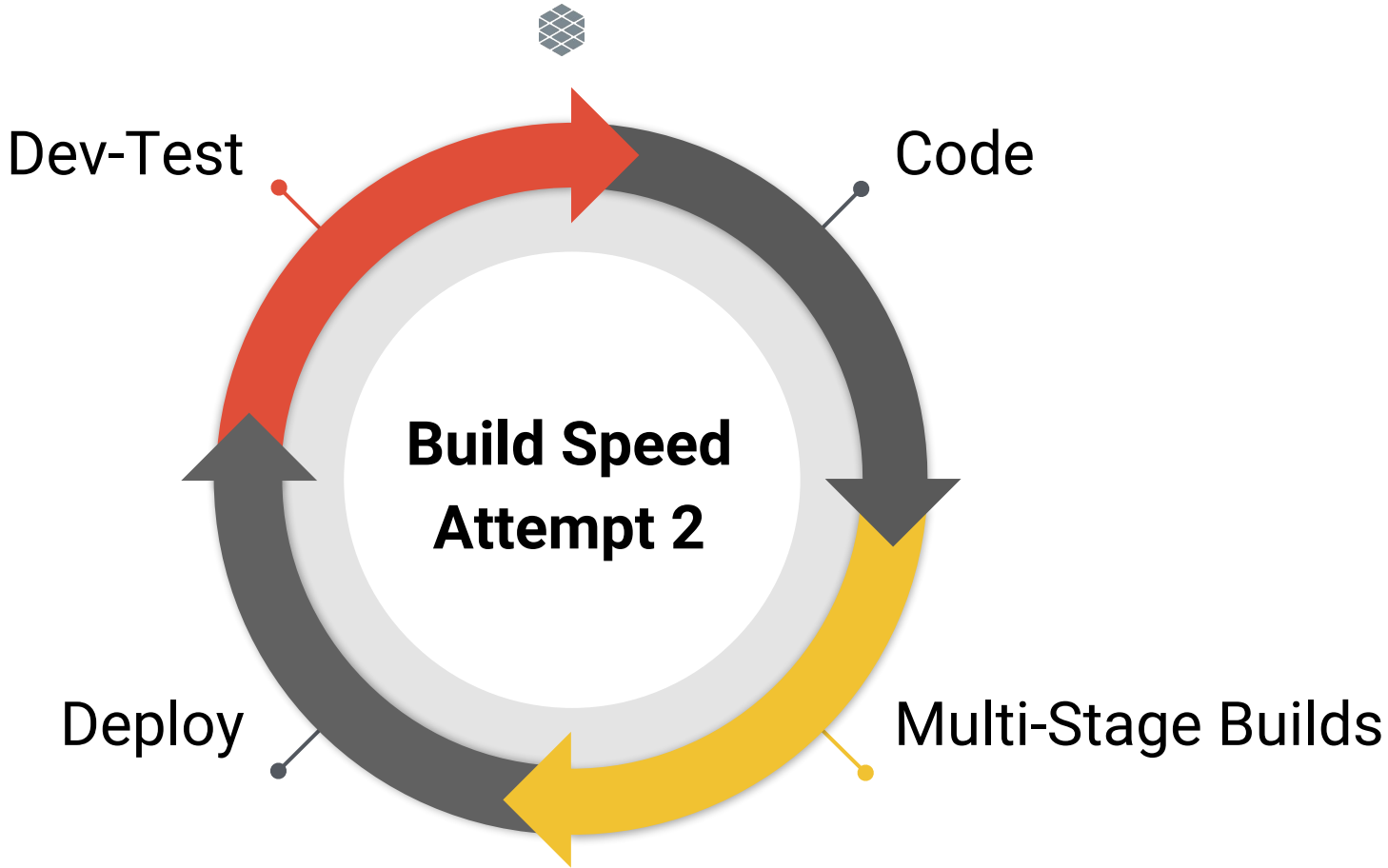
# 120 seconds is Optimistic

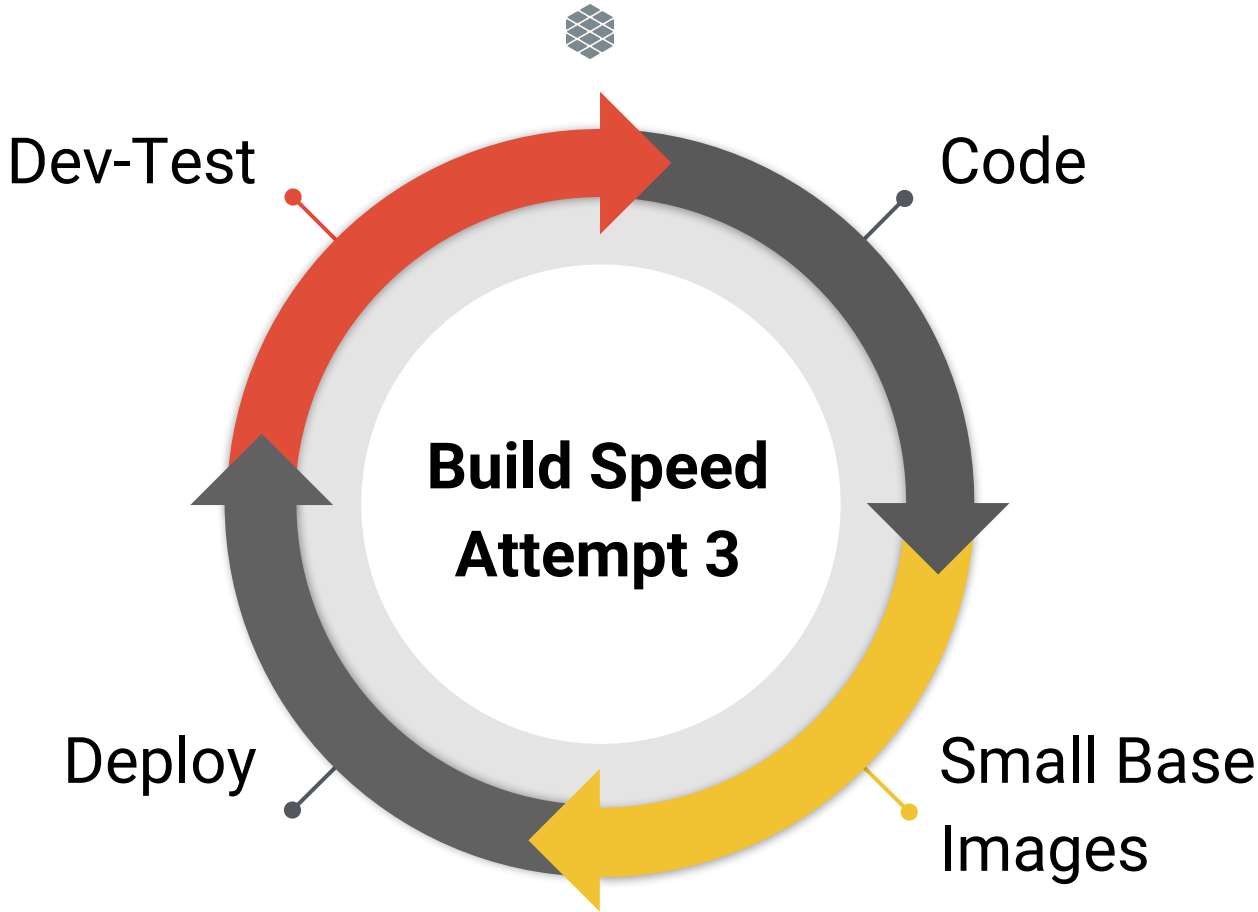
- (Java) Compile in around 30 seconds? Nope. Maven is still downloading the world 30 seconds in.
- (Python) Native dependencies require all kinds of build toolchain stuff to be installed.
- Slow networks…

# And it’s really 120s x # of Rebuilds

Adds up real quick! Easy to throw away an hour or more just waiting.


Dev-Test
Code
Build Speed
Attempt 1
Deploy
Optimize
Dockerfile


Dev-Test
Code
Build Speed
Attempt 2
Deploy
Multi-Stage Builds


Dev-Test
Code
Build Speed
Attempt 3
Deploy
Small Base
Images

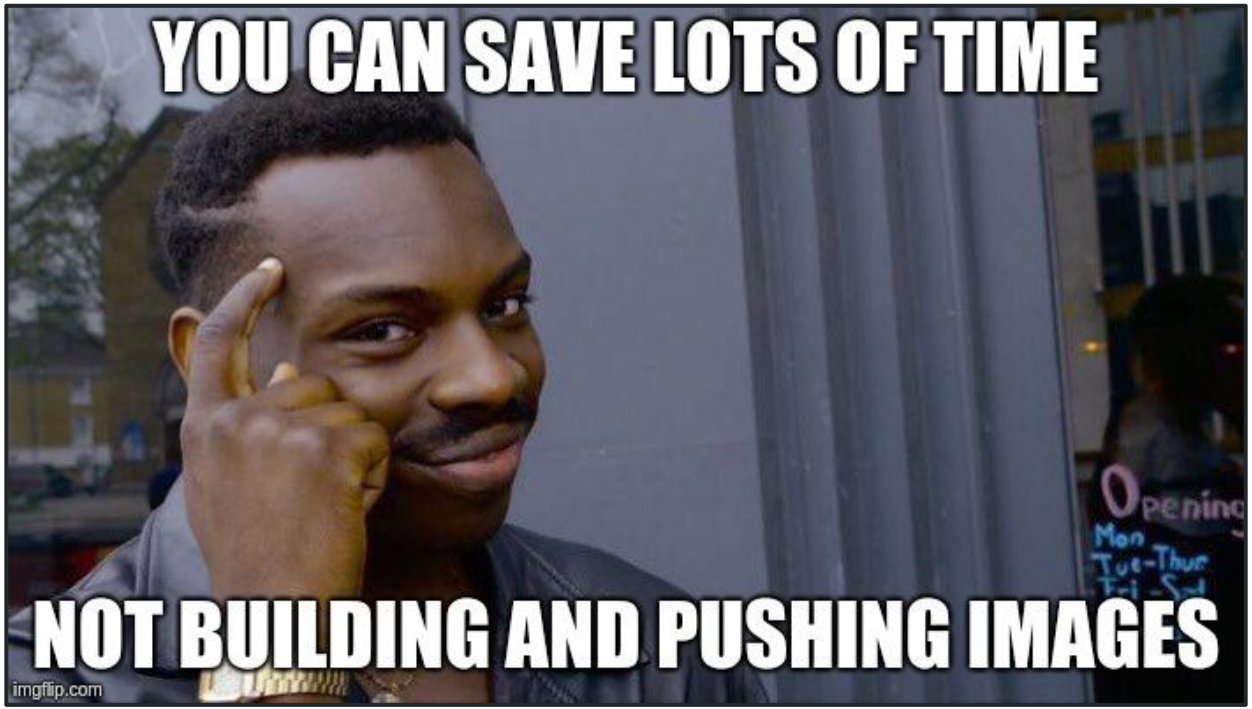
# Not Enough!

- Shaves some seconds… but not enough.

- Even the fastest builds gets stuck in time sink of registry push, redeploy, registry pull.

# Faster Builds… Hail Mary Pass

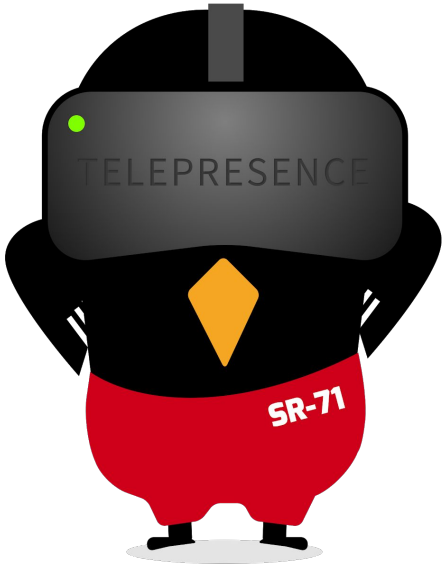
1. Wrote a custom tool that did some crazy stuff around managing Docker layers and continuously rebuilding…
2. Faster builds but generated monster sized images that took ages to push and pull… so net gain of nothing.


YOU CAN SAVE LOTS OF TIME
NOT BUILDING AND PUSHING IMAGES
imgflip.com

# What if...

Your dev machine ran your code and it just worked the same as if you ran it on Kubernetes?

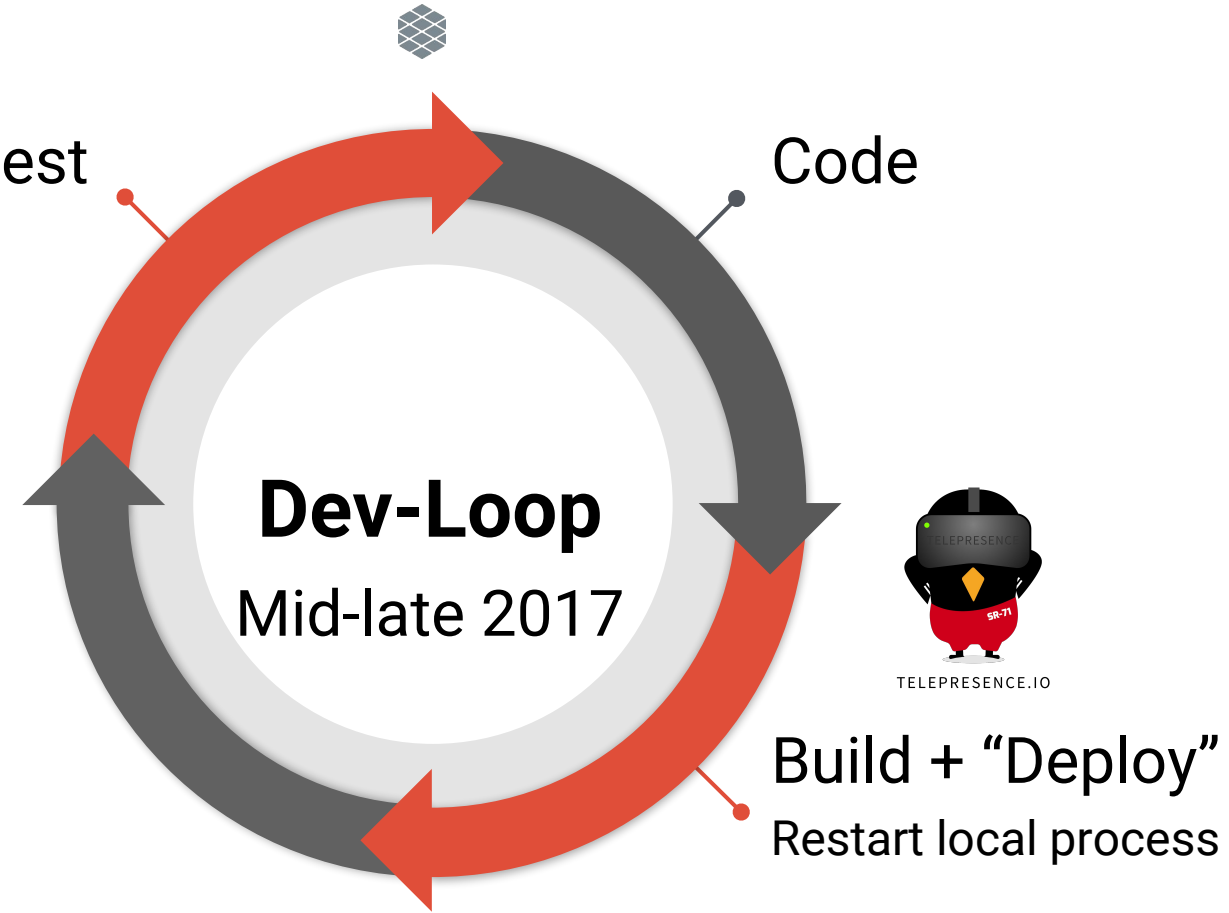
# What if...



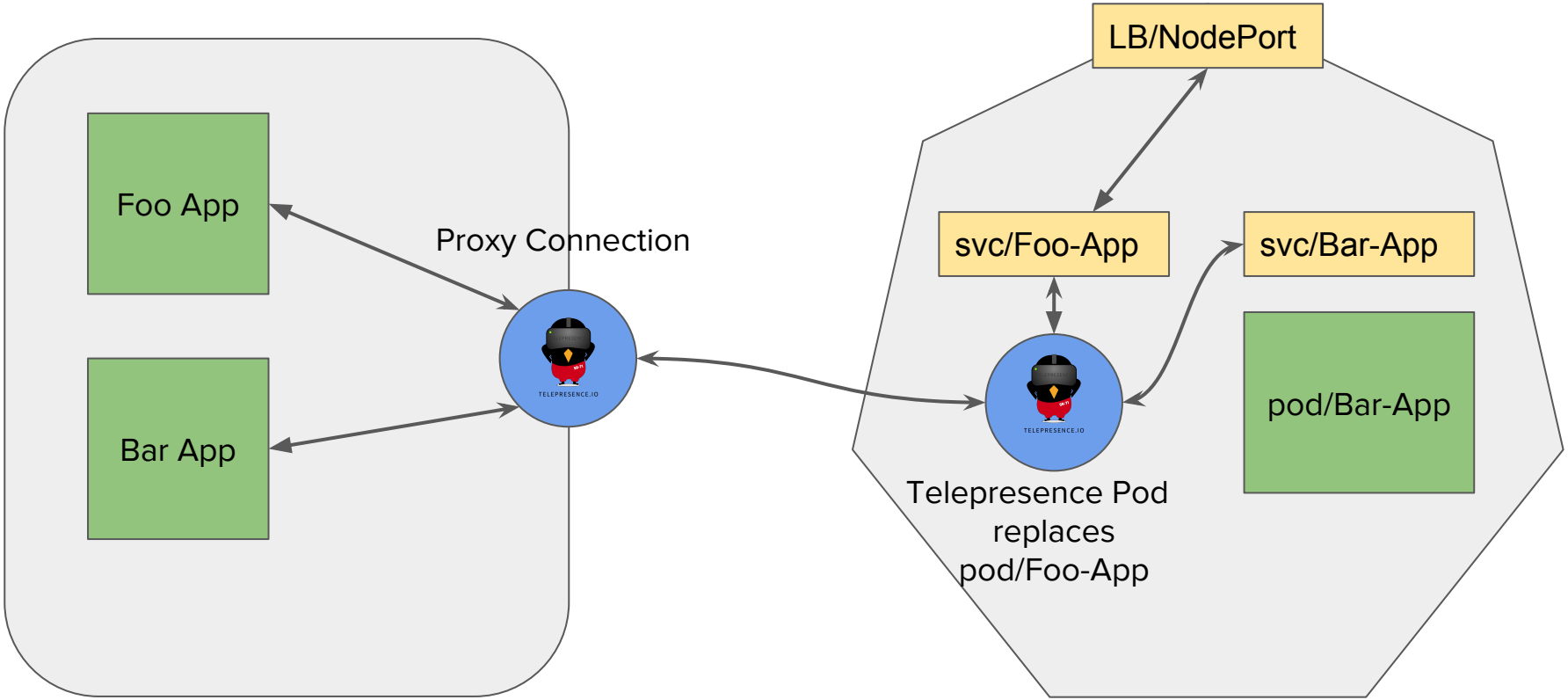

TELEPRESENCE.IO

CNCF Sandbox!

Open Source!


Dev-Test
Code
Dev-Loop
Mid-late 2017
TELEPRESENCE.IO
Build + “Deploy”
Restart local process

# Telepresence Topology



# Kill Two Birds w/ One Stone

- “Deploy” is merged into Build because “Deploy” just means relaunching local process.

- Drastically simplifies the problem.

# Secondary Benefit!

- Code is on YOUR machine... so...
- You can run a debugger from your local machine easily...
- No more `printf` debugging and `grepping` logs!
  - Connect a debugger to your local process and voila breakpoints! Stack information!
  - Debugging is an often a painful part of Dev-Test so the value of this cannot be overstated.

# Expected Outcome... :D

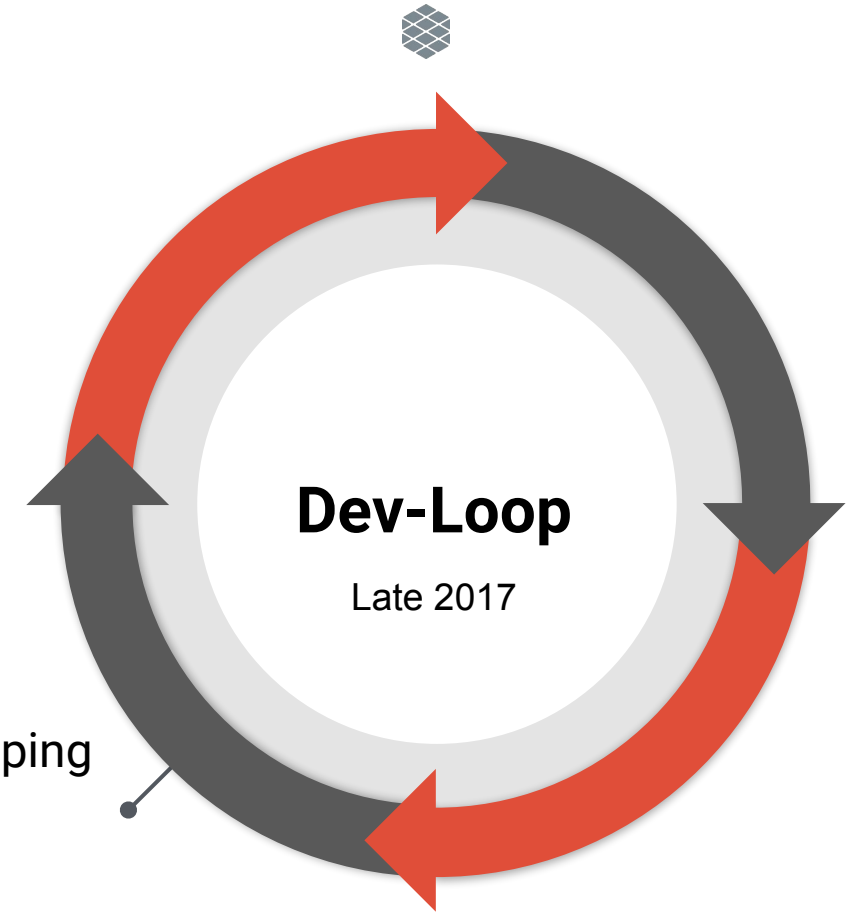
# Actual Outcome... :D


MISSION ACCOMPLISHED

# Not Quite...

- Devs are never satisfied!

- Build speed is great but Getting Started and Bootstrapping are real problems.




Deploy: Bootstrapping

It is a big problem!

# Attempt #1

- Use Minikube on local machine
- Apply manifests of stuff
- It should all just work out right?

# Outcome...

- Minikube is fragile and tends to break in mysterious ways…

- Our deployment logic for stuff is not as clean as we would hope. Tends to also make assumptions about running from a well-configured CI env and not a dev machine.

- Developers banging their heads against their desks because of bootstrapping pain.

# Minikube is for Unicorns

# For the rest of us… remote shared environments

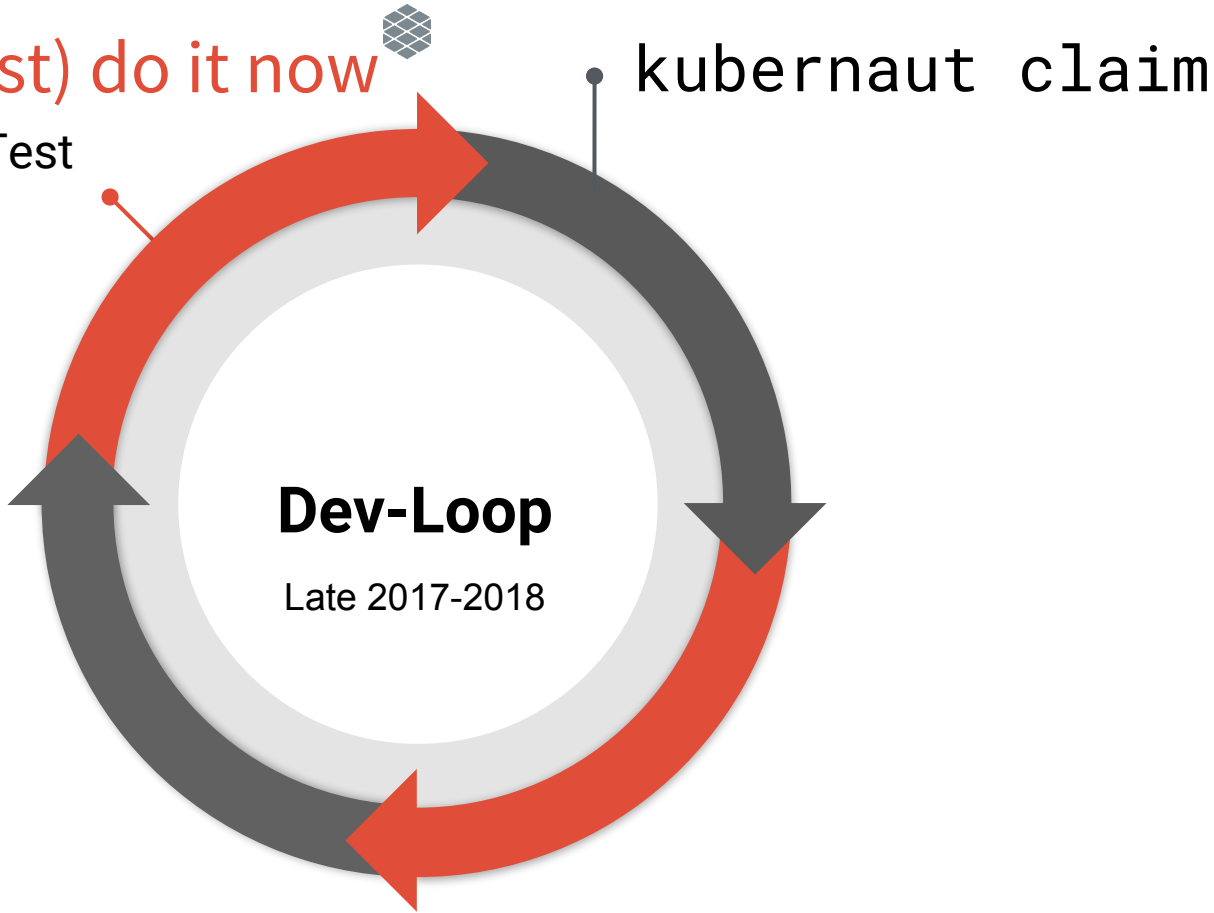
- If you have any cloud services you likely already have a deploy pipeline for production.

- Leverage the deploy tools to spin remote dev-test environments for individuals or teams.

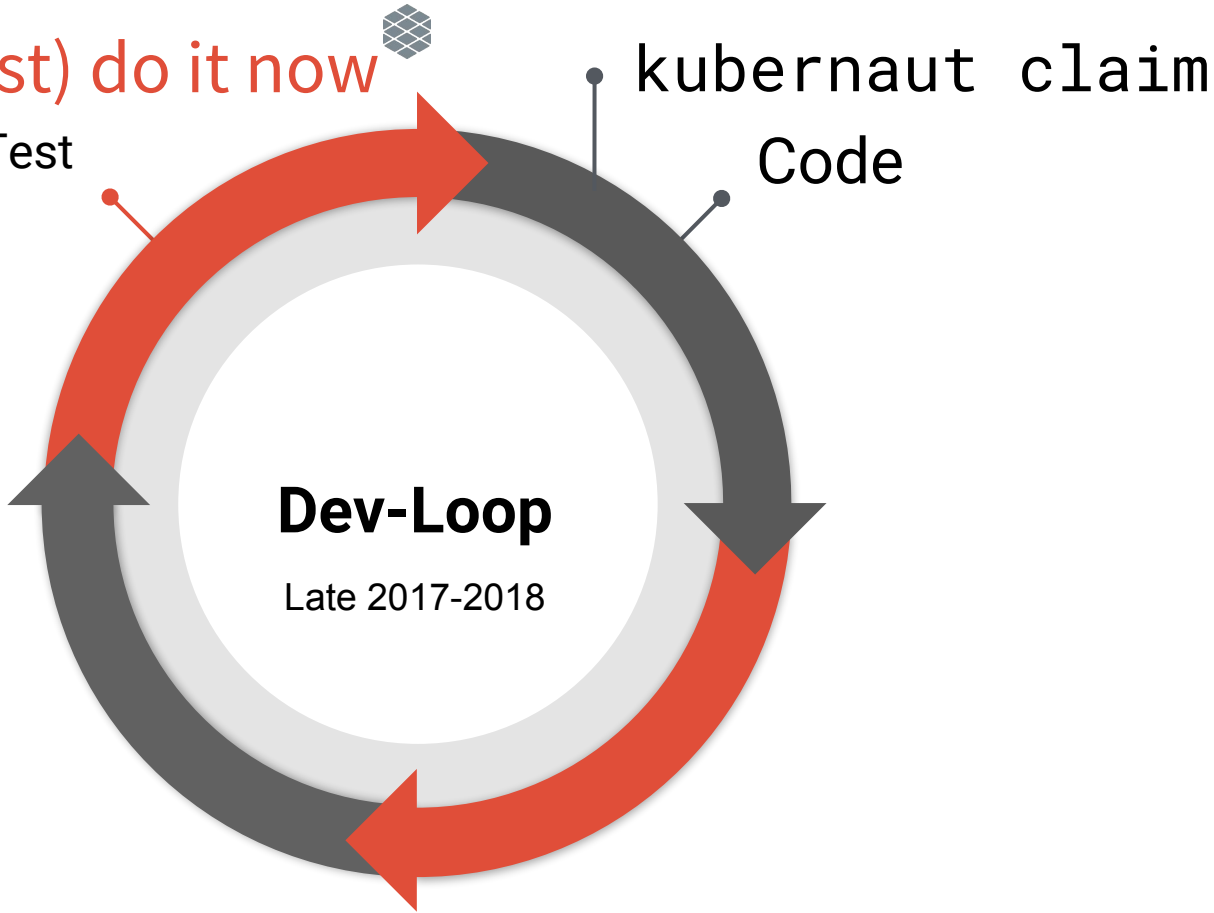
# Attempt #2: Kubernaut

- Remote clusters that take <5 seconds to access.
- Runs a pool of clusters in the cloud.
- Developers claim (start exclusive lock) or discard (end exclusive lock) a cluster.
- Pluggable to support a range of cluster configs and backends
- Going Open Source in 2019

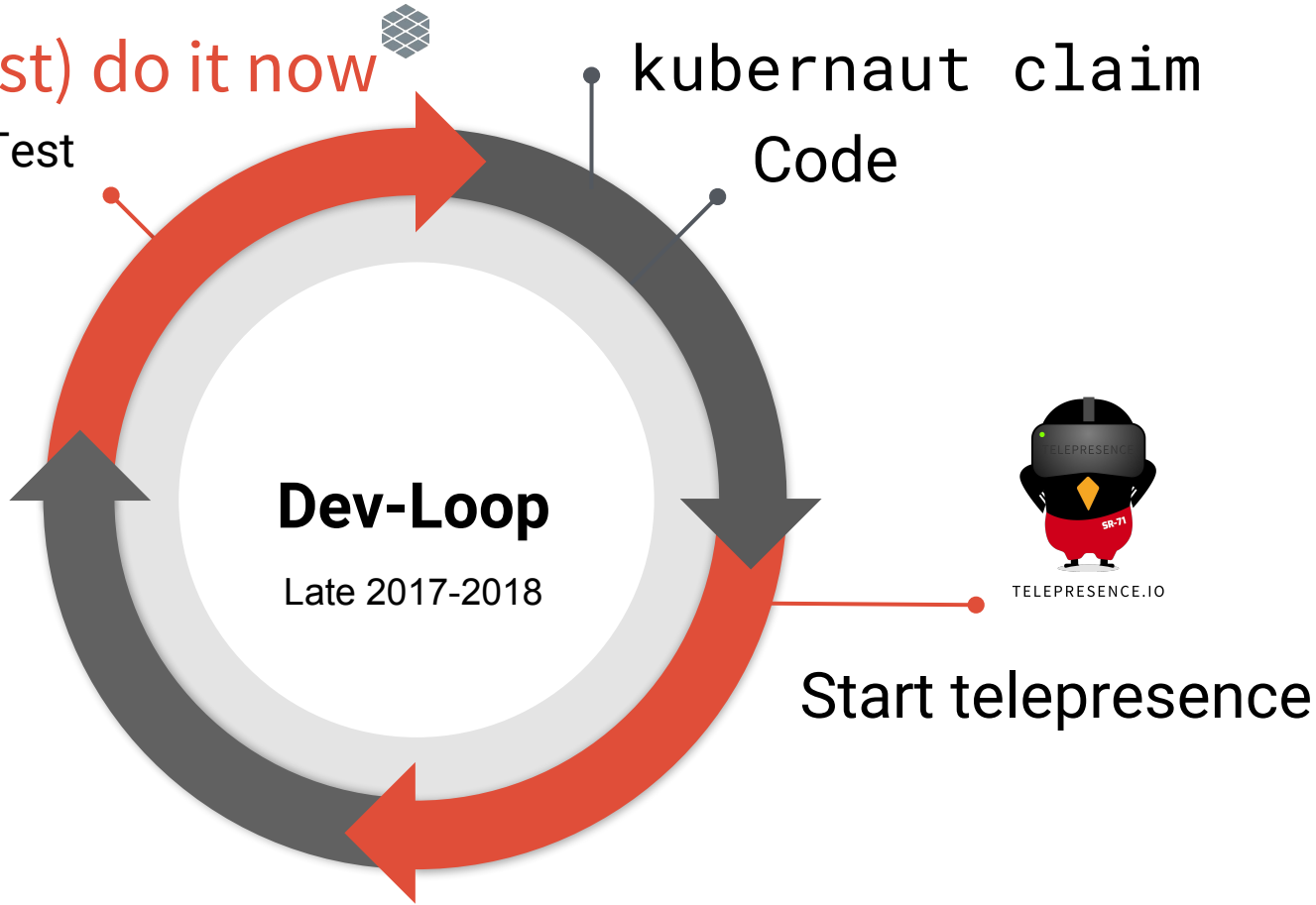
# How we (almost) do it now



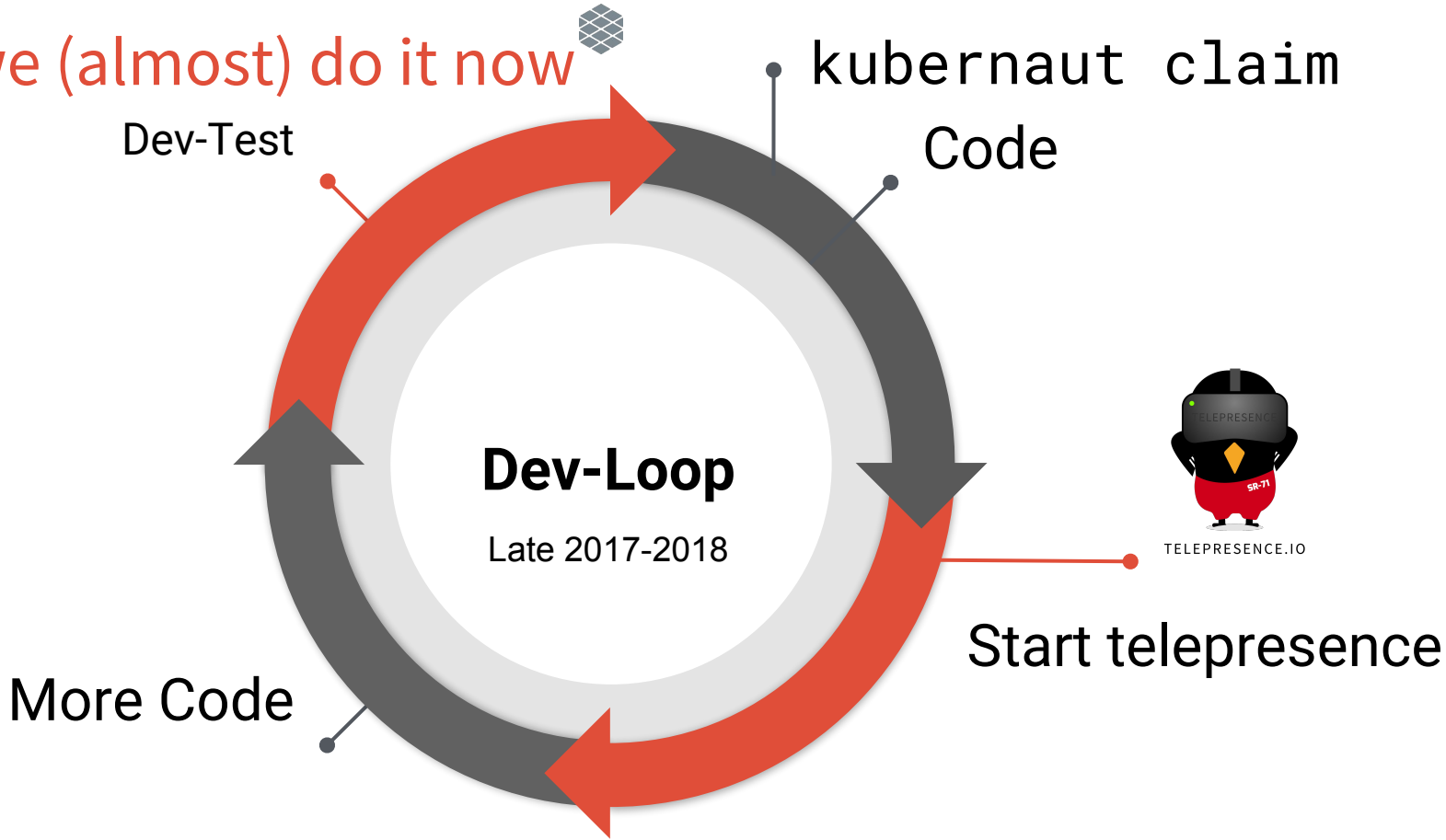
# How we (almost) do it now



# How we (almost) do it now

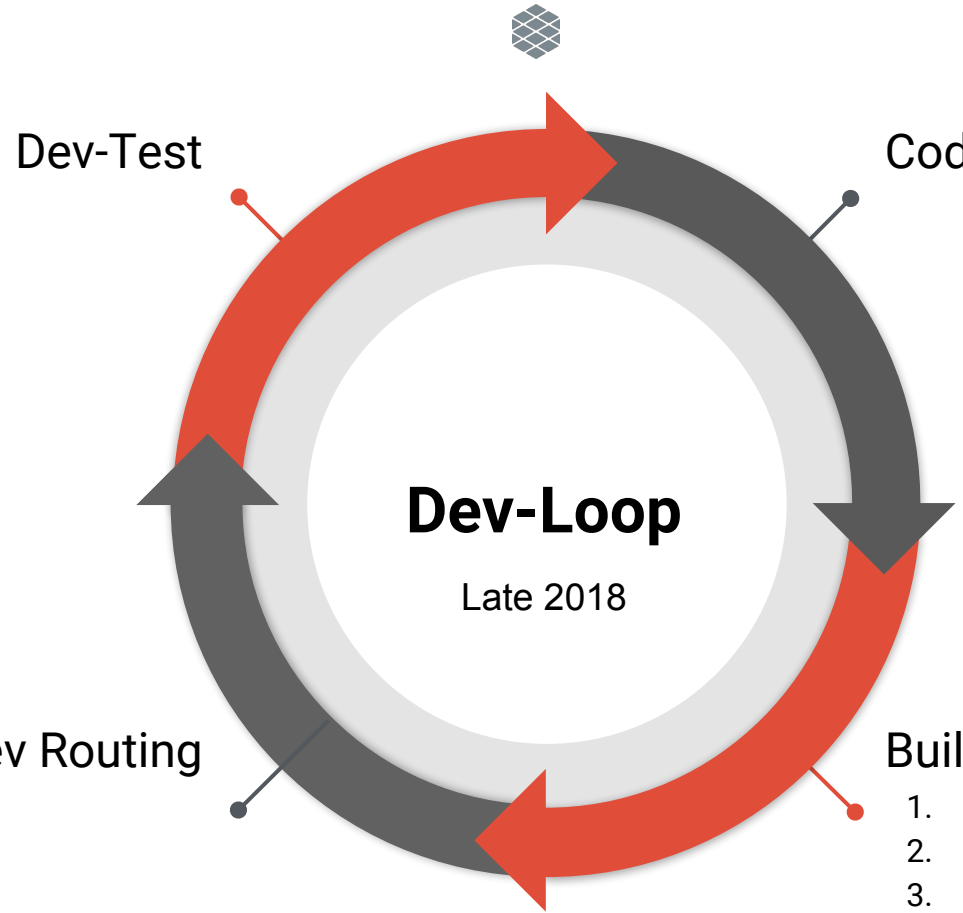


# How we (almost) do it now



# The Ugly: Dev Testing

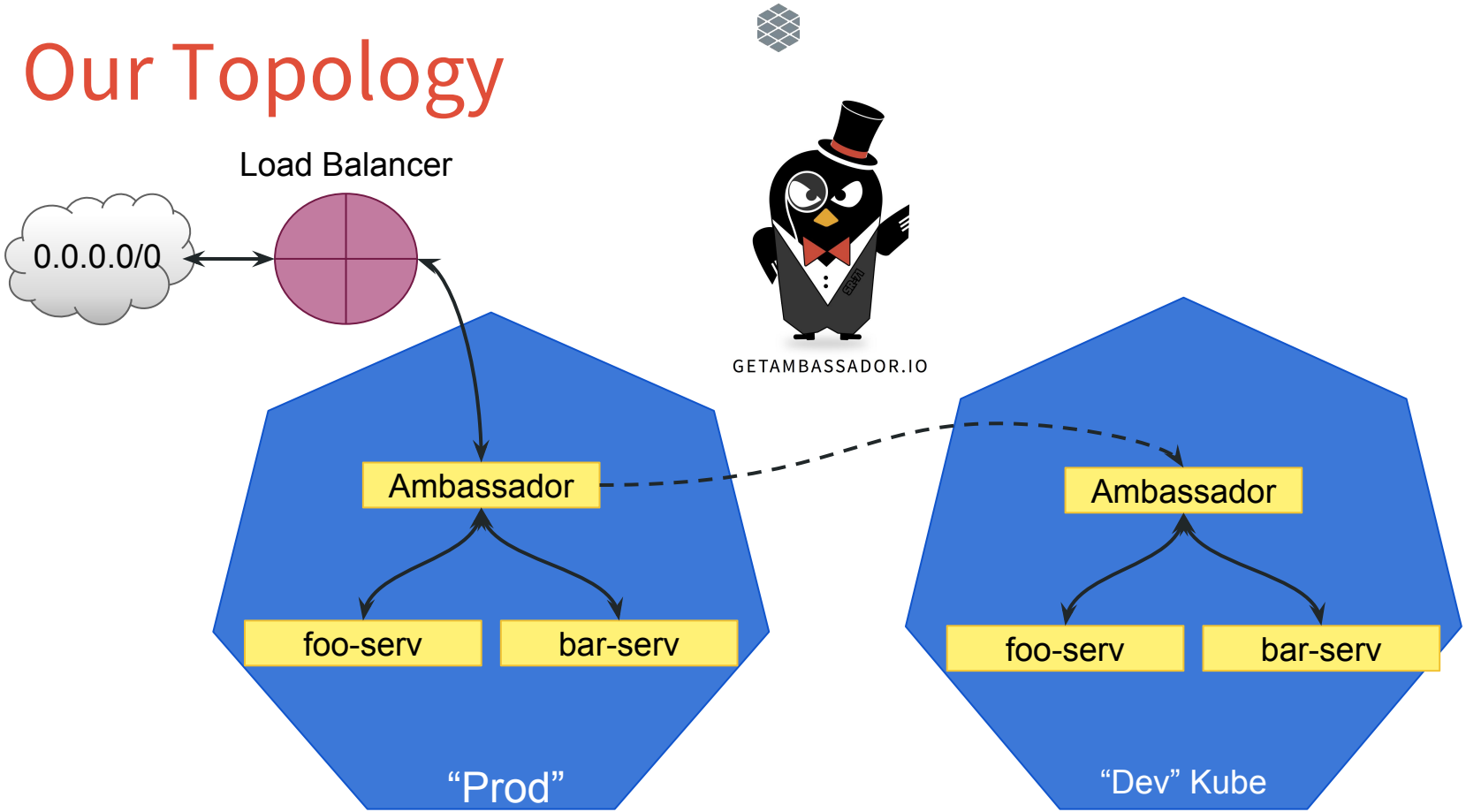
- Dev-Testing is painful without shared remote infrastructure
- Recommended way of doing it
  - Nested API Gateways and some combination of HTTP Host/Header/Path based routing.
- Your API Gateway needs to be self-service for developers and easy to safely reconfigure.



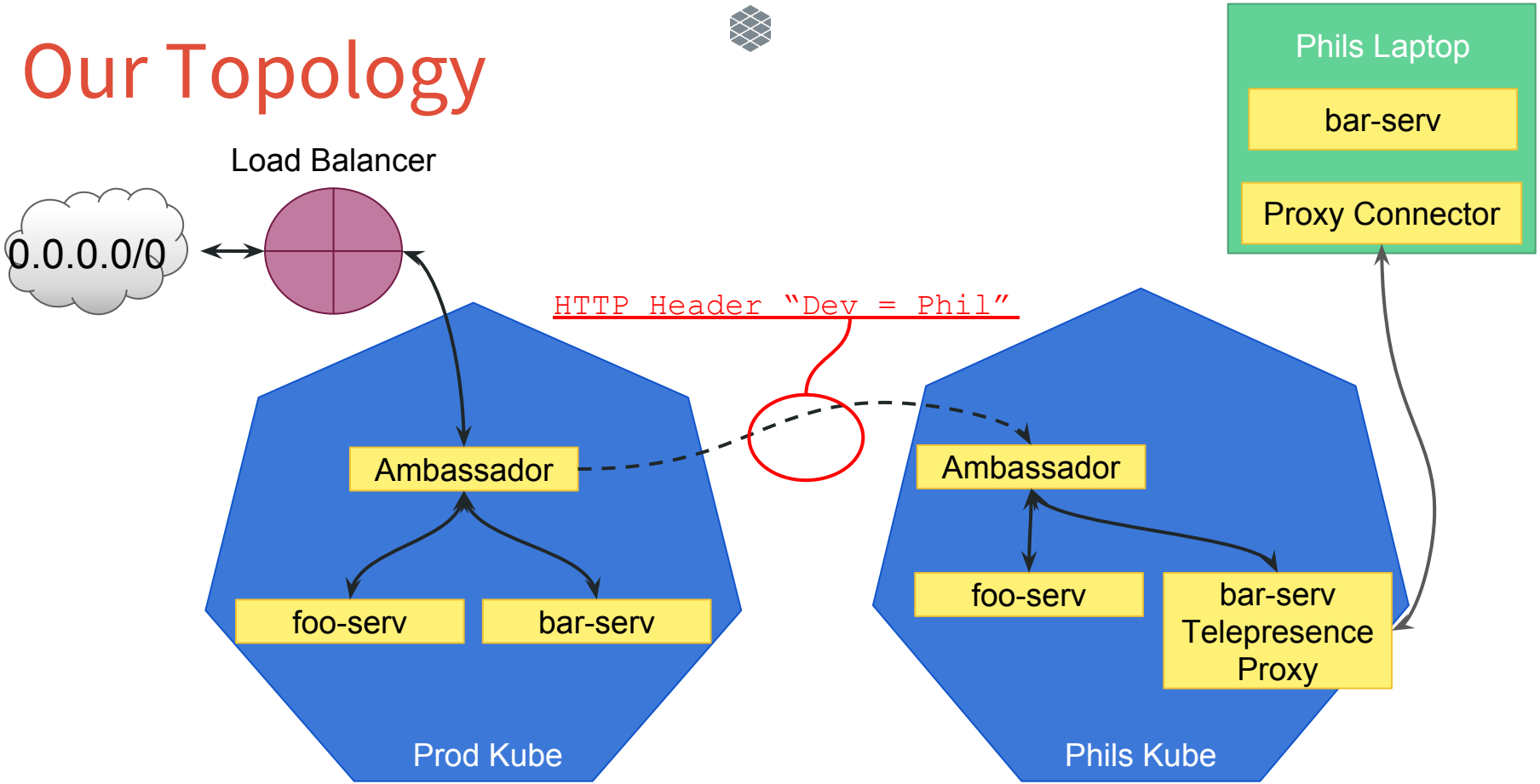

GETAMBASSADOR.IO

TELEPRESENCE.IO

# Our Topology



# Our Topology



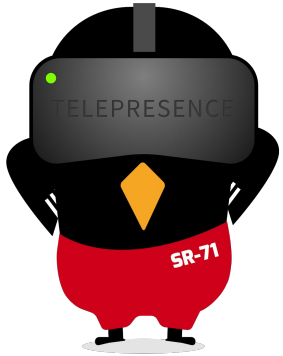
# That’s where we are

- Still plenty of work to do
- Devs are way more productive than when we started: **minimal waiting** for things to happen
- Keep evolving this as we scale out our team.

# Takeaways

- Rome was not built in a day
- Identify and fix the right problems!
- Identify your loops slow points… tackle low hanging fruit first. Repeat…!
- You’re most likely going to need remote development clusters for your projects.

# Thanks!




TELEPRESENCE.IO

I will be at the **Datawire.io** booth

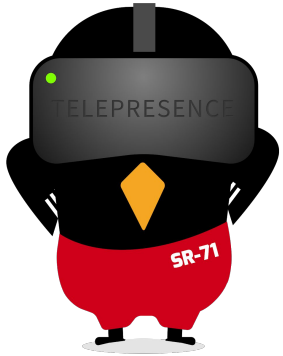
Come and chat :)

Check out Ambassador and Telepresence

Tweet me! **@TheBigLombowski**


GETAMBASSADOR.IO

# Thanks!




TELEPRESENCE.IO

I will be at the **Datawire.io** booth

Come and chat :)

Check out Ambassador and Telepresence

Tweet me! **@TheBigLombowski**

GETAMBASSADOR.IO