Effective Kubernetes Development

Turbocharge Your Dev Loop

Philip Lombardi
@TheBigLombowski

DATAWIRE

$(whoami)

- Engineer at Datawire.io
- ... Own the Dev tooling and infrastructure problem
- ... Occasionally hack on product
- Very impatient... on a mission to make Cloud Native development less cumbersome

@TheBigLombowski

Datawire.io?

- Since 2015 tackling cloud native development problems
- Created the CNCF-sponsored **Telepresence** project
- Created the **Ambassador** API Gateway
- Done a bunch of other stuff too, check us out!

Journey

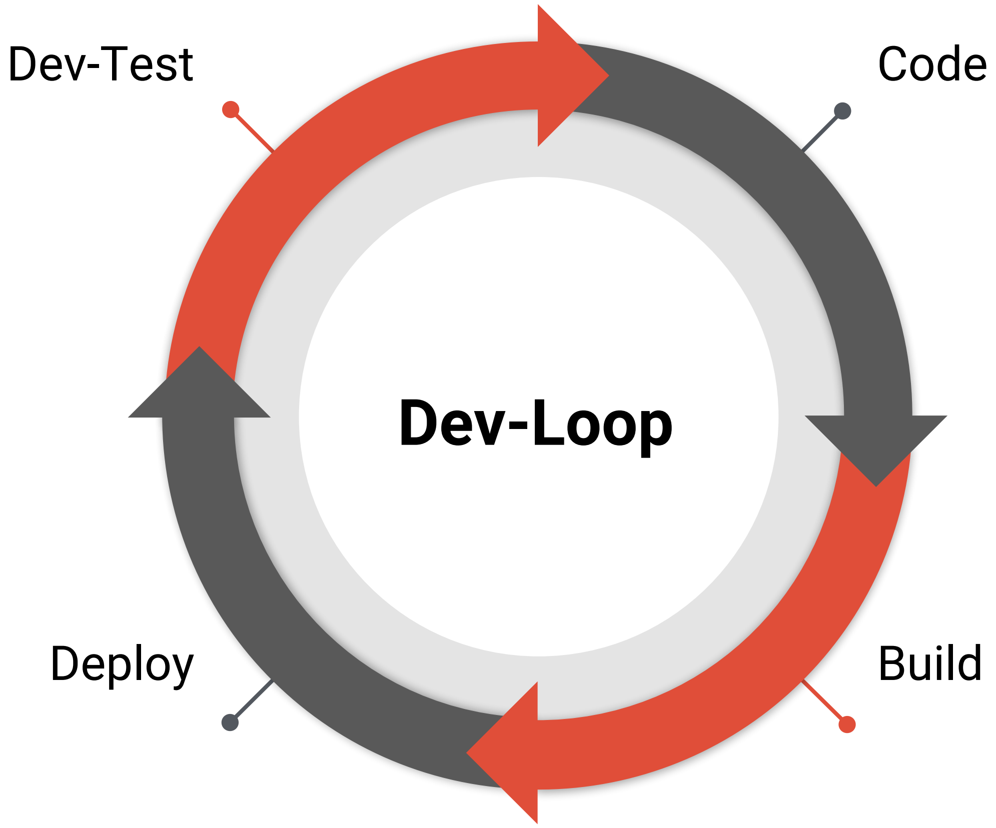
- The story of building a well-liked and efficient dev loop over three years…

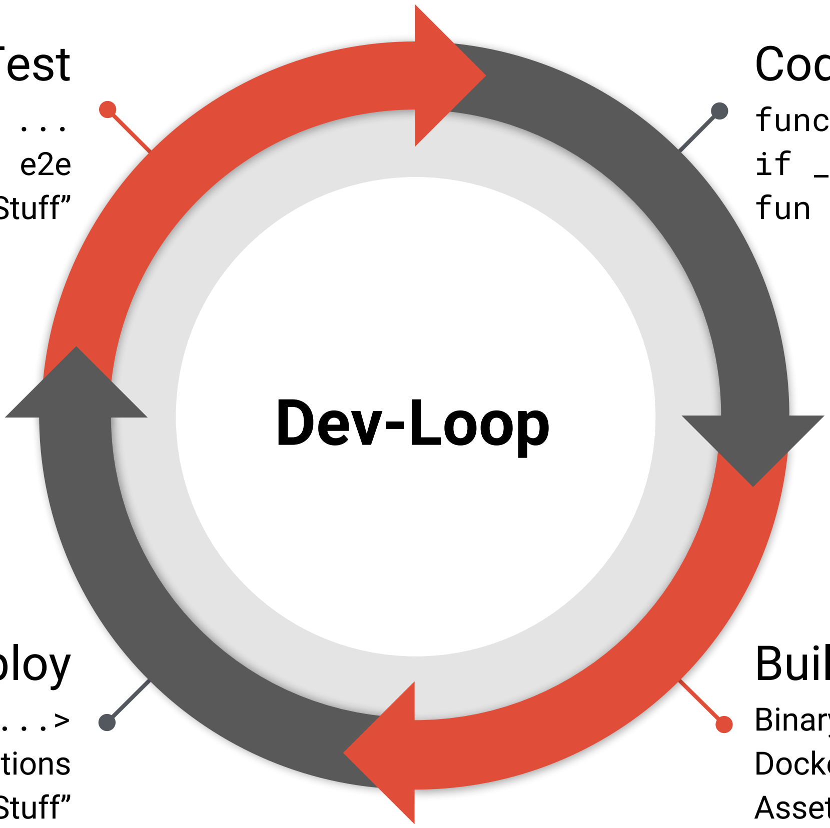
  (… and It's still not done)

- There have been a lot of:
  - Arguments
  - Failures
  - Successes
  - Lessons Learned

- Fortunately for you, I've done all the hard work :)

@TheBigLombowski


Dev-Test
Code
Dev-Loop
Deploy
Build



Code

```
func main() { ... }
if __name__ == “main”
fun main(args: Array<String>)
```

Build

Binary
Docker Image
Assets

Deploy

```
kubectl <...>
```

Data migrations
“Stuff”

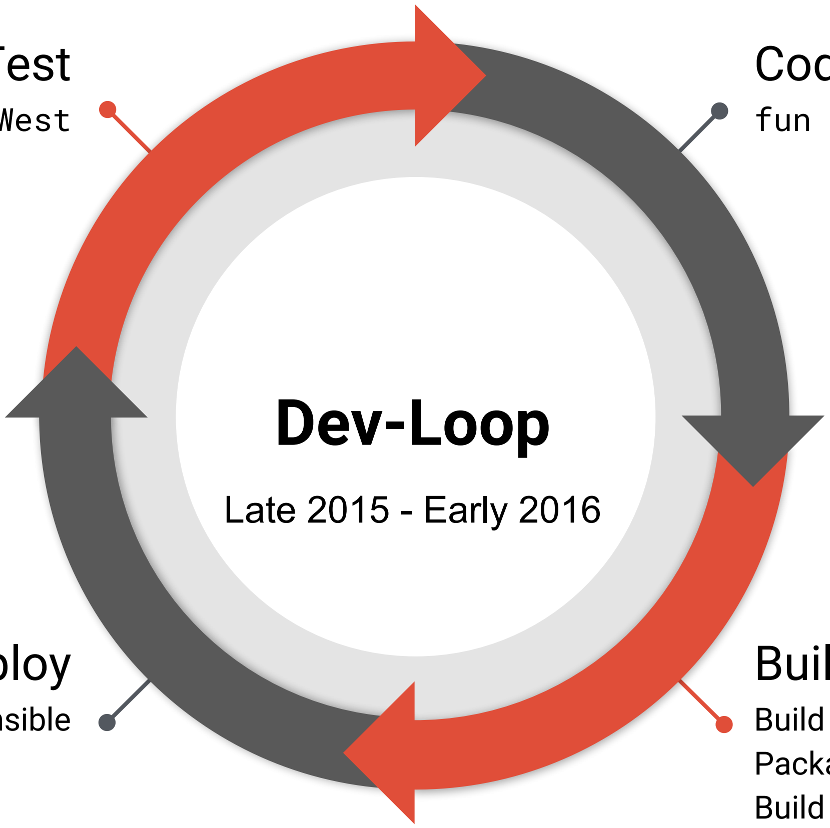
Dev-Test

```
curl ...
```

e2e
“Stuff”

Late 2015

- We are not on Kubernetes yet (kube 1.0 has just been announced)
- Bootstrapping backend services...
- Ship fast! Ship often! (Or that's what we want).
- Dev loop is very primitive...



Dev-Test

`Wild West`

Code

`fun main(args: Array<String>)`

Build

Build Binary
Package Binary
Build AMI (VM Image)

Deploy

Mix of: Shell, Terraform and Ansible

Problems 1.0

1. VM images take *forever* to create (around 10 min)

2. When the build fails… start all over again (another 10 minutes!)

3. Developers hate dealing with `packer` as part of their dev loop.

4. The Shell + Terraform + Ansible stuff is pretty janky too and causes problems.

@TheBigLombowski

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.

@TheBigLombowski

Reality...

Me

CTO + Coworkers

@TheBigLombowski

Lessons 1.0

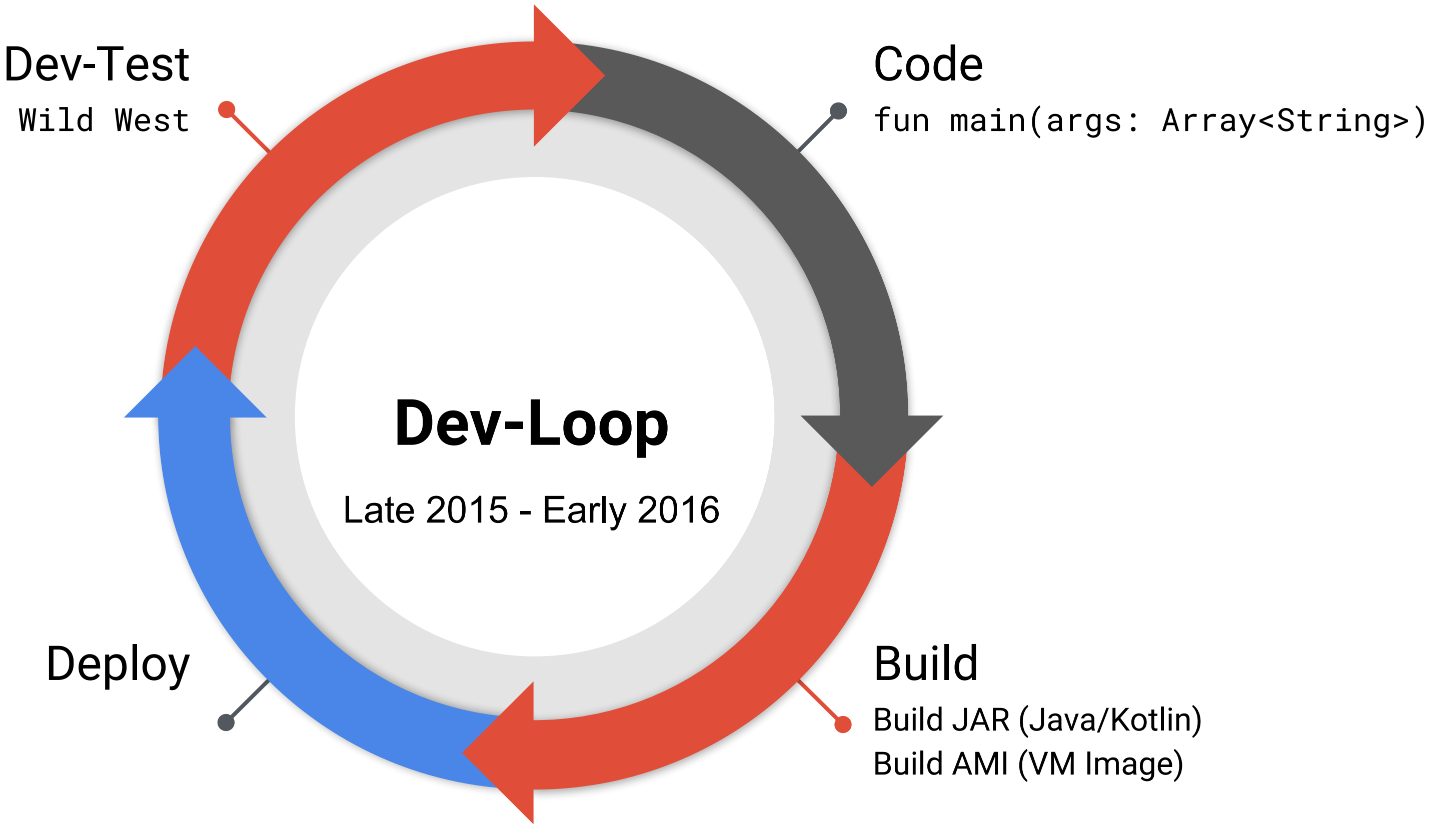
1. I shouldn’t ignore my developers!

2. I need to listen to their problems more…

Problems 1.1 and 1.2

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

Problem: It is totally possible for myself and a team of really bright engineers to identify, solutioneer and agree to fix the wrong problem...


Dev-Test
Wild West
Code
fun main(args: Array<String>)
Dev-Loop
Late 2015 - Early 2016
Build
Build JAR (Java/Kotlin)
Build AMI (VM Image)
Deploy
Python! (swap.py)

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

We threw all the hot tech of the time at the problems created by swap.py!

Spinnaker

EC2

Expected Outcome… :D

@TheBigLombowski

… Actual Reaction D:

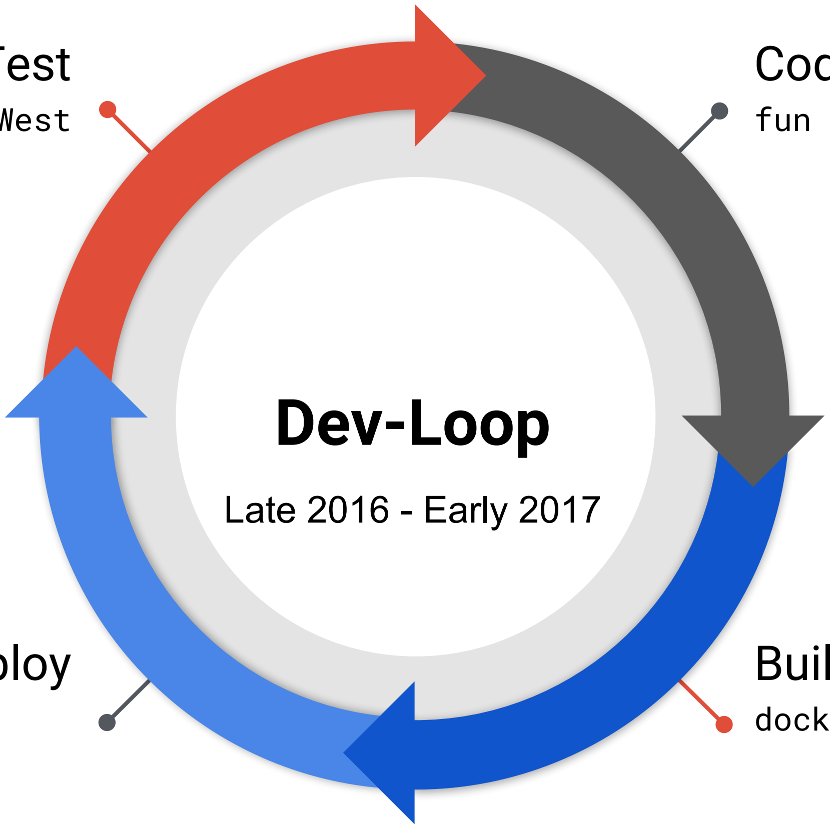
@TheBigLombowski

Lessons 2.0

- I’m thankful my bosses are patient people ;)
- We need to do a better job identifying the 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

`fun main(args: Array<String>)`

Build

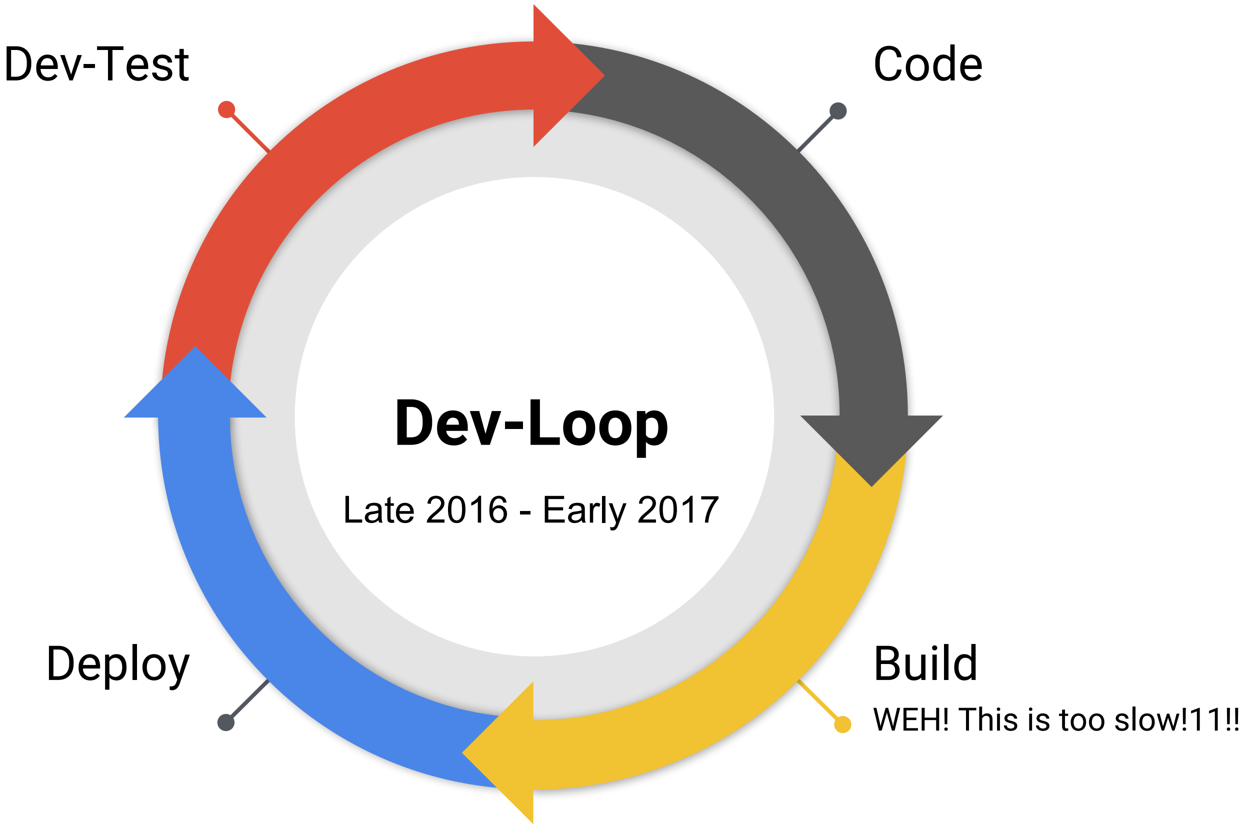
`docker build ...`

Deploy

`Kubectl <...>`

Good News!

- Scrapping + Rethink a success!
- Devs Happy! Velocity increased!


Dev-Test
Code
Dev-Loop
Late 2016 - Early 2017
Deploy
Build
WEH! This is too slow!11!!

Docker build...

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

120 seconds*

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

120 Seconds is Optimistic For Most

- (Java) Compile in around 30 seconds if you're a Java dev? Nope. Maven is still downloading the world 30 seconds in.
- (Python) Most Docker images do some kind of system update in an an early RUN statement (e.g. `apt-get install gcc`) then compile a postgresql lib.
- Slow networks…

And it’s really 120s x # of Reloads

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

Faster Builds… Attempt #1

1. Optimize the use of `Dockerfile` instructions (`COPY, RUN, etc`) for better caching.

2. Multi-stage

3. Shrink the size of the image to speed up registry push… lots of well known techniques
    a. Alpine Linux
    b. `FROM scratch`
    c. multi-stage

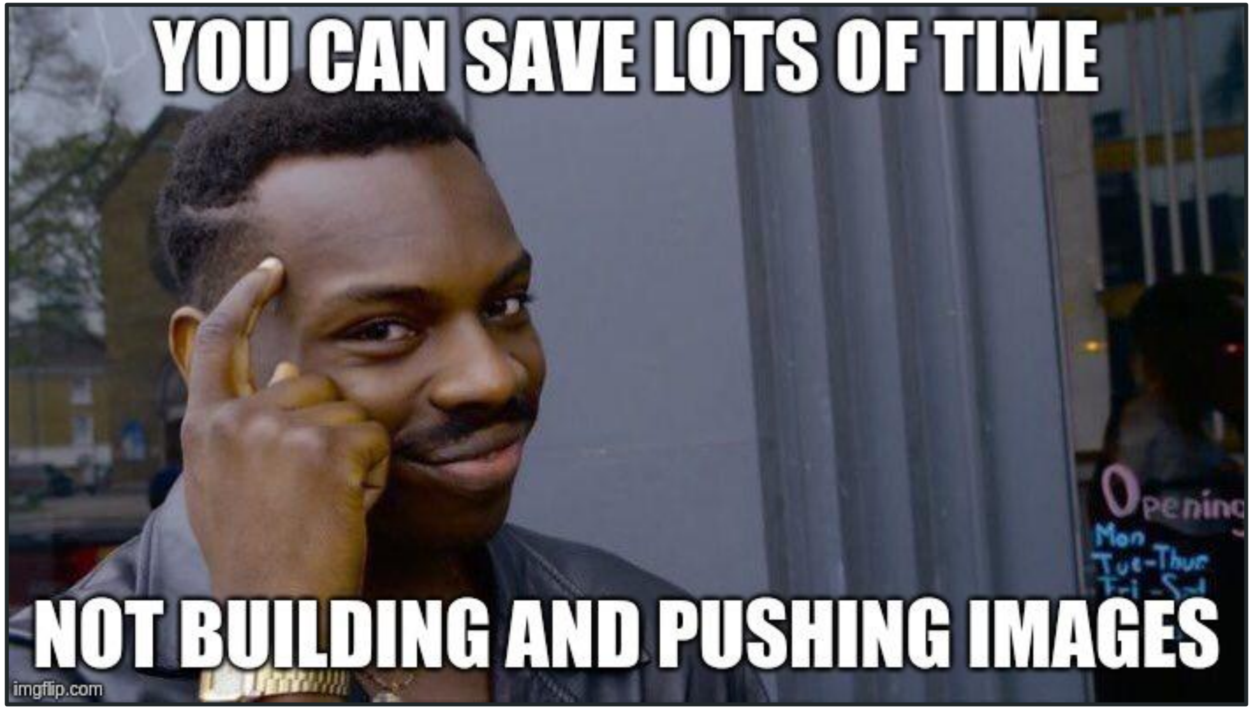
Not Enough!

- Shaves some seconds… especially caching layers and multi-stage.
- Can still be very frustrating though if you’re doing it enough.
- Even the fastest builds gets stuck in time sink of registry push, redeploy, registry pull.

Faster Builds… Attempt #2

1. Wrote a custom tool that did some crazy stuff around managing Docker layers and and continuously rebuilding…

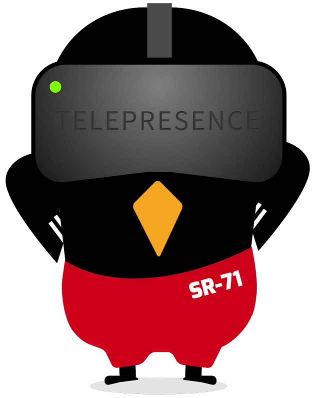
2. Faster builds but generated monster sized images that took ages to push and pull… so net gain of nothing.



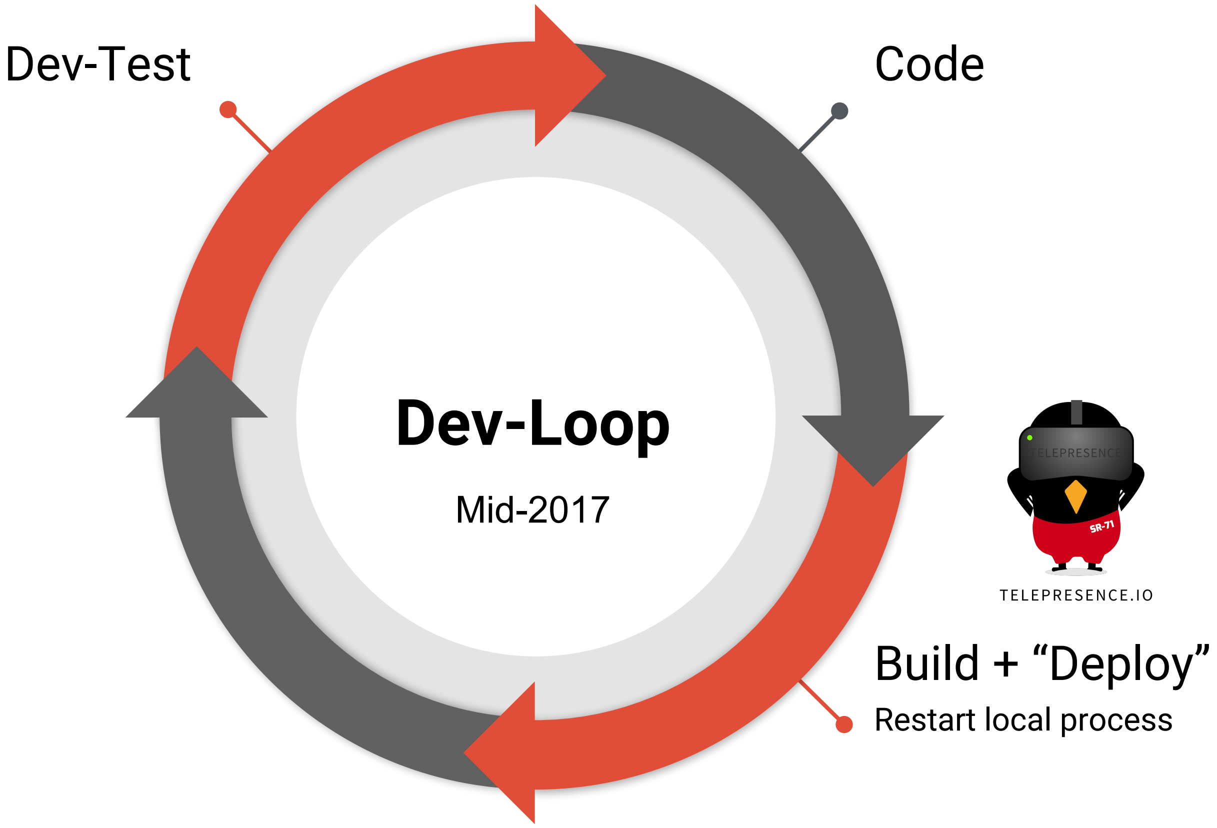
@TheBigLombowski

What if...

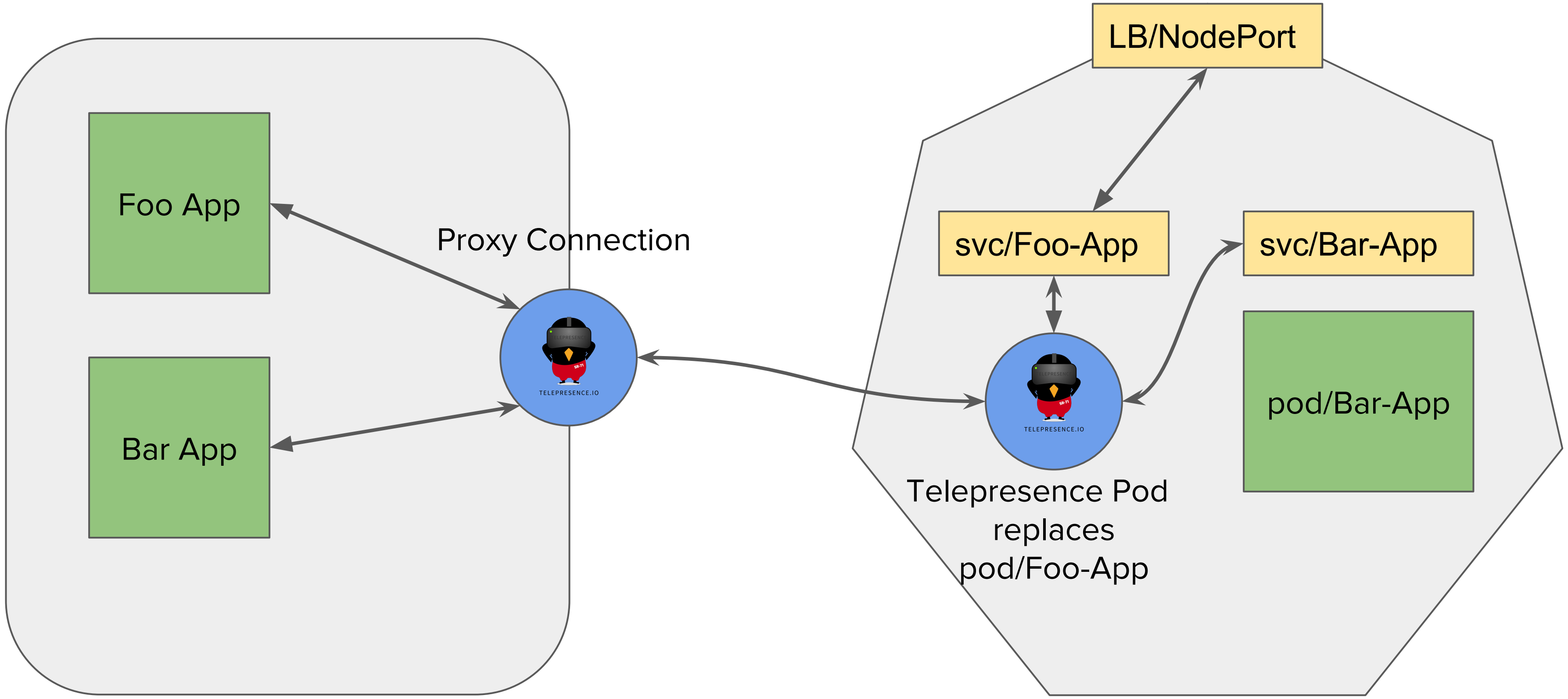
Your dev machine ran your code and it just worked the same as if you ran it on Kubernetes?



TELEPRESENCE.IO


Dev-Test
Code
Dev-Loop
Mid-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 dev 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.
- I die a little inside everytime I see `printf` debugging.

Expected Outcome… :D

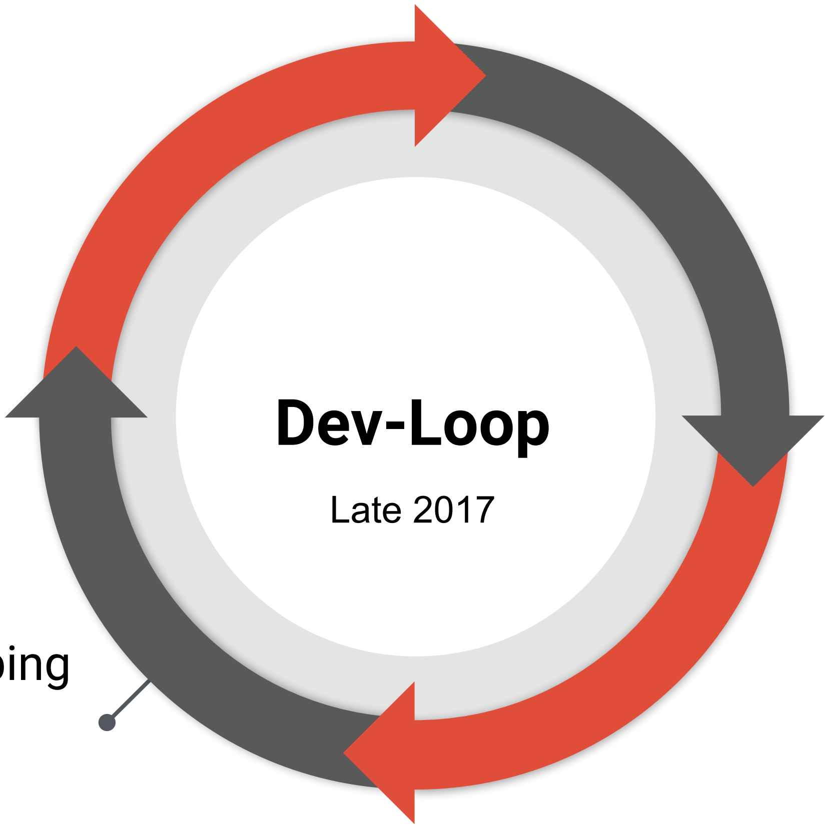
@TheBigLombowski

Actual Outcome... :D

@TheBigLombowski

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.

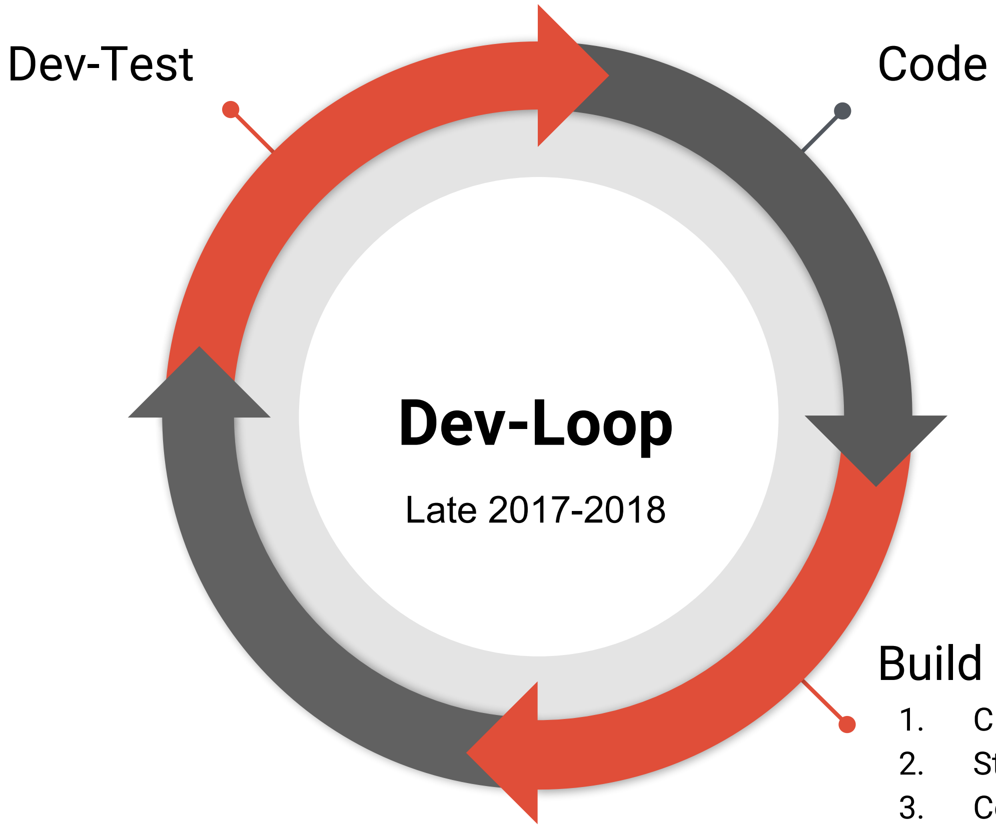
Minikube is for Unicorns

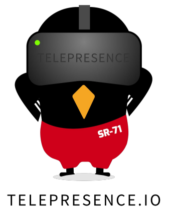
@TheBigLombowski

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.





Build + “Deploy”

1. Claim Cluster
2. Start Telepresence
3. Code

How we do it @ Datawire.io?

- We run about 20 fresh Kubernetes clusters in a pool.
    - Yes this is expensive… but so is developer time
    - Costs mitigated with Spot Instances (AWS) and Preemptible VMs (Google)
- Devs use one-line command to “claim” or “discard” a cluster. Dev gets a `kubeconfig` file and puts an exclusive lock on the cluster.
    - `kubernaut claims create “issue/940”`
- No waiting for cluster to start. Everything is “ready” the moment they receive the cluster.
- New clusters are added to the pool as capacity requirements increase.

The Cloud Native Loop @ Datawire.io

`kubernaut claim …`	(gets a cluster)
`Deploy Ambassador`	(sets up an API gateway)
`Start telepresence`	(links local to cluster)

`Start hacking on code locally!`

`(There's a bit more to it… but details for next Kubecon ;)`

@TheBigLombowski

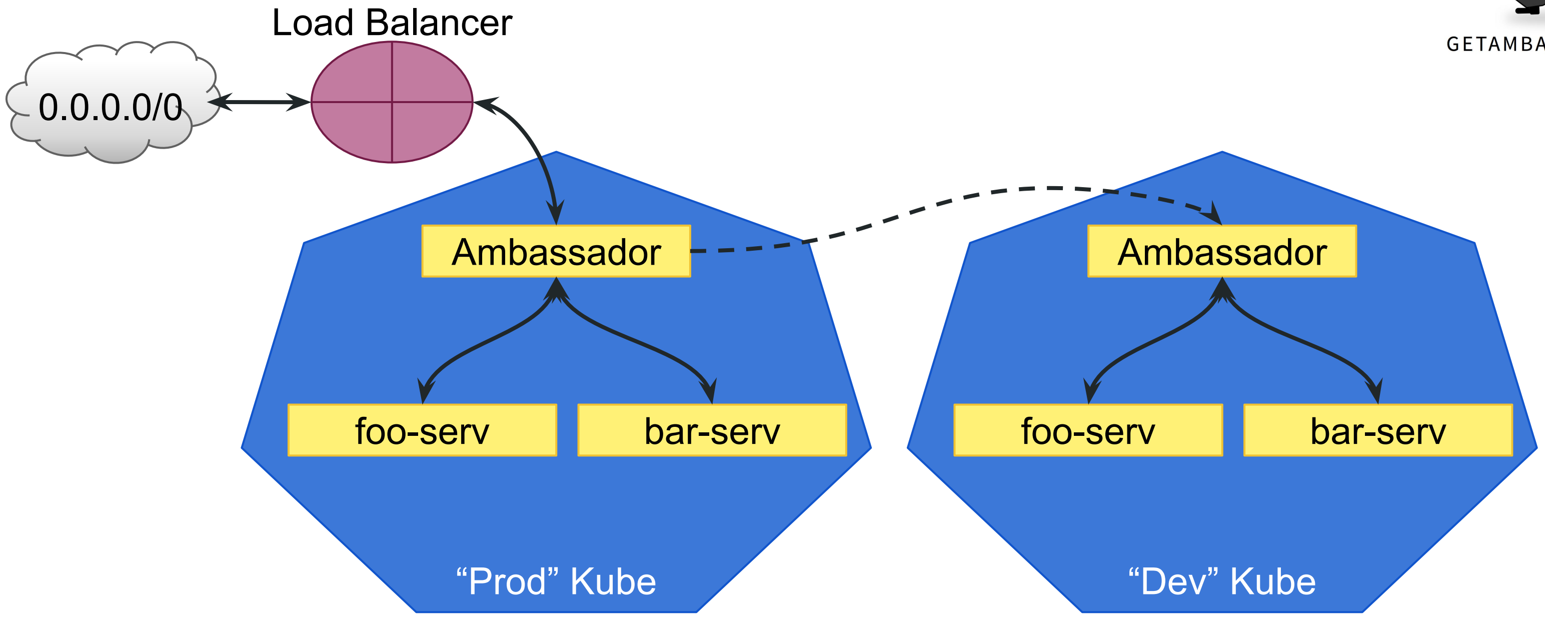
The Ugly: Dev Testing

- Dev-Testing is painful without shared remote infrastructure

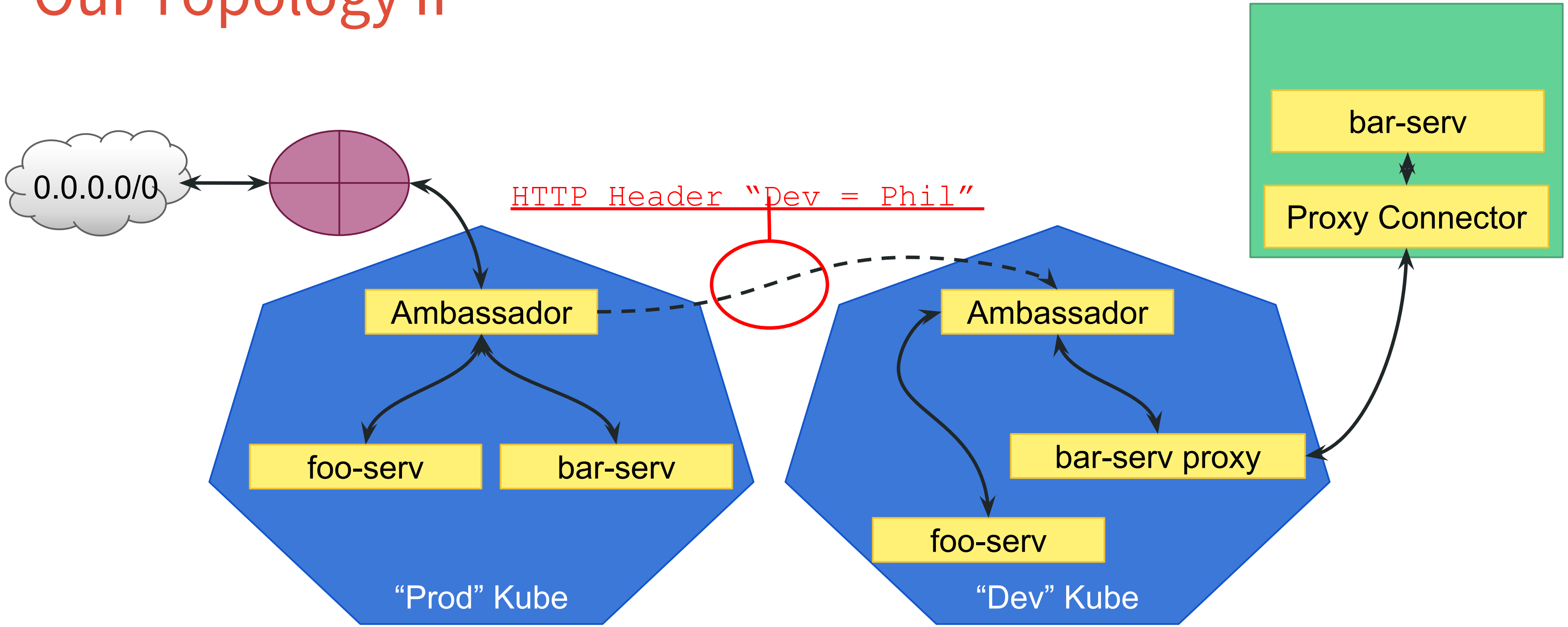
- Recommended way of doing it
  - Nested API Gateways and some combination of HTTP Host/Header/Path based routing.

- Your API Gateway needs to be self-service for developers and easy to safely reconfigure.

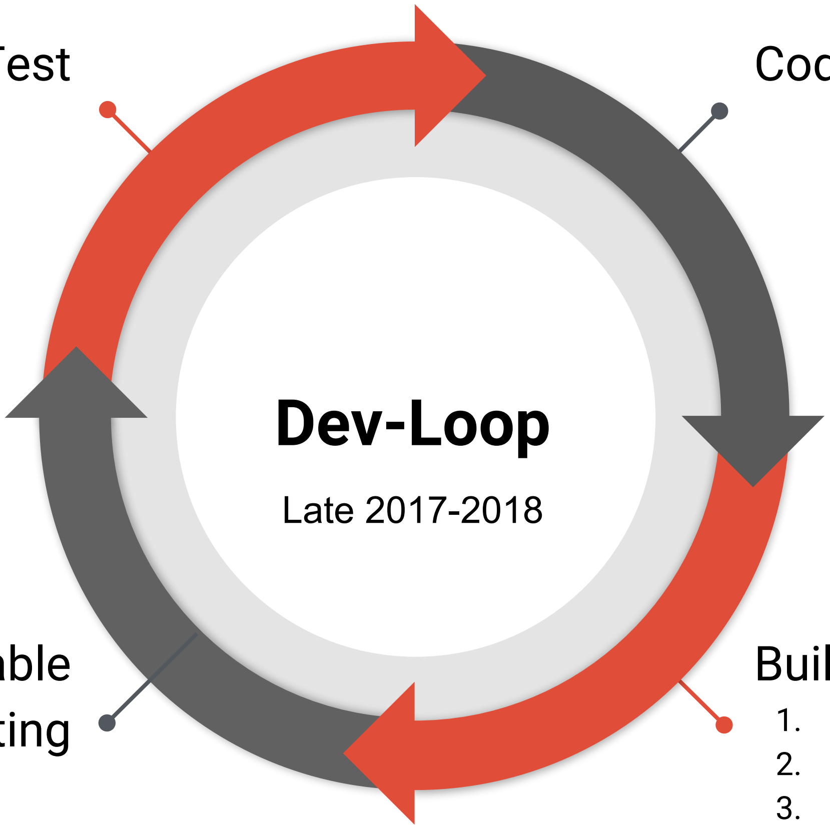
Our Topology



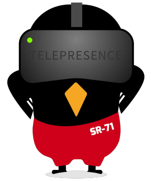
Our Topology II



@TheBigLombowski



1. Claim Cluster
2. Start Telepresence
3. Code



And that’s where we are today...

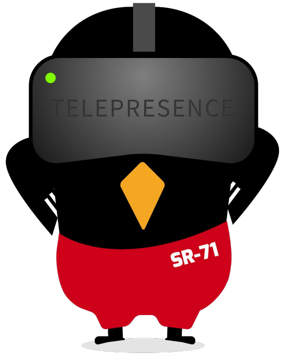
- Still painful things to address
- Devs are way more productive than when we started: minimal waiting for things to happen.
- We intend to keep evolving this as we scale out our engineering team.

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!

- I will be at the **Datawire.io** booth, come and chat :)
- Check out Ambassador and Telepresence!
- Tweet me! **@TheBigLombowski**



TELEPRESENCE.IO

GETAMBASSADOR.IO