Ellen Körbes

The Quest For The Fastest Deployment Time

Why?

The best indicator of a healthy development workflow is a **short feedback loop**.

One of the reasons behind **Go** is to address the absurd **compile time** of C++.

Go has achieved that goal.

But...

Why?

...**there's more** to feedback loop than just compile time.

Looking at you, **Kubernetes.**

So...

Let's fix this.

L
@l_korbes

Assuming your team uses #Kubernetes:

How long does it take between changing a line of code, and that code running in your development cluster? (Mark the closest answer.)

Feel free to share details of your setup 😍

30 minutes	22.6%
1 minute	21.6%
2 seconds	4.9%
Results	**50.9%**

1,931 votes · Final results

3:38 PM · Nov 29, 2019 · Twitter Web App

How?

Follow **three** important rules:

1. **Don’t rely on CI** for development feedback.
2. **Use an MDX* tool** like **Tilt**, Skaffold, Telepresence, or Garden.
3. Then use **every hack, cheat, and dirty trick** in the book.

* Multiservice Development eXperience

Goal:

This talk is an exploration into **how** to achieve the **shortest** possible **feedback loop**.

The Initial Setup

The Initial Setup

- A simple **Go** app
- Running on **Kubernetes** (microk8s & GCP)
- Set up for **Tilt**
- With **benchmark trickery**
- github.com/tilt-dev/pixeltilt

Attempt #0: Basic

Basic — Performance

	Local	Δ-Initial	Remote	Δ-Initial
Basic	56.8s	—	2m10s	—

Basic — How To Improve

Downloading **dependencies** takes forever.

Let’s try...

Attempt #1: Image Layer Cache

Image Layer Cache — Performance

	Local	Δ-Initial	Remote	Δ-Initial
• Basic	56.8s	—	2m10s	—
○ Layer Cache	26.1s	46%	1m16s	58%

Image Layer Cache — How To Improve

Changing dependencies:

- Brings update time to **square #1**.
- And it **re-downloads everything**;
  while vendoring is incremental.

Let's try...

Attempt #2: Vendoring

Vendoring — Performance

	Local	Δ-Initial	Remote	Δ-Initial
Basic	56.8s	—	2m10s	—
Layer Cache	26.1s	46%	1m16s	58%
Vendoring	29.5s	52%	1m11s	55%

Vendoring — How To Improve

Awesome! But can we **cache**

more than just dependencies?

What if we could use the...

Attempt #4: Compiler Cache

Compiler Cache — Performance

	Local	Δ-Initial	Remote	Δ-Initial
Basic	56.8s	—	2m10s	—
Layer Cache	26.1s	46%	1m16s	58%
Vendoring	29.5s	52%	1m11s	55%
Compiler Cache	17.6s	31%	53.2s	41%

Compiler Cache — How To Improve

Great! But...

All this **building and shipping containers** seems **wasteful**.

What if we keep the same container running and just...

The Quest For The Fastest Deployment Time

Attempt #5: Hot-Reload The Source Code

Live Update — Performance

	Local	Δ-Initial	Remote	Δ-Initial
Basic	56.8s	—	2m10s	—
○ Layer Cache	26.1s	46%	1m16s	58%
○ Vendoring	29.5s	52%	1m11s	55%
○ Compiler Cache	17.6s	31%	53.2s	41%
Live Update	3.41s	6%	6.34s	5%

Live Update — Caveats

Heck yeah!

But the images are **different from production images**.

And less CPU means...

The Quest For The Fastest Deployment Time

Attempt #5½: Hot-Reload With Less CPU

Live Update <CPU — Performance

	Local	Δ-Initial	Remote	Δ-Initial
• Basic	56.8s	—	2m10s	—
○ Layer Cache	26.1s	46%	1m16s	58%
○ Vendoring	29.5s	52%	1m11s	55%
○ Compiler Cache	17.6s	31%	53.2s	41%
• Live Update	3.41s	6%	6.34s	5%
○ <CPU	**F O R E V E R**			

Live Update <CPU — How To Improve

What if...

Instead of **syncing** the source **code** and **compiling** inside the **container**...

What if we...

The Quest For The Fastest Deployment Time

Attempt #6:
Compile Locally & Hot-Reload The Binary

Local Build — Performance

	Local	Δ-Initial	Remote	Δ-Initial
Basic	56.8s	—	2m10s	—
○ Layer Cache	26.1s	46%	1m16s	58%
○ Vendoring	29.5s	52%	1m11s	55%
○ Compiler Cache	17.6s	31%	53.2s	41%
Live Update	3.41s	6%	6.34s	5%
Local Build	3.69s	6%	21.5s	17%

L
@l_korbes

Problem:
/app # ls
binary
/app # ./binary
sh: ./binary: not found

* 30 minutes later *

Solution:
CGO_ENABLED=0

🤦‍♀️ #golang

9:36 PM · Jan 12, 2020 · Twitter Web App

View Tweet activity

13 Retweets **126** Likes

Local Build — Caveats

Check out Jérôme Petazzoni's blog series **Reducing Image Size**.

With **large binaries**, moving them around becomes a **bottleneck**.

In that case, how about we...

The Quest For The Fastest Deployment Time

Attempt #7:
Remove Debugging Artifacts

No Debugging Artifacts — Performance

	Local	Δ-Initial	Remote	Δ-Initial
• Basic	56.8s	—	2m10s	—
◦ Layer Cache	26.1s	46%	1m16s	58%
◦ Vendoring	29.5s	52%	1m11s	55%
◦ Compiler Cache	17.6s	31%	53.2s	41%
• Live Update	3.41s	6%	6.34s	5%
• Local Build	3.69s	6%	21.5s	17%
◦ No Debugging	2.89s	5%	19.5s	15%

No Debugging Artifacts — How To Improve

You call **that** small?

I meant **SMALL!!!!11**

Let's use the...

The Quest For The Fastest Deployment Time

Attempt #7: UPX
The Ultimate Packer for eXecutables!

UPX — Performance

	Local	Δ-Initial	Remote	Δ-Initial
• Basic	56.8s	—	2m10s	—
◦ Layer Cache	26.1s	46%	1m16s	58%
◦ Vendoring	29.5s	52%	1m11s	55%
◦ Compiler Cache	17.6s	31%	53.2s	41%
• Live Update	3.41s	6%	6.34s	5%
• Local Build	3.69s	6%	21.5s	17%
◦ No Debugging	2.89s	5%	19.5s	15%
◦ UPX	4.93s	9%	13.6s	10%

UPX — Caveats

Compressing a binary takes time.

It **makes sense when using a remote cluster**, not when using a local one.

The Quest For The Fastest Deployment Time

Who's The Winner?

Final Score

	Local	Δ-Initial	Remote	Δ-Initial
• Basic	56.8s	—	2m10s	—
◦ Layer Cache	26.1s	46%	1m16s	58%
◦ Vendoring	29.5s	52%	1m11s	55%
◦ Compiler Cache	17.6s	31%	53.2s	41%
• Live Update	**3.41s**	**6%**	**6.34s**	**5%**
• Local Build	3.69s	6%	21.5s	17%
◦ No Debugging	**2.89s**	**5%**	19.5s	15%
◦ UPX	4.93s	9%	**13.6s**	**10%**

Thank you!

Ellen Körbes

Developer Relations

- l@tilt.dev
- @ellenkorbes
- they/them
- #tilt@slack.k8s.io