KubeCon | CloudNativeCon

North America 2023

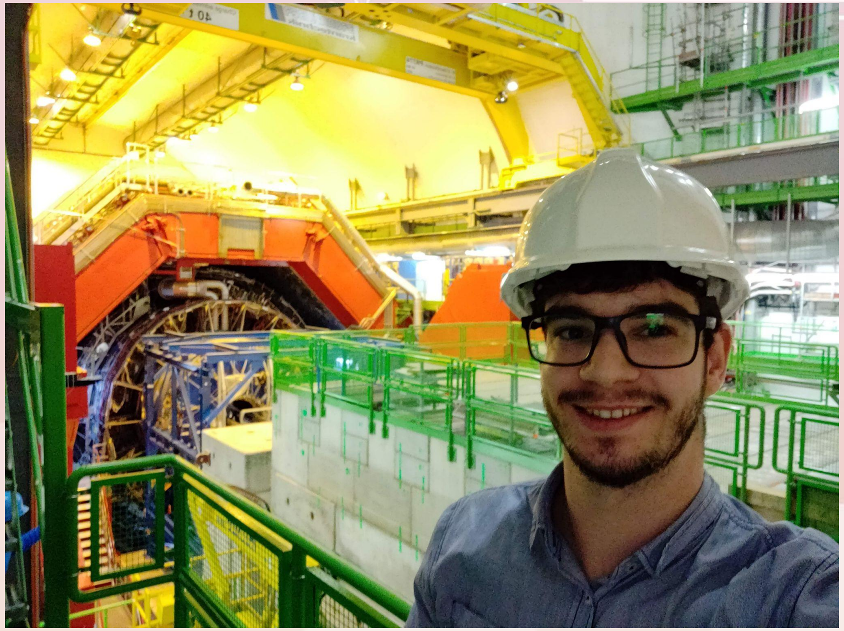
KubeCon | CloudNativeCon

North America 2023

Turbocharging Compilation Times: Kubernetes Orchestration with Distributed Compilation Tools

Diogo Guerra, formerly @ CERN

Who



Diogo Guerra

Former Computing Engineer, Kubernetes team @ CERN

www.linkedin.com/in/diogoguerraist

dy090.guerra@gmail.com

Once Upon a Time…

Source: https://www.reddit.com/media?url=https%3A%2F%2Fi.redd.it%2F38iqur4116x41.jpg

- What makes compilation time-consuming?
- When to go distributed? Are there ready-to-use solutions available?
- Can we use Kubernetes to scale a distributed compilation service on-demand?

Motivation

CERN develops and maintains many applications:

- ROOT - framework to analyse and visualize large amounts of data
- CVMFS - read-only file system for software distribution

This solutions are usually big and resource-consuming to compile.

CERN Root homepage

Teams have constrained resources:

- Cost of infrastructure for the team/organization
  - Expensive to set up, maintain and run
- Users’ default is to opt for the easiest solution

CERN CVMFS homepage

Challenges

Compute challenges:

- Small teams have lower budget for resources
- Big computer node (64+ threads) or distributed compilation required

Organization challenges:

- Solution easily integratable with existing compile definitions/workflows
- No ‘standard’ way offered to the community
- Use of public machines in shared Hypervisors for Heavy Workloads

Landscape of Some Distributed Compile Tools

Criteria:

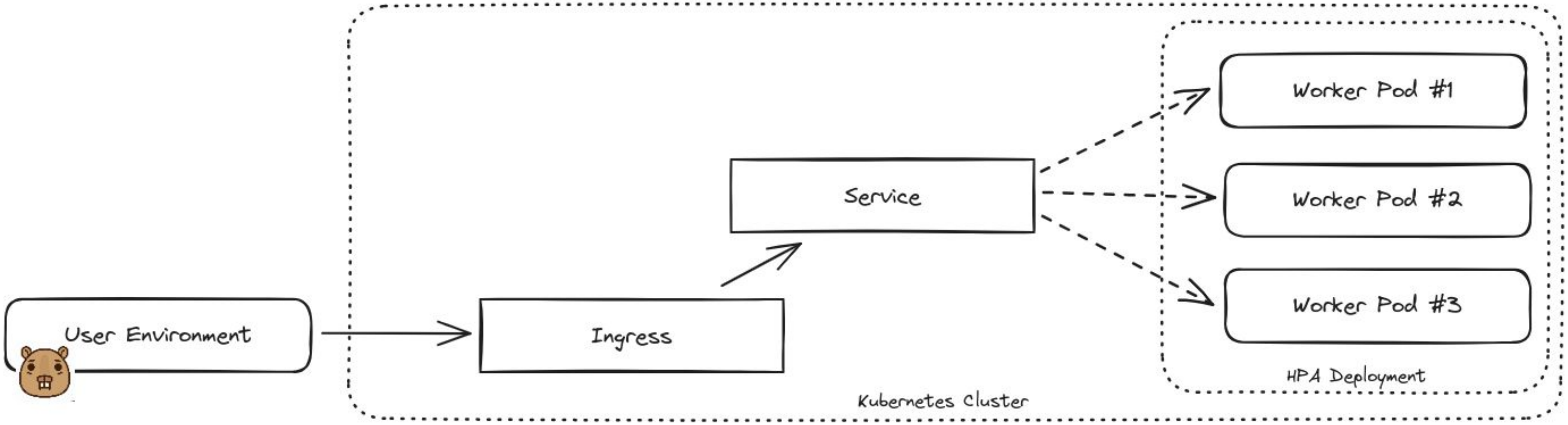
- Compatibility
- Ease of configuration
- Does it work with Kubernetes

Explored tools:

- distcc
- icecc
- sccache
- gg

Source: https://twitter.com/AmyfStuart/status/735108951138766849/photo/1

Setup - distcc



Key Takeaways - distcc

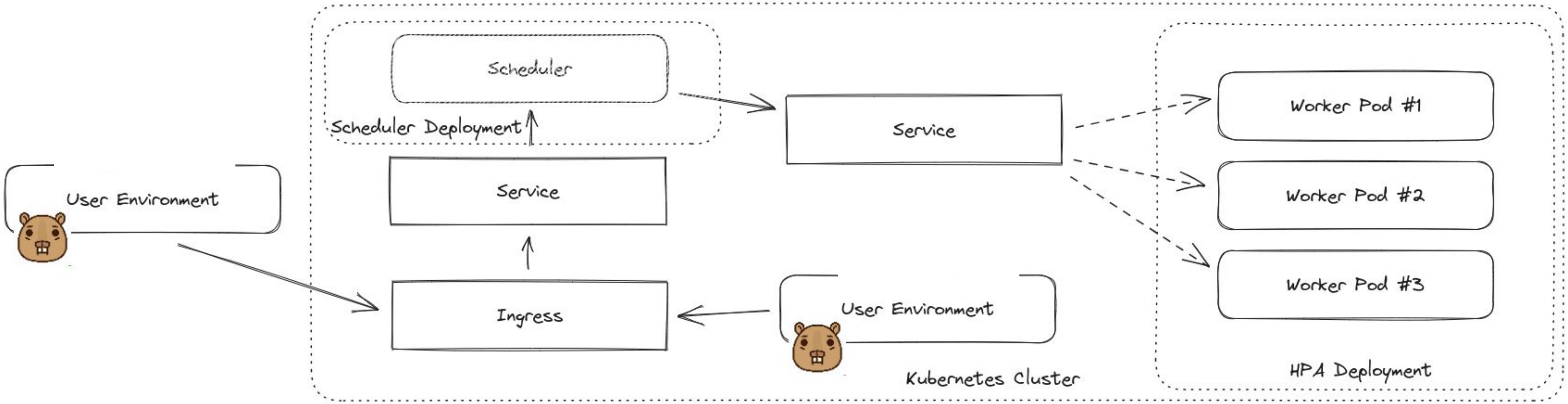
Pros:

- Most known and widely used tool

Cons:

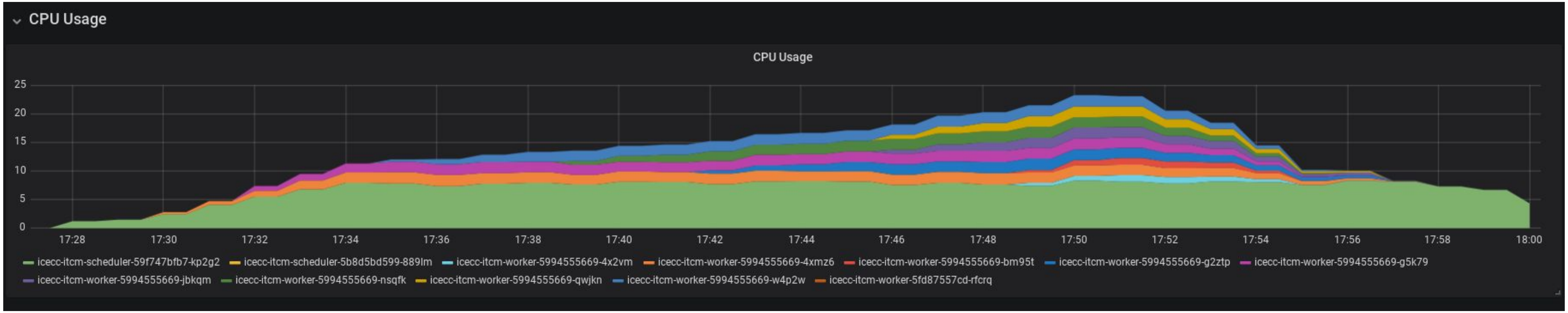
- distcc is static
- Kubernetes service will assign jobs to random workers
- The client controls the whole compile job

Setup - icecream

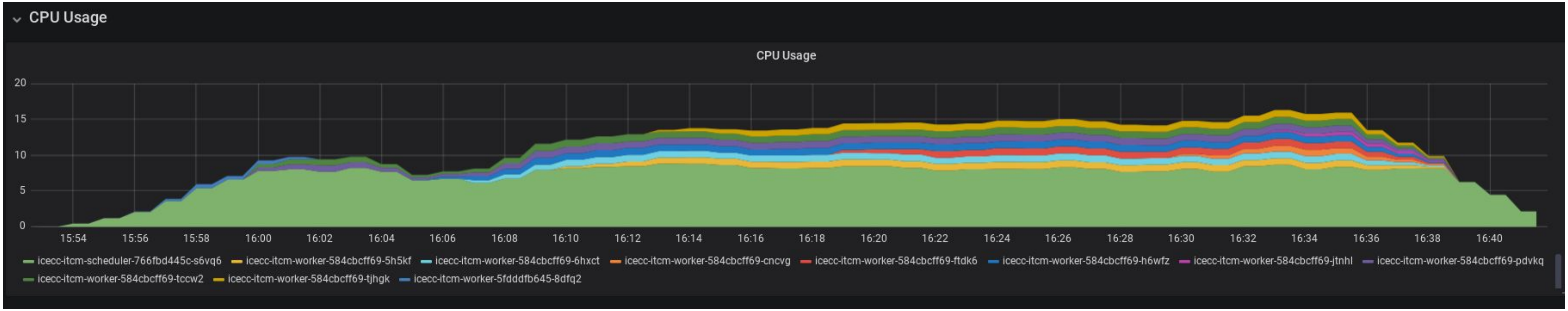


Testing - icecream

8 CPU - Scheduler & 2 CPU - Workers



8 CPU - Scheduler & 1 CPU - Workers



Key Takeaways - icecream

Pros:

- Icecc supports worker discovery
- Failed jobs fallback to local computer

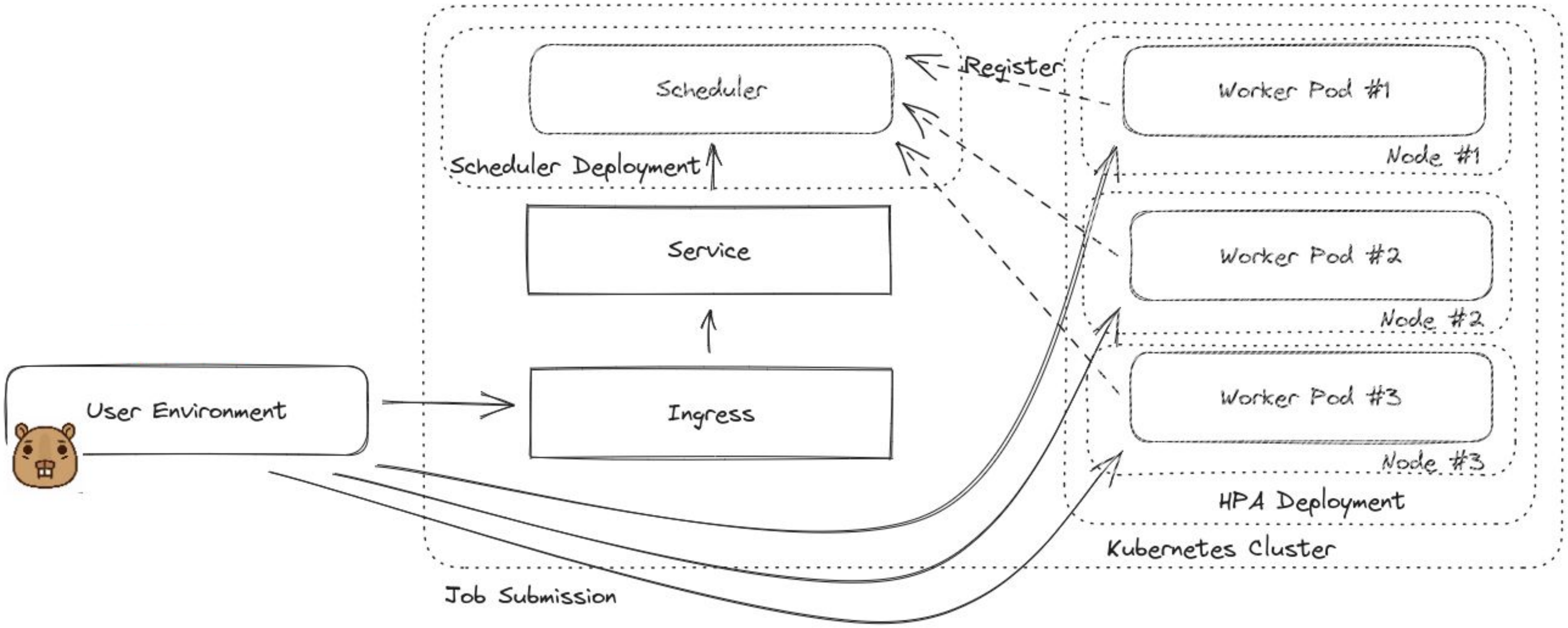
Cons:

- Requirement of setting up dependencies
- Scaling is a problem

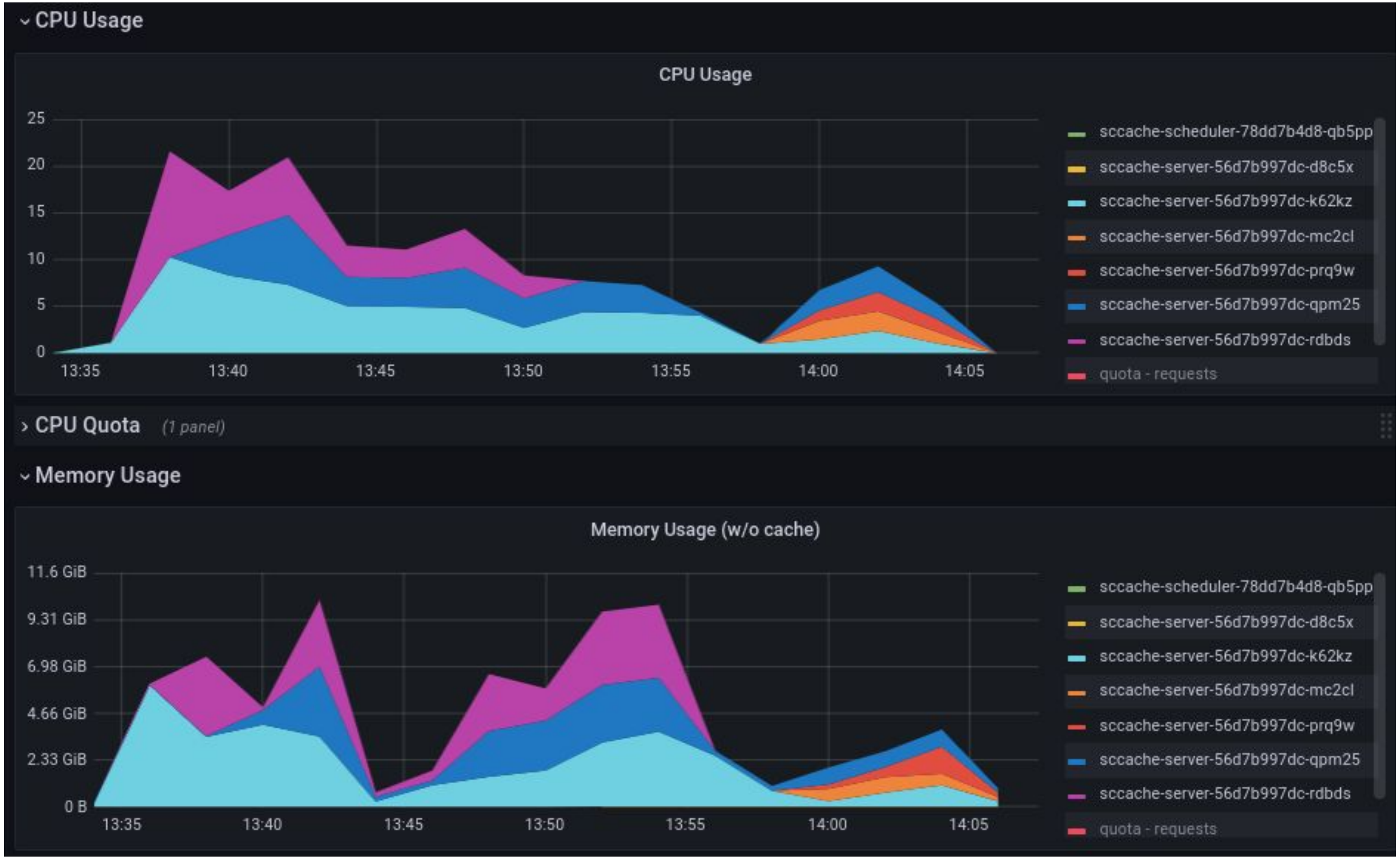
No real stable solution…

Source: https://images.app.goo.gl/SEr8n38eKqioXQ9s8

Setup - sccache



Testing - sccache


CPU Usage
CPU Usage
25
20
15
10
5
0
13:35
13:40
13:45
13:50
13:55
14:00
14:05
sccache-scheduler-78dd7b4d8-qb5pp
sccache-server-56d7b997dc-d8c5x
sccache-server-56d7b997dc-k62kz
sccache-server-56d7b997dc-mc2cl
sccache-server-56d7b997dc-prq9w
sccache-server-56d7b997dc-qpm25
sccache-server-56d7b997dc-rdbds
quota - requests
CPU Quota (1 panel)
Memory Usage
Memory Usage (w/o cache)
11.6 GiB
9.31 GiB
6.98 GiB
4.66 GiB
2.33 GiB
0 B
13:35
13:40
13:45
13:50
13:55
14:00
14:05
sccache-scheduler-78dd7b4d8-qb5pp
sccache-server-56d7b997dc-d8c5x
sccache-server-56d7b997dc-k62kz
sccache-server-56d7b997dc-mc2cl
sccache-server-56d7b997dc-prq9w
sccache-server-56d7b997dc-qpm25
sccache-server-56d7b997dc-rdbds
quota - requests

Key Takeaways - sccache

Pros:

- No dependency management needed
- Compatible with rust projects

Cons:

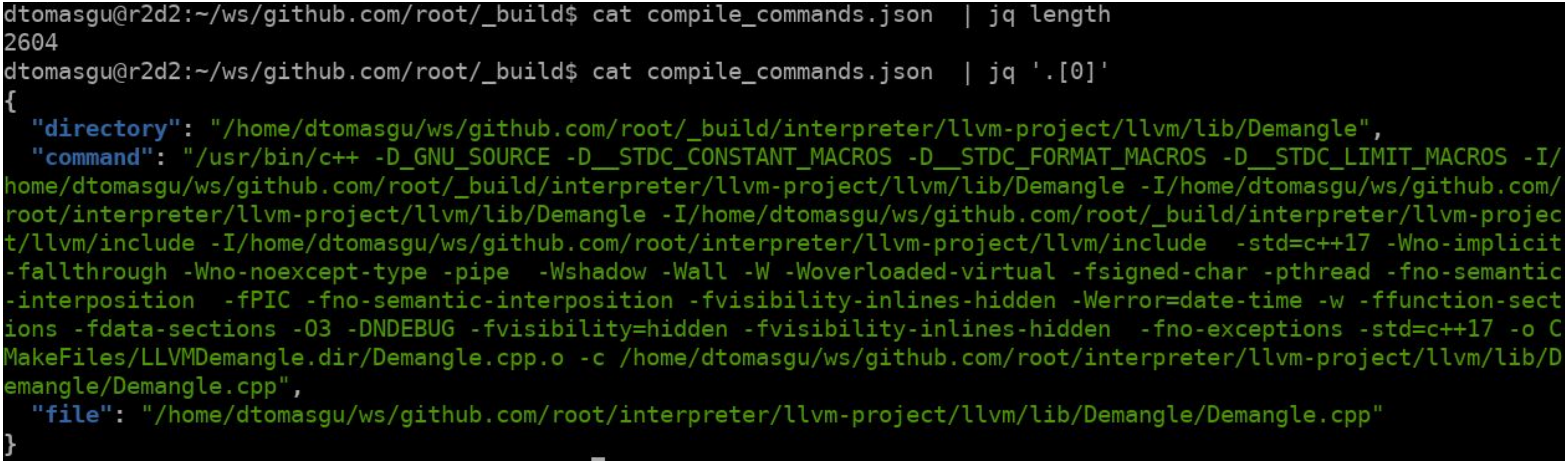
- Client needs access to worker node
- sccache uses bubblewrap
  - Access to namespaces & privileges
- Job limit not supported

Can we go ‘Cloud Native?’

Source: https://www.reddit.com/r/capybara/comments/vmt4ax/capybara_on_vacation/

Compilation Database

- Many Compilation Tools create a compilation database.

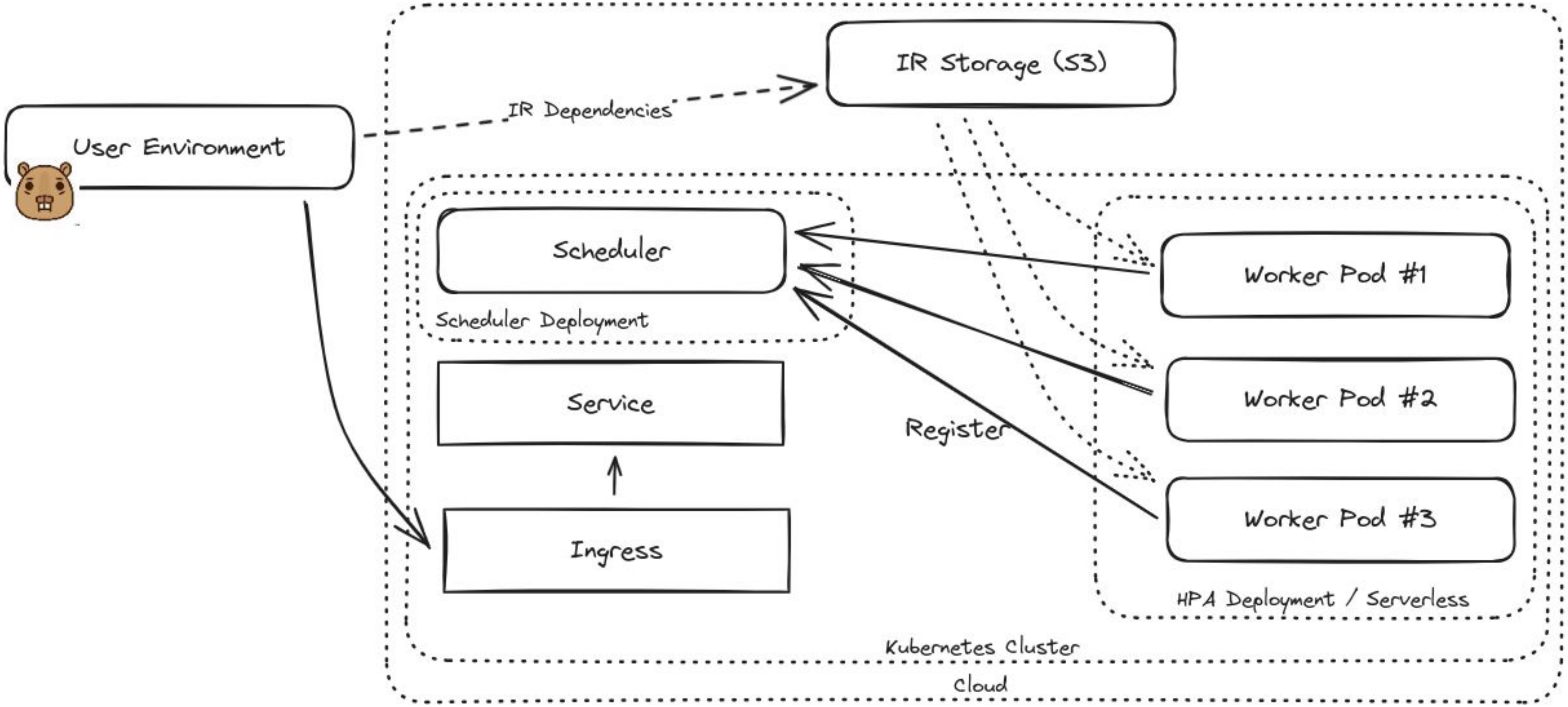


```
dtomasgu@r2d2:~/ws/github.com/root/_build$ cat compile_commands.json  | jq length
2604
dtomasgu@r2d2:~/ws/github.com/root/_build$ cat compile_commands.json  | jq '.[0]'
{
  "directory": "/home/dtomasgu/ws/github.com/root/_build/interpreter/llvm-project/llvm/lib/Demangle",
  "command": "/usr/bin/c++ -D_GNU_SOURCE -D__STDC_CONSTANT_MACROS -D__STDC_FORMAT_MACROS -D__STDC_LIMIT_MACROS -I/
home/dtomasgu/ws/github.com/root/_build/interpreter/llvm-project/llvm/lib/Demangle -I/home/dtomasgu/ws/github.com/
root/interpreter/llvm-project/llvm/lib/Demangle -I/home/dtomasgu/ws/github.com/root/_build/interpreter/llvm-projec
t/llvm/include -I/home/dtomasgu/ws/github.com/root/interpreter/llvm-project/llvm/include  -std=c++17 -Wno-implicit
-fallthrough -Wno-noexcept-type -pipe  -Wshadow -Wall -W -Woverloaded-virtual -fsigned-char -pthread -fno-semantic
-interposition  -fPIC -fno-semantic-interposition -fvisibility-inlines-hidden -Werror=date-time -w -ffunction-sect
ions -fdata-sections -O3 -DNDEBUG -fvisibility=hidden -fvisibility-inlines-hidden  -fno-exceptions -std=c++17 -o C
MakeFiles/LLVMDemangle.dir/Demangle.cpp.o -c /home/dtomasgu/ws/github.com/root/interpreter/llvm-project/llvm/lib/D
emangle/Demangle.cpp",
  "file": "/home/dtomasgu/ws/github.com/root/interpreter/llvm-project/llvm/lib/Demangle/Demangle.cpp"
}
```

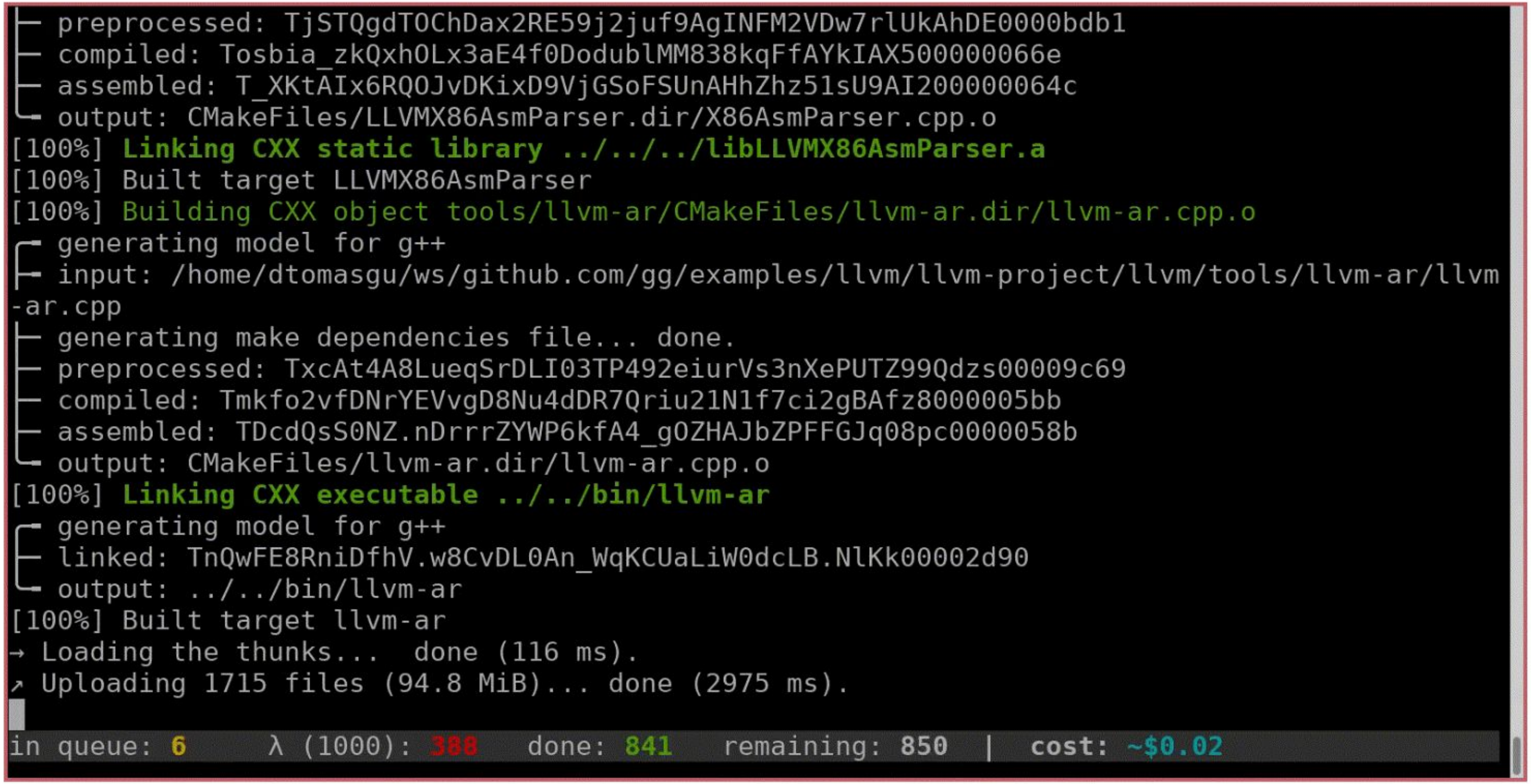
We were having lunch…

Source: https://www.youtube.com/watch?v=UZeaR2N4l8c&ab_channel=%E9%95%B7%E5%B4%8E%E3%83%90%E3%82%A4%E3%82%AA%E3%83%91%E3%83%BC%E3%82%AF%E5%85%AC%E5%BC%8F

Setup - gg


IR Storage (S3)
IR Dependencies
User Environment
Scheduler
Scheduler Deployment
Service
Ingress
Register
Worker Pod #1
Worker Pod #2
Worker Pod #3
HPA Deployment / Serverless
Kubernetes Cluster
Cloud

Testing - gg



```
├─ preprocessed: TjSTQgdTOChDax2RE59j2juf9AgINFM2VDw7rlUkAhDE0000bdb1
├─ compiled: Tosbia_zkQxhOLx3aE4f0DodublMM838kqFfAYkIAX500000066e
├─ assembled: T_XKtAIx6RQOJvDKixD9VjGSoFSUnAHhZhz51sU9AI200000064c
└─ output: CMakeFiles/LLVMX86AsmParser.dir/X86AsmParser.cpp.o
[100%] Linking CXX static library ../../../libLLVMX86AsmParser.a
[100%] Built target LLVMX86AsmParser
[100%] Building CXX object tools/llvm-ar/CMakeFiles/llvm-ar.dir/llvm-ar.cpp.o
┌─ generating model for g++
├─ input: /home/dtomasgu/ws/github.com/gg/examples/llvm/llvm-project/llvm/tools/llvm-ar/llvm
-ar.cpp
├─ generating make dependencies file... done.
├─ preprocessed: TxcAt4A8LueqSrDLI03TP492eiurVs3nXePUTZ99Qdzs00009c69
├─ compiled: Tmkfo2vfDNrYEVvgD8Nu4dDR7Qriu21N1f7ci2gBAfz8000005bb
├─ assembled: TDcdQsS0NZ.nDrrrZYWP6kfA4_gOZHAJbZPFFGJq08pc0000058b
└─ output: CMakeFiles/llvm-ar.dir/llvm-ar.cpp.o
[100%] Linking CXX executable ../../bin/llvm-ar
┌─ generating model for g++
├─ linked: TnQwFE8RniDfhV.w8CvDL0An_WqKCUaLiW0dcLB.NlKk00002d90
└─ output: ../../bin/llvm-ar
[100%] Built target llvm-ar
→ Loading the thunks...  done (116 ms).
↗ Uploading 1715 files (94.8 MiB)... done (2975 ms).

in queue: 6      λ (1000): 388   done: 841   remaining: 850  |  cost: ~$0.02
```

Key Takeaways - gg

Pros:

- Fastest or equal speed
- Fits very well in the ‘Cloud Native’ ideology
- Scalable and flexible
- No dependency management needed

Cons:

- Recent project (2019) with no active users/development
- Some lack of supporting documentation
- Integration work required for more complex projects

Conclusions

- Bigger nodes that accommodate the workload are ALWAYS preferable
- Distributed compilations are useful when:
  - User compile machine is not powerful enough
  - Infrastructure is intended to be shared across an organization
- Legacy distributed compilation tools do not work well in modern architectures
- New paradigm to take advantage of the dynamic presented by cloud native environments

Sources

SCCACHE Helm

StanfordSNR/gg src and Paper

Special thanks

Source: https://www.reddit.com/media?url=https%3A%2F%2Fi.redd.it%2Fxkifu5yylqx41.jpg

Diana Gaponcic

KubeCon | CloudNativeCon

North America 2023

Thank you!