Columbo: A Reasoning Framework for Kubernetes' Configuration Space

Matthijs Jansen, Sacheendra Talluri, Krijn Doekemeijer, Nick Tehrany, Alexandru Iosup, Animesh Trivedi

m.s.jansen@vu.nl

https://atlarge-research.com/mjansen/

@Large Research
Massivizing Computer Systems

VU VRIJE UNIVERSITEIT AMSTERDAM

Columbo
Open-source Code

Containerization

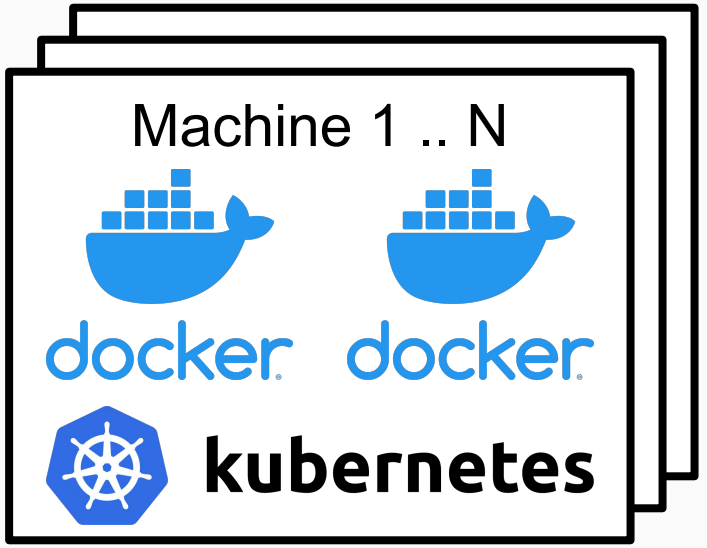
VU VRIJE UNIVERSITEIT AMSTERDAM

Container benefits:

- Virtualization with **little overhead**
- **Portable** and **reproducible** packaging and execution

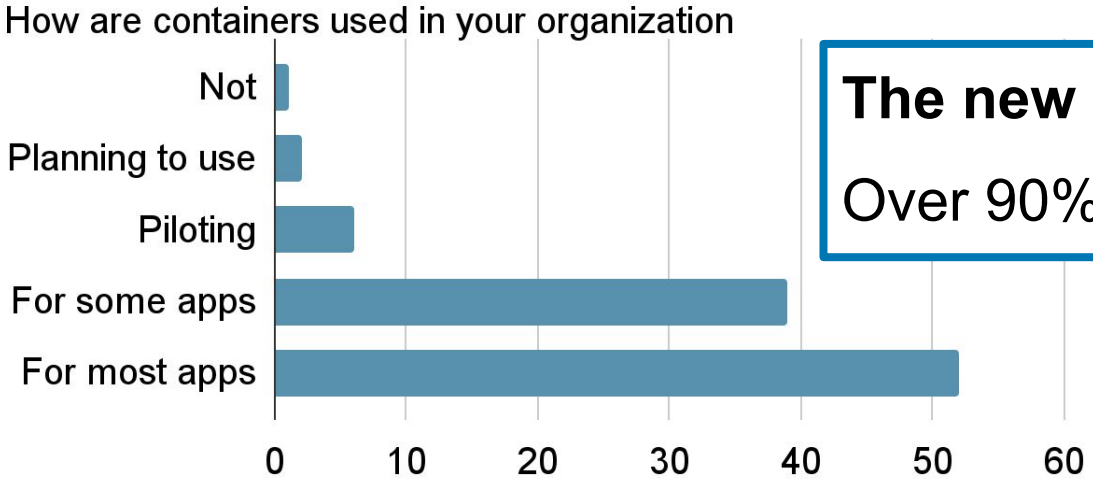
In data centers:

- Many containers across many machines
- **Resource managers** govern life cycle



CNCF 2024 Annual Survey

VU VRIJE UNIVERSITEIT AMSTERDAM

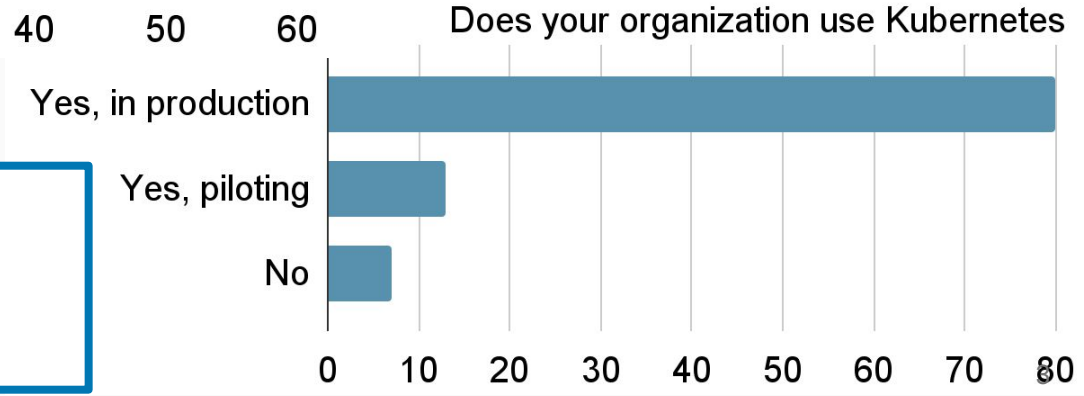


The new normal

Over 90% uses containers

The production standard

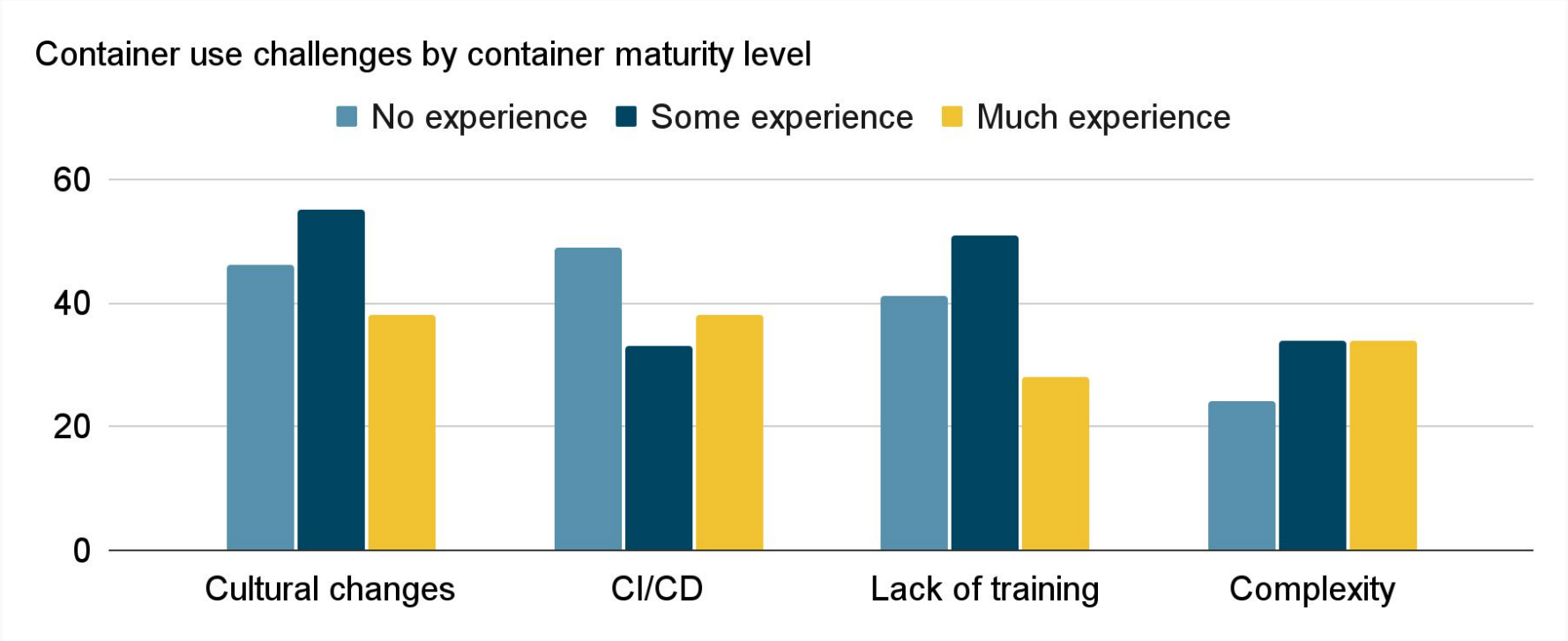
80% uses Kubernetes



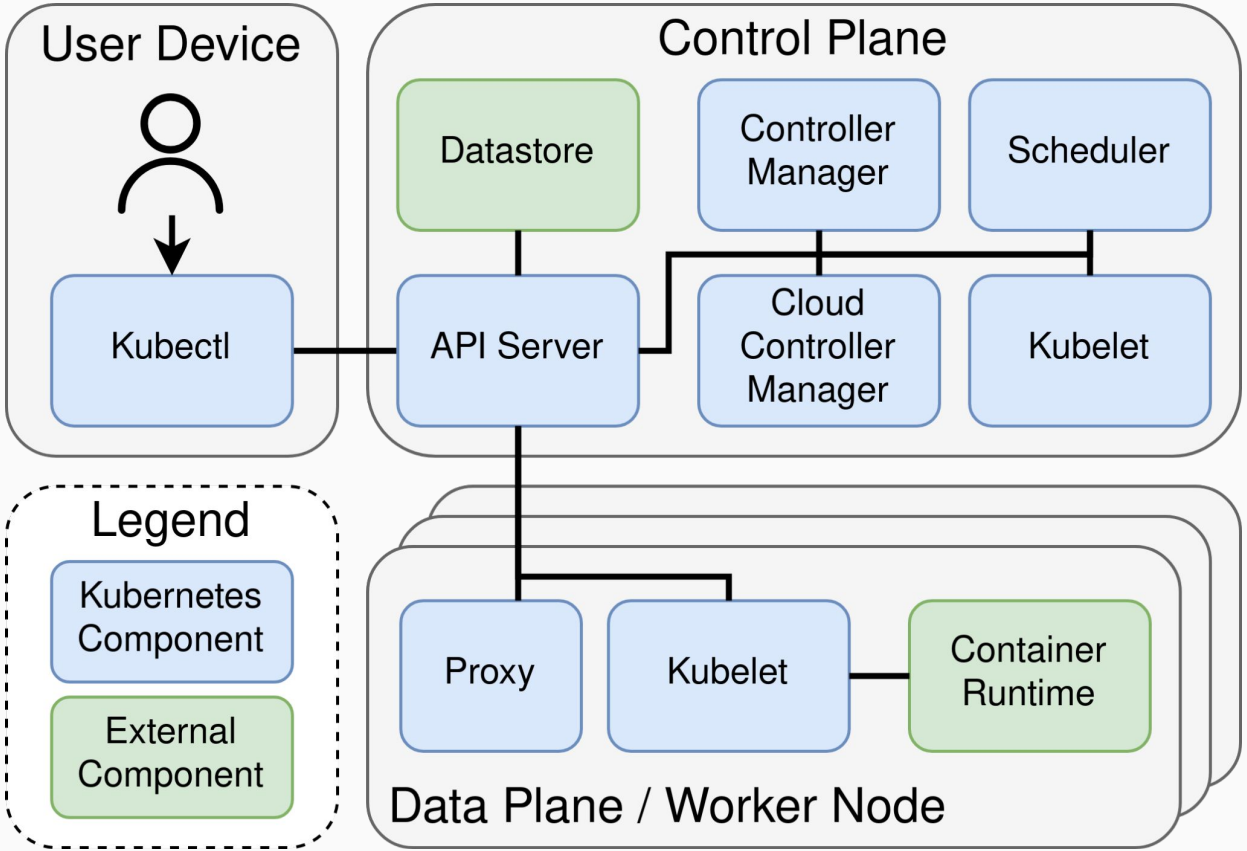
CNCF 2024 Annual Survey

VU VRIJE UNIVERSITEIT AMSTERDAM

But: Containers are **difficult to use**



Kubernetes Configuration Complexity



Architecture with 9 distributed components

- **Control plane**: Decision making
- **Data plane**: Decision execution

Configuration API:

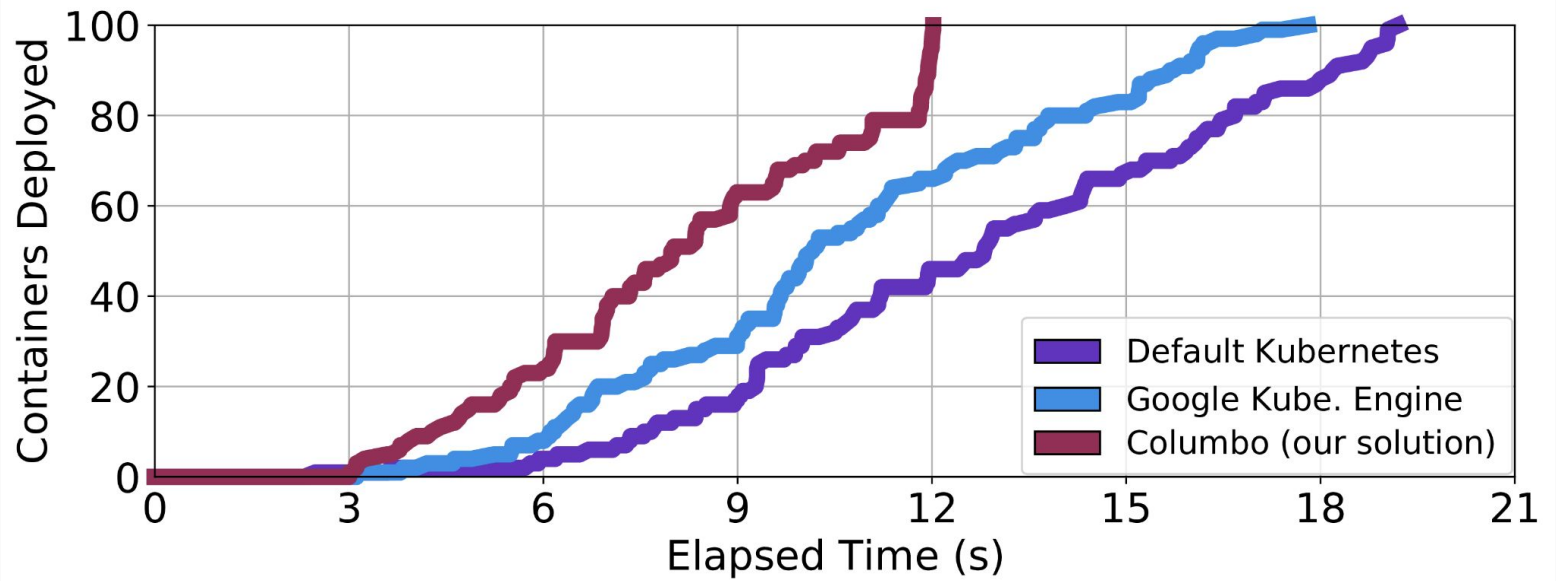
- **234 resources**
- **1598 parameters**

Configurations Are Important

VU VRIJE UNIVERSITEIT AMSTERDAM

Goal: Tune configurations to optimize container deployment latency

- **Google Kubernetes Engine** deploys only **7% faster**
- Configuration tuning with **our approach**: Deploys **37% faster**



Configuration Tuning Strategy

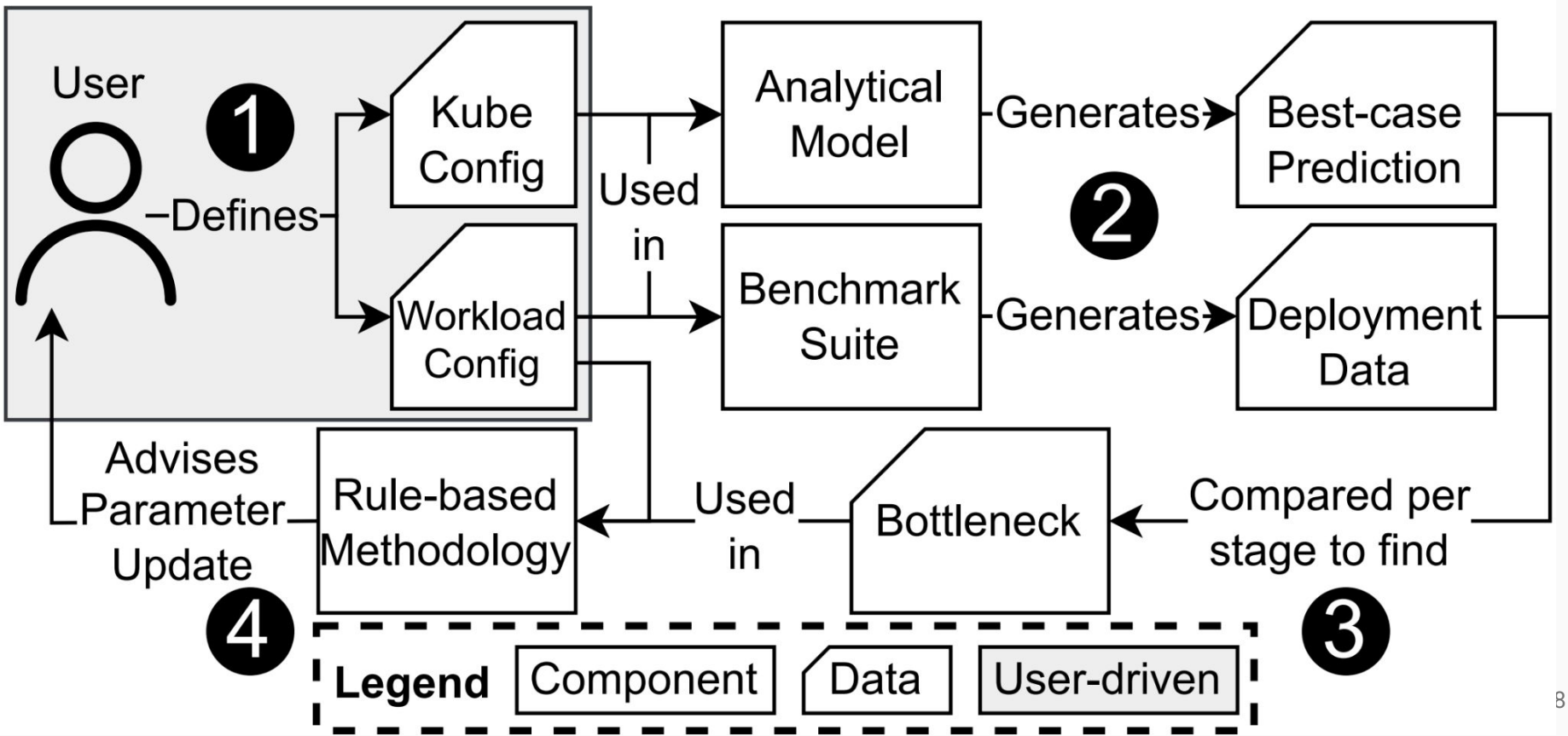
VU VRIJE UNIVERSITEIT AMSTERDAM

Approaches

- **Brute force? No** → Too many parameters, slow updates
- **Users to restrict parameter space? No** → Requires expertise
- **Root-cause analysis? Yes** → Fast and user-friendly

Pipeline Stage	Parameter	Object	Current value	Demanded
3	*Batch Size*	Pods	**5**	100

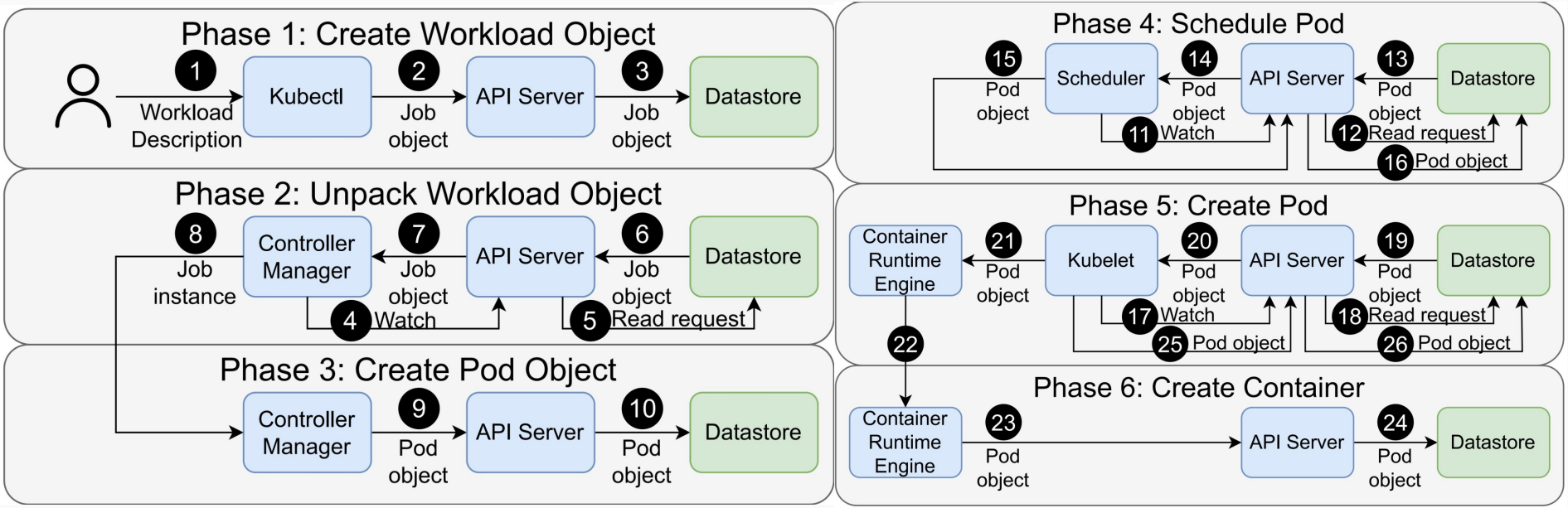
Columbo's Configuration Tuning



Workload Deployment Pipeline

VU VRIJE UNIVERSITEIT AMSTERDAM

6 stages: Unique data object / components control loop
26 steps: Move data between components

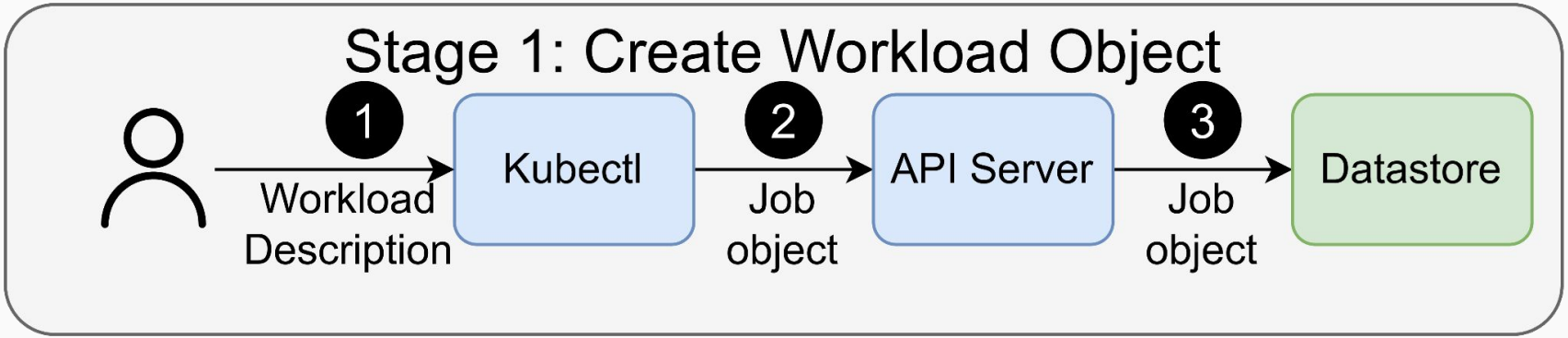


Analytical Performance Model

Best case: Objects are **independent** and are processed **in parallel**
So: Benchmark execution of **1 container, extrapolate** to demand

$$T_{s1} = T_{start-kubectl} + T_{translate} + Latency + T_{write}$$

$$T_{total} = T_{s1} \times \lceil jobs/CPUs \rceil$$

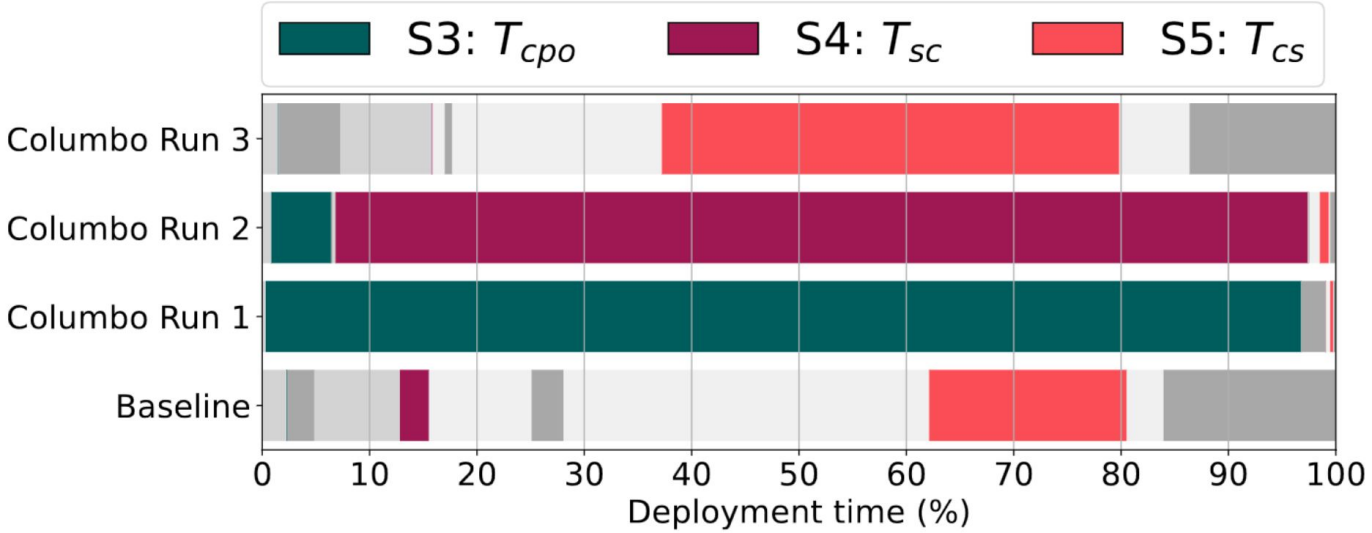


Columbo Day-to-day Operation

VU VRIJE UNIVERSITEIT AMSTERDAM

Use case: 1600 containers over 16 worker, 1 control plane node
Bottleneck detection:

- Benchmark finds **97% execution time in phase 3**, step 9
- Analytical baseline predicts **<1% of execution time in this step**



Resolve Bottleneck with Parameter Rules

VU VRIJE UNIVERSITEIT AMSTERDAM

1. Map configuration parameter to pipeline stages / steps
   a. Automatic analysis of documentation and source code
   b. Filter for performance-sensitive parameters
2. Define parameters' current value and matching object
   a. Extract from documentation / source code and pipeline

One-time effort → Expert can augment this automated approach

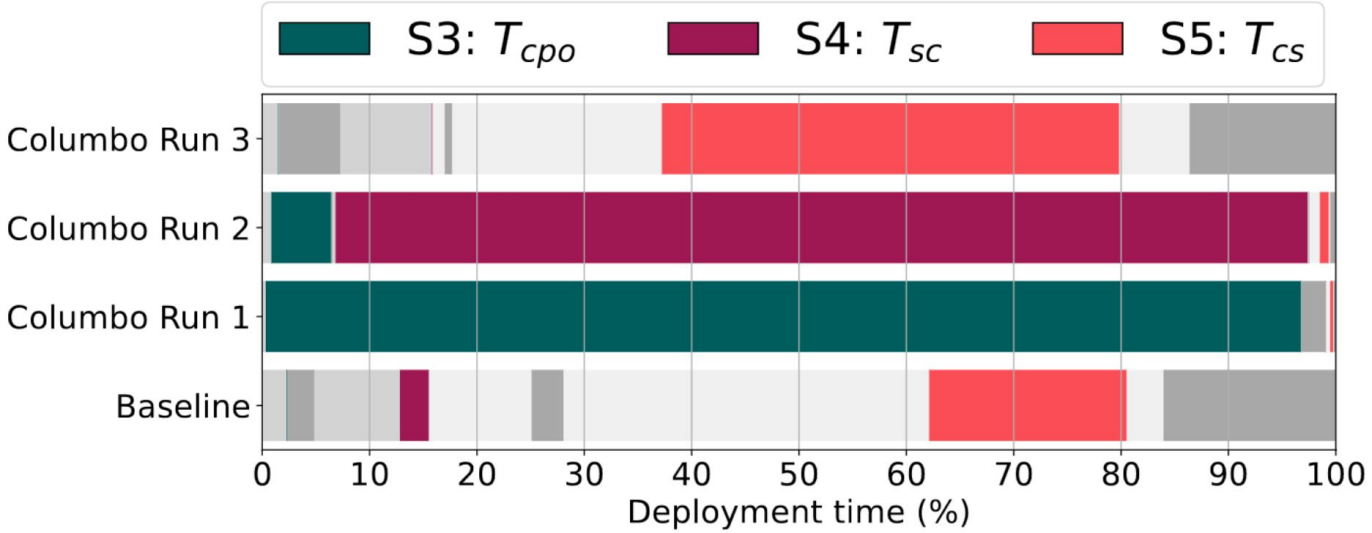
Stage	Step	Parameter	Current Value	Scales with
3	8	*InitialBatchSize*	1	#pods
4	14	*KubeApiQPS*	50	#pods

We have 1600 pods

Columbo Day-to-day Operation

Update *InitialBatchSize* to 1600 to resolve bottleneck
Columbo Run 2: 80.9 seconds to 31.1 seconds

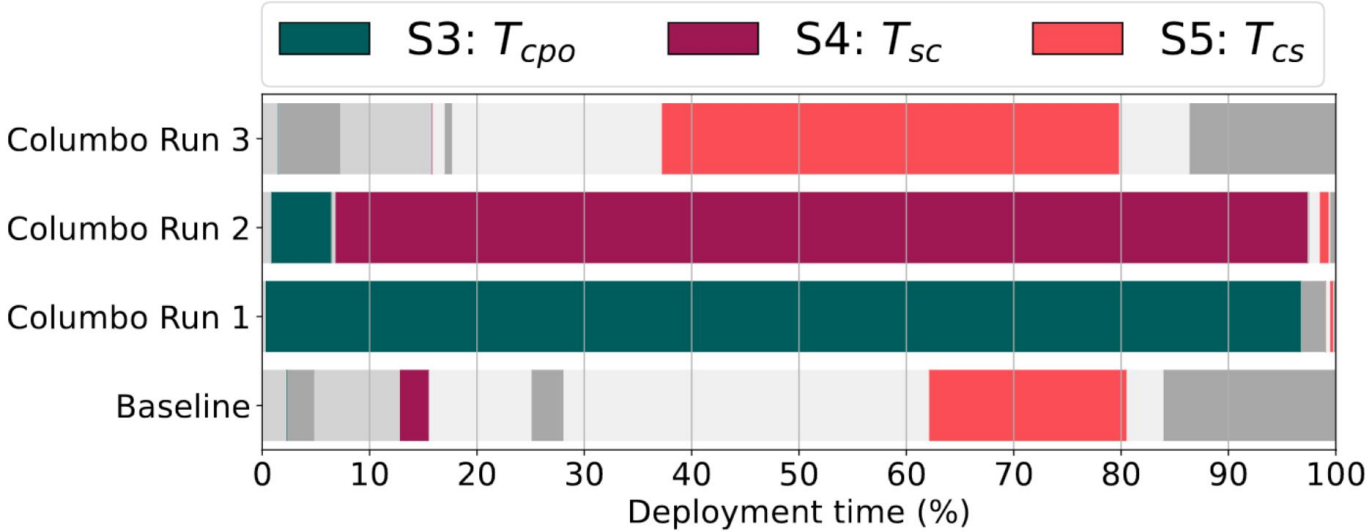
- Benchmark finds 97% execution time in phase 4, step 14
- Analytical baseline predicts ~6% of execution time in this step



Columbo Day-to-day Operation

Update *KubeApiQPS* to 1600 to resolve bottleneck
Columbo Run 3: 80.9 seconds to 16.9 seconds → 79.1% reduction

- CPU utilization at 100%
- No further optimization possible unless resources are added

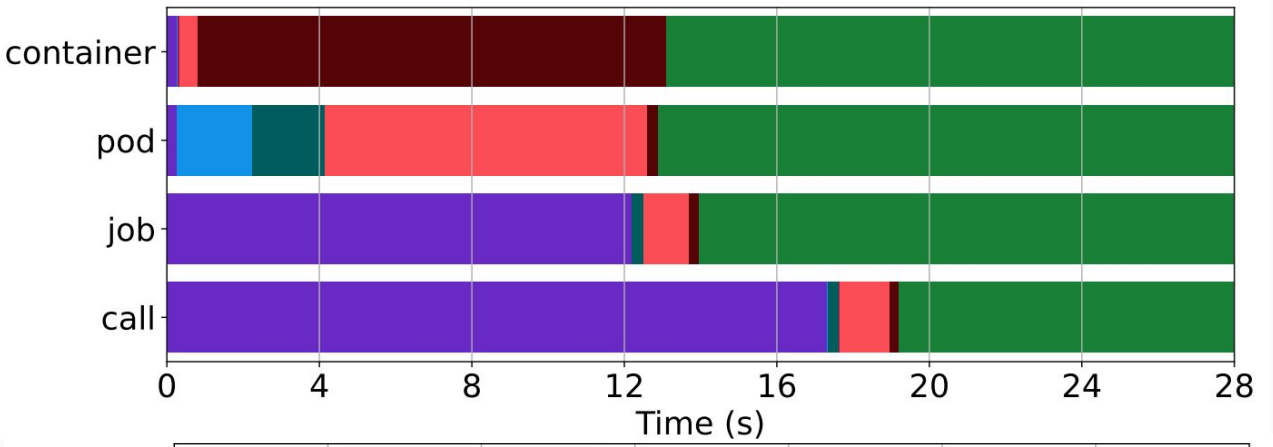


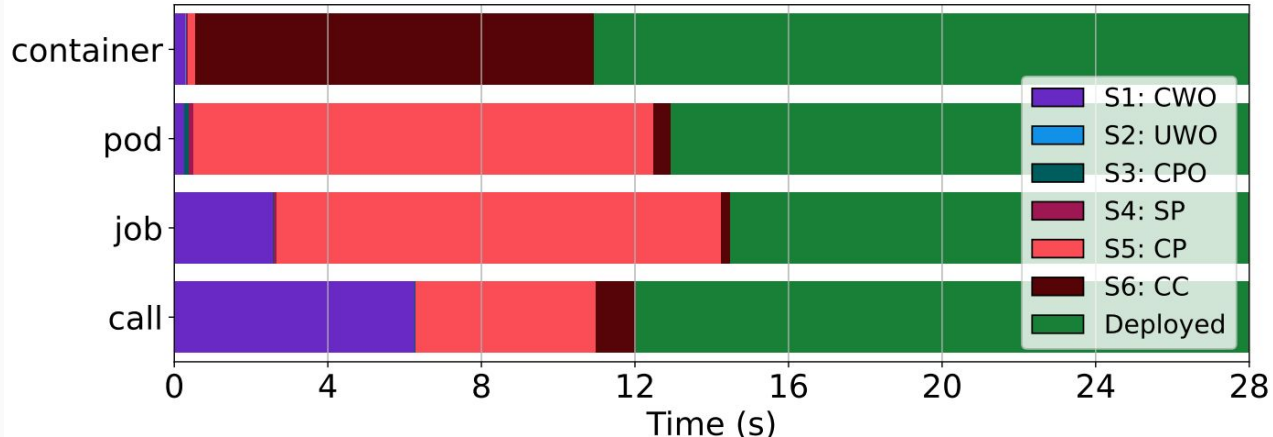
Optimize Deployment Method

VU VRIJE UNIVERSITEIT AMSTERDAM

100 containers over:
- 100 kubectl calls
- 1 call, 100 jobs
- 1 job, 100 pods
- 1 pod, 100 cont.

Results:
- Cont.: **-16%**
- Pod: +1%
- Job: +5%
- Call: **-37%**

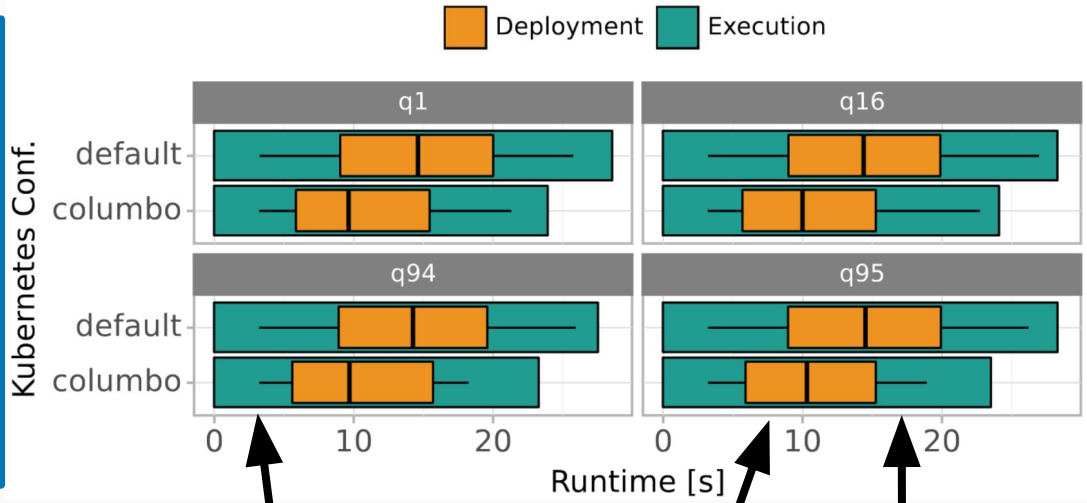




End-to-end Impact of Columbo

VU VRIJE UNIVERSITEIT AMSTERDAM

Setup:
- 100GB TPC-DS on Spark
- Spark control plane and workers on Kubernetes
- 400 Spark workers run 4 queries with cold starts



Spark startup

Processing

K8s deploys workers

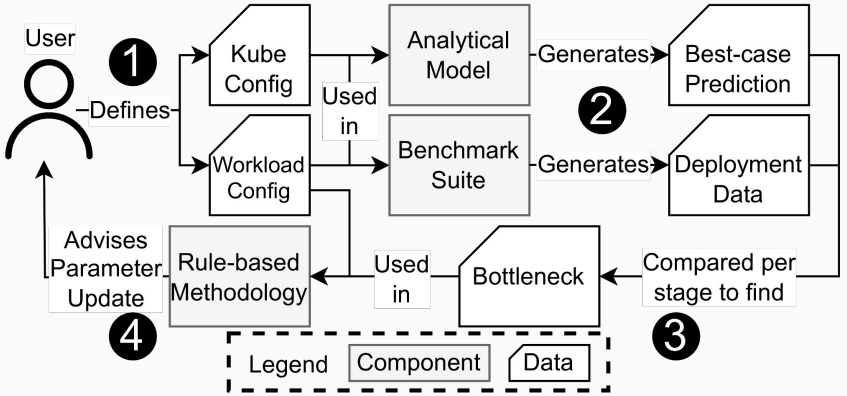
Findings:
- **Avg. deployment time: -29%**
  - Similar to microbench.
- **Avg. end-to-end time: -17%**

Conclusion

We present Columbo:

- Automatically detect and resolve bottlenecks for Kubernetes configurations
- Columbo reduces deployment time (avg. 28%) and total execution time (17% for Spark)

Open-source and free-to-use



m.s.jansen@vu.nl

atlarge.science/mjansen

Columbo
Open-source Code