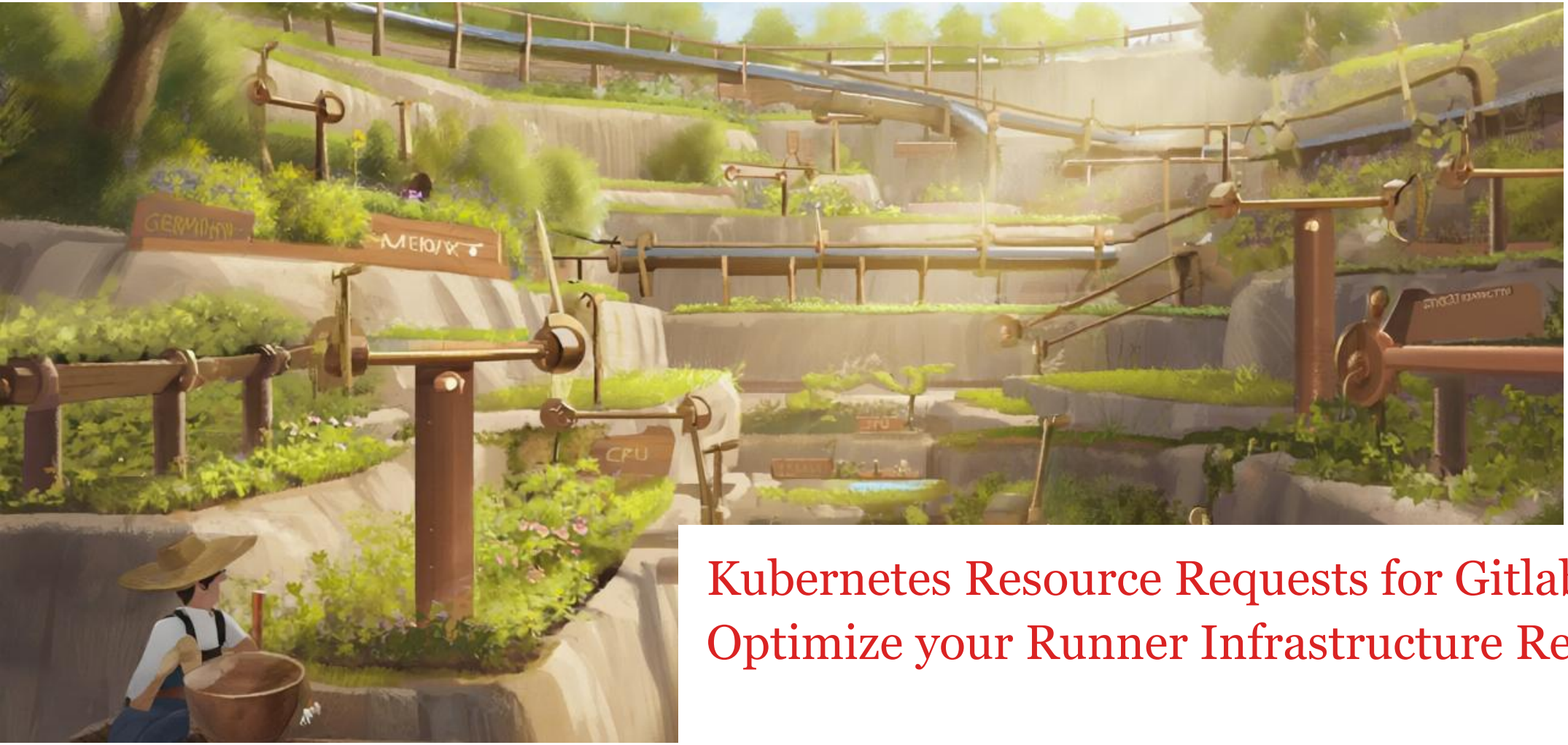
die Mobiliar



Kubernetes Resource Requests for Gitlab Jobs - Optimize your Runner Infrastructure Resources

die Mobiliar

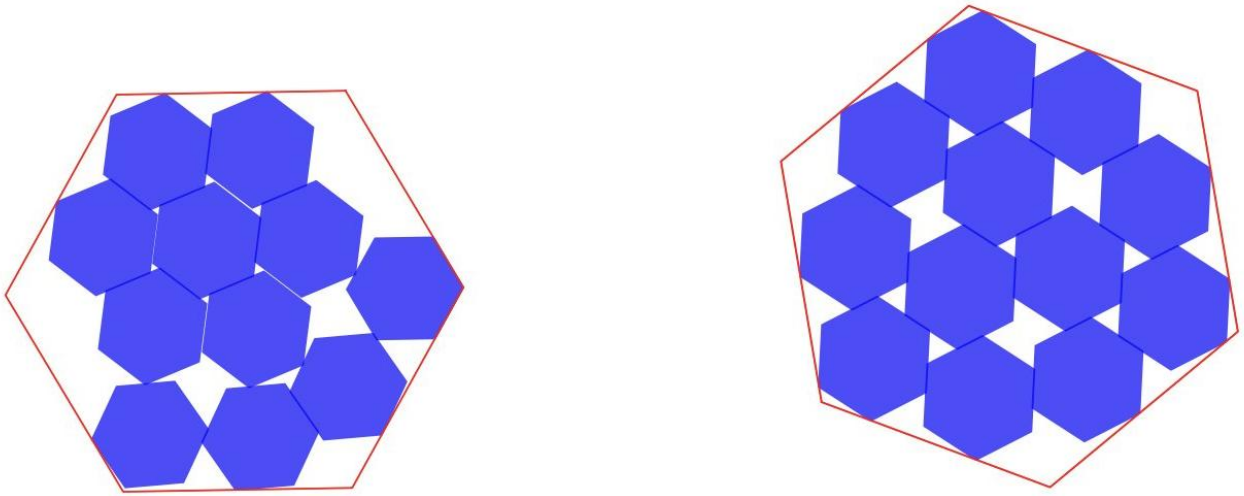


Figure 11 | Constructions of the packing problems found by *AlphaEvolve*. Left: Packing 11 unit hexagons into a regular hexagon of side length 3.931. Right: Packing 12 unit hexagons into a regular hexagon of side length 3.942.

… be like AlphaEvolve – solve the Packing Problem (more efficiently)

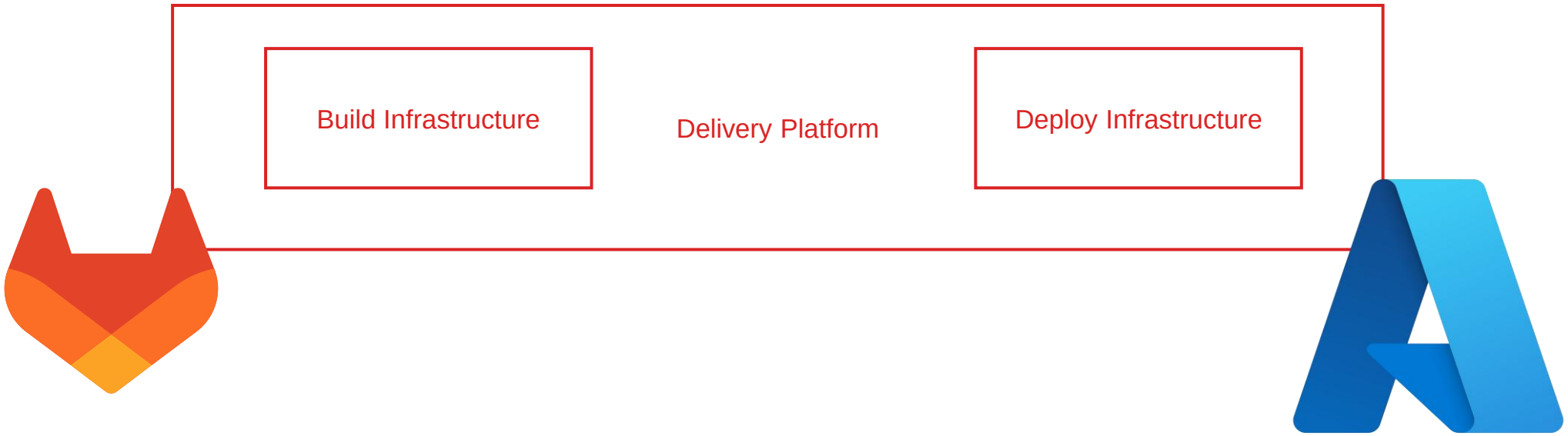
About us

Jan Wolfensberger
Software Engineer @ DieMobiliar

Sebastian Plattner
Cloud Architect @ Puzzle ITC AG

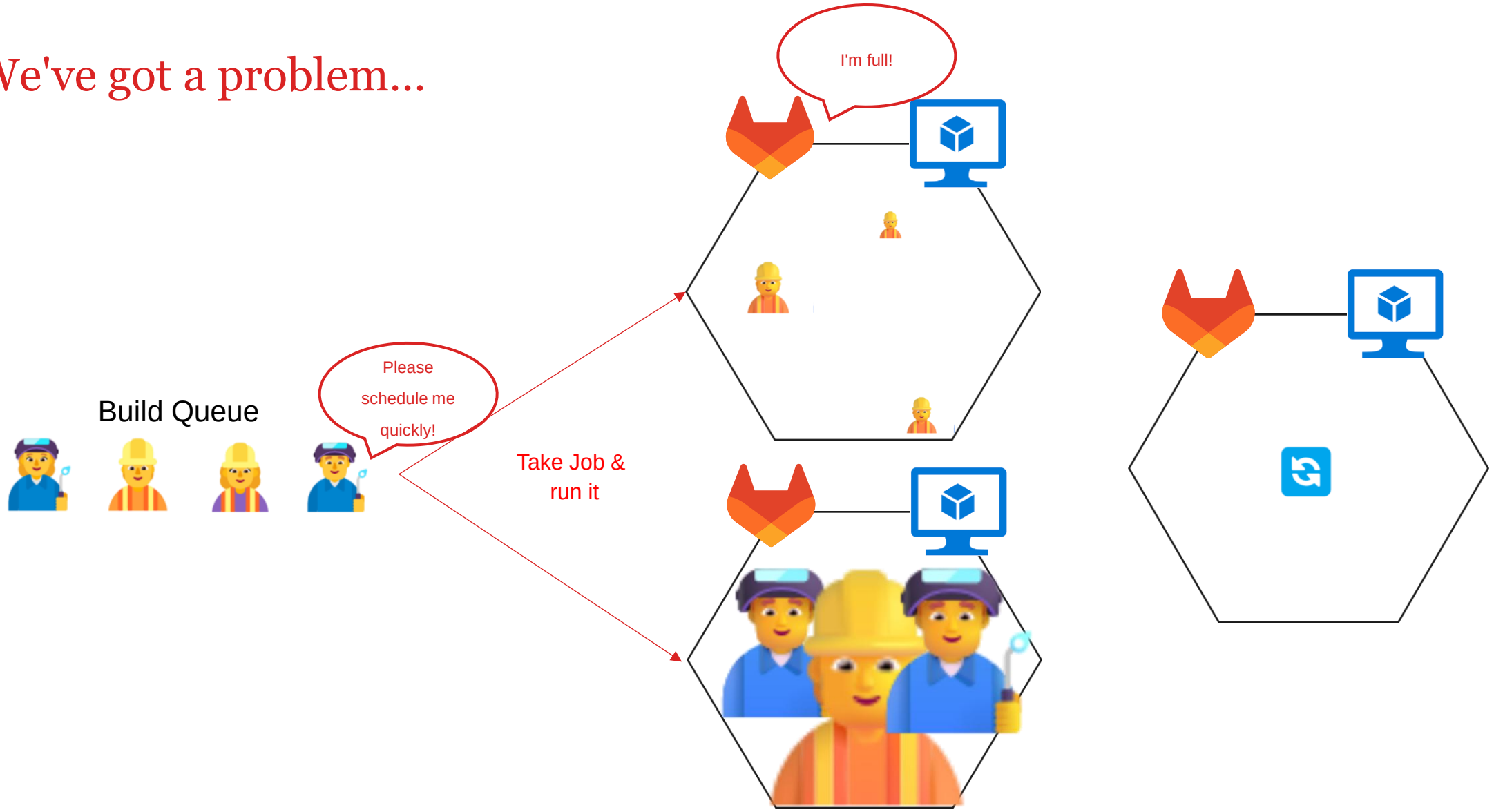
What we do

Code → Push → Build → Test → Publish → Deploy → Release

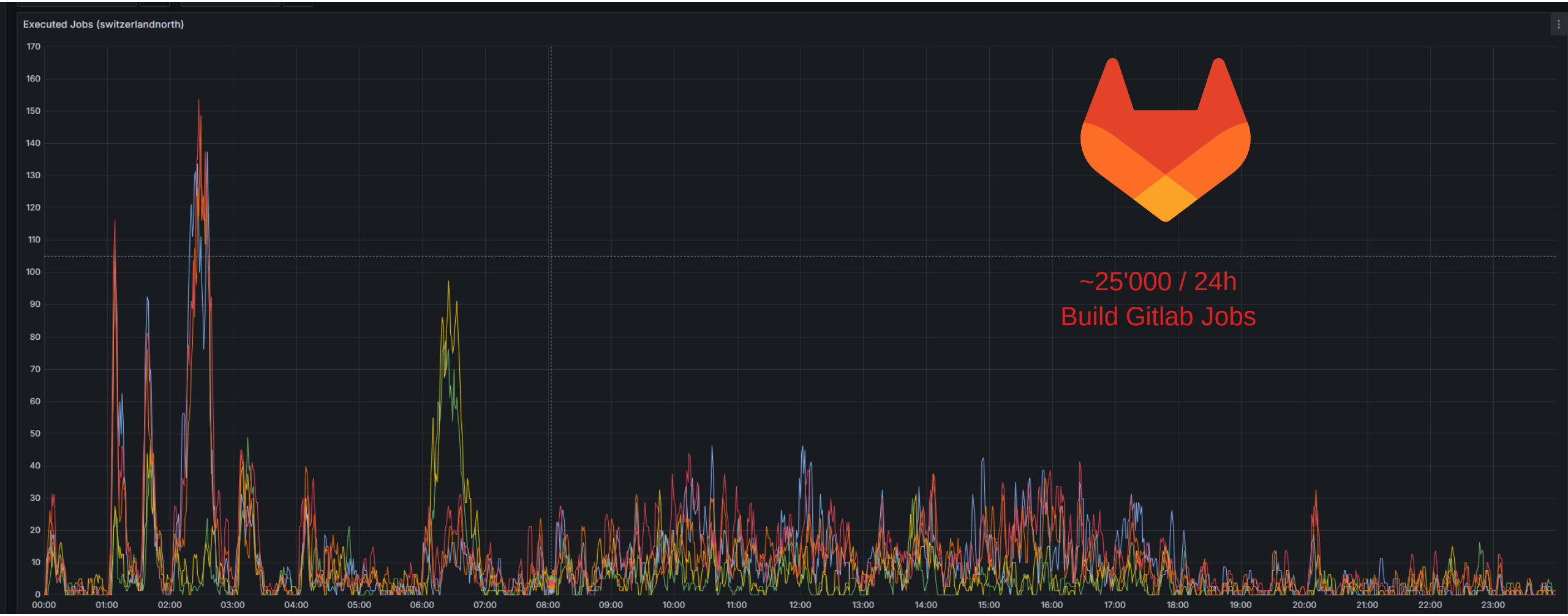


We've got a problem…

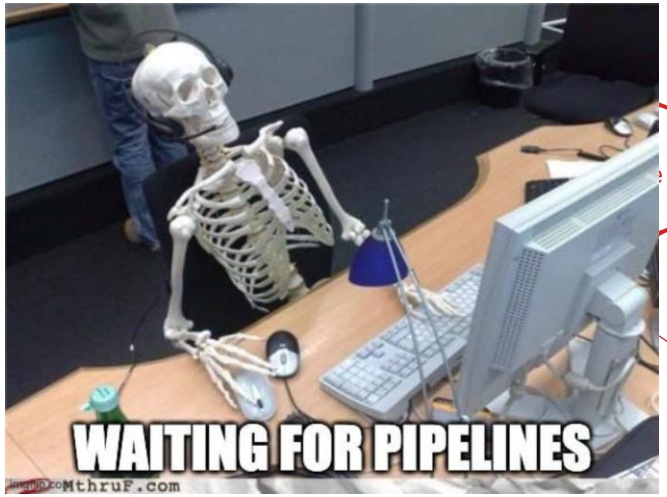
We've got a problem...



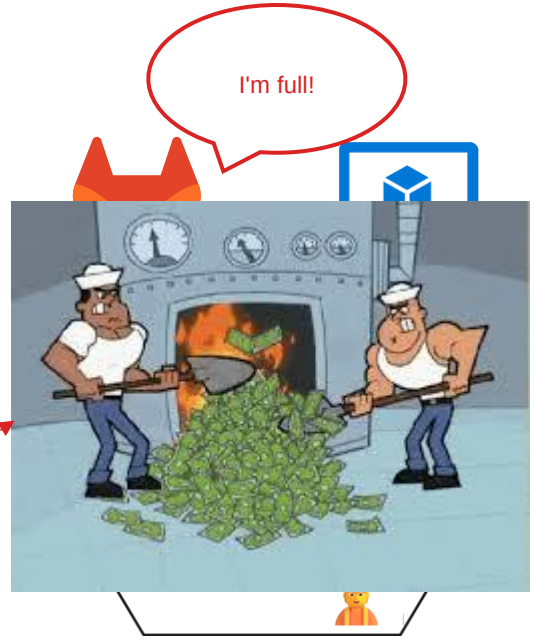
Facts and Figures

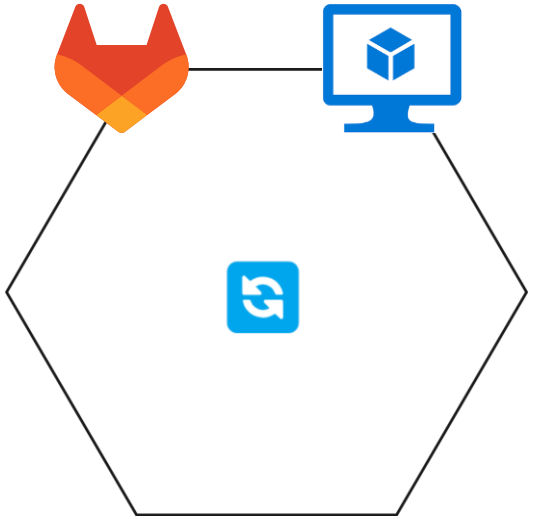


We've got a problem...



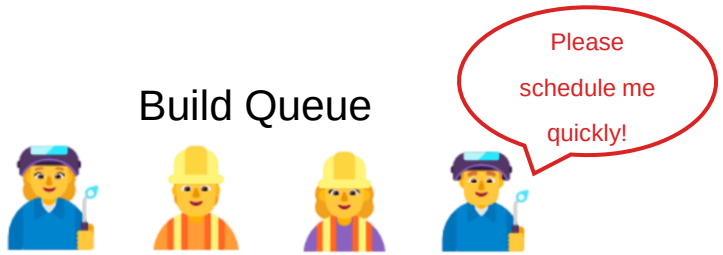
Take Job & run it



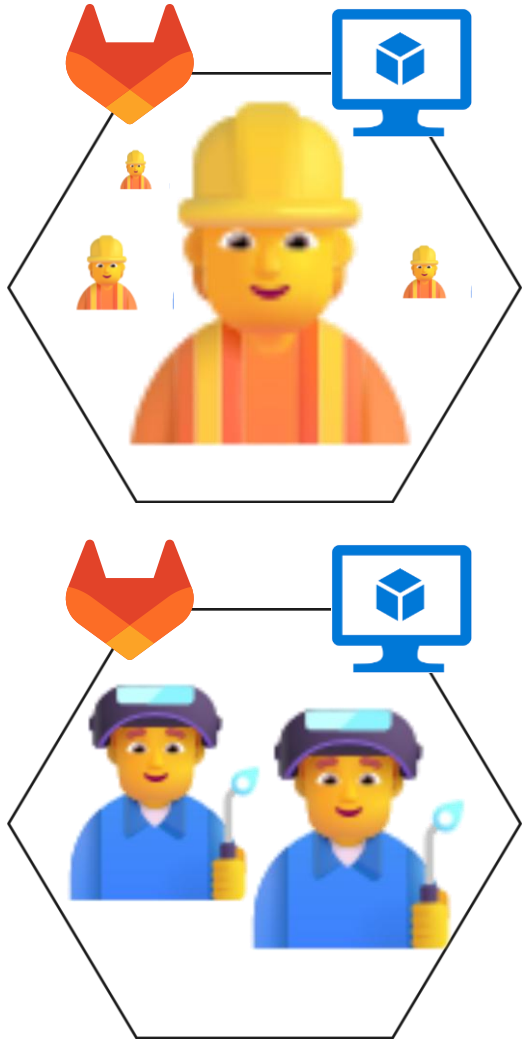


This would be better!

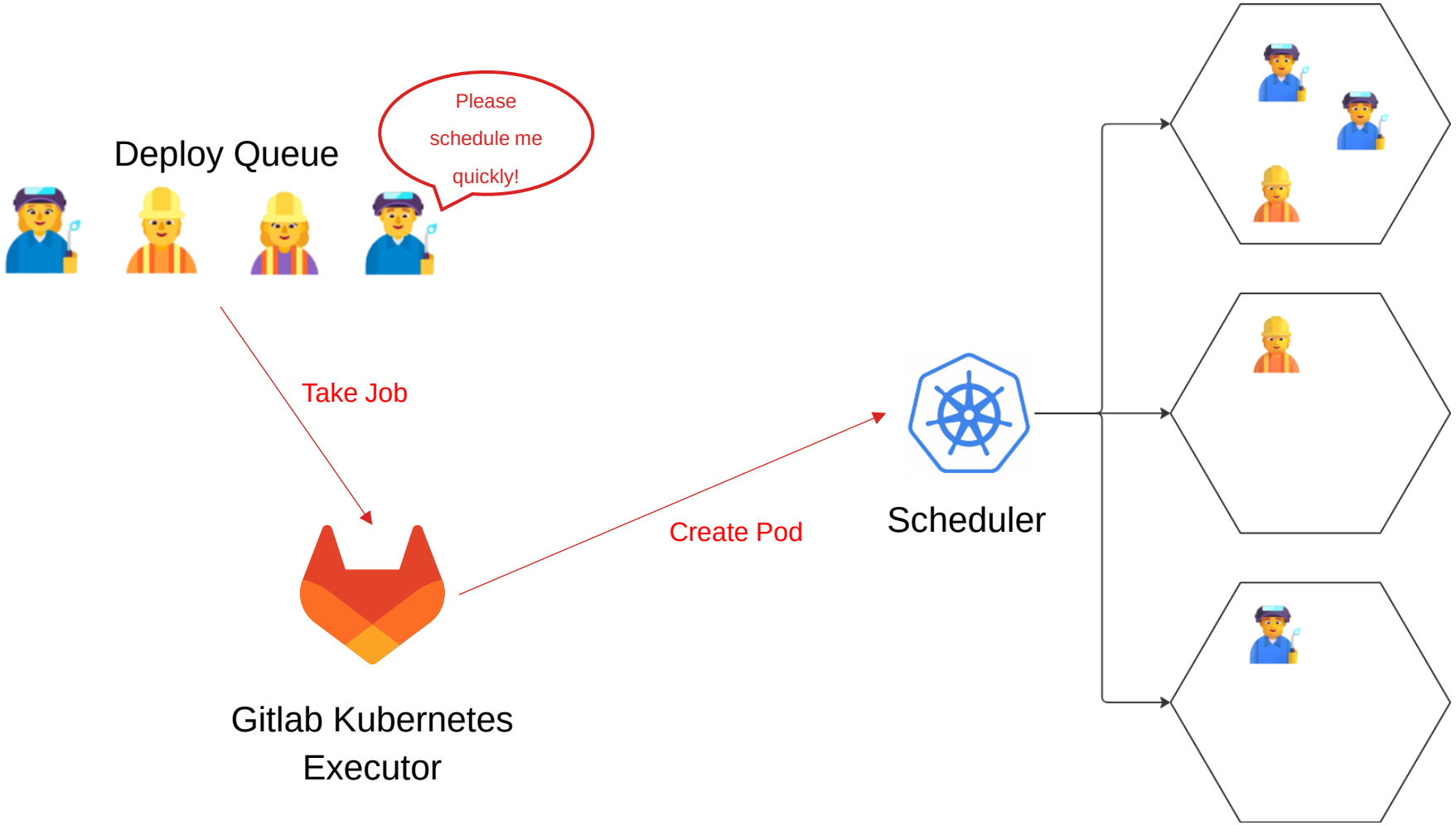
Build Queue



We need a way to properly schedule Jobs!



Kubernetes?!


Please
schedule me
quickly!
Deploy Queue
Take Job
Create Pod
Scheduler
Gitlab Kubernetes
Executor

Why using the Kubernetes Executor

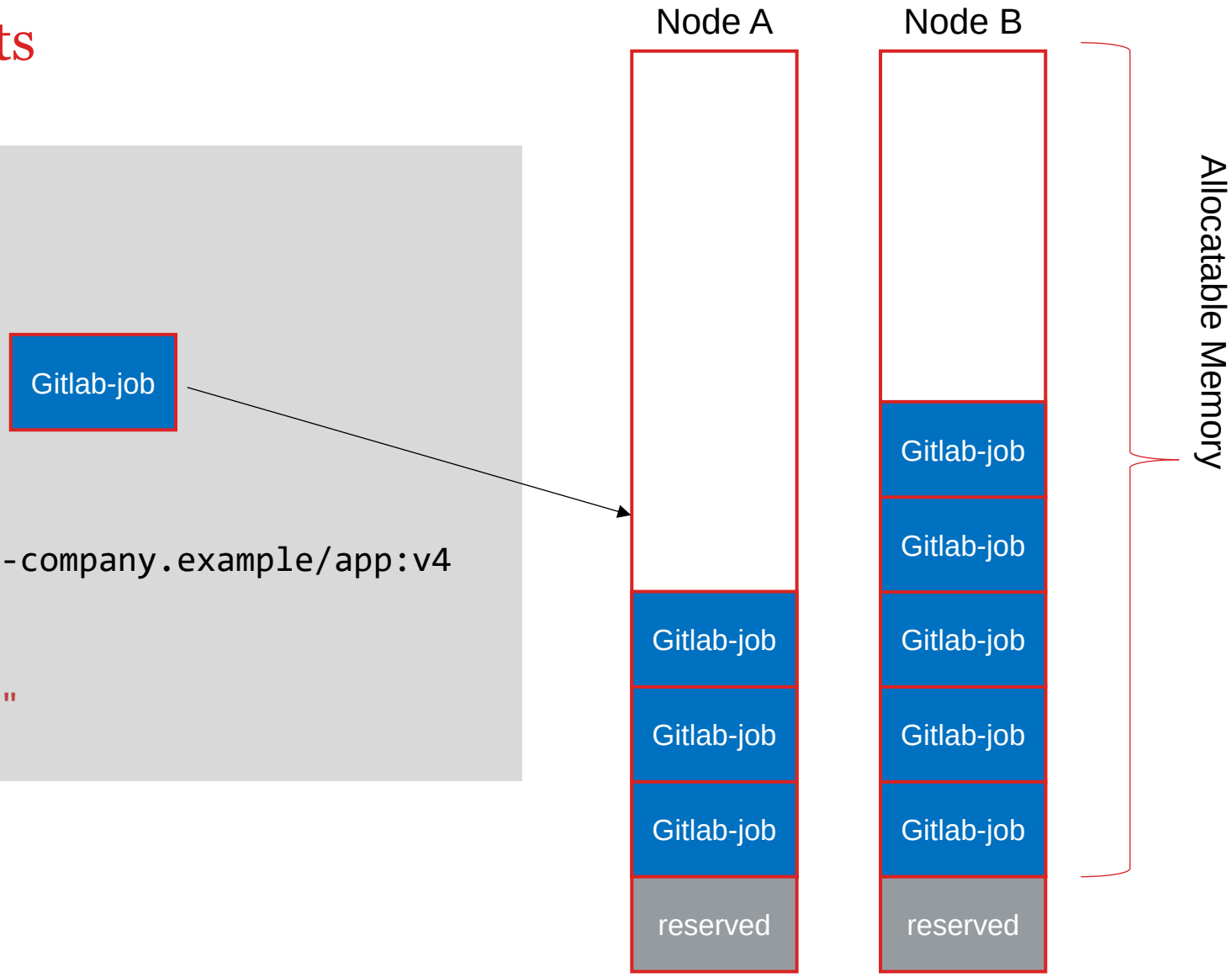
- Scalability and Resource Management
  The Kubernetes executor **automatically scales** runners based **on job demand**, spinning up pods as needed and terminating them when jobs complete.
- Cost Efficiency
  You **only consume resources when actually needed** as we are also using the Kubernetes Cluster Autoscaler
- Advanced Scheduling
  Kubernetes provides **sophisticated scheduling capabilities** including node affinity, resource requests/limits
- All in all, this helps us:
  – To Give our Developer the Resources they need to run their build jobs fast and stable
  – While optimize our Infrastructure usage (less overprovisioning)

Kubernetes Scheduling & Resource Requests

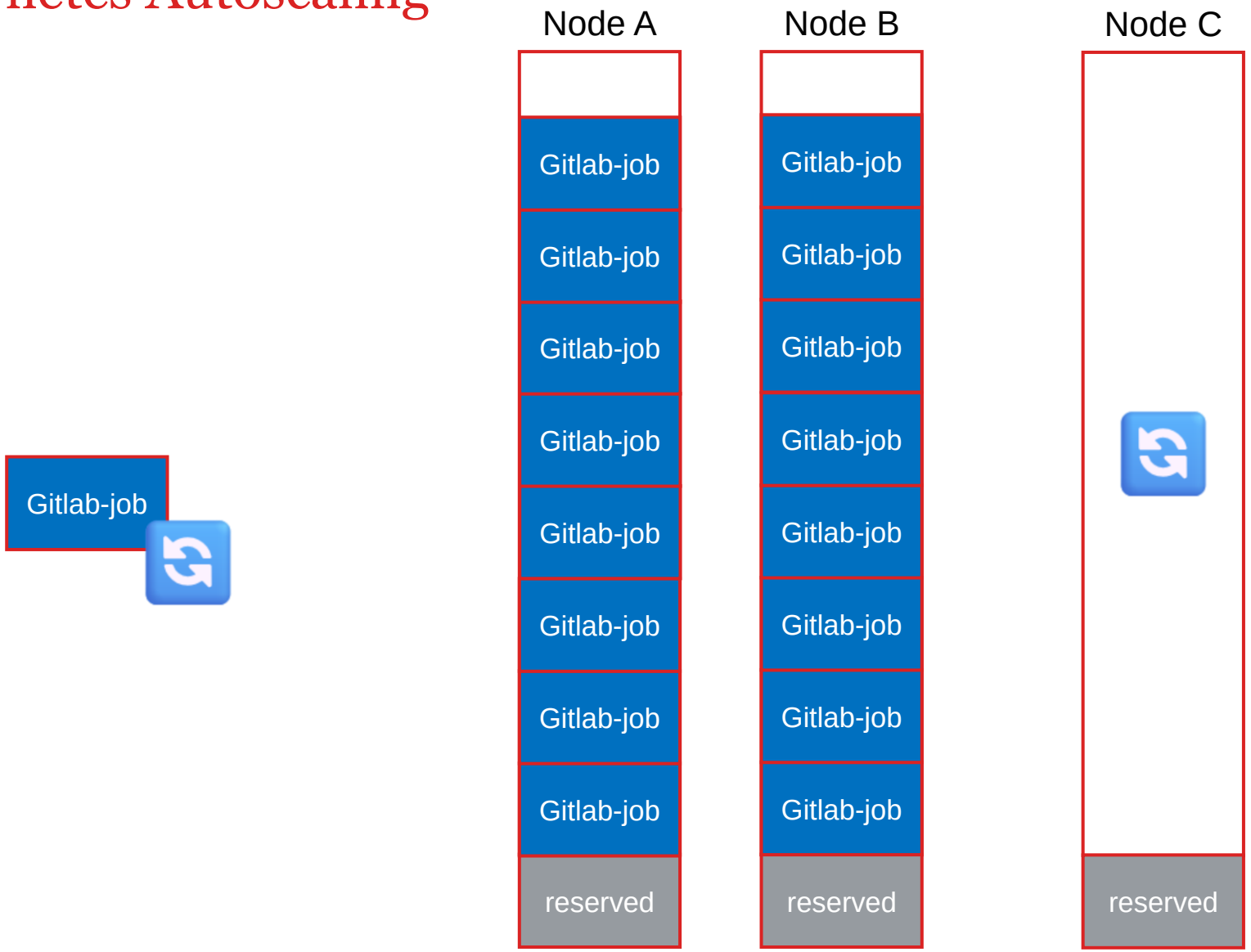
«Only place me on a node that can guarantee at least this much CPU and Memory»

Kubernetes Scheduling Resource Requests

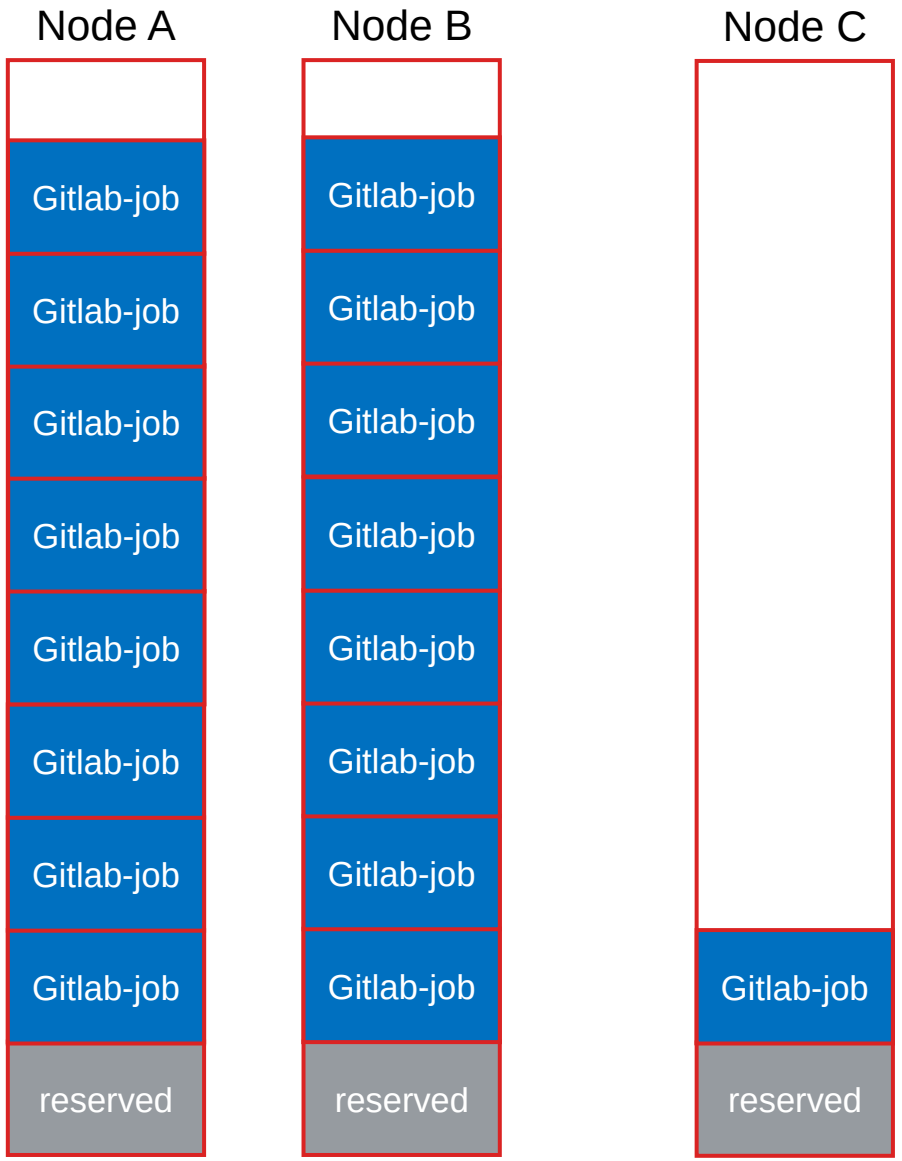
```yaml
---
apiVersion: v1
kind: Pod
metadata:
  name: gitlab-job
spec:
  containers:
    - name: build
      image: images.my-company.example/app:v4
      resources:
        requests:
          memory: "1Gi"
          cpu: "500m"
```



Kubernetes Autoscaling



Kubernetes Autoscaling



Solution:
Move everything to Kubernetes!

Simply configure Requests for heavy Candidates

Example .gitlab-ci.yml

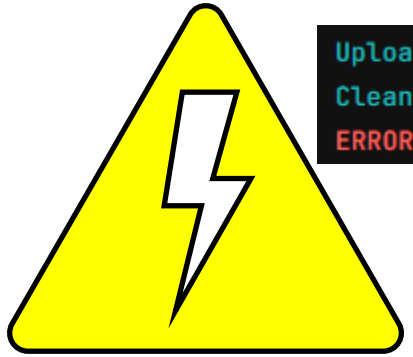
```
my-job:
 stage: test
 image: node:14
script:
 - run-some-test.sh
variables:
  KUBERNETES_CPU_REQUEST: 4
  KUBERNETES_MEMORY_REQUEST: 12Gi
```

Defaults for all Jobs

```
[runners.kubernetes]
  memory_request = "256Mi"
  cpu_request = "300m"
```

Problem solved!

Not really...

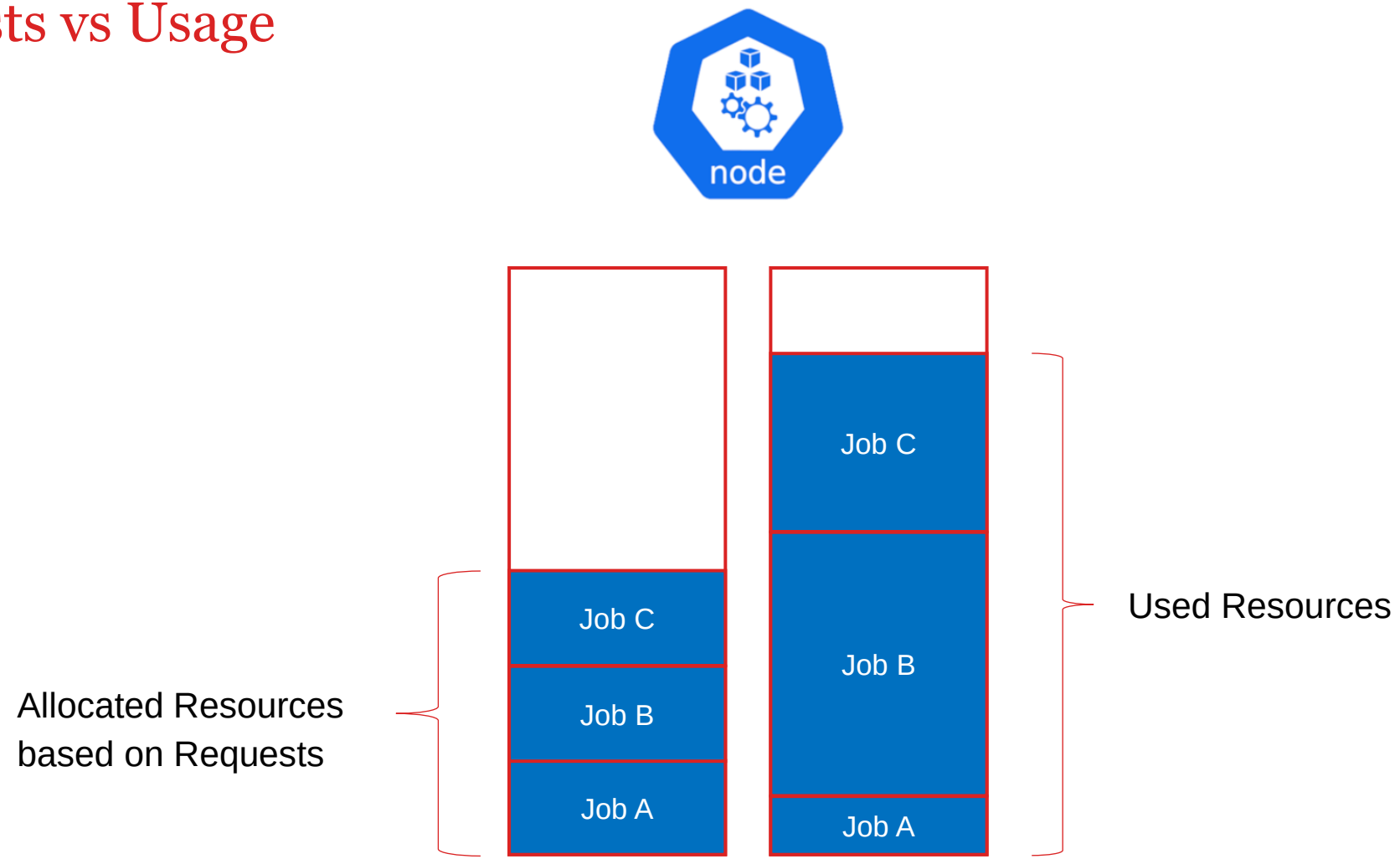


```
Uploading artifacts for failed job
Cleaning up project directory and file based variables
ERROR: Job failed (system failure): Error in container helper: exit code: 137, reason: 'OOMKilled'
```

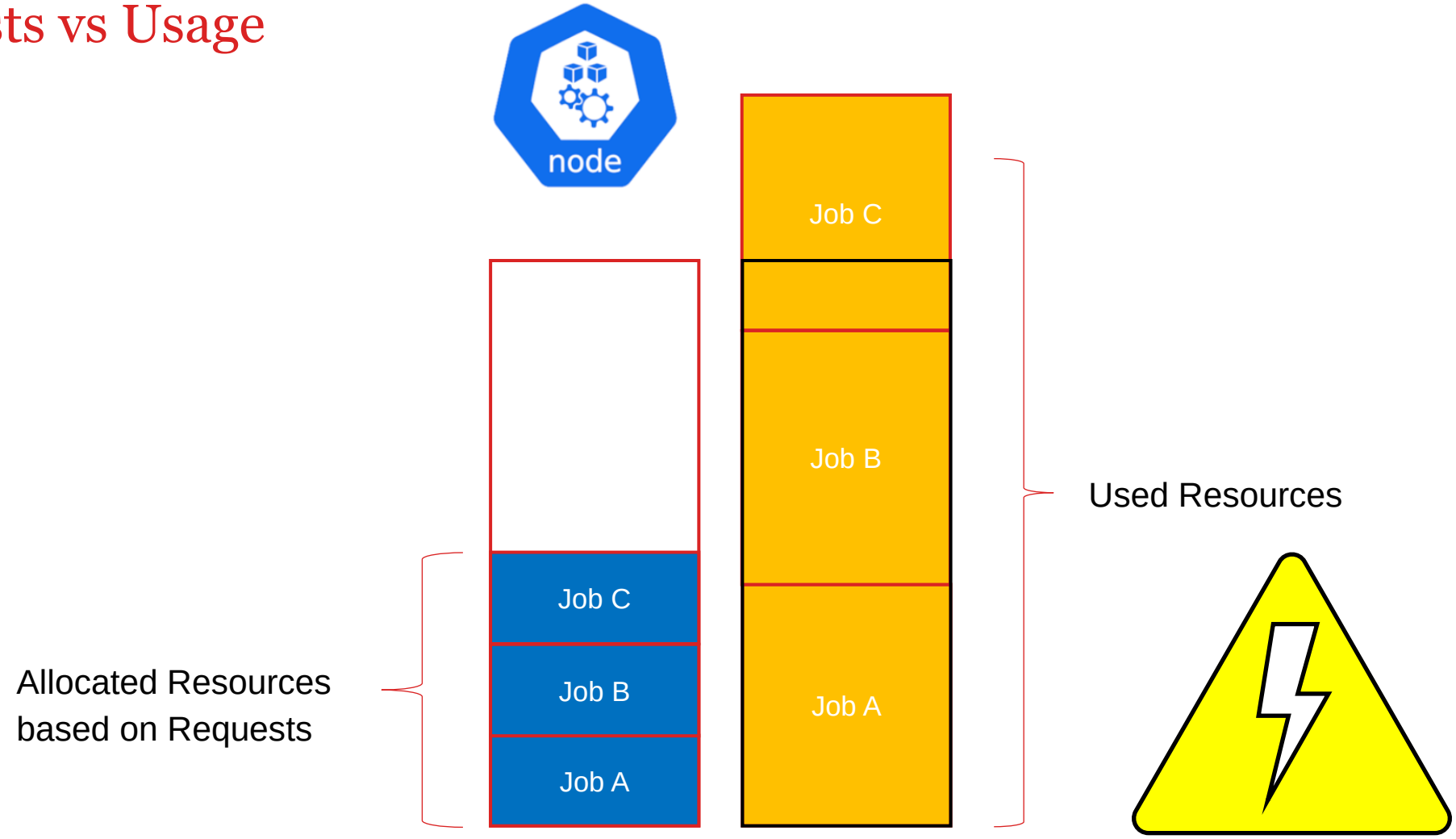
```
WARNING: Event retrieved from the cluster: 0/11 nodes are available: 1 node(s) had untolerated taint {ToBeDeletedByCl
usterAutoscaler: 1749513767}, 1 node(s) had untolerated taint {node.cilium.io/agent-not-ready: }, 3 Insufficient cpu,
6 node(s) didn't match Pod's node affinity/selector. preemption: 0/11 nodes are available: 3 No preemption victims fo
und for incoming pod, 8 Preemption is not helpful for scheduling.
```

```
FATAL  [16:20:54.924859477] Failed to publish image                    error="Failed to check publish gate: failed
to download CDXVulnsFiltered attestations: error fetching attached attestation: fetching payload: read tcp 10.2.100.1
33:34938->20.209.231.1:443: read: connection reset by peer"
Running after_script                                                                                          00:00
Cleaning up project directory and file based variables                                                        00:01
ERROR: Job failed: command terminated with exit code 1
```

Kubernetes Scheduling
Requests vs Usage



Kubernetes Scheduling
Requests vs Usage



… and it does not scale!

Nice Idea

- Let the User configure Resource Requests
- Set reasonable defaults

Problems

- Decentralized
- Does not scale
- Dev's don't care 💸 💸

Fix:
The Rise of Centralization

Get into the Pod Lifecycle

```
my-job:
 variables:
  KUBERNETES_CPU_REQUEST: 4
  KUBERNETES_MEMORY_REQUEST: 12Gi
```

Get Job

Gitlab Kubernetes Executor

Create Pod

```
---
kind: Pod
metadata:
  name: gitlab-job
spec:
  containers:
    - resources:
        requests:
          memory: "12Gi"
          cpu: "4"
```

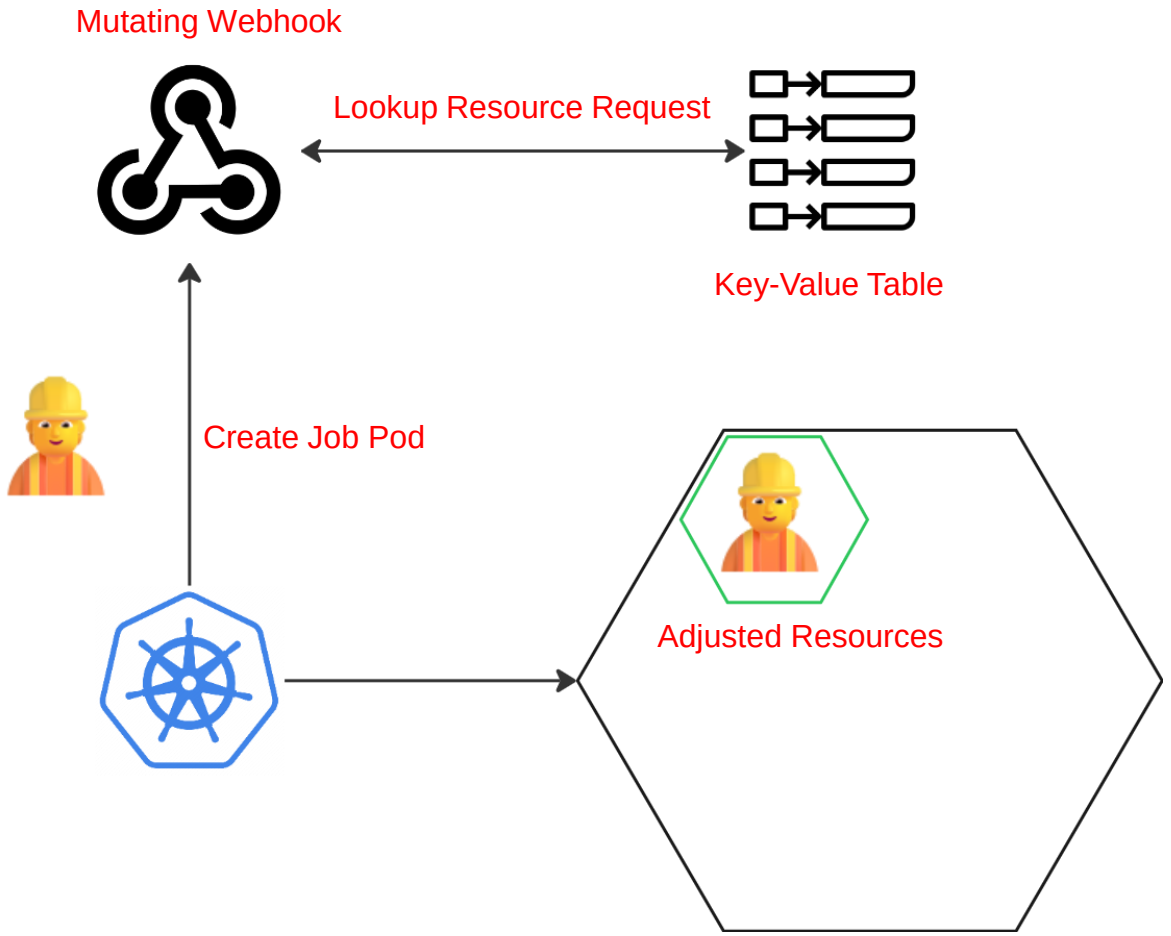
Kubernetes Mutating Webook

- Admission Controller that intercepts API Requests to the Kubernetes API Server
- Can modify (mutate) the resource object (Gitlab Job Pod) before they are persisted

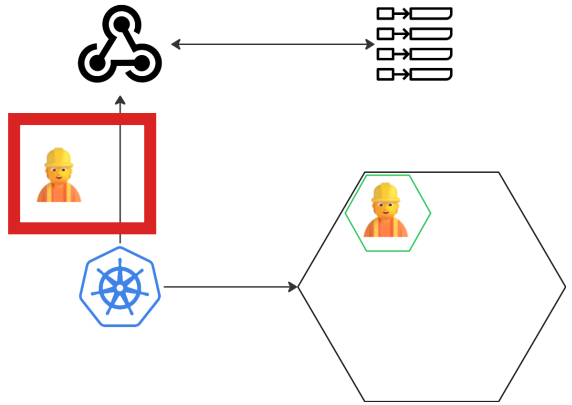
```
apiVersion: admissionregistration.k8s.io/v1
kind: MutatingWebhookConfiguration
metadata:
  name: resourceallocator
webhooks:
  - clientConfig:
      service:
        name: admission-server
        namespace: resourceallocator
        path: /mutate
       port: 443
    failurePolicy: Ignore
    timeoutSeconds: 30
```

```
rules:
- apiGroups:
  - ""
  apiVersions:
  - v1
  operations:
  - CREATE
  resources:
  - pods
  scope: "*"
objectSelector:
  matchExpressions:
  - key: runnertype
    operator: In
    values:
    - build-worker
```

The fixed Design



Passing along information

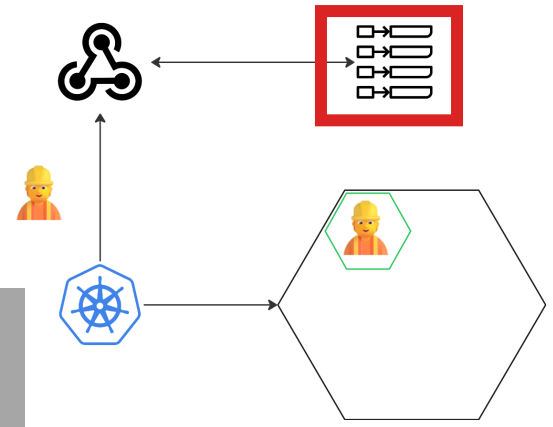


```
[runners.kubernetes.pod_labels]
"azure.workload.identity/use" = "true"
"runnertype" = "build-worker"
"runneractive" = "${runner_active}"
"capability" = "${capability}"
"gitlab-project-path" = "$CI_PROJECT_PATH"
"gitlab-project-name" = "$CI_PROJECT_NAME"
"gitlab-job-name" = "$CI_JOB_NAME"
"stackid" = "$STACK_ID"
```

Centralized Configuration

- YAML based
- Configuration per
  - Stack or
  - Project
- Memory and CPU Request per Container in Runner Pod (build, docker, helper) for each Gitlab Job

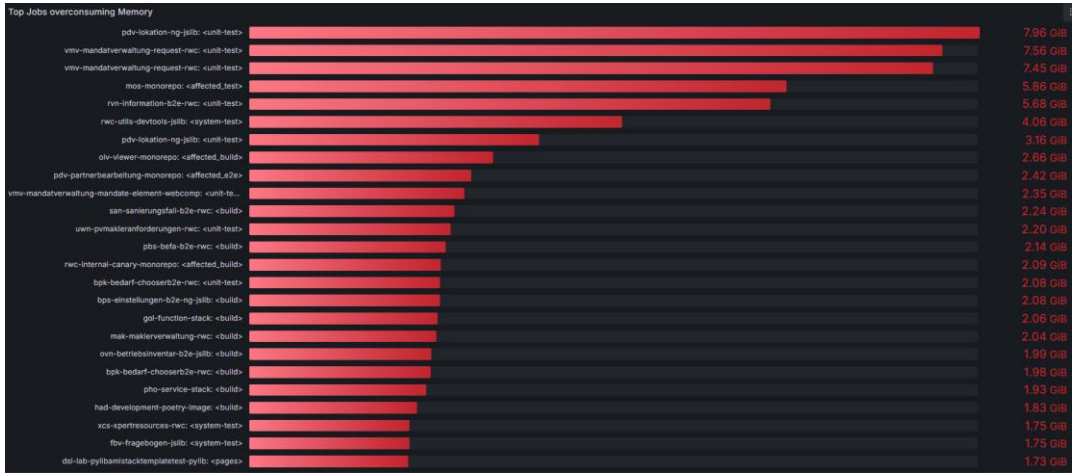
```
# stack gol-fn
go-function: # stack id
  sast_go:
    build-memory: "1Gi"
    build-cpu: "0.6"

# projects gol-fn
dlp-gitlabgov-permission-fn: # project name
  lint_go:
    build-memory: "4Gi"
    build-cpu: "1.5"
  sast_go: # job name
    build-memory: "4Gi"
    build-cpu: "1.5"
```



Problem solved!

Not really…

- Our Configuration YAML File:
  - **666 Lines of YAML** and rapidly growing
  - Manual lookup (by us) in Grafana Dashboard
  - Constantly updating for new projects or changed requests



Go and adjust the values – every day…

Solution
Why not use the metrics we already have?

Prometheus for the win

```
sum by (pod)
(
  container_memory_working_set_bytes{namespace="build-runner",
container="build"
  * on(pod)
  group_left(label_gitlab_job_name, label_gitlab_project_name)
  kube_pod_labels{
    namespace="build-runner",
    label_gitlab_project_name= "$CI_PROJECT_NAME"
    label_gitlab_job_name ="$CI_JOB_NAME"
  }
)
```

More Details

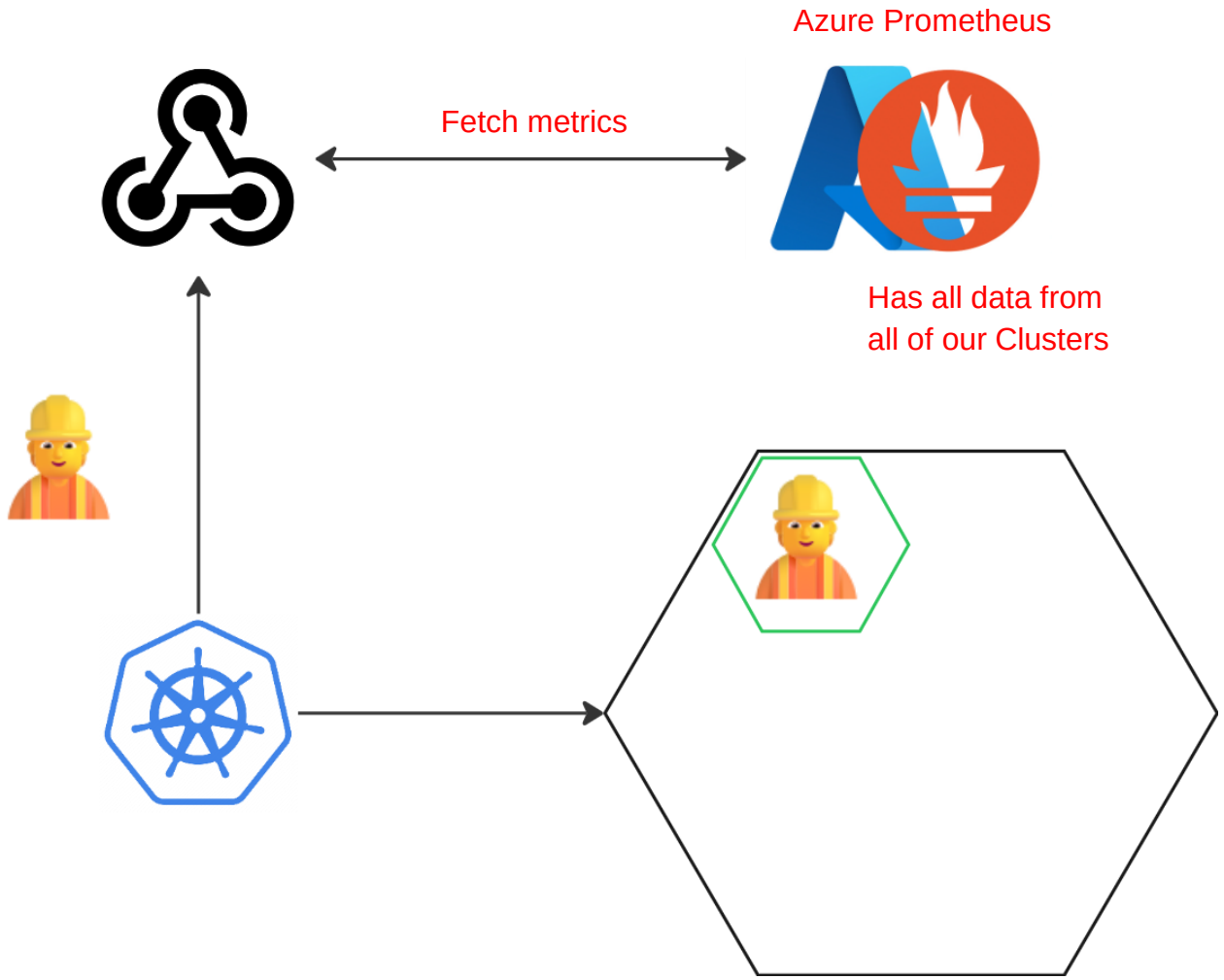
Last 20 datapoints

0.2, 3, 4.8, 5, 6.5, …

80th percentile

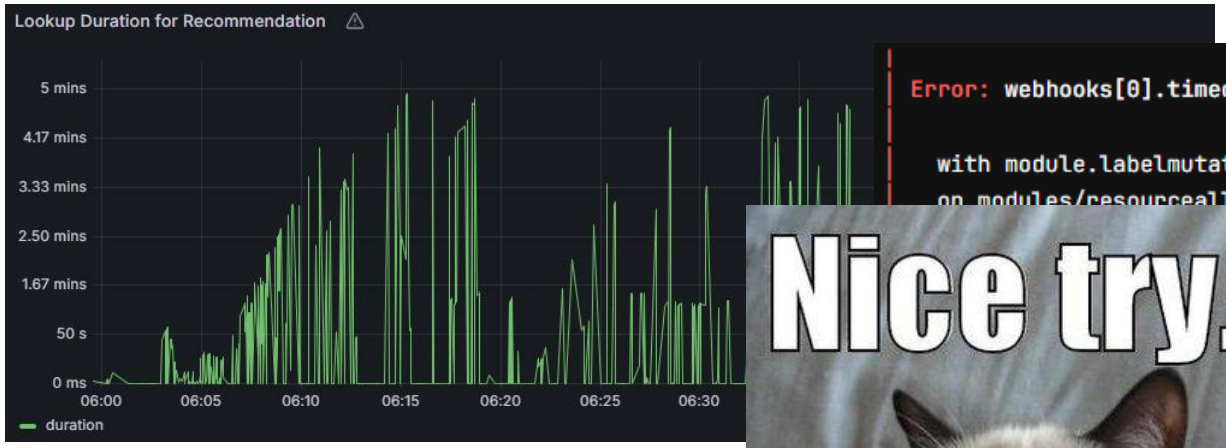
5

Get those Metrics directly



Problem solved!

Not really...




```
Error: webhooks[0].timeoutSeconds

  with module.labelmutator_cluster_a_untrusted[0].kubernetes_manifest.manifests["webhooks.yaml"],
  on modules/resourceallocator/main.tf line 5, in resource "kubernetes_manifest" "manifests":
                                                 est" "manifests" {

                                                 lue must be between 1 and 30 seconds
```

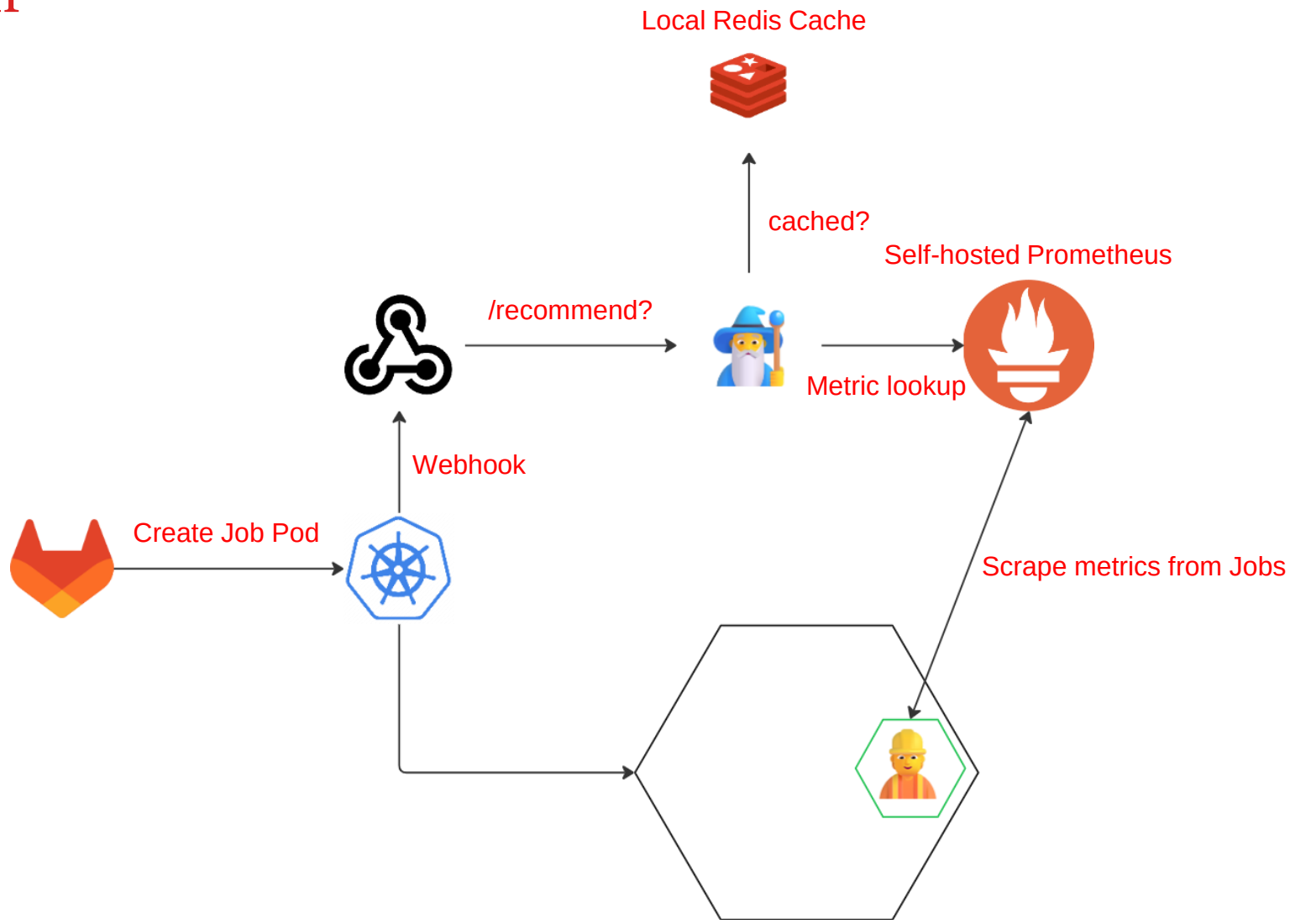
A
Status: 429. Message: Too Many Requests
(Trace ID: 6f939f55c3214de0b8cf21addba
0594b20b096bbcf

A
Status: 429. Message: Too Many Requests
(Trace ID: 6f939f55c3214de0b8cf21addba
0594b20b096bbcf

B
Status: 429. Message: Too Many Requests: Account has exceeded the query data limit.. RequestId: b0a6b42b39a1fa5e9d1a24f59f6c1cde
(Trace ID: 6f939f55c3214de0b8cf21addba4502a)

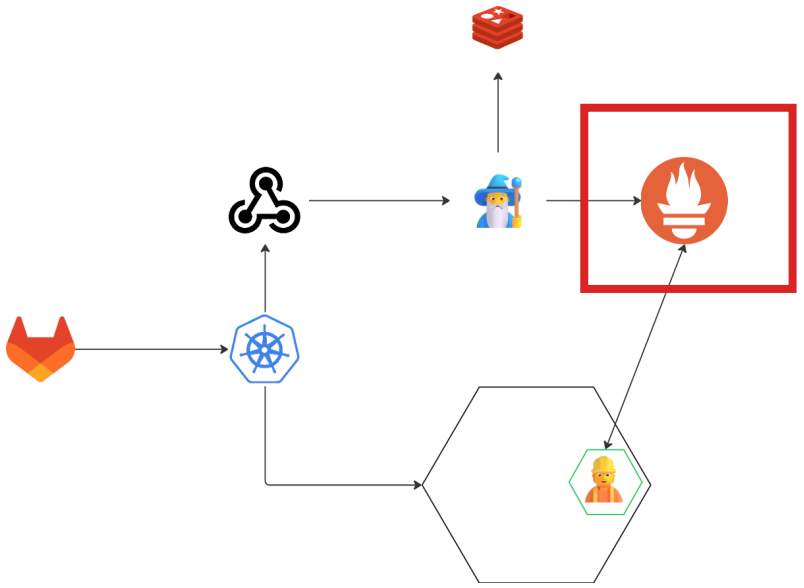
Solution
The final one

Do it yourself



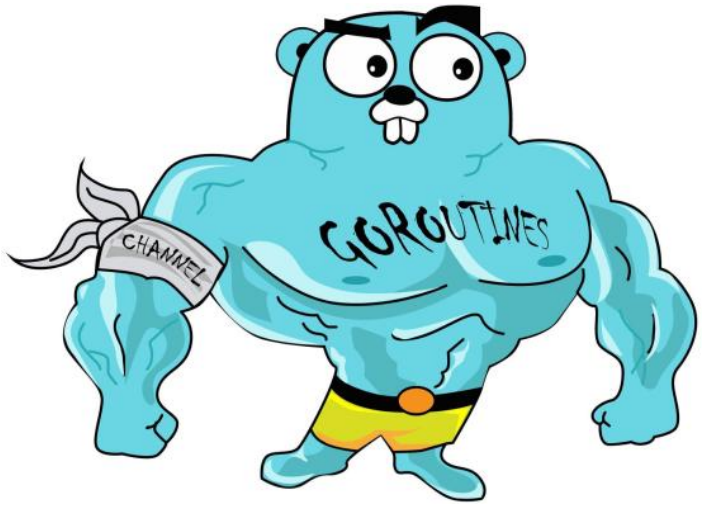
Do it yourself - Prometheus

container_cpu_usage_seconds_total{**cpu**="total", **instance**="aks-buildrunner-10570765-vmss0002vj", **job**="kubernetes-nodes-cadvisor", **namespace**="build-runner", **pod**="runner-t2f5hlya-project-53800315-concurrent-0-ehrfjkx7"}	0.018717

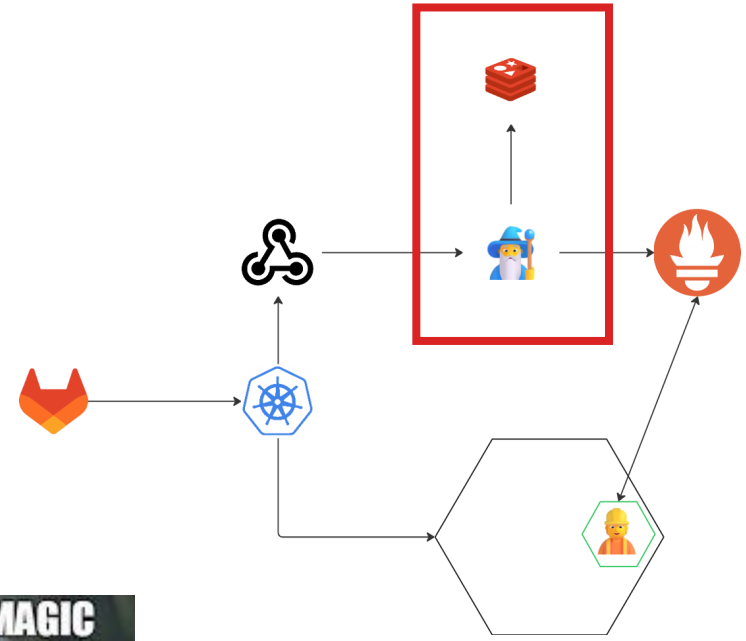


kube_pod_labels{**instance**="10.0.40.129:8080", **job**="kubernetes-service-endpoints", **label_capability**="dind", **label_gitlab_job_name**="sca_licenses_v2", **label_gitlab_project_name**="dlp-information-status-canary", **label_gitlab_project_path**="diemobiliar_it_dlp_information_dlp-information-status-canary", **label_stackid**="Docker-Base-Image", **namespace**="build-runner", **pod**="runner-t2bgjyk5-project-61930578-concurrent-0-fshz0edq", **service**="ama-metrics-ksm"}	1

Do it yourself – The Magic

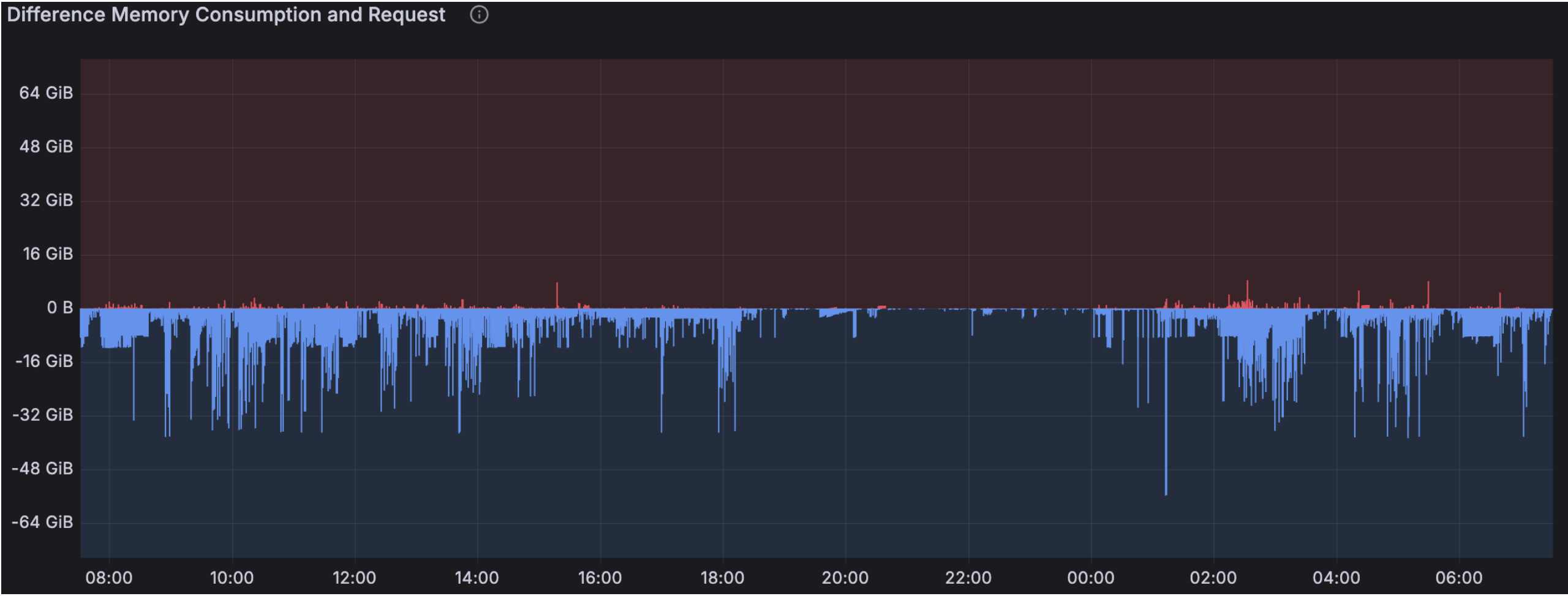


- Cache fetched data for faster access
- Control concurrency
- Refetch data periodically

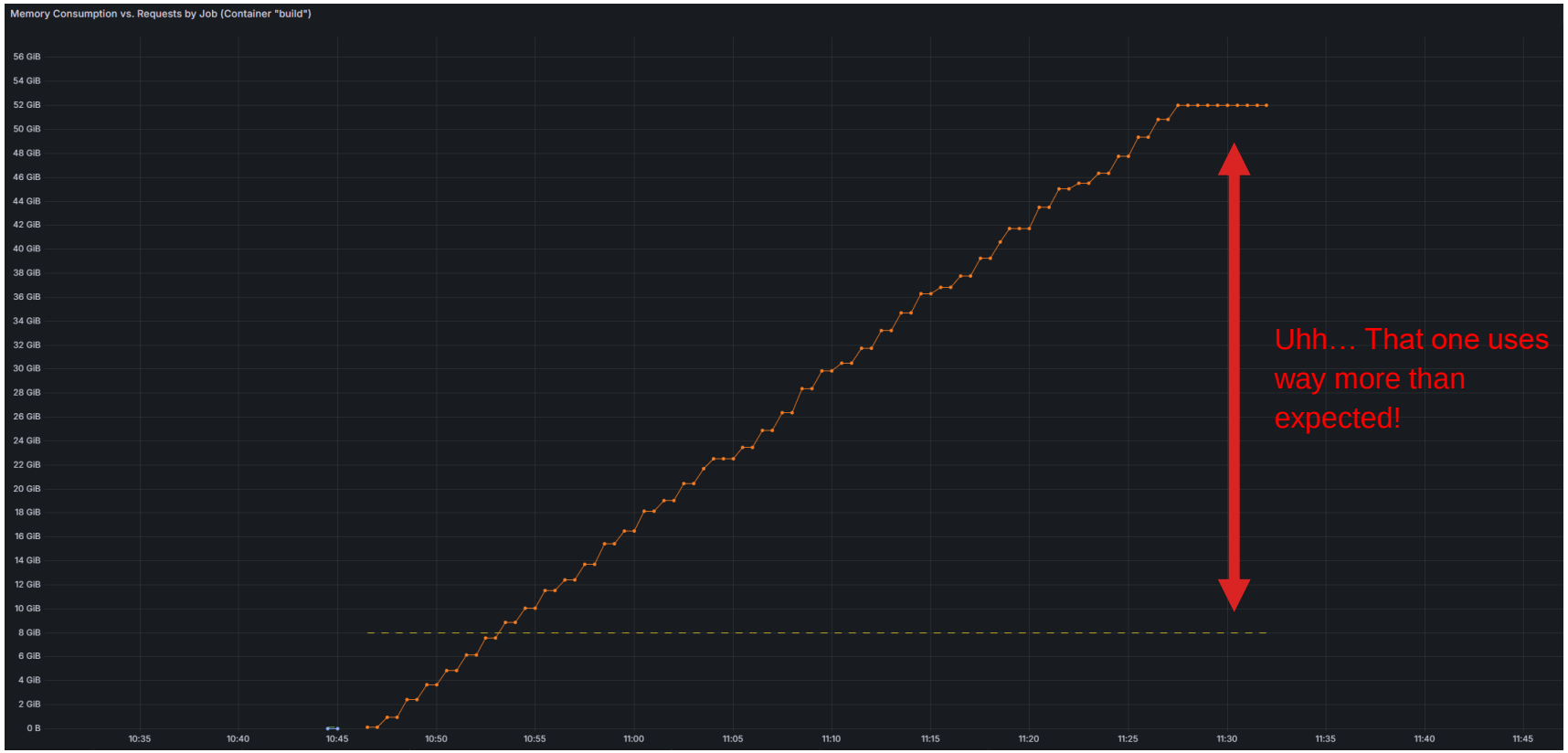


Final Solution?

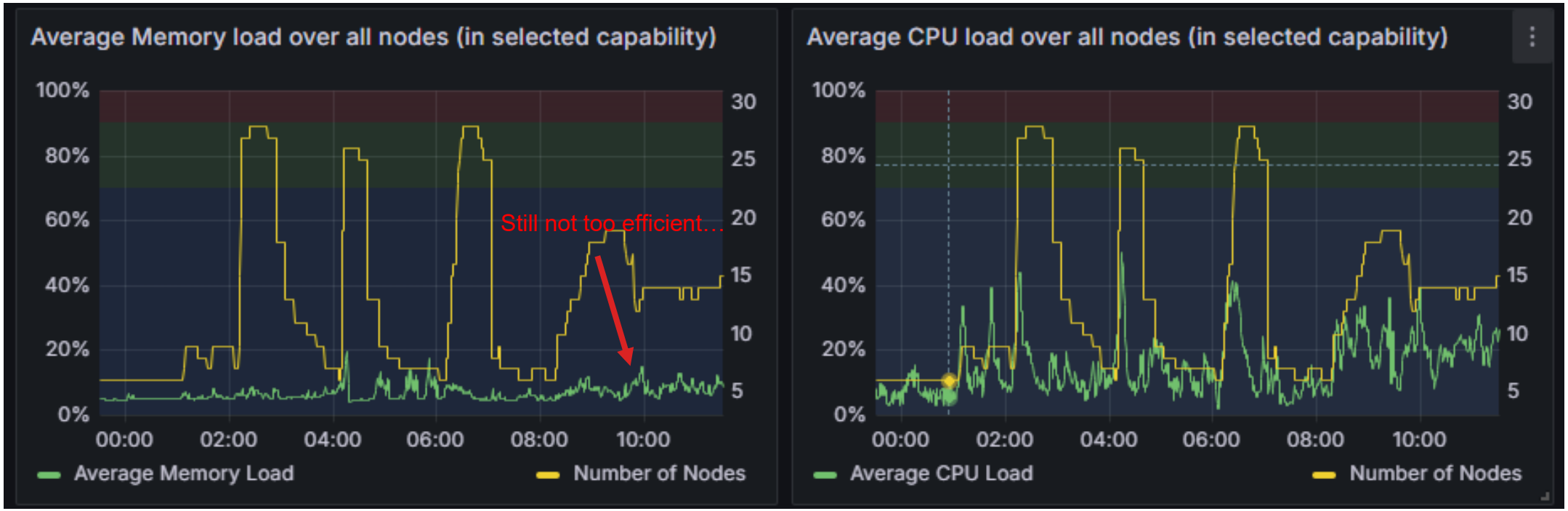
It kinda works…



Still got some challenges…



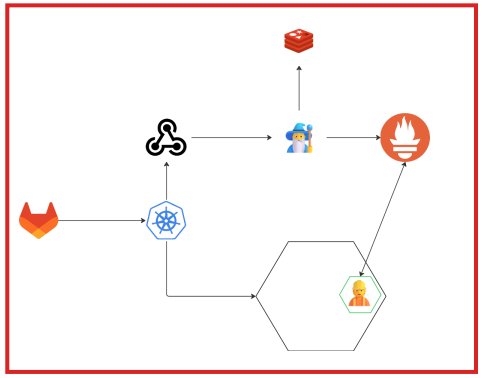
Still got some challenges…



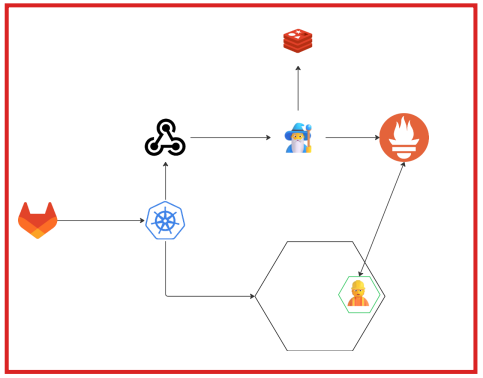
Still got some challenges…

A / B Deployment (Testing)

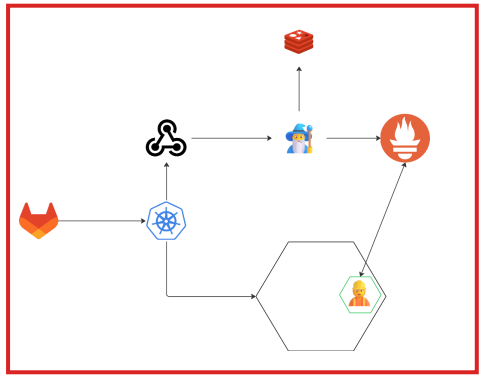
High Availability
(Multi Region)



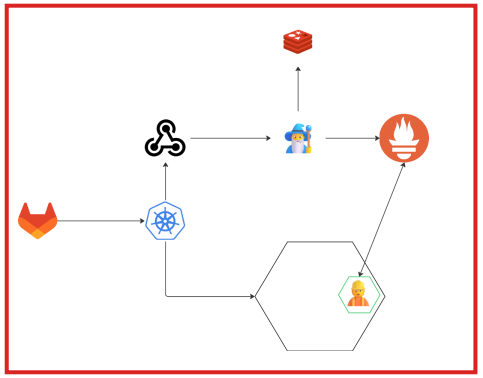
Cluster A – Region 1



Cluster B – Region 1



Cluster A – Region 2

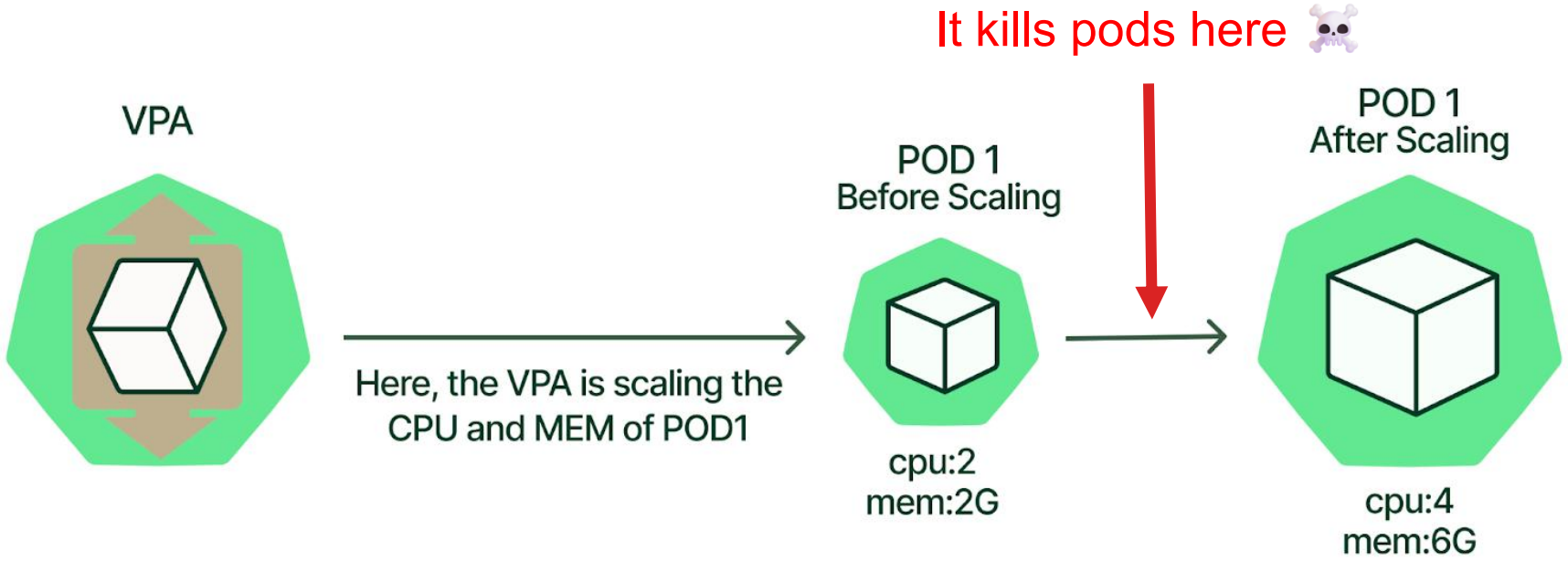


Cluster B – Region 2

- Data is not shared…
- One job only runs on one cluster…

What to do if a job has no data?

BTW why not using Kubernetes VPA?



Thank you for packing with us!

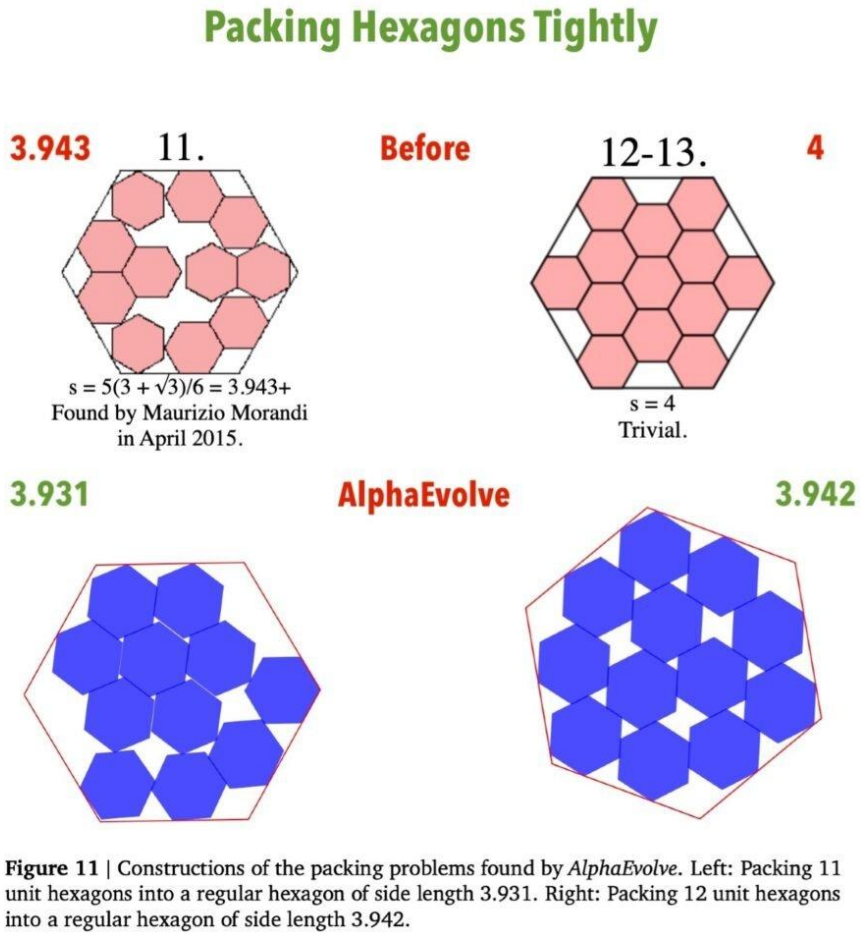


Figure 11 | Constructions of the packing problems found by *AlphaEvolve*. Left: Packing 11 unit hexagons into a regular hexagon of side length 3.931. Right: Packing 12 unit hexagons into a regular hexagon of side length 3.942.

Questions?