



Scalable CI/CD Infra With EKS and Karpenter

KCD Ukraine Fundraiser 2023

About Me: Serhii Vasylenko

Developer Experience Engineer at  Grammarly

Specialization: CI/CD, AWS, Terraform

Contributor to  and  communities



Agenda

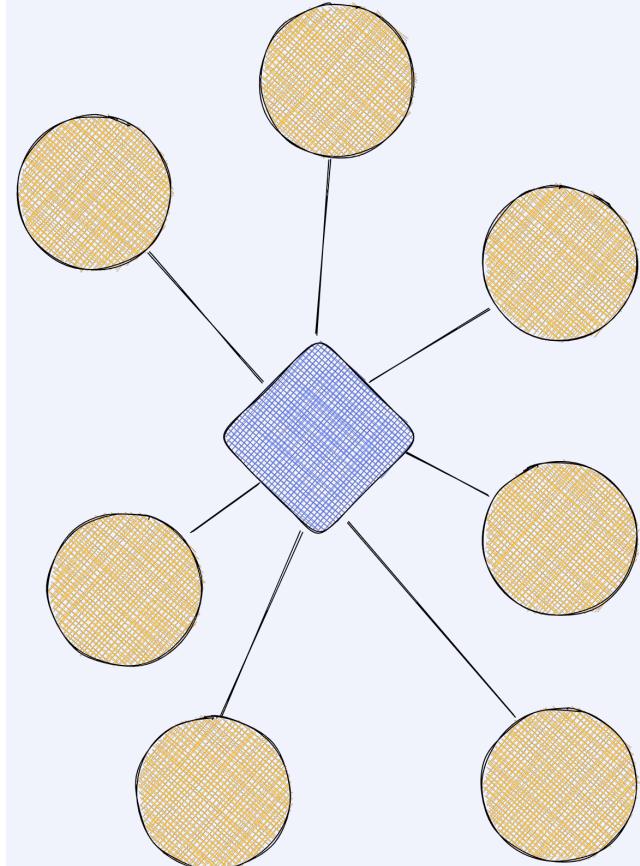
- 
- Context: why and where**
 - One CI account “to rule them all”**
 - Dynamic provisioning with Karpenter**
 - Extra CI run-time tweaks with Kyverno**
 - Deployment and self-service**

Context: why and where

AWS Environment

Multi-account AWS organization

Around 200 accounts and growing



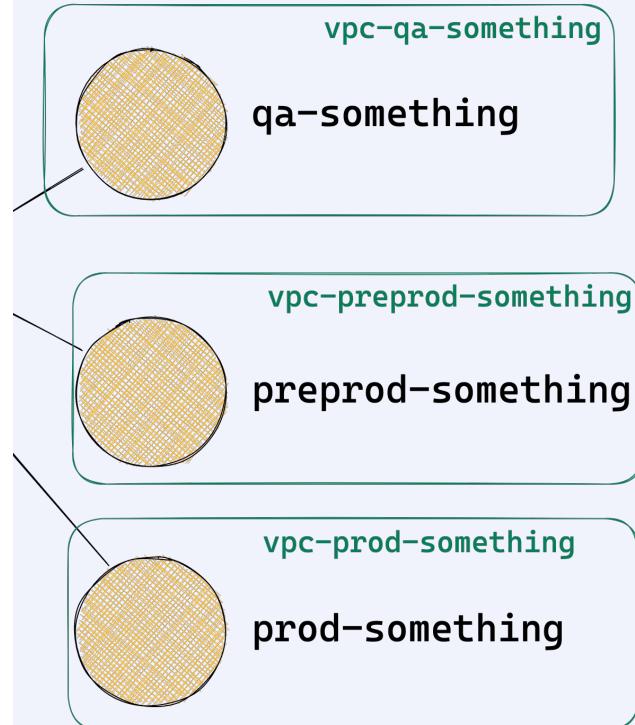
Context: why and where

AWS Environment

AWS account per project's environment

Each account has its own VPC

Transit Gateways used for cross-VPC communication



Context: why and where

Why: Code Management

Self-hosted GitLab server:
1,500 projects and 450 users

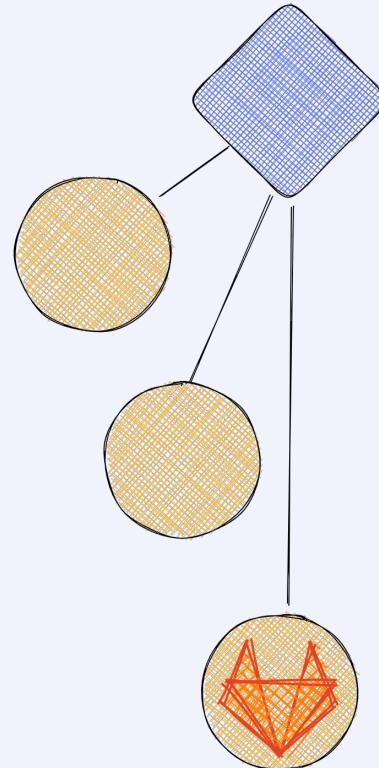
Projects grouped by teams

*CI runners access
most of AWS accs*

Everyone can contribute to any project

*Security
and Audit*

*And deploy
QA / Preprod*



Why: Old Infra Management

CI runners deployed in AWS every account

No auto-scaling

“Best guess” provisioning, no capacity review later

Duplicated runners for access to the same AWS account
from different GitLab projects

Agenda

- 
- Context: why and where
 - One CI account “to rule them all”
 - Dynamic provisioning with Karpenter
 - Extra CI run-time tweaks with Kyverno
 - Deployment and self-service

One CI account "to rule them all"

CI/CD Runners EKS Cluster

**Centralized
resource
management**

**Access
to all
AWS accounts**

**Customizable
CI environment
options**

**Fast scaling
and
cost efficiency**



One CI account "to rule them all"

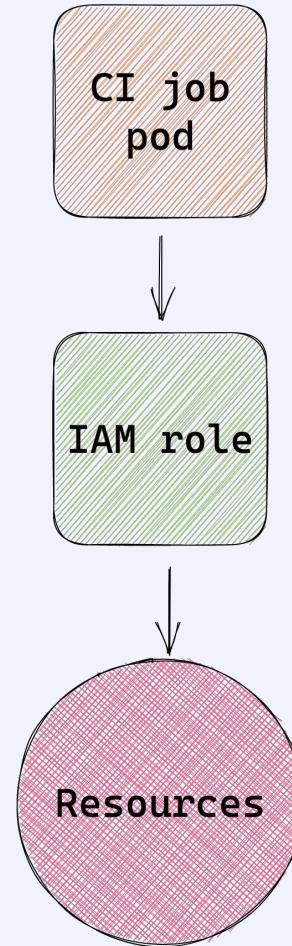
Access Control With IAM

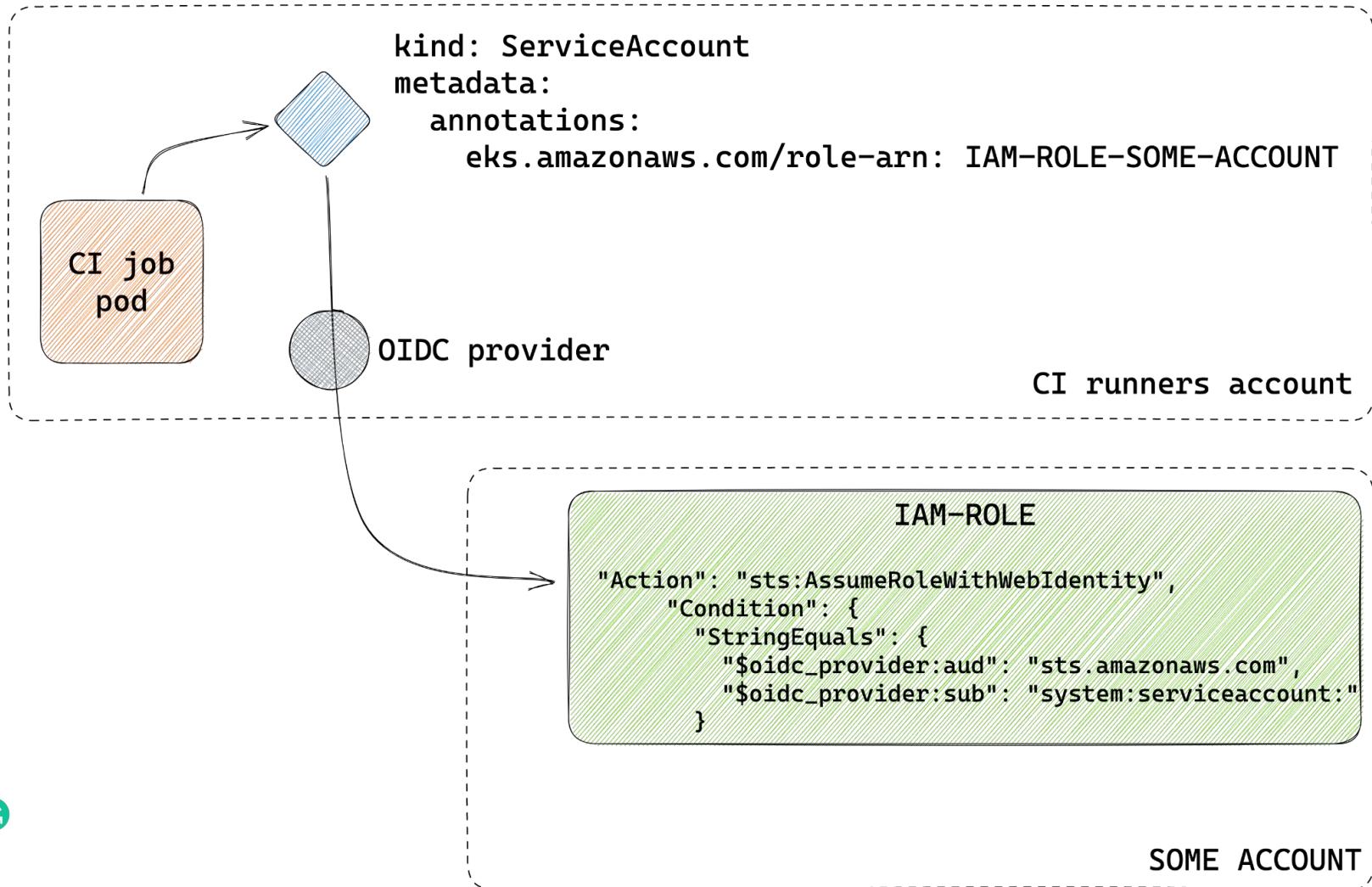
Pod with a CI job uses service accounts (SA)

SA authenticates through OpenID Connect provider to get to the target AWS account

Target AWS account has IAM role for EKS CI pods

IAM role trusts only CI pods, and users can extend policies





One CI account "to rule them all"

How It Looks in a CI Job

```
app-build-job:  
  variables:  
    KUBERNETES_SERVICE_ACCOUNT_OVERWRITE: "account-name-here"  
    KUBERNETES_CPU_REQUEST: "3"  
    KUBERNETES_CPU_LIMIT: "5"  
    KUBERNETES_MEMORY_REQUEST: "2Gi"  
    KUBERNETES_MEMORY_LIMIT: "4Gi"  
  image: "repo/image:tag"  
  script:  
    - my build command
```

Access to an AWS account

And what about this?



Agenda

- 
- Context: why and where
 - One CI account “to rule them all”
 - Dynamic provisioning with Karpenter**
 - Extra CI run-time tweaks with Kyverno
 - Deployment and self-service

What Is It?

A bit smarter than CAS:

- Group-less node provisioning
- Highly customizable

Picks most suitable and cheap EC2 instances to meet the requirements

Coexists with nodes provisioned by other provisioners



Multiple provisioners configuration

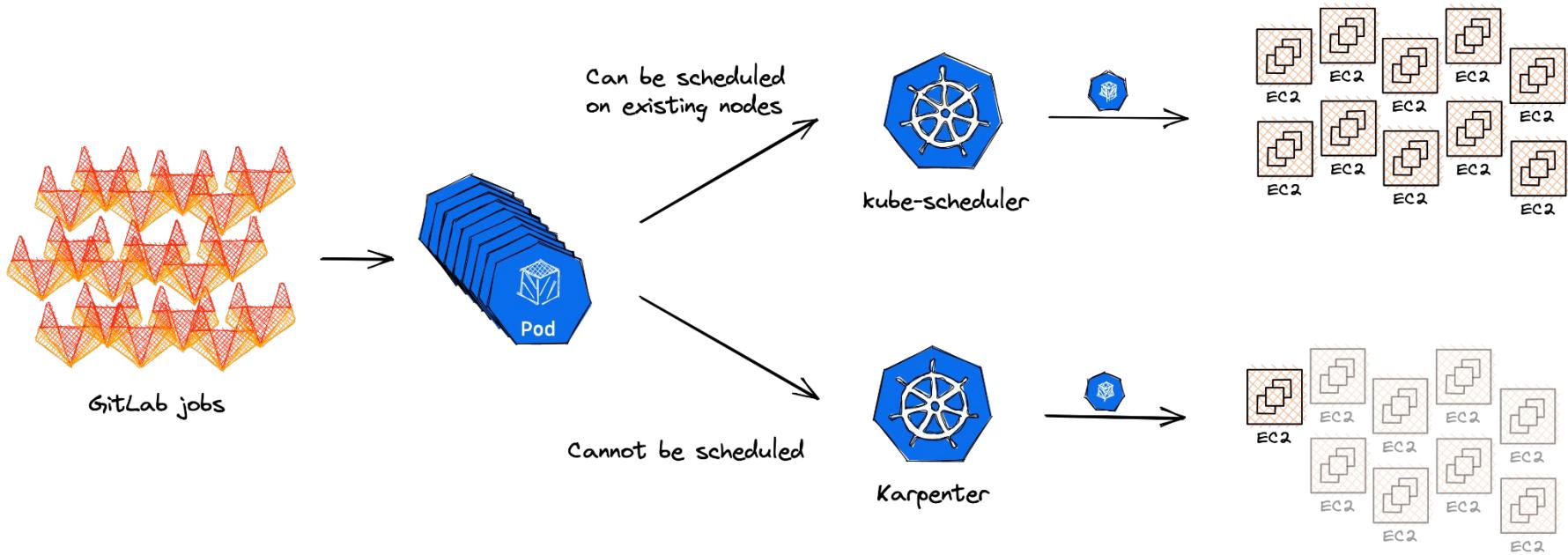


Karpenter Provisioner Example

```
apiVersion: karpenter.sh/v1alpha5
kind: Provisioner
...
requirements:
  - key: "karpenter.k8s.aws/instance-family"
    operator: In
    values: ["c5", "c5a", "c6i", "c6a", "m5", "m5a"]
  - key: "karpenter.k8s.aws/instance-size"
    operator: NotIn
    values: ["nano", "micro", "24xlarge", "32xlarge", "48xlarge"]
  - key: "kubernetes.io/arch"
    operator: In
    values: ["amd64"]
  - key: "karpenter.sh/capacity-type"
    operator: In
    values: ["on-demand", "spot"]
```



Dynamic CI Runners Provisioning

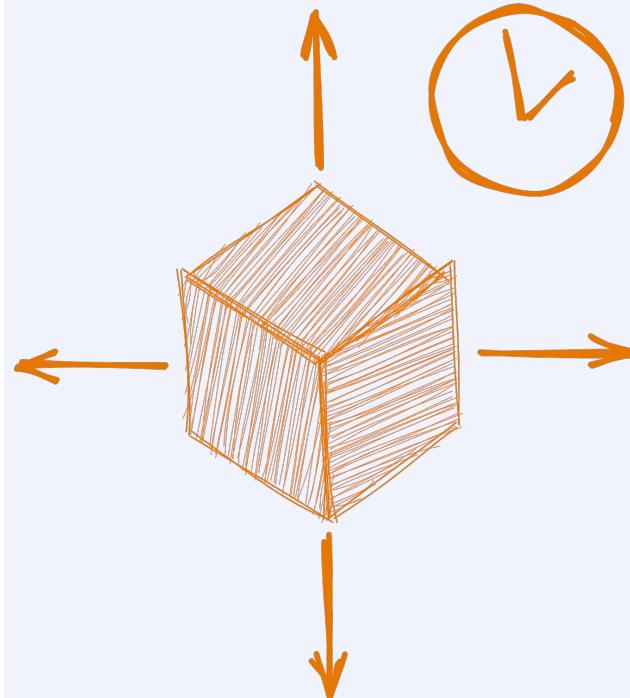


Warm Pools for Better UX

Group of nodes to handle around a half of the jobs demand

Two node groups of different sizes

Scale to zero on weekends



More for Customizable CI

Capacity select:

- On-demand
- Spot

GPU
for ML-related CI

Specific
instance
type
(for snowflakes)

Evenly spread
across AZs



But GitLab and Karpenter need an "assistant"

Agenda

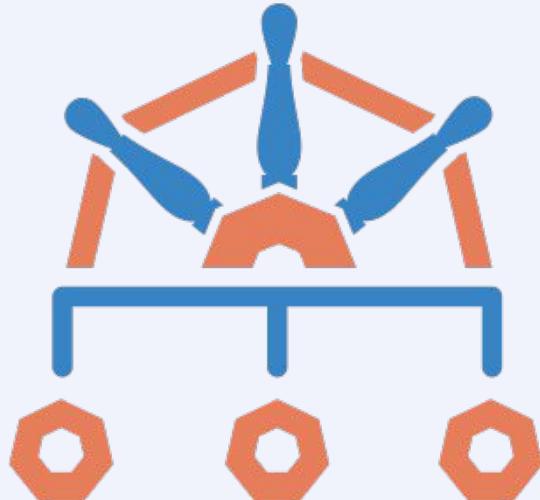
- 
- Context: why and where
 - One CI account “to rule them all”
 - Dynamic provisioning with Karpenter
 - Extra CI run-time tweaks with Kyverno**
 - Deployment and self-service

Kyverno — Policy Engine

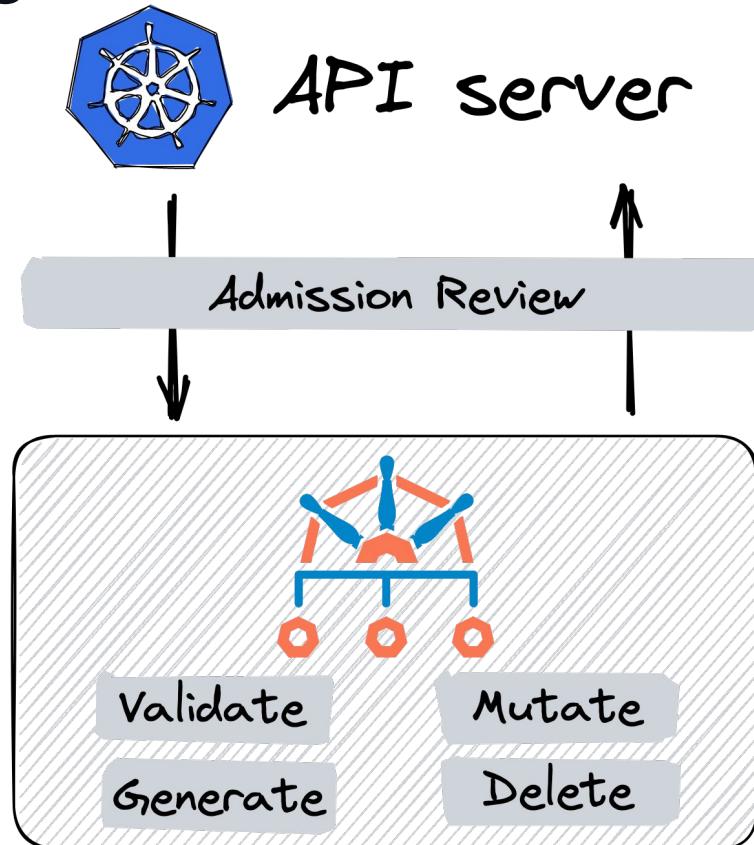
Policies are another kind of Kubernetes resources

What policies do:

Validate, mutate, generate, or delete other K8s resources



How It Works



Example: Spot Instances for CI Jobs

```
apiVersion: kyverno.io/v1
kind: ClusterPolicy
metadata:
  name: set-capacity-type
spec:
  rules:
    - name: set-spot
      match:
        any:
          - resources:
              annotations:
                k8s.gitlab.runner/capacity-type: "spot"
              kinds:
                - Pod
      mutate:
        patchesJson6902: |-
          - op: add
            path: "/spec/tolerations/-"
            value: {"key":"spot","operator":"Exists","effect":"NoSchedule"}
          - op: add
            path: "/spec/nodeSelector"
            value: {"karpenko.sh/capacity-type": "spot"}  
  
This is checked by Karpenter
```



Example: Spot Instances for CI Jobs

```
● ● ● GitLab CI configuration  
my-ci-job:  
  variables:  
    KUBERNETES_POD_ANNOTATIONS_1: "k8s.gitlab.runner/capacity-type=spot"  
    KUBERNETES_MEMORY_REQUEST: 1Gi  
    KUBERNETES_MEMORY_LIMIT: 8Gi  
  script:  
    - some command to execute
```

Example: Spread CI Pods Evenly

```
apiVersion: kyverno.io/v1
kind: ClusterPolicy
metadata:
  name: set-topology-spread-zone
spec:
  rules:
    - name: "set-topology-spread-zone"
      match:
        any:
          - resources:
              kinds:
                - Pod
              selector:
                matchLabels:
                  app: "gitlab-runner-job"
      mutate:
        patchStrategicMerge:
          spec:
            topologySpreadConstraints:
              - labelSelector:
                  matchLabels:
                    app: "gitlab-runner-job"
            maxSkew: 1
            topologyKey: "topology.kubernetes.io/zone"
            whenUnsatisfiable: ScheduleAnyway
```

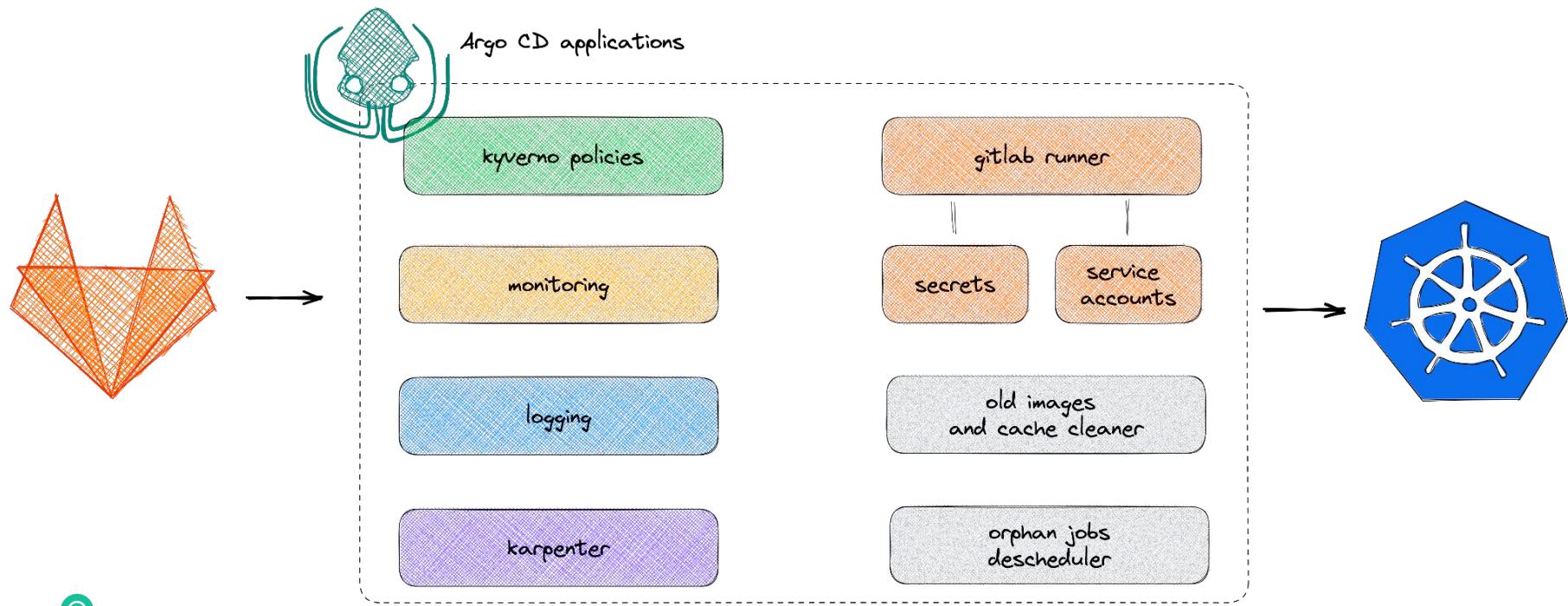
Set for every CI pod

Read by Karpenter

Agenda

- 
- Context: why and where
 - One CI account “to rule them all”
 - Dynamic provisioning with Karpenter
 - Extra CI run-time tweaks with Kyverno
 - Deployment and self-service

GitOps With Argo CD

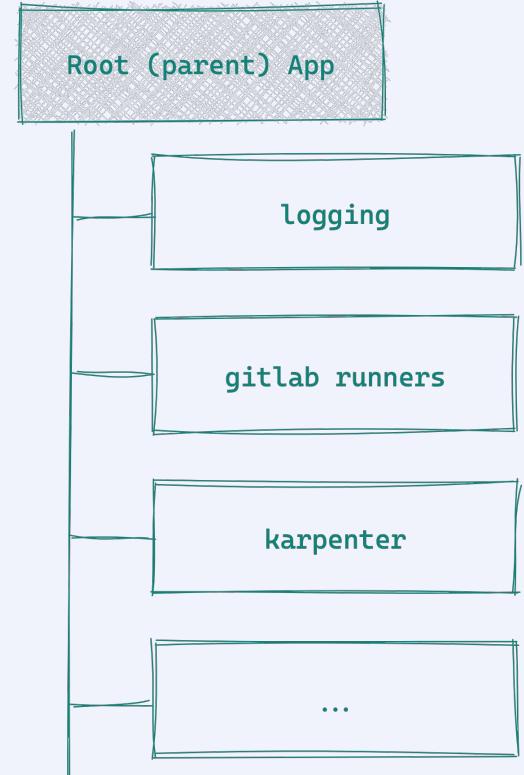


Approaches: App of Apps

The parent app — helm chart with specs for children

Child apps — separate subprojects, also Helm charts

Works fine for static set of applications



Approaches: Application Set

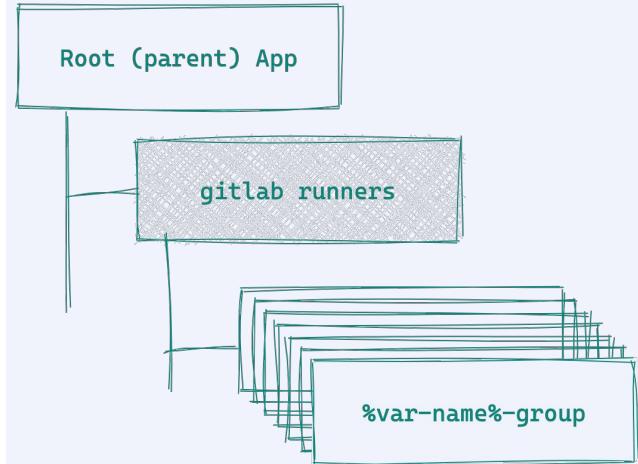
Generates application configs from templates

Combines (if needed) multiple configuration sources

Generators: list, matrix, git (files/folders), clusters

Our case:

- Register runners on GitLab groups
- Add service accounts for runners to allow AWS access

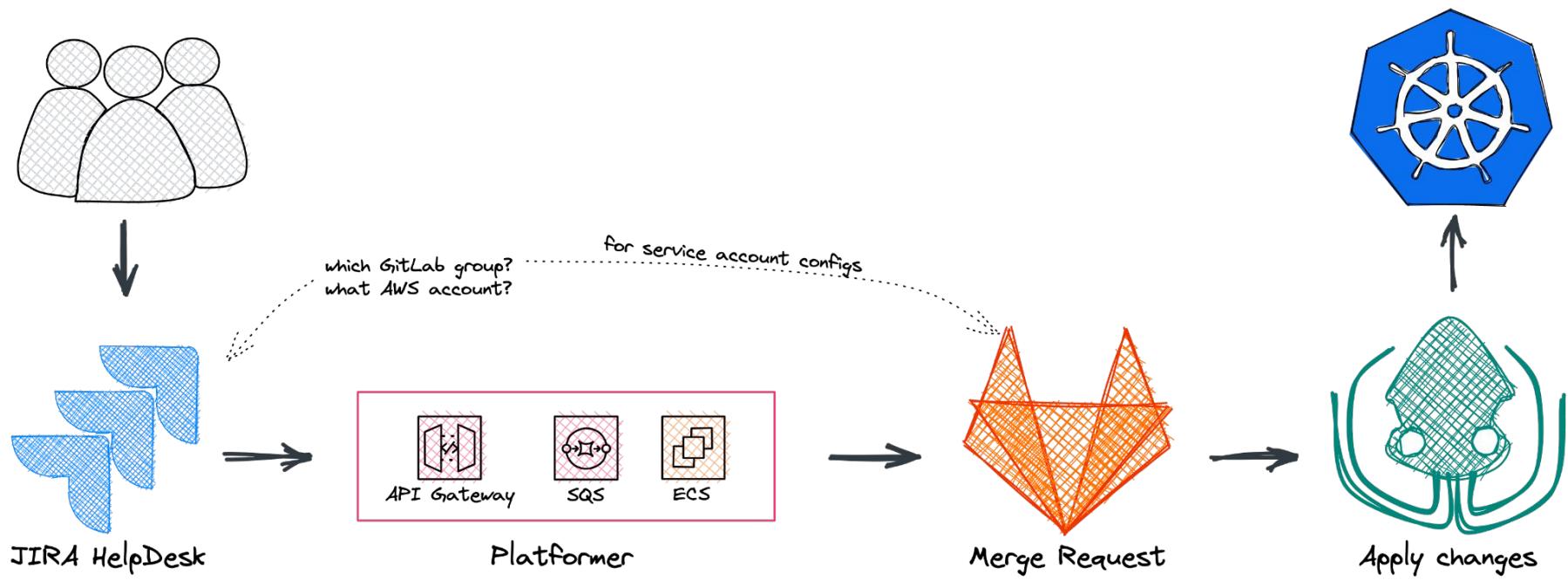


Example: Application Set

```
...
spec:
  generators:
    - git:
        repoURL: redacted-git-url
        revision: main
        files:
          - path: "redacted-path-to-config.json"
template:
  metadata:
    name: 'gitlab-group-runners-{{name}}-app'
  spec:
    project: default
    syncPolicy:
      automated:
        prune: true
    syncOptions:
      - CreateNamespace=true
  source:
    repoURL: redacted-git-url
    path: "charts/gitlab-runners"
    targetRevision: main
    helm:
      ignoreMissingValueFiles: true
      valueFiles:
        - redacted-path-to-values.yaml # chart default values
        - redacted-path-to-group-{{name}}-values.yaml # chart customized values
        - .redacted-path-to-values.yaml # namespace values
    releaseName: gitlab-runner
    parameters:
      - name: 'redacted-parameter-name'
        value: 'k8s-gl-{{name}}'
destination:
  server: https://kubernetes.default.svc
  namespace: 'gitlab-group-runners-{{name}}'
```

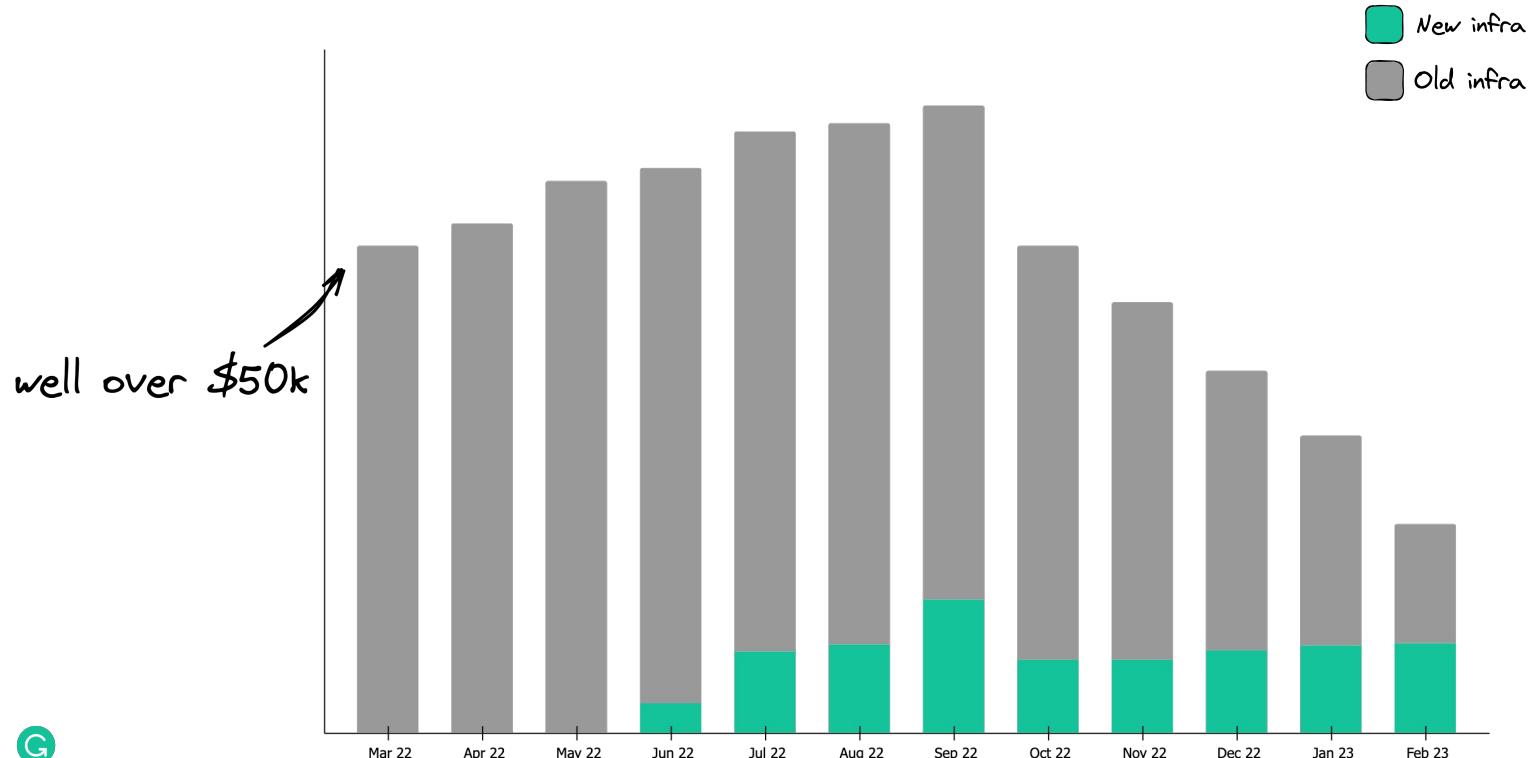


When Users Need Runners



Context: why and where

What About Costs?



**Thank you
for being here today!**

Q&A

Let's connect!

 devDosvid.blog

 linkedin.com/in/svasylenko

