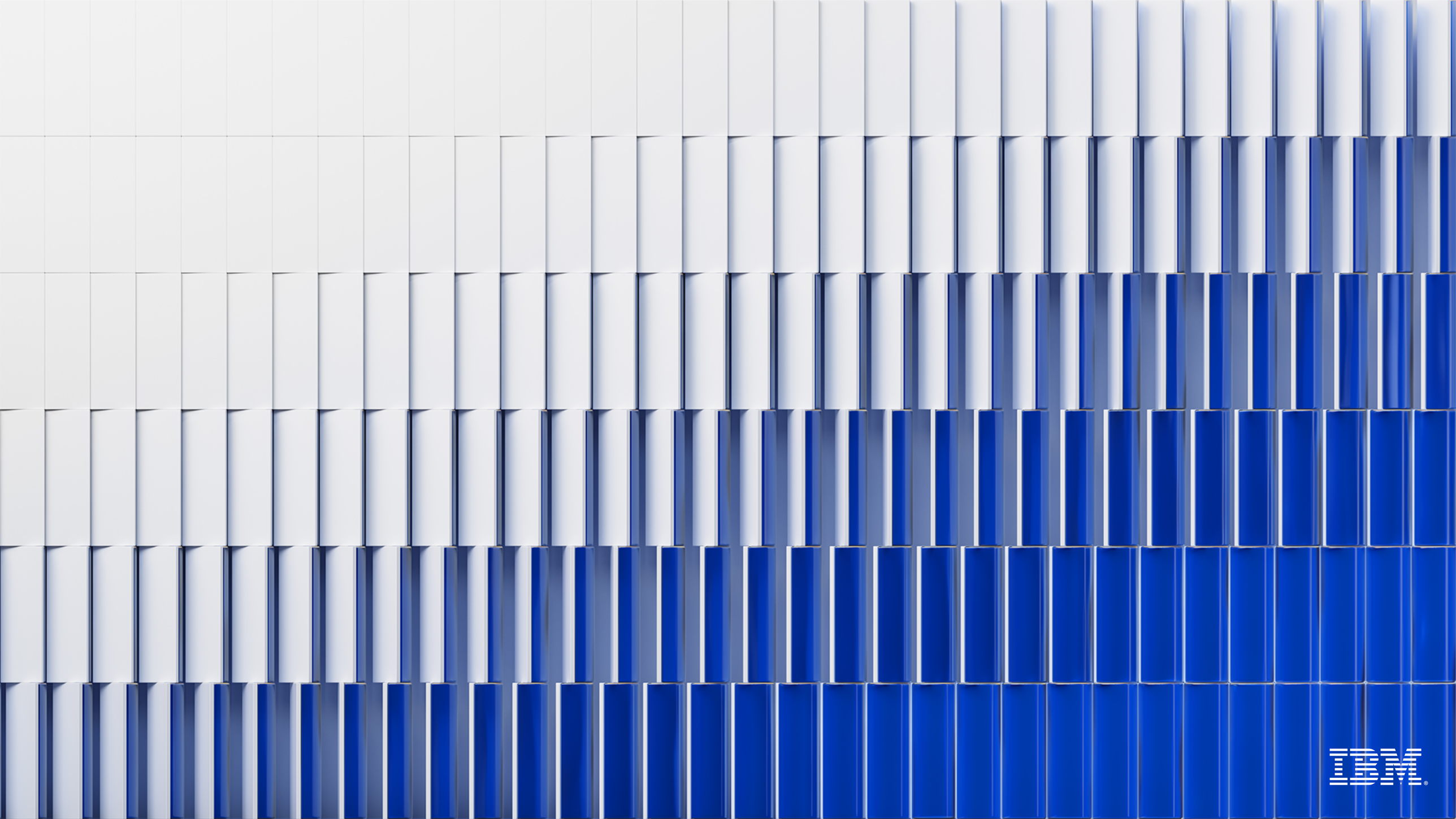


Taming the Wild West of Research Computing: How Policies Saved Us a Thousand Headaches

Alessandro Pomponio
Research Software Engineer

KyvernoCon
NORTH AMERICA

IBM

A bit of context

Accelerated Discovery

- ⚛️ Accelerated Discovery explores the intersection between *Artificial Intelligence*, *Hybrid Cloud*, and *Quantum Computing*.
- 🚀 It aims to ***drastically increase how quickly we can discover solutions to tackle today's most urgent problems***.
- 🤝 We collaborate with researchers coming from many different backgrounds: AI, HPC, chemistry, mathematics, and more.
- 🛠️ ***My team manages two bare-metal OpenShift clusters for our colleagues to run their workloads on***.

Challenges in Research Computing

👨‍🔬 *Researchers ≠ Kubernetes Experts*

Researchers used to HPC environments often create interactive pods to run GPU workloads.

Pods are left running long after training ends, leaving GPUs idle.

🏋️ *Bursty Workloads & Resource Contention*

Workloads are bursty, spiking near conference deadlines.

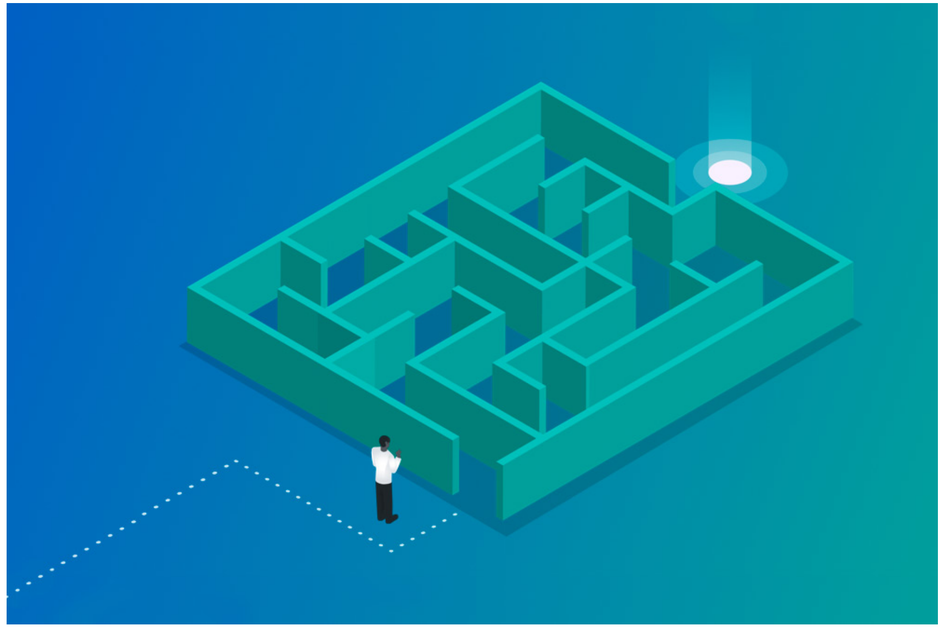
Large CPU jobs often spill over onto GPU nodes, causing scheduling issues.

🦹‍♂️*The side gig problem*

Manual processes left admins handling every user request, *twice*.

The volume of requests made it increasingly difficult to focus on our own research.

What’s the plan?



Shift namespace-management decisions to users

Introduce automation and tooling that enables self-service namespace management even for non-experts users.

Prevent CPU jobs from running amok

Introduce safeguards to prevent CPU-only workloads from overwhelming the cluster or interfering with GPU scheduling.

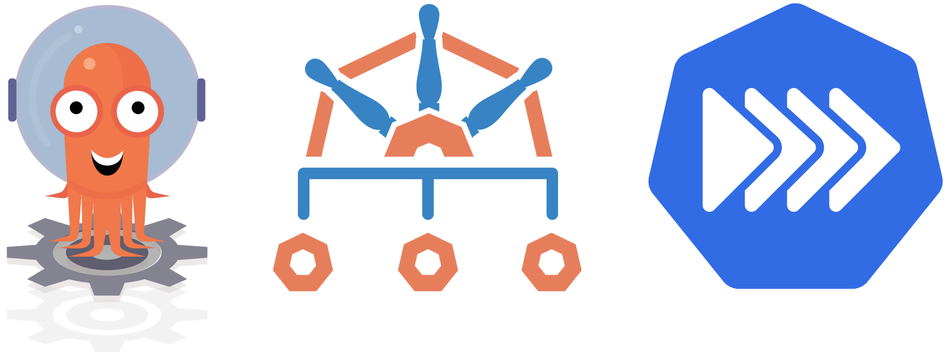
Discourage interactive GPU usage

Prevent the creation of long-lived, interactively accessed pods in favour of more efficient, Job-based workflows that free up (GPU) resources once they’re complete.

Make GPU access fair

Improve resource allocation to ensure GPU access is fair, providing a more predictable, HPC-like experience that minimises contention.

What’s the plan?



Shift namespace-management decisions to users

Introduce automation and tooling that enables self-service namespace management even for non-experts users.

Prevent CPU jobs from running amok

Introduce safeguards to prevent CPU-only workloads from overwhelming the cluster or interfering with GPU scheduling.

Discourage interactive GPU usage

Prevent the creation of long-lived, interactively accessed pods in favour of more efficient, Job-based workflows that free up (GPU) resources once they’re complete.

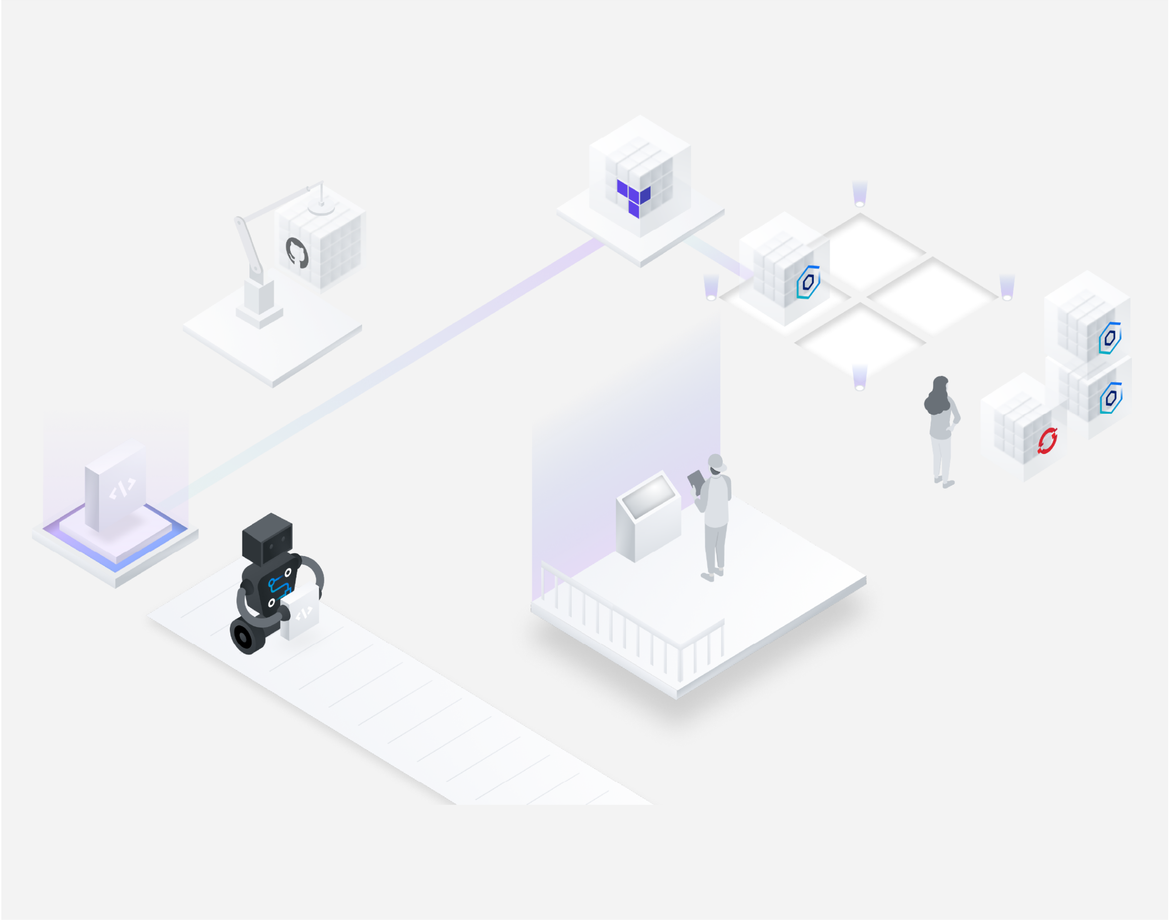
Make GPU access fair

Improve resource allocation to ensure GPU access is fair, providing a more predictable, HPC-like experience that minimises contention.

Automating

Our goals

- ***Let users own their namespaces***. Users submit change requests for admins review and approval.
- ***Eliminate manual deployments***. Automation saves time, removes guesswork, and guarantees consistency.
- ***Be easy to use.*** It should not impose any toll on non-expert users. Admins should take care of the heavy lifting.
- ***Minimise maintenance overhead***. Configurations and tooling should be easy to set up, maintain, and extend.

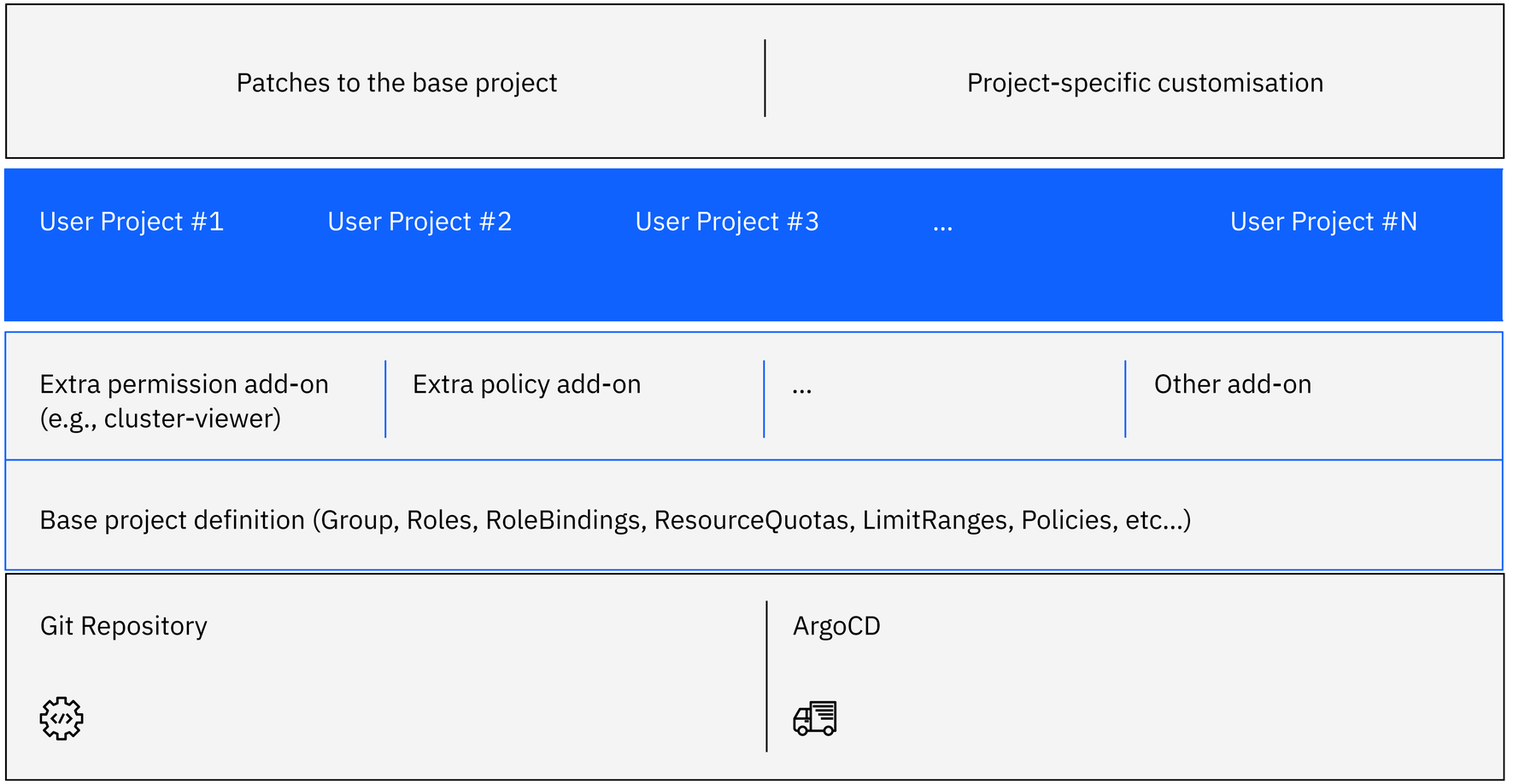


Automating

The idea

Users

Admins

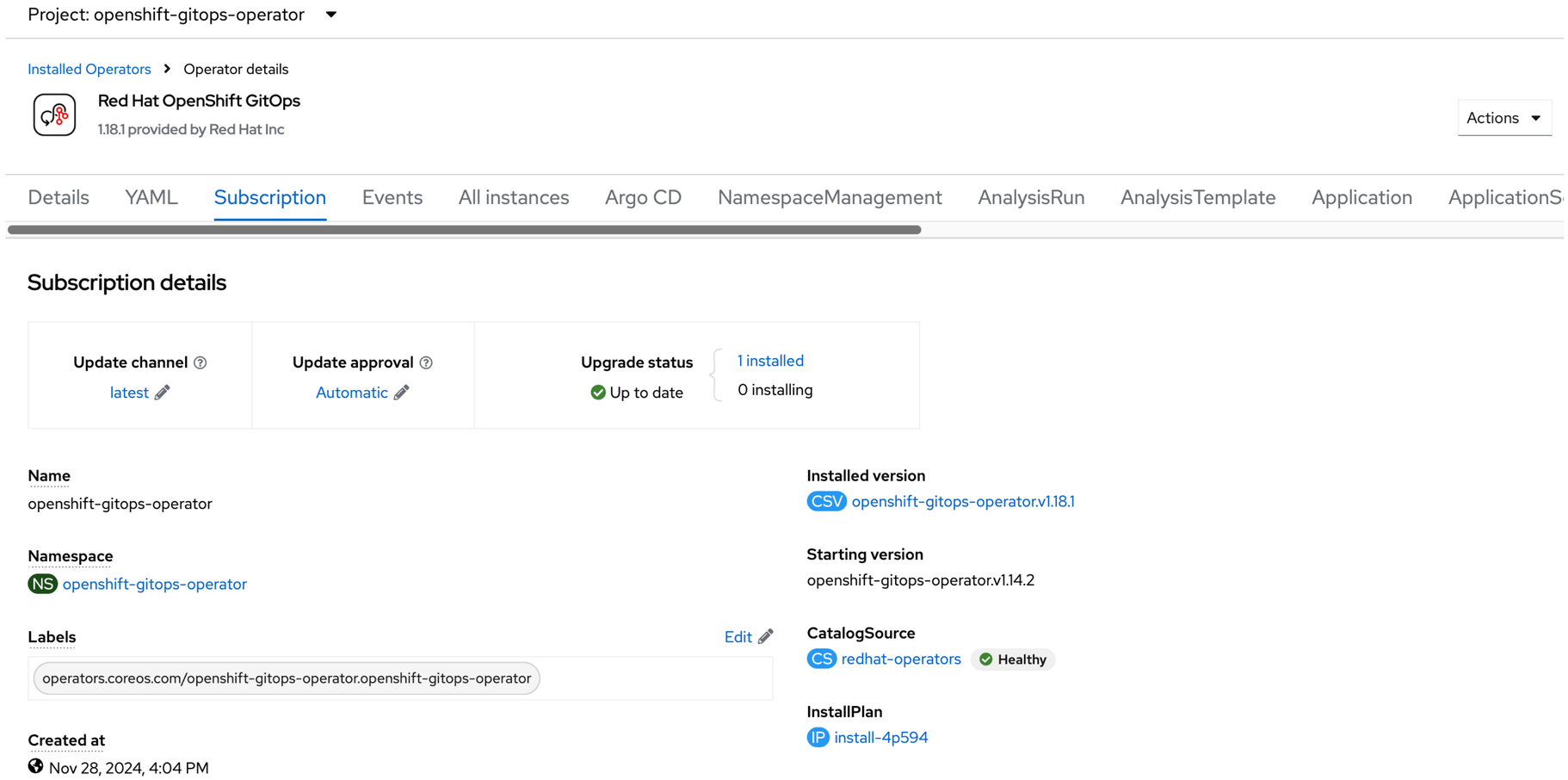


*Note: we want all of this while **minimising effort for everyone***

Automating

Minimising maintenance

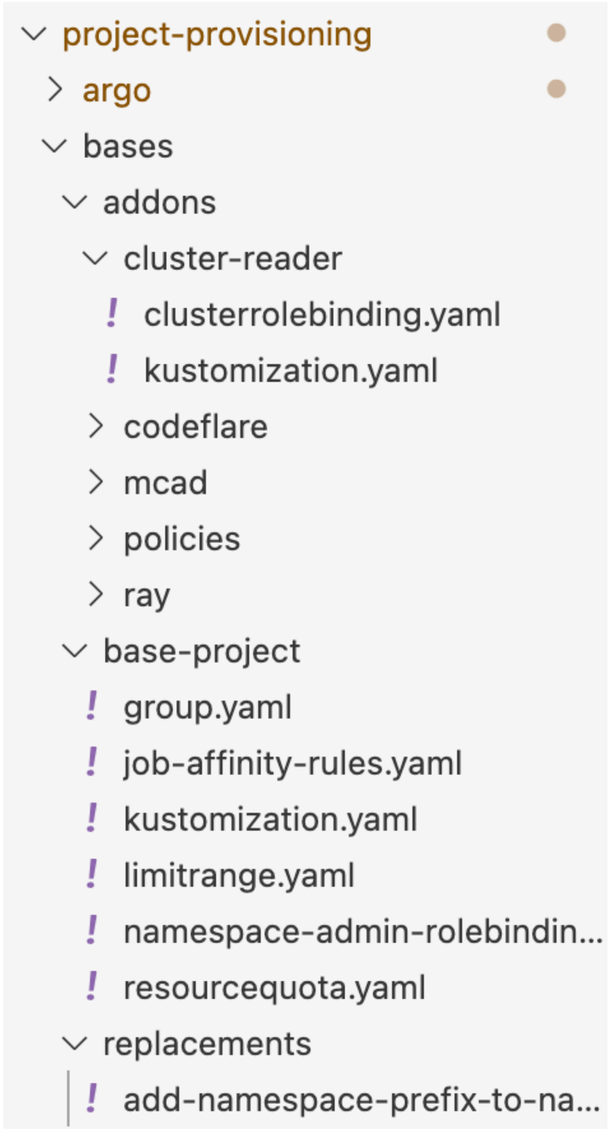
- Install your GitOps tool of choice in a way that minimises overhead for configuring, patching, etc.



Automating

Extensibility and ease-of-use

- *Kustomize is easy for both users and admins*. Keep it simple and use a LEGO-style approach.
- *Have everything ready*. Require users to change as little as possible. **Provide a fully-fledged example to copy/paste from**.
- *Provide a set of useful add-ons* that users can use by adding just one line in their kustomization.yaml file.



Automating

Extensibility and ease-of-use

- *Kustomize is easy for both users and admins*. Keep it simple and use a LEGO-style approach.
- *Have everything ready*. Require users to change as little as possible. **Provide a fully-fledged example to copy/paste from**.
- *Provide a set of useful add-ons* that users can use by adding just one line in their kustomization.yaml file.

```yaml
apiVersion: user.openshift.io/v1
kind: Group
metadata:
  name: group
users: []
```

```yaml
apiVersion: v1
kind: LimitRange
metadata:
  name: default-resources-for-containers
spec:
  limits:
    - default:
        cpu: 1
        memory: 2Gi
      defaultRequest:
        cpu: 500m
        memory: 512Mi
      type: Container
```

```yaml
resources:
  - group.yaml
  - namespace-admin-rolebinding.yaml
  - resourcequota.yaml
  - limitrange.yaml
  # Policies
  - job-affinity-rules.yaml
```

Automating

Extensibility and ease-of-use

- ***Automate the creation of Argo Applications*** using Git Directory Generators and ApplicationSets.

- ***Prevent drift*** by enabling self-healing and automated pruning (⚠️).

```yaml
apiVersion: argoproj.io/v1alpha1
kind: ApplicationSet
metadata:
  name: user-projects
spec:
  goTemplate: true
  goTemplateOptions: ["missingkey=error"]
  syncPolicy:
    automated:
      prune: true
      selfHeal: true
  generators:
    - git:
        repoURL: ssh://git@example.com/org/repo
        revision: HEAD
        directories:
          - path: project-provisioning/user-projects/*
  template:
    metadata:
      name: "{{.path.basename}}"
    spec:
      project: "provisioning"
      source:
        repoURL: ssh://git@example.com/org/repo
        targetRevision: HEAD
        path: "{{.path.path}}"
      destination:
        server: https://kubernetes.default.svc
        namespace: "{{.path.basename}}"
      syncPolicy:
        syncOptions:
          - CreateNamespace=true
        automated:
          prune: true
          selfHeal: true
```

Automating

Extensibility and ease-of-use

- ***Automate the creation of Argo Applications*** using Git Directory Generators and ApplicationSets.

- ***Prevent drift*** by enabling self-healing and automated pruning (⚠️).

```yaml
apiVersion: argoproj.io/v1alpha1
kind: ApplicationSet
metadata:
  name: user-projects
spec:
  goTemplate: true
  goTemplateOptions: ["missingkey=error"]
  syncPolicy:
    automated:
      prune: true
      selfHeal: true
  generators:
    - git:
        repoURL: ssh://git@example.com/org/repo
        revision: HEAD
        directories:
          - path: project-provisioning/user-projects/*
  template:
    metadata:
      name: "{{.path.basename}}"
    spec:
      project: "provisioning"
      source:
        repoURL: ssh://git@example.com/org/repo
        targetRevision: HEAD
        path: "{{.path.path}}"
      destination:
        server: https://kubernetes.default.svc
        namespace: "{{.path.basename}}"
      syncPolicy:
        syncOptions:
          - CreateNamespace=true
        automated:
          prune: true
          selfHeal: true
```

Automating

Extensibility and ease-of-use

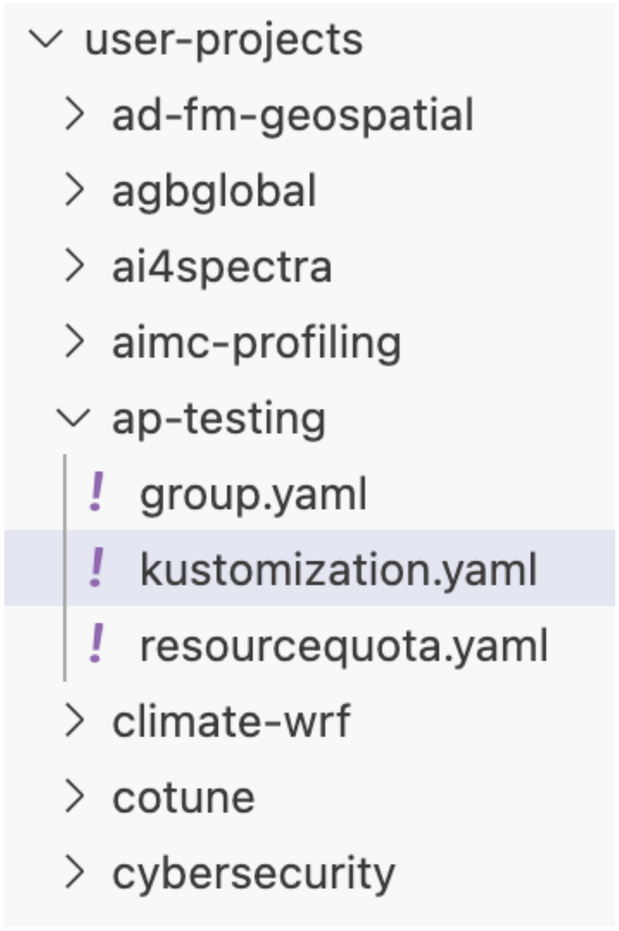
- ***Automate the creation of Argo Applications*** using Git Directory Generators and ApplicationSets.

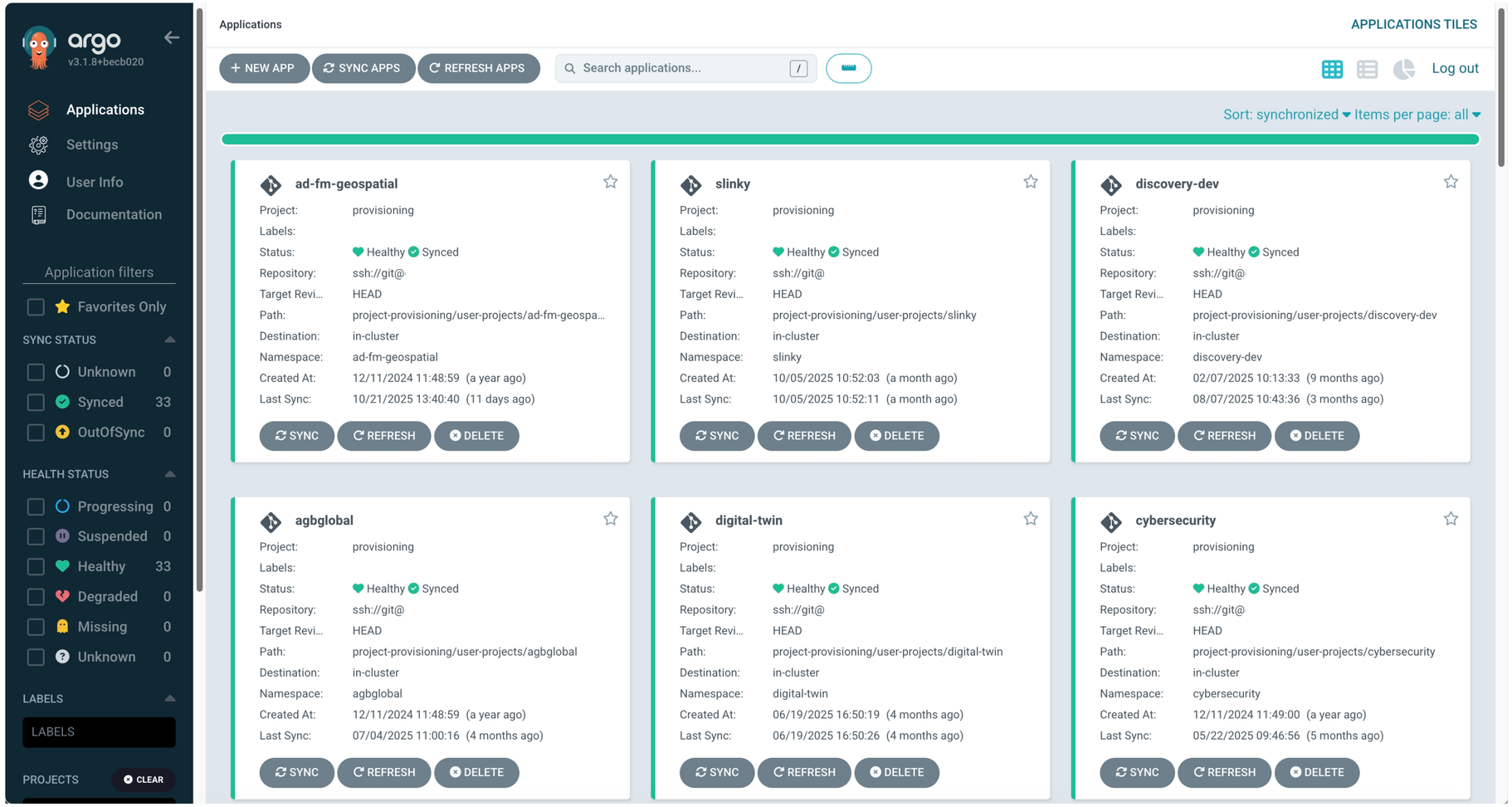
- ***Prevent drift*** by enabling self-healing and automated pruning (⚠️).

```yaml
apiVersion: argoproj.io/v1alpha1
kind: ApplicationSet
metadata:
  name: user-projects
spec:
  goTemplate: true
  goTemplateOptions: ["missingkey=error"]
  syncPolicy:
    automated:
      prune: true
      selfHeal: true
  generators:
    - git:
        repoURL: ssh://git@example.com/org/repo
        revision: HEAD
        directories:
          - path: project-provisioning/user-projects/*
  template:
    metadata:
      name: "{{.path.basename}}"
    spec:
      project: "provisioning"
      source:
        repoURL: ssh://git@example.com/org/repo
        targetRevision: HEAD
        path: "{{.path.path}}"
      destination:
        server: https://kubernetes.default.svc
        namespace: "{{.path.basename}}"
      syncPolicy:
        syncOptions:
          - CreateNamespace=true
        automated:
          prune: true
          selfHeal: true
```

Automating

Results





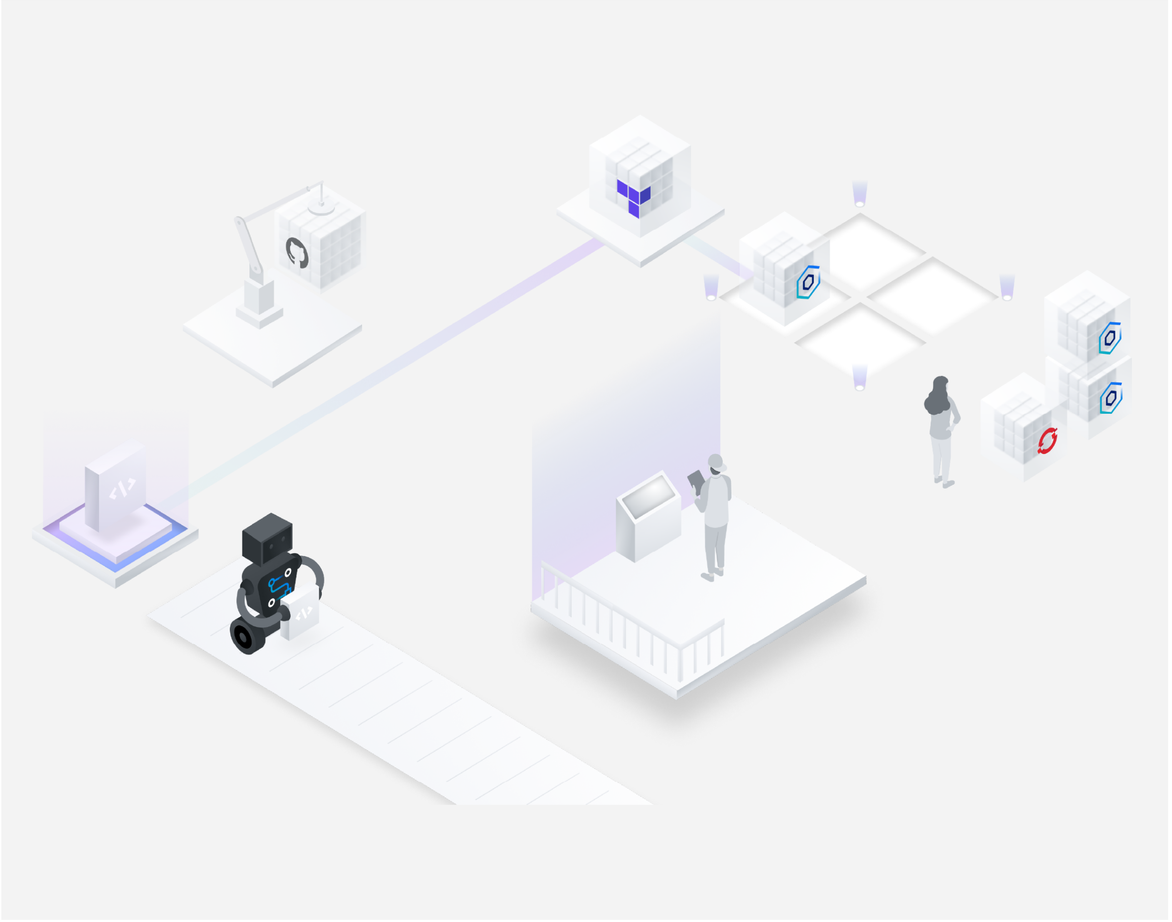
Automating

Results

- ***Users manage their namespaces.*** They submit PRs that require just a review and changes are deployed instantly. Kustomize is intuitive for both admins and users.

- ***Compliance is guaranteed.*** A shared project definition guarantees compliance and additions there cascade to all namespaces. Self-healing prevents manual changes from persisting.

- ***Maintenance effort is minimised***. Manual deployments are gone, ArgoCD auto-updates. Slack requests for simple changes have effectively been zeroed.



What’s the plan?



✅ Reduce the need for admin intervention

Introduce automation and tooling that enables self-service namespace management even for non-experts users.

Prevent CPU jobs from running amok

Introduce safeguards to prevent CPU-only workloads from overwhelming the cluster or interfering with GPU scheduling.

Discourage interactive GPU usage

Prevent the creation of long-lived, interactively accessed pods in favour of more efficient, Job-based workflows that free up (GPU) resources once they’re complete.

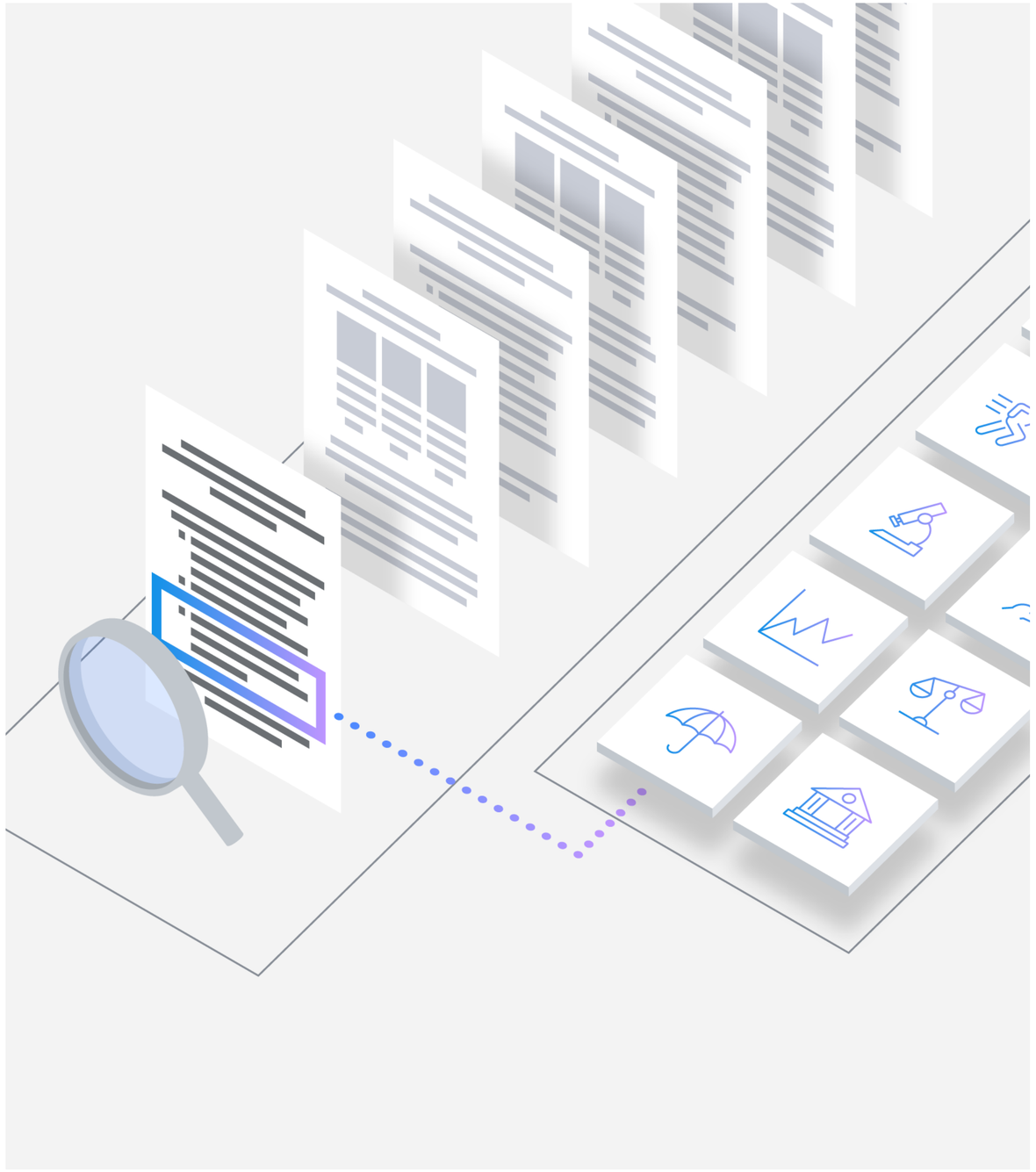
Make GPU access fair

Improve resource allocation to ensure GPU access is fair, providing a more predictable, HPC-like experience that minimises contention.

Policing

Our goals

- ***Fix the root causes of our issues.*** Make sure we help the scheduler start pods on appropriate nodes and prevent interactive GPU pods.
- ***Not be overly disruptive***. Our policies should not do more harm than good. Exception mechanisms and observability are a must.
- ***Get up and running quickly***. Avoid having to learn a new language, do as much copy/pasting as possible.



Policing

Handling batch Jobs

How to deal with large CPU-only Jobs jailing GPUs on GPU nodes?

Do not schedule their pods on them:

→ Automatically inject (anti-)affinity rules!

```yaml
apiVersion: kyverno.io/v1
kind: Policy
metadata:
  name: job-affinity-rules
spec:
  background: false
  rules:
    - name: avoid-gpu-nodes-for-jobs-with-no-gpus
      match:
        any:
          - resources:
              kinds:
                - Job
      context:
        - name: gpu_requests
          variable:
            # Use || [0] to return an array with just a zero if the field does not exist.
            # This is required because the sum function in Kyverno requires at least one element
            # in the array, which would otherwise be empty.
            jmesPath: 'request.object.spec.template.spec.containers[].resources.requests."nvidia.com/gpu" || [0]'
            default: [0]
      preconditions:
        all:
          - key: "{{ sum(gpu_requests) }}"
            operator: Equals
            value: 0
      mutate:
        patchStrategicMerge:
          spec:
            template:
              spec:
                affinity:
                  nodeAffinity:
                    requiredDuringSchedulingIgnoredDuringExecution:
                      nodeSelectorTerms:
                        - matchExpressions:
                            - key: nvidia.com/gpu.product
                              operator: DoesNotExist
```

Policing

Handling batch Jobs

How to deal with large CPU-only Jobs jailing GPUs on GPU nodes?

Do not schedule their pods on them:

→ Automatically inject (anti-)affinity rules!

```yaml
apiVersion: kyverno.io/v1
kind: Policy
metadata:
  name: job-affinity-rules
spec:
  background: false
  rules:
    - name: avoid-gpu-nodes-for-jobs-with-no-gpus
      match:
        any:
          - resources:
              kinds:
                - Job
      context:
        - name: gpu_requests
          variable:
            # Use || [0] to return an array with just a zero if the field does not exist.
            # This is required because the sum function in Kyverno requires at least one element
            # in the array, which would otherwise be empty.
            jmesPath: 'request.object.spec.template.spec.containers[].resources.requests."nvidia.com/gpu" || [0]'
            default: [0]
      preconditions:
        all:
          - key: "{{ sum(gpu_requests) }}"
            operator: Equals
            value: 0
      mutate:
        patchStrategicMerge:
          spec:
            template:
              spec:
                affinity:
                  nodeAffinity:
                    requiredDuringSchedulingIgnoredDuringExecution:
                      nodeSelectorTerms:
                        - matchExpressions:
                            - key: nvidia.com/gpu.product
                              operator: DoesNotExist
```

Policing

Handling interactive pods

How to deal with interactive pods running indefinitely?

- Deny *sleep infinity*?
- Deny *tail -f /dev/null*?

→Make it impossible to use them!

Bonus: started off from [one of the example policies](#)!

KyvernoCon NORTH AMERICA

Policing

Handling interactive pods

How to deal with interactive pods running indefinitely?

- Deny *sleep infinity*?
- Deny *tail -f /dev/null*?

→Make it impossible to use them!

Bonus: started off from [one of the example policies](#)!

```yaml
apiVersion: kyverno.io/v1
kind: ClusterPolicy
metadata:
  name: exec-only-from-cluster-admins
  namespace: kyverno-admin
spec:
  validationFailureAction: Enforce
  background: false
  rules:
    - name: exec-only-from-cluster-admins
      context:
        - name: exec-namespace-exceptions
          configMap:
            name: exec-namespace-exceptions
            namespace: kyverno-admin
      match:
        any:
          - resources:
              kinds:
                - Pod/exec
      preconditions:
        all:
          - key: "{{ request.operation || 'BACKGROUND' }}"
            operator: Equals
            value: CONNECT
          - key: "{{ request.clusterRoles.contains(@, 'cluster-admin') }}"
            operator: NotEquals
            value: true
          - key: "{{ request.namespace }}"
            operator: AnyNotIn
            value:
              '{{ "exec-namespace-exceptions".data."exceptions" | parse_json(@) }}'
      validate:
        message:
          Executing a command in a container is forbidden for Pods running in
          this Namespace. To request an exception, reach out to the admins.
        deny: {}
```

Policing

Handling interactive pods

How to deal with interactive pods running indefinitely?

- Deny *sleep infinity*?
- Deny *tail -f /dev/null*?

→Make it impossible to use them!

Bonus: started off from [one of the example policies](#)!

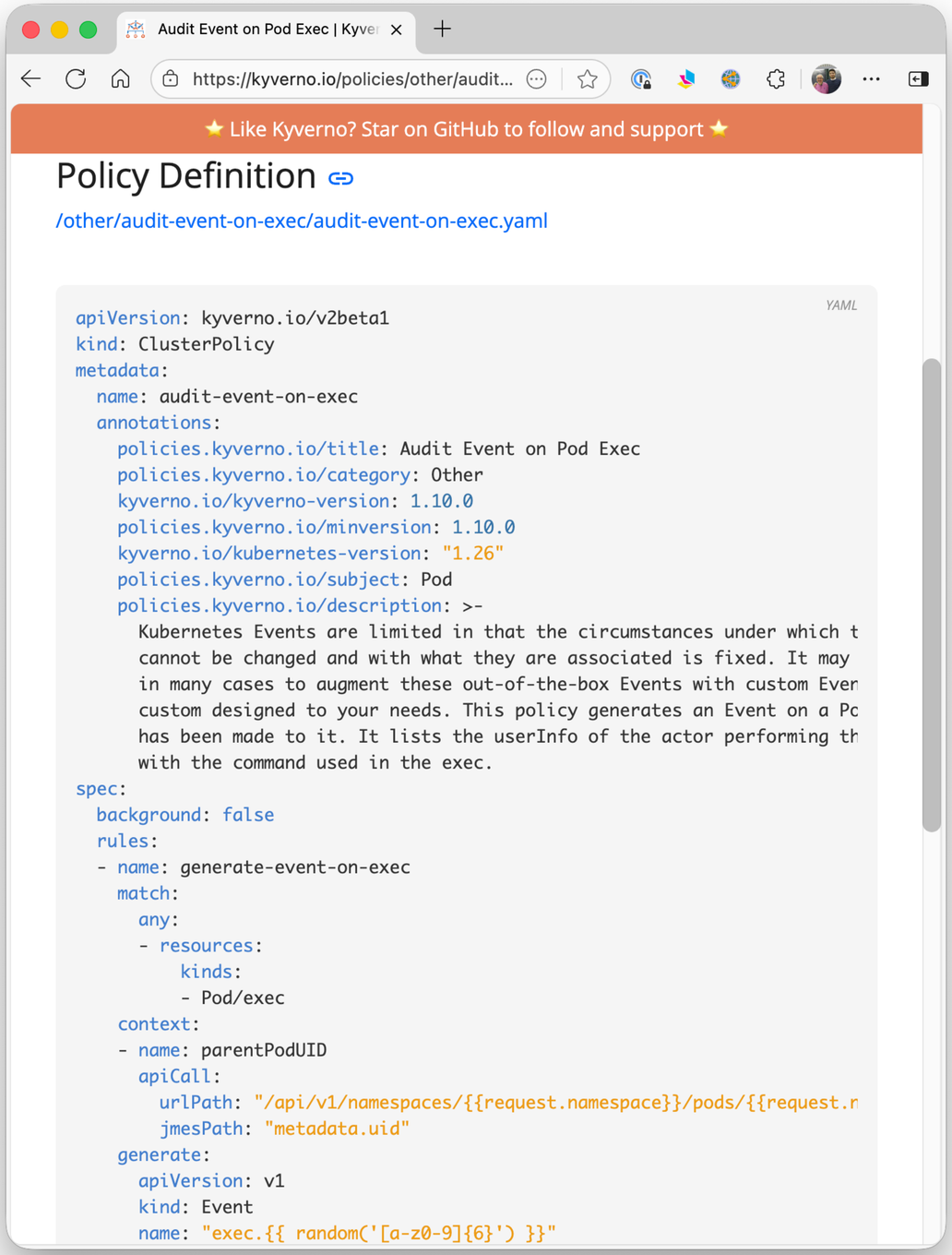
```yaml
apiVersion: kyverno.io/v1
kind: ClusterPolicy
metadata:
  name: exec-only-from-cluster-admins
  namespace: kyverno-admin
spec:
  validationFailureAction: Enforce
  background: false
  rules:
    - name: exec-only-from-cluster-admins
      context:
        - name: exec-namespace-exceptions
          configMap:
            name: exec-namespace-exceptions
            namespace: kyverno-admin
      match:
        any:
          - resources:
              kinds:
                - Pod/exec
      preconditions:
        all:
          - key: "{{ request.operation || 'BACKGROUND' }}"
            operator: Equals
            value: CONNECT
          - key: "{{ request.clusterRoles.contains(@, 'cluster-admin') }}"
            operator: NotEquals
            value: true
          - key: "{{ request.namespace }}"
            operator: AnyNotIn
            value:
              '{{ "exec-namespace-exceptions".data."exceptions" | parse_json(@) }}'
      validate:
        message:
          Executing a command in a container is forbidden for Pods running in
          this Namespace. To request an exception, reach out to the admins.
        deny: {}
```

Policing

Handling interactive pods

But... what if it's too much?

- *Create audit events*.
- *Make a test in a namespace with power users*.



Policing

Handling interactive pods

But... what if it's too much?

- *Create audit events*.
- *Make a test in a namespace with power users*.

KyvernoCon NORTH AMERICA

Policing

Handling interactive pods

It was too much!

- Blocking *oc cp* and *oc rsync*.
- Blocking *exec* on non-GPU pods.

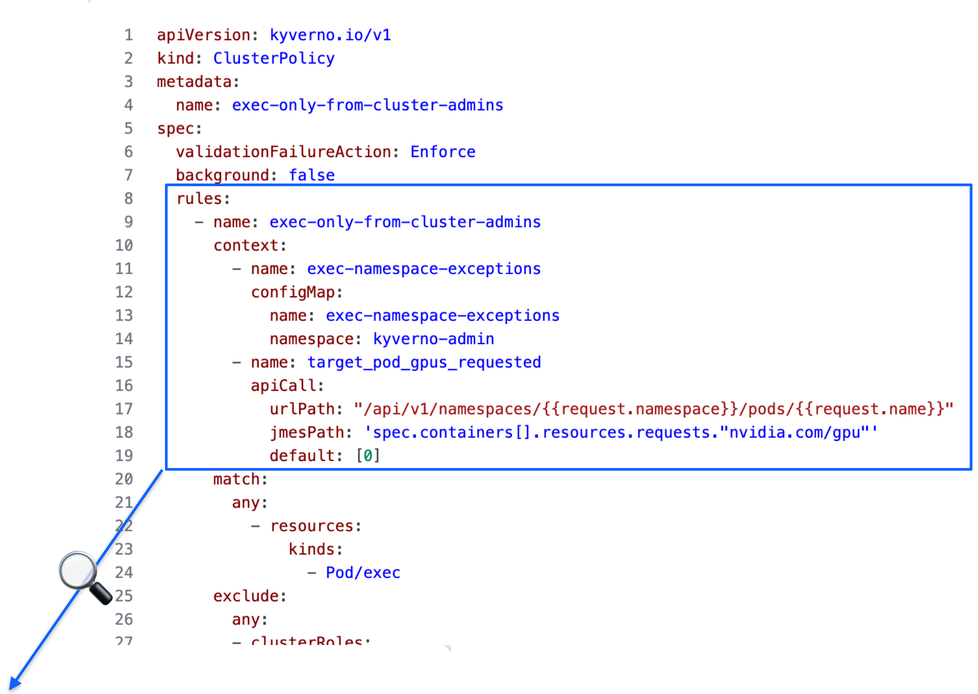
```yaml
apiVersion: kyverno.io/v1
kind: ClusterPolicy
metadata:
  name: exec-only-from-cluster-admins
spec:
  validationFailureAction: Enforce
  background: false
  rules:
    - name: exec-only-from-cluster-admins
      context:
        - name: exec-namespace-exceptions
          configMap:
            name: exec-namespace-exceptions
            namespace: kyverno-admin
        - name: target_pod_gpus_requested
          apiCall:
            urlPath: "/api/v1/namespaces/{{request.namespace}}/pods/{{request.name}}"
            jmesPath: 'spec.containers[].resources.requests."nvidia.com/gpu"'
            default: [0]
      match:
        any:
          - resources:
              kinds:
                - Pod/exec
      exclude:
        any:
        - clusterRoles:
          - cluster-admin
      preconditions:
        all:
          - key: "{{ request.operation || 'BACKGROUND' }}"
            operator: Equals
            value: CONNECT
          # Some namespaces are exempted from this rule
          - key: "{{ request.namespace }}"
            operator: AnyNotIn
            value:
              '{{ "exec-namespace-exceptions".data."exceptions" | parse_json(@) }}'
          # Allow rsync and tar
          - key: "{{ request.object.command[0] }}"
            operator: AllNotIn
            value:
              - rsync
              - tar
      validate:
        message:
          Users in this Namespace are not allowed to execute commands in pods that use GPUs.
          To request an exception, reach out to the admins.
        deny:
          conditions:
            all:
              # Deny GPU pods
              # We need to check the length of the array and ensure it's greater than 0
              # otherwise, the sum function might complain that it's expecting a non-empty array
              - key: "{{ length(target_pod_gpus_requested) }}"
                operator: GreaterThan
                value: 0
              - key: "{{ sum(target_pod_gpus_requested) }}"
                operator: GreaterThan
                value: 0
```

Policing

Handling interactive pods

It was too much!

- Blocking *oc cp* and *oc rsync*.
- Blocking *exec* on non-GPU pods.



```
apiVersion: kyverno.io/v1
kind: ClusterPolicy
metadata:
  name: exec-only-from-cluster-admins
spec:
  validationFailureAction: Enforce
  background: false
  rules:
    - name: exec-only-from-cluster-admins
      context:
        - name: exec-namespace-exceptions
          configMap:
            name: exec-namespace-exceptions
            namespace: kyverno-admin
        - name: target_pod_gpus_requested
          apiCall:
            urlPath: "/api/v1/namespaces/{{request.namespace}}/pods/{{request.name}}"
            jmesPath: 'spec.containers[].resources.requests."nvidia.com/gpu"'
            default: [0]
      match:
        any:
          - resources:
              kinds:
                - Pod/exec
      exclude:
        any:
        - clusterRoles:
```

```
rules:
  - name: exec-only-from-cluster-admins
    context:
      - name: exec-namespace-exceptions
        configMap:
          name: exec-namespace-exceptions
          namespace: kyverno-admin
      - name: target_pod_gpus_requested
        apiCall:
          urlPath: "/api/v1/namespaces/{{request.namespace}}/pods/{{request.name}}"
          jmesPath: 'spec.containers[].resources.requests."nvidia.com/gpu"'
          default: [0]
```

```
eration || 'BACKGROUND' }}"

 exempted from this rule
mespace }}"

ce-exceptions".data."exceptions" | parse_json(@) }}'

ject.command[0] }}"

ce are not allowed to execute commands in pods that use GPUs.
on, reach out to the admins.

k the length of the array and ensure it's greater than 0
sum function might complain that it's expecting a non-empty array
      - key: "{{ length(target_pod_gpus_requested) }}"
        operator: GreaterThan
        value: 0
      - key: "{{ sum(target_pod_gpus_requested) }}"
        operator: GreaterThan
        value: 0
```

Policing

Handling interactive pods

```yaml
exclude:
  any:
  - clusterRoles:
    - cluster-admin
preconditions:
  all:
    - key: "{{ request.operation || 'BACKGROUND' }}"
      operator: Equals
      value: CONNECT
    # Some namespaces are exempted from this rule
    - key: "{{ request.namespace }}"
      operator: AnyNotIn
      value:
        '{{ "exec-namespace-exceptions".data."exceptions" | parse_json(@) }}'
    # Allow rsync and tar
    - key: "{{ request.object.command[0] }}"
      operator: AllNotIn
      value:
        - rsync
        - tar
validate:
  message:
    Users in this Namespace are not allowed to execute commands in pods that use GPUs.
    To request an exception, reach out to the admins.
  deny:
    conditions:
      all:
        # Deny GPU pods
        # We need to check the length of the array and ensure it's greater than 0
        # otherwise, the sum function might complain that it's expecting a non-empty array
        - key: "{{ length(target_pod_gpus_requested) }}"
          operator: GreaterThan
          value: 0
        - key: "{{ sum(target_pod_gpus_requested) }}"
          operator: GreaterThan
          value: 0
```

```yaml
apiVersion: kyverno.io/v1
kind: ClusterPolicy
metadata:
  name: exec-only-from-cluster-admins
spec:
  validationFailureAction: Enforce
  background: false
  rules:
    - name: exec-only-from-cluster-admins
      context:
        - name: exec-namespace-exceptions
          configMap:
            name: exec-namespace-exceptions
            namespace: kyverno-admin
        - name: target_pod_gpus_requested
          apiCall:
            urlPath: "/api/v1/namespaces/{{request.namespace}}/pods/{{request.name}}"
            jmesPath: 'spec.containers[].resources.requests."nvidia.com/gpu"'
            default: [0]
      match:
        any:
          - resources:
              kinds:
                - Pod/exec
      exclude:
        any:
        - clusterRoles:
          - cluster-admin
      preconditions:
        all:
          - key: "{{ request.operation || 'BACKGROUND' }}"
            operator: Equals
            value: CONNECT
          # Some namespaces are exempted from this rule
          - key: "{{ request.namespace }}"
            operator: AnyNotIn
            value:
              '{{ "exec-namespace-exceptions".data."exceptions" | parse_json(@) }}'
          # Allow rsync and tar
          - key: "{{ request.object.command[0] }}"
            operator: AllNotIn
            value:
              - rsync
              - tar
      validate:
        message:
          Users in this Namespace are not allowed to execute commands in pods that use GPUs.
          To request an exception, reach out to the admins.
        deny:
          conditions:
            all:
              # Deny GPU pods
              # We need to check the length of the array and ensure it's greater than 0
              # otherwise, the sum function might complain that it's expecting a non-empty array
              - key: "{{ length(target_pod_gpus_requested) }}"
                operator: GreaterThan
                value: 0
              - key: "{{ sum(target_pod_gpus_requested) }}"
                operator: GreaterThan
                value: 0
```

Policing

A small bonus

Knowing who created a certain object can be very helpful.

```yaml
apiVersion: v1
data:
  .dockerconfigjson: data
kind: Secret
metadata:
  annotations:
    owner: Alessandro.Pomponio1-at-ibm.com
  creationTimestamp: "2025-03-21T09:21:02Z"
  name: name
  namespace: namespace
  resourceVersion: "619018630"
  uid: uid
type: kubernetes.io/dockerconfigjson
```

```yaml
apiVersion: kyverno.io/v1
kind: ClusterPolicy
metadata:
  name: add-owner-to-resources
spec:
  background: false
  rules:
    - name: add-owner-to-resources
      match:
        any:
          - resources:
              kinds:
                - Job
                - Deployment
                - DeploymentConfig
                - StatefulSet
                - DaemonSet
                - ReplicaSet
                - Pod
                - Secret
                - ConfigMap
                - PersistentVolumeClaim

      preconditions:
        all:
          - key: "{{ request.userInfo.username || '' }}"
            operator: NotEquals
            value: ""
      mutate:
        patchStrategicMerge:
          metadata:
            annotations:
              owner:
                "{{ replace_all('{{ request.userInfo.username }}', '@', '-at-') }}"

```

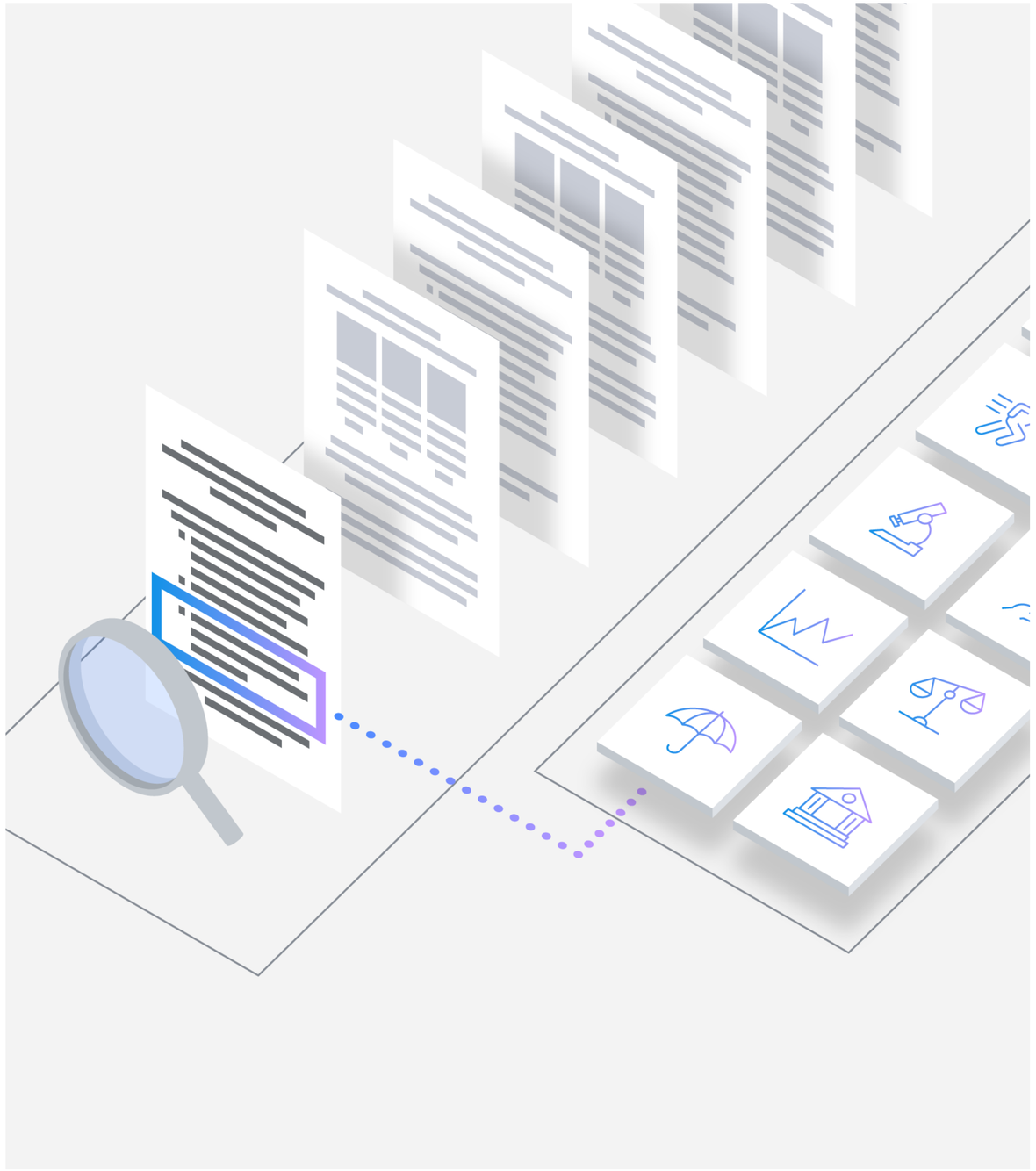
Policing

Results

- ***GPU Jobs have exclusive access to GPU nodes.*** Affinity rules are auto-applied instead of relying on users remembering to add them.

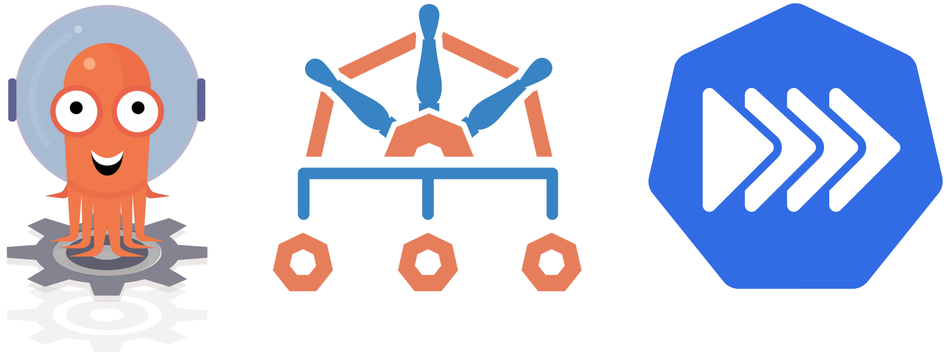
- ***Interactive GPU pods are mostly a thing of the past***. Working with our colleagues we have identified a few workloads that *really* need this interactivity, but it's now off by default. Other, non-GPU workloads are unaffected, as well as file copying mechanisms.

- ***Users are largely unaware of the policies***. Governance is transparent and does not hinder their workloads.



Addressing the challenges

What’s the plan?



✅ Reduce the need for admin intervention

Introduce automation and tooling that enables self-service namespace management even for non-experts users.

✅ Prevent CPU jobs from running amok

Introduce safeguards to prevent CPU-only workloads from overwhelming the cluster or interfering with GPU scheduling.

✅ Discourage interactive GPU usage

Prevent the creation of long-lived, interactively accessed pods in favour of more efficient, Job-based workflows that free up (GPU) resources once they’re complete.

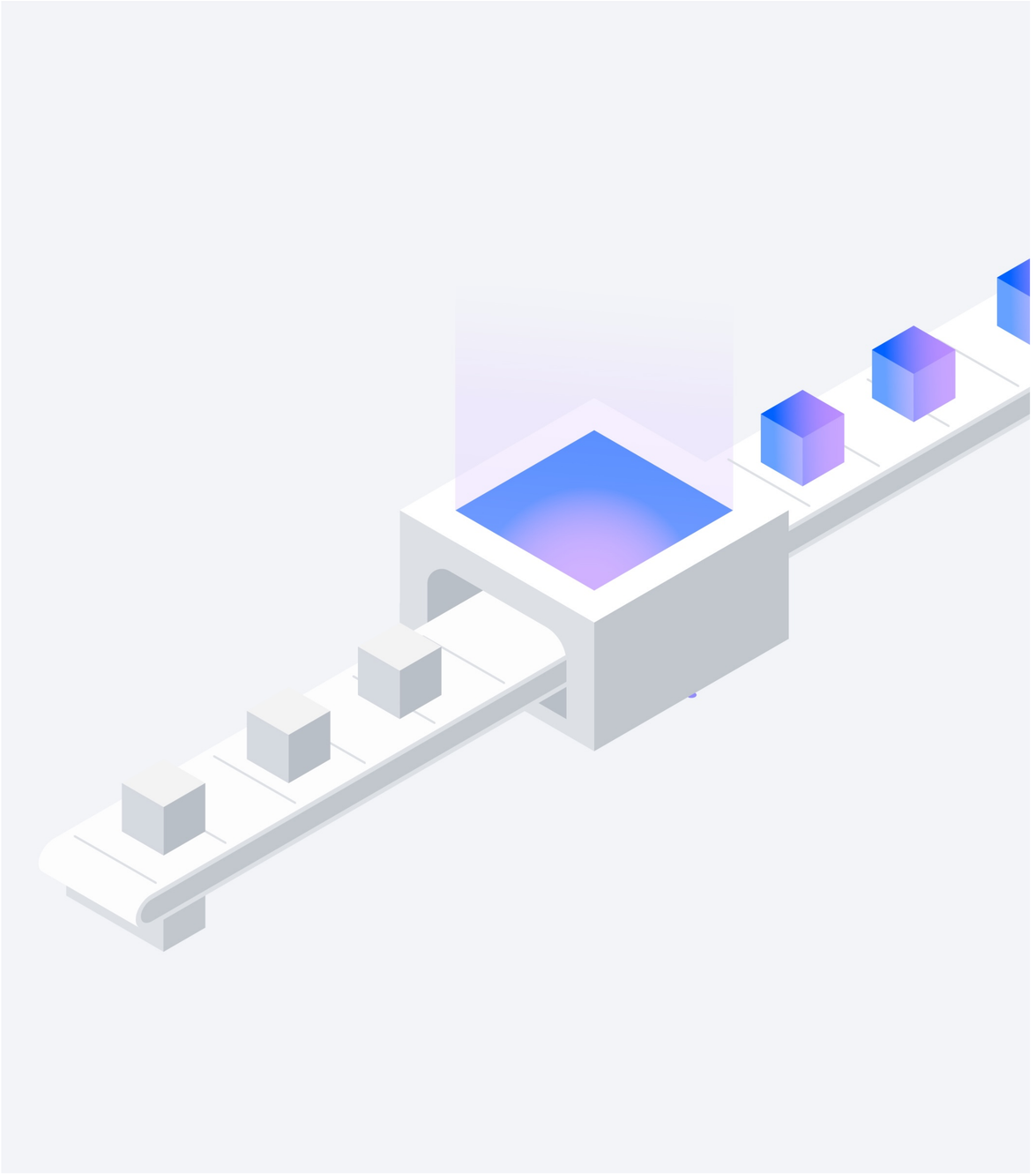
Make GPU access fair

Improve resource allocation to ensure GPU access is fair, providing a more predictable, HPC-like experience that minimises contention.

Queueing

Our goals

- ***Support HPC-like queueing semantics***. Workloads are often not time-sensitive and researchers just *eventually* want the results.
- ***Support specific GPU requests***. Workloads often have specific hardware requirements (e.g., at least 80GB of VRAM).
- ***Support generic GPU requests***. When a generic GPU is sufficient, queue times will be lower.



Queueing

Accessing specific GPUs

Our clusters have multiple GPU types we have acquired over the years:

- T4s
- V100s
- A100s 40GB
- A100s 80GB
- H100s 80GB

```yaml
apiVersion: kueue.x-k8s.io/v1beta1
kind: ResourceFlavor
metadata:
  name: "nvidia-h100-80gb-pcie"
spec:
  nodeLabels:
    nvidia.com/gpu.product: NVIDIA-H100-PCIe
---
apiVersion: kueue.x-k8s.io/v1beta1
kind: ClusterQueue
metadata:
  name: "h100-cluster-queue"
spec:
  cohort: gpu-cohort
  namespaceSelector:
    matchLabels:
      kueue-enable: gpu-cluster-queue
  resourceGroups:
    - coveredResources: ["cpu", "memory", "nvidia.com/gpu"]
      flavors:
        - name: "nvidia-h100-80gb-pcie"
          resources:
            - name: "cpu"
              nominalQuota: 160
            - name: "memory"
              nominalQuota: 400Gi
            - name: "nvidia.com/gpu"
              nominalQuota: 4
```

Queueing

Accessing specific GPUs

Our clusters have multiple GPU types we have acquired over the years:

- T4s
- V100s
- A100s 40GB
- A100s 80GB
- H100s 80GB

```yaml
apiVersion: kueue.x-k8s.io/v1beta1
kind: ResourceFlavor
metadata:
  name: "nvidia-h100-80gb-pcie"
spec:
  nodeLabels:
    nvidia.com/gpu.product: NVIDIA-H100-PCIe
---
apiVersion: kueue.x-k8s.io/v1beta1
kind: ClusterQueue
metadata:
  name: "h100-cluster-queue"
spec:
  cohort: gpu-cohort
  namespaceSelector:
    matchLabels:
      kueue-enable: gpu-cluster-queue
  resourceGroups:
    - coveredResources: ["cpu", "memory", "nvidia.com/gpu"]
      flavors:
        - name: "nvidia-h100-80gb-pcie"
          resources:
            - name: "cpu"
              nominalQuota: 160
            - name: "memory"
              nominalQuota: 400Gi
            - name: "nvidia.com/gpu"
              nominalQuota: 4
```

Queueing

Accessing specific GPUs

Our clusters have multiple GPU types we have acquired over the years:

- T4s
- V100s
- A100s 40GB
- A100s 80GB
- H100s 80GB

```yaml
apiVersion: kueue.x-k8s.io/v1beta1
kind: ResourceFlavor
metadata:
  name: "nvidia-h100-80gb-pcie"
spec:
  nodeLabels:
    nvidia.com/gpu.product: NVIDIA-H100-PCIe
---
apiVersion: kueue.x-k8s.io/v1beta1
kind: ClusterQueue
metadata:
  name: "h100-cluster-queue"
spec:
  cohort: gpu-cohort
  namespaceSelector:
    matchLabels:
      kueue-enable: gpu-cluster-queue
  resourceGroups:
    - coveredResources: ["cpu", "memory", "nvidia.com/gpu"]
      flavors:
        - name: "nvidia-h100-80gb-pcie"
          resources:
            - name: "cpu"
              nominalQuota: 160
            - name: "memory"
              nominalQuota: 400Gi
            - name: "nvidia.com/gpu"
              nominalQuota: 4
```

Queueing

Accessing a generic GPU

Our clusters have multiple GPU types we have acquired over the years:

- T4s
- V100s
- A100s 40GB
- A100s 80GB
- H100s 80GB

```yaml
apiVersion: kueue.x-k8s.io/v1beta1
kind: ClusterQueue
metadata:
  name: "gpu-cluster-queue"
spec:
  cohort: gpu-cohort
  namespaceSelector:
    matchLabels:
      kueue-enable: gpu-cluster-queue
  resourceGroups:
    - coveredResources: ["cpu", "memory", "nvidia.com/gpu"]
      flavors:
        - name: "nvidia-a100-80gb-pcie"
          resources:
            - name: "cpu"
              nominalQuota: 0
            - name: "memory"
              nominalQuota: 0Gi
            - name: "nvidia.com/gpu"
              nominalQuota: 0
        - name: "nvidia-h100-80gb-pcie"
          resources:
            - name: "cpu"
              nominalQuota: 0
            - name: "memory"
              nominalQuota: 0Gi
            - name: "nvidia.com/gpu"
              nominalQuota: 0
```

Queueing

Accessing a generic GPU

Our clusters have multiple GPU types we have acquired over the years:

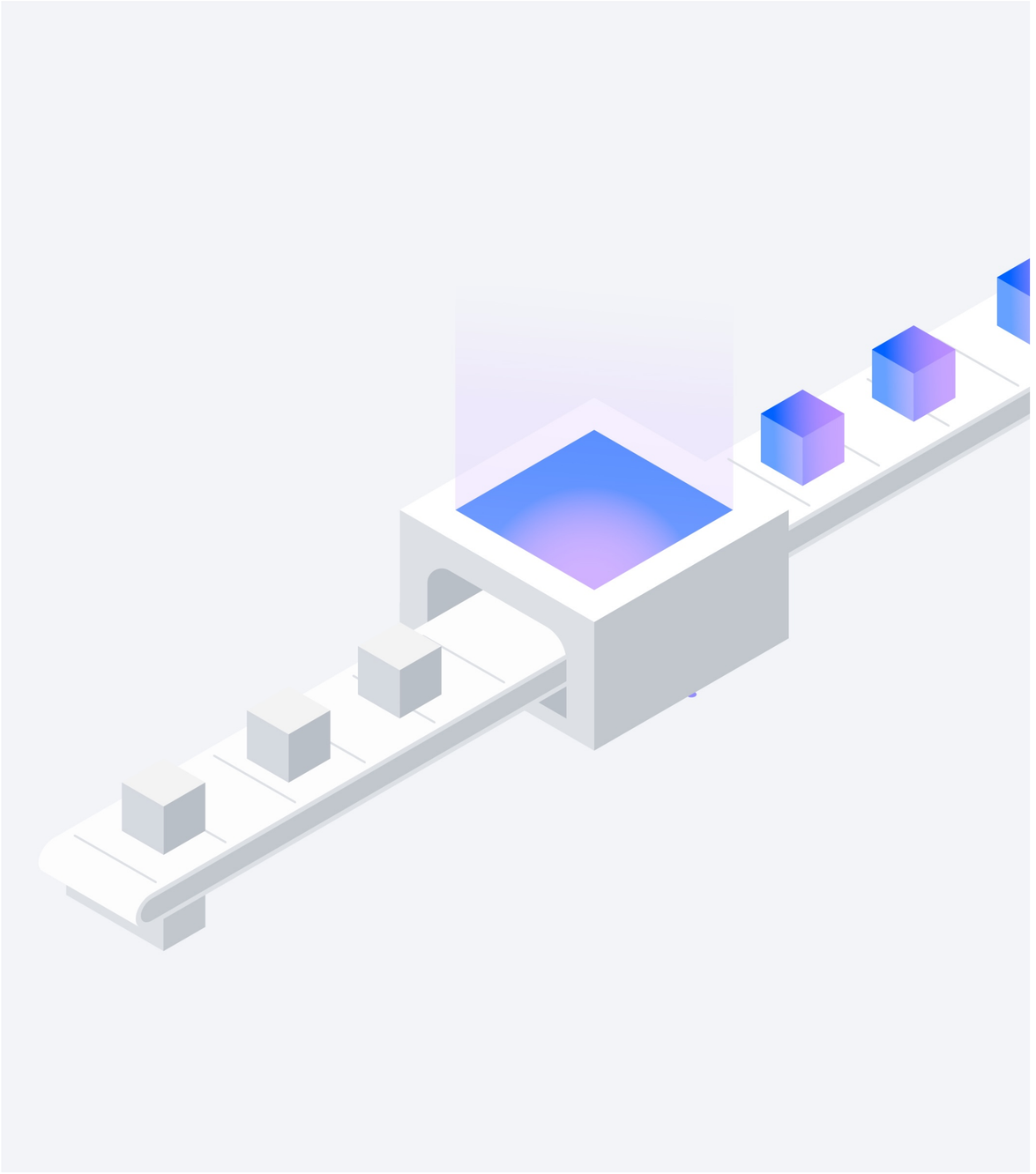
- T4s
- V100s
- A100s 40GB
- A100s 80GB
- H100s 80GB

```yaml
apiVersion: kueue.x-k8s.io/v1beta1
kind: ClusterQueue
metadata:
  name: "gpu-cluster-queue"
spec:
  cohort: gpu-cohort
  namespaceSelector:
    matchLabels:
      kueue-enable: gpu-cluster-queue
  resourceGroups:
    - coveredResources: ["cpu", "memory", "nvidia.com/gpu"]
      flavors:
        - name: "nvidia-a100-80gb-pcie"
          resources:
            - name: "cpu"
              nominalQuota: 0
            - name: "memory"
              nominalQuota: 0Gi
            - name: "nvidia.com/gpu"
              nominalQuota: 0
        - name: "nvidia-h100-80gb-pcie"
          resources:
            - name: "cpu"
              nominalQuota: 0
            - name: "memory"
              nominalQuota: 0Gi
            - name: "nvidia.com/gpu"
              nominalQuota: 0
```

Queueing

Results

- ***Resource access is now fair***. Workloads are enqueued and eventually execute.
- ***Specific and generic queues are provided***. Users can choose to queue for specific GPUs or for a generic one.



“I’m hungry and I dozed off for the past 25 minutes. TL;DR?”

Audience member

To recap...

Admins have less headaches:

- User support queries have basically been zeroed.
- Deployments are not manual anymore. A GitOps approach guarantees visibility, audibility, and compliance.
- Policies automatically solve misconfigurations and prevent problematic behaviour.

Users are happier:

- Namespace-level changes are much quicker.
- Policies greatly reduced resource contention on the clusters.
- GPUs are accessed more fairly.

Thank you