Kubeflow: Multi-Tenant, Self-Serve, ML Platform

kunming@google.com
kam.d.kasravi@intel.com

Enabling the end user

- **Multi-tenant self-serve workspaces for developers and data scientists**
- **Do not saddle the end user with k8s details**
- **Deploy job to the right CPU/Accelerated hardware**
- **Kustomize overlays for different cpu/accelerated hardware combinations**

Kubeflow: A platform for building ML products

- **Leverage containers and Kubernetes to solve the challenges of building ML products**
  - Reduce the time and effort to get models launched
- **Why Kubernetes**
  - Kubernetes has won
  - Kubernetes runs everywhere
  - Enterprises can adopt shared infrastructure and patterns for ML and non ML services
  - Knowledge transfer across the organization
- Kubeflow is open
  - No lock in
  - 200 Members
  - 20+ Organizations
  - Stats available @ http://devstats.kubeflow.org

Kubeflow Cloud Providers

- Google Kubernetes Engine
- AWS
- Azure

Kubeflow Native K8

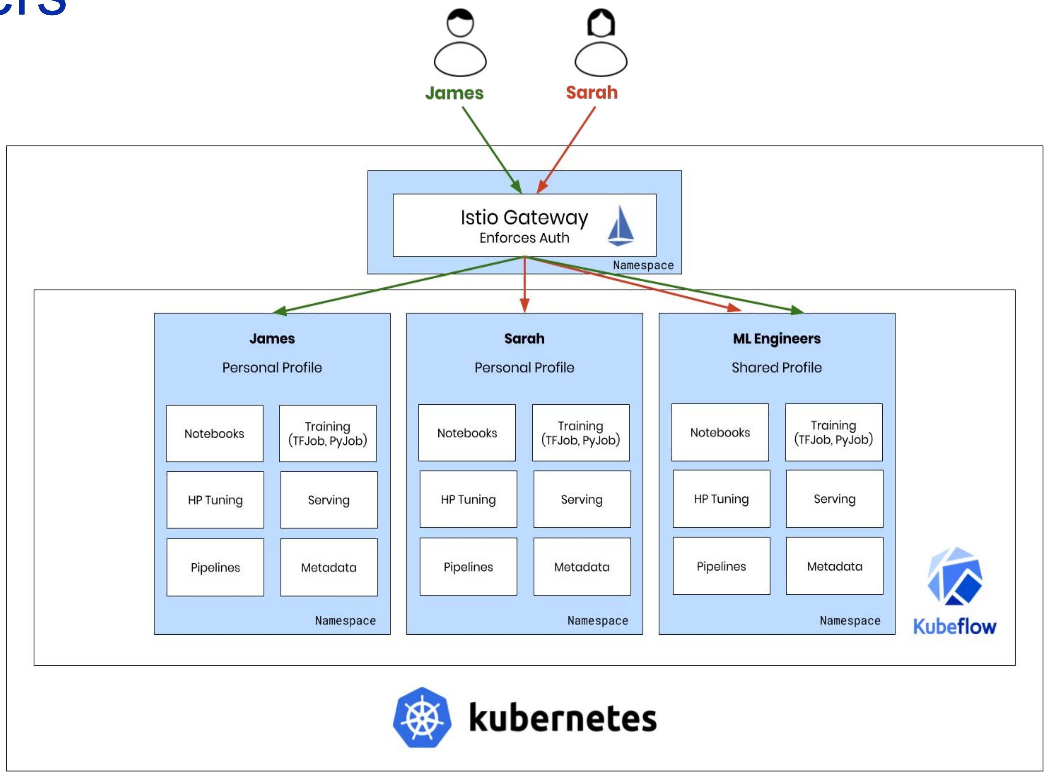
- Deployable to any k8 existing cluster

ML Applications

- Goal: applications for every stage of ML
- Examples:
  - Experimentation / Data Exploration
    - Jupyter / Notebook Spawner
  - Training
    - Tensorflow & Pytorch distributed training managed through K8s CRDs
    - Katib - HP Tuning
  - Workflows:
    - Pipelines
  - Metadata
    - Tracking and managing metadata of ML workflows
  - Feature Store
    - Feast (from GOJEK)

Multi Tenancy for End Users

Users will operate on same k8s cluster while each user has their own workspace hosting their services. Workspaces are logically isolated: each user can only access services to their own workspace.



K8s Multi-Tenancy Challenges

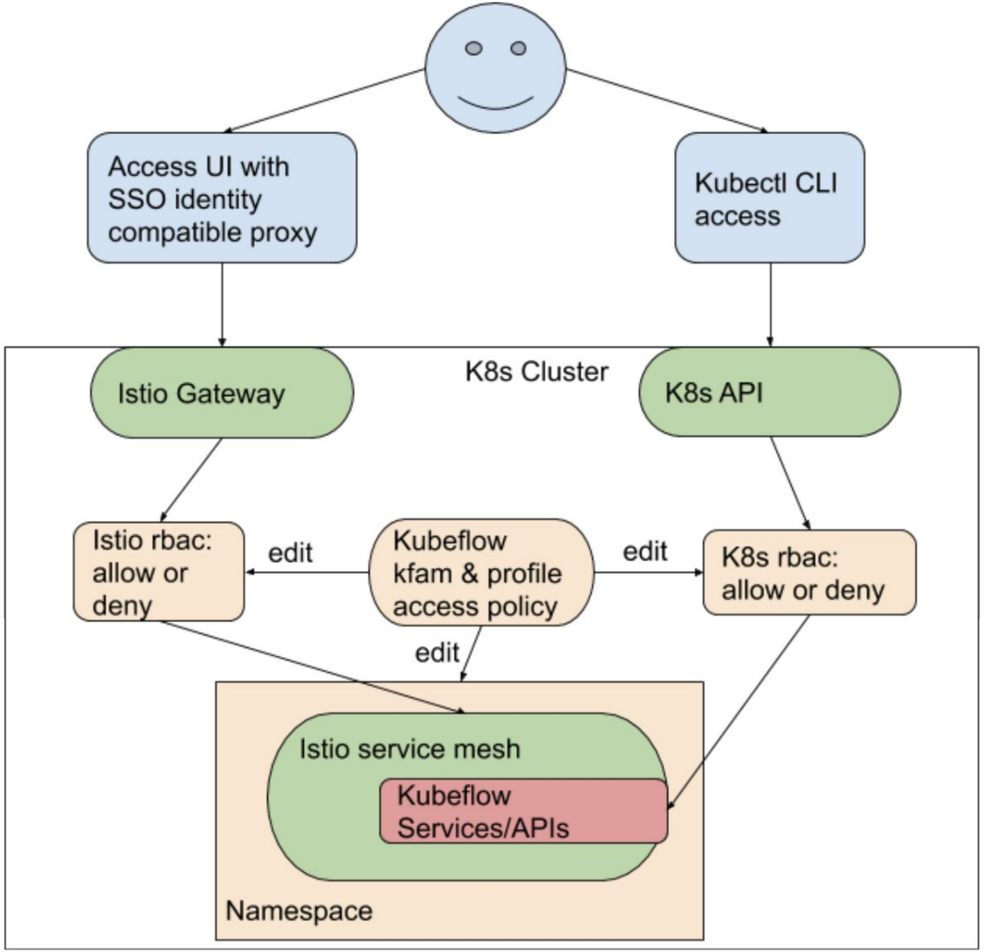
- Define clear user workspace boundary for access isolation
- K8s in-cluster network is transparent
  - Services are by default visible from all pods
  - Need to establish network access control
- Access control around traffic through shared ingress
  - Users might access services in their own workspaces through same ingress.
  - Need to establish access control behind ingress: user can only access workspace after permission check
- Workspace access sharing & revoke
  - Each workspace owner should be able to share/revoke workspace access
  - Access sharing should not leak owner privilege while allow invited user operating on CRs
- All policies, roles and bindings involved should behave in consistency.

Kubeflow Multi-Tenancy (Profiles)

- Define user workspace as namespace and build access control around it
  - Manage user access to namespace through k8s rbac policy.
- Leverage Istio to control in-cluster traffic
  - By default requests to user workspaces are denied unless allowed by Istio Rbac
- Leverage Identity-Aware Proxy and Istio to control traffic through ingress
  - Identity user request through Identity-Aware Proxy.
  - Istio then do rbac check on request target workspace and identity
- Enable workspace access sharing & revoke
  - Workspace owners can share/revoke workspace access with other users through kubeflow UI
  - Invited users will have k8s edit permission plus permission to operate kubeflow CRs
- Self-serve
  - New user can self-register to create and own their workspace through kubeflow UI
- Kubeflow Profile CR to control all policies, roles and bindings involved and guarantee consistency.
  - Offer plugin interface to manage external resource/policy outside k8s, eg. access control of public cloud APIs

Kubeflow Access Control

- User access through kubectl: controlled by k8s rbac policy.
- User access through browser: controlled by istio rbac policy.
- Kubeflow multi-tenancy is implemented k8s-native way, new services can be integrated easily.



Kubeflow Profile

Created by the user via cli: *kubectl apply -f myprofile.yaml* or *kubeflow UI*

```
apiVersion: kubeflow.org/v1alpha1
kind: Profile
metadata:
  name: $(name)
spec:
  owner: $(owner)
```

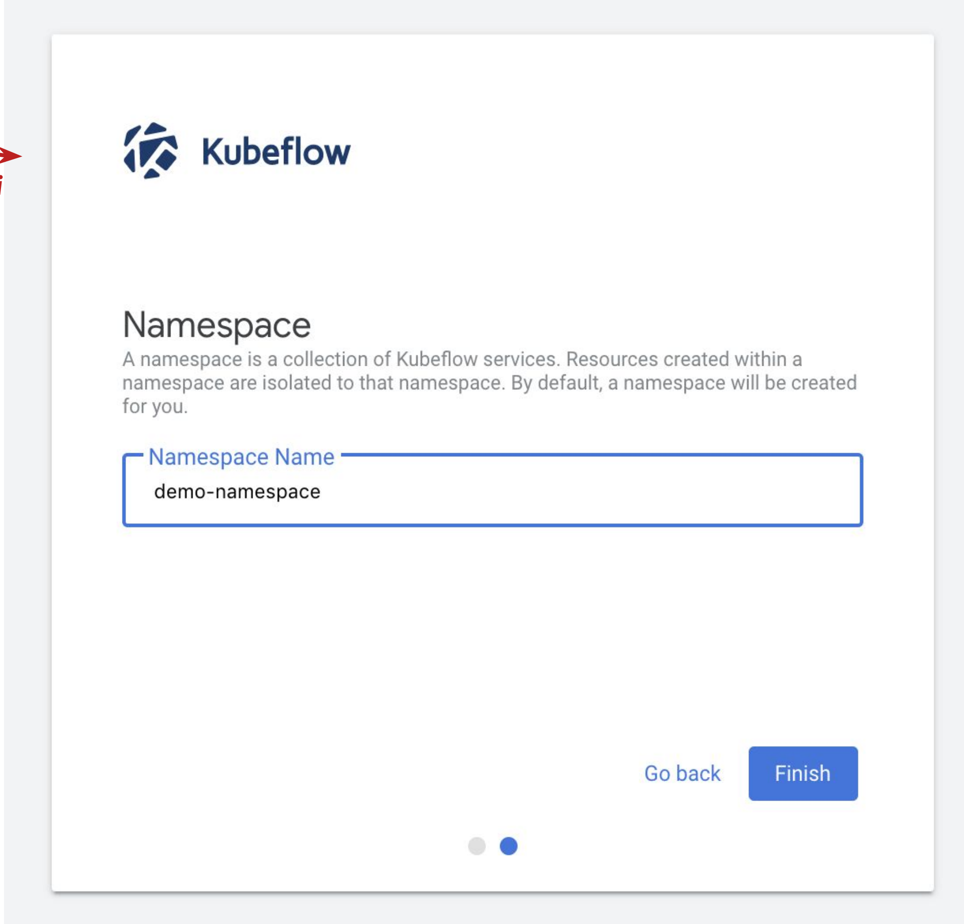
Data scientists use Profiles to create various types of workspaces, where they can run training, inference, etc.

Create Kubeflow Profile

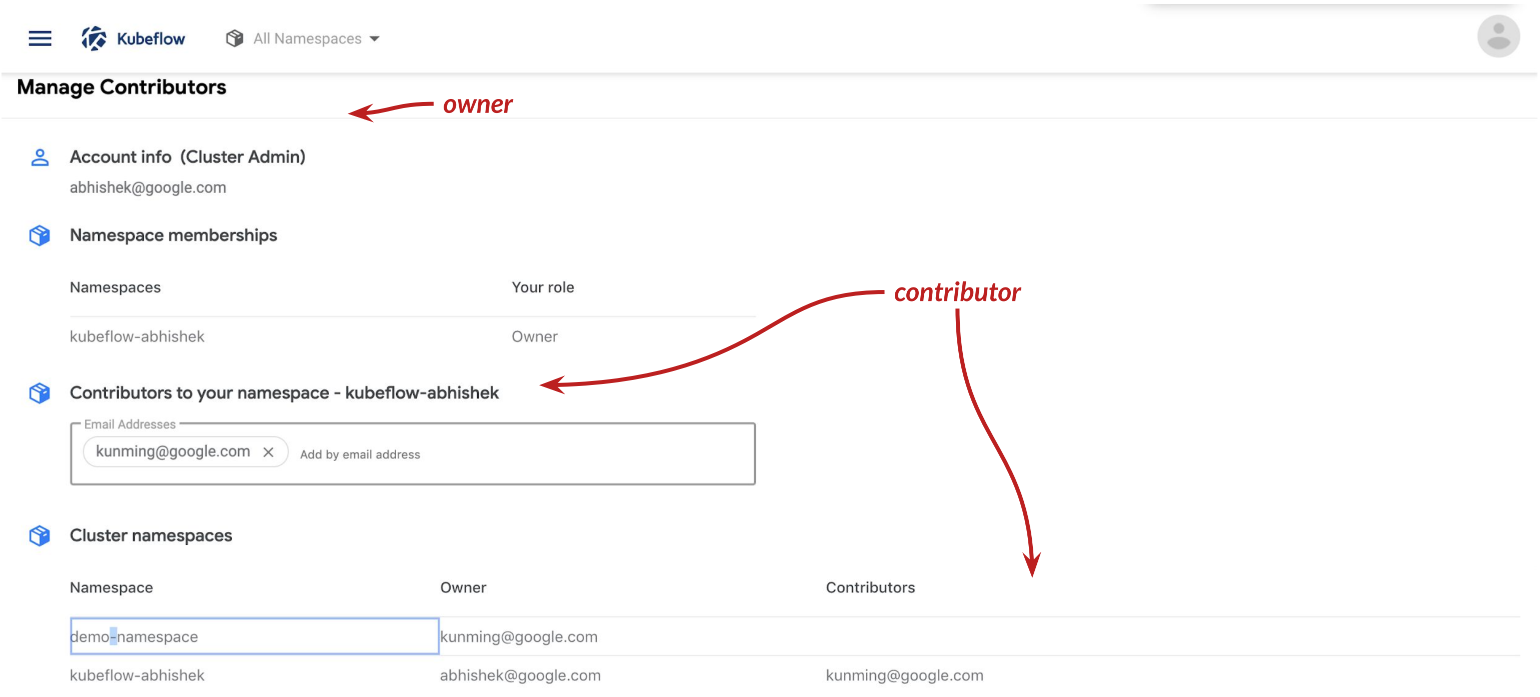
yaml

```yaml
apiVersion: kubeflow.org/v1beta1
kind: Profile
metadata:
  name: demo-namespace   # profile name is also namespace name
spec:
  owner:
     kind: User
     name: user1@email.com   # replace with the email of the user
  plugins:
  - kind: WorkloadIdentity
    spec:
       gcpServiceAccount: user1-gcp@project-id.iam.gserviceaccount.com
```

ui



Share Kubeflow Profile with other users



Kubeflow Device Overlays (accelerators, cpus)

- Device Overlays into Profiles, Pods
- Uses Profile extensions, Tekton Pipelines
- Can also be applied to Argo workflows and other pipeline engines

Kubeflow Profile adds cpu/accelerator quotas

Has a quotas section that is added to the namespace

```
apiVersion: kubeflow.org/v1alpha1
kind: Profile
metadata:
  name: $(name)
spec:
  owner: $(owner)
  quota:
    hard:
      requestsCpu: $(requestsCpu)
      requestsMemory: $(requestsMemory)
      requestsGpu: $(requestsGpu)
      limitsCpu: $(limitsCpu)
      limitsMemory: $(limitsMemory)
      <vendor/device>: <value>
```

This could be added by an admission-controller or gitops

Tekton Pipeline to run a model

kfctl can deploy manifest files from different repos or on disk

```
tk/tk-pipeline-run/overlays/cpu-node-selector
├── kustomization.yaml
├── params.env
├── params.yaml
└── pipeline-run.yaml
```

Tekton Pipeline to run a model

Adds a podTemplate.nodeSelector.cpu value

tk/tk-pipeline-run/overlays/cpu-node-selector/pipeline-run.yaml

```yaml
apiVersion: tekton.dev/v1alpha1
kind: PipelineRun
metadata:
  name: $(generateName)
spec:
  podTemplate:
    nodeSelector:
      cpu: "$(cpuType)"
```

Tekton Pipeline to run a model

kfctl can deploy manifest files from different repos or on disk

tk/tk-pipeline/overlays/run-model-task

```
├── config-map.yaml
├── kustomization.yaml
├── params.env
├── params.yaml
├── pipeline_patch.yaml
└── task.yaml
```

tk/tk-pipeline/overlays/run-model-task-nvidia

```
├── kustomization.yaml
├── params.env
├── params.yaml
└── task_patch.yaml
```

Tekton Pipeline to run a model

run-model example

tk/tk-pipeline-run/overlays/run-model-task/task.yaml

```
apiVersion: tekton.dev/v1alpha1
kind: Task
metadata:
  name: run-model
spec:
  inputs:
    params:
    - name: imageName
      type: string
  steps:
  - name: run-model
    image: $(inputs.params.imageName)
    command: ["/bin/bash", "/run-model/run-model.sh"]
```

Tekton Pipeline to run a model

task is patched to add gpu info

tk/tk-pipeline-run/overlays/run-model-task-nvidia/task_patch.yaml

```yaml
- op: add
  path: /spec/steps/0/resources
  value:
    limits:
      nvidia.com/gpu: $(accelerator_count)
```

Tekton Pipeline to run a model

config file to run a model on a node with cpu=skylake, nvidia.com/gpu

kfctl apply -f kfdef/run-model-gpu.yaml

```
- kustomizeConfig:
    overlays:
    - run-model-task
    - run-model-task-nvidia
    parameters:
    - name: accelerator_count
      value: 1
    repoRef:
      name: manifests
      path: tk/tk-pipeline
  name: tk-pipeline
```

```
- kustomizeConfig:
    overlays:
    - application
    - cpu-node-selector
    parameters:
    - name: cpuType
      value: skylake
    repoRef:
      name: manifests
      path: tk/tk-pipeline-run
  name: tk-pipeline-run
```

DEMO

- A kubeflow deployment that created a GKE cluster with 2 nodes

CPU Platform	Accelerator Type	Machine Type	Image Type
Intel Skylake	gpu nvidia-tesla-t4	n1-standard-8	cos
Intel Cascade Lake	-	c2-standard-8	ubuntu

- Run the same tensorflow model within a Profile but with different overlays

Pod limits selects the accelerator type (nvidia.com/gpu: ‘1’)
Pod affinity selects the cpu platform (cpu: cascadelake)

Thank You

Kubeflow

- Kubeflow website - https://www.kubeflow.org/
- Code - https://github.com/kubeflow/kubeflow