Outline

- Objectives and user stories
- An interTwin k8s virtual node
- State of the art
- First pilot experiences
- Current status and plans

interTwin

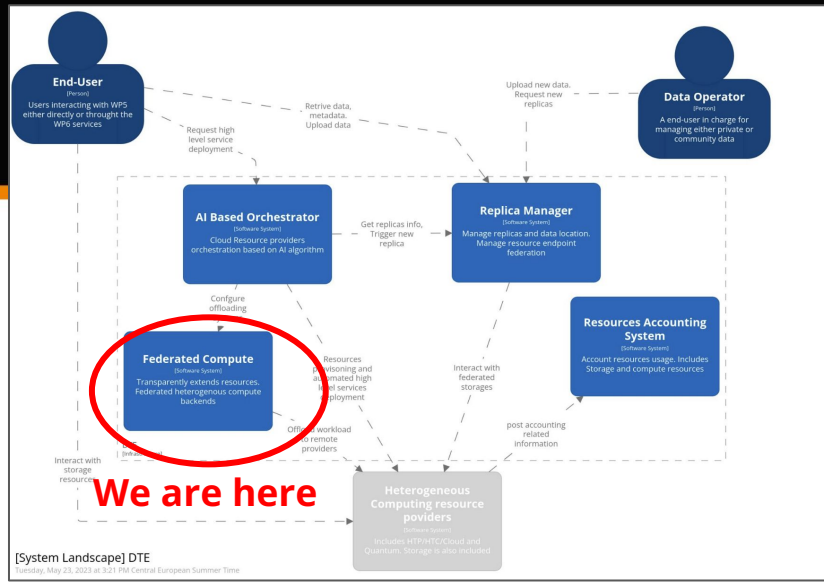
Enabling transparent access to heterogeneous computing providers via interLink

Diego Ciangottini - INFN
On behalf of the interTwin WP5 group

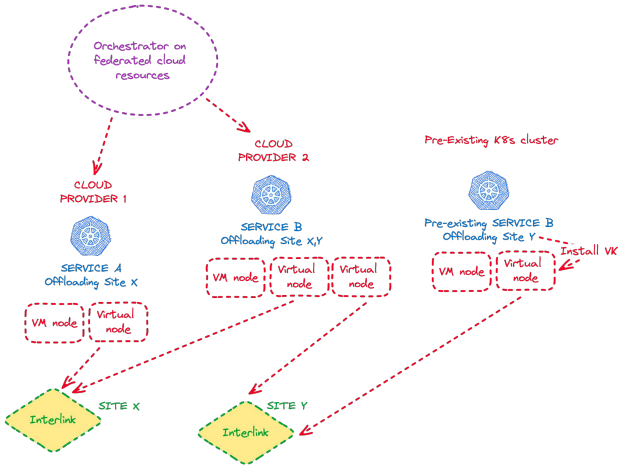
interTwin is funded by Horizon Europe under grant agreement n° 101058386

Objectives

- Federate a set of highly heterogeneous and disparate providers
  - enabling a "transparent payload offloading"
- Flexible extension of existing computing clusters to remote resources
  - E.g. HPC, specialized hardware…



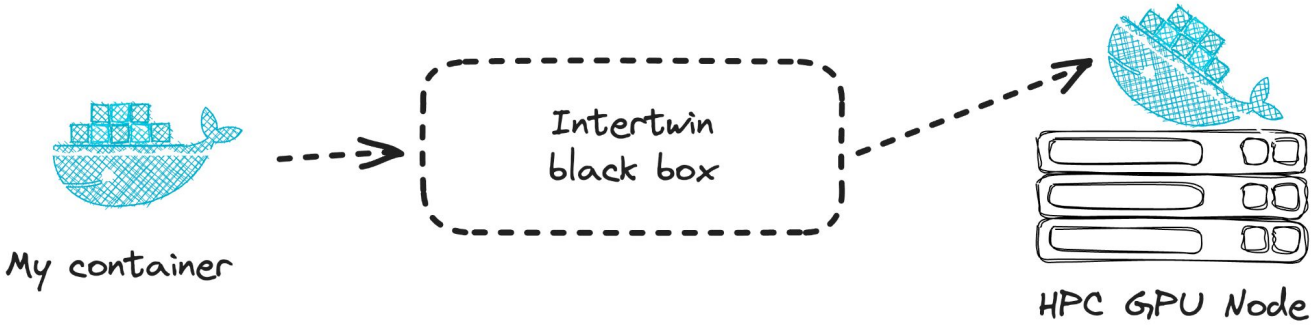
- Terminology
  - Containers as the unit of compute
    - Meaning starting from a strawman model where all the payloads are containerized
  - Offloading:
    - A seed of resources available on cloud providers, the heavylift is done "transparently" delegating (offloading) the execution of the containers on hpc/remote resources



User stories: Scheduling K8s POD on HPC

This is the most generic one:

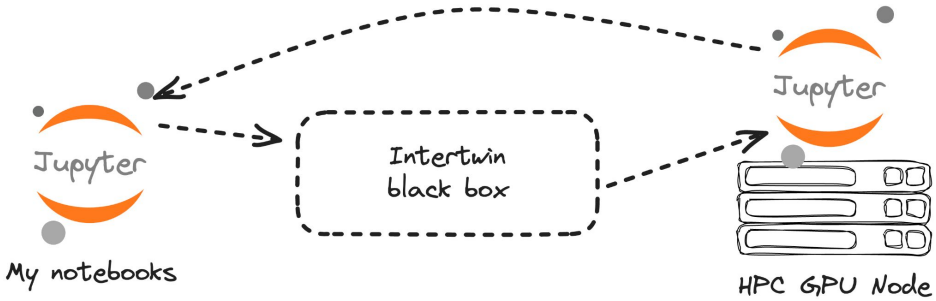
- ***I have a container and I want it executed on a node with N GPUs and M GB of RAM***
- I want to create a simple container to be scheduled on a remote Slurm batch on an HPC center



User story: Jupyter notebooks

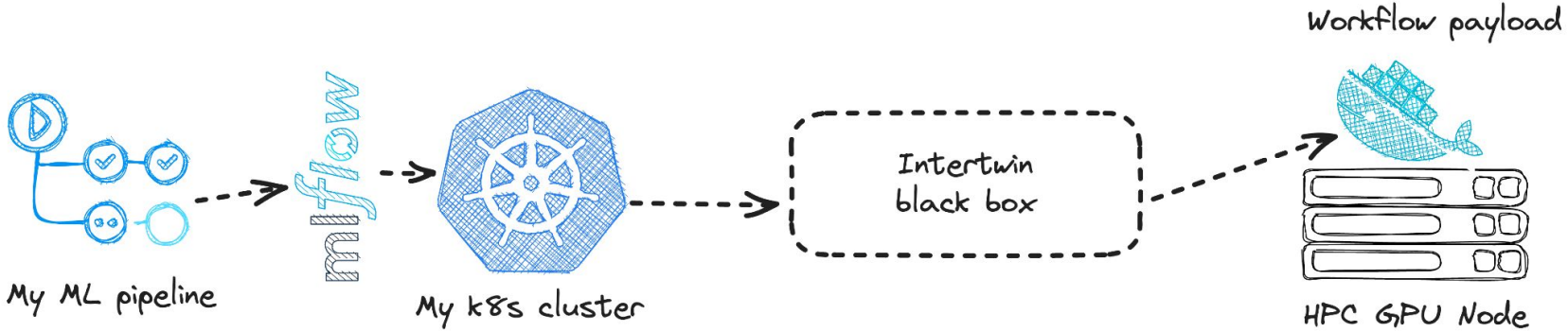
For development of ML pipelines, Jupyter notebooks are a common solutions

- ***I want access to a hub where I can ask for the spawning of a JupyterLab instance on a node with GPUs***
- **In other words I want to schedule a JupyterLab container on a remote slurm cluster**
  - **Similar to the previous one, with the exception of bidirectional connection needed b/w Lab and HUB that is a bit tricky**



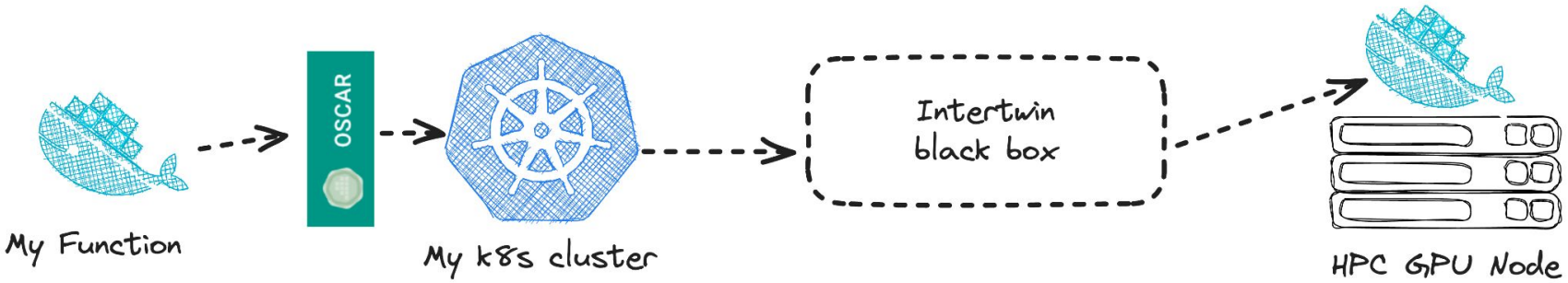
User story: pipelines/workflows

- Frameworks for DAG/workflow managements are usually well integrated with K8s APIs
  - Airflow/Kubeflow pipelines/Argo wokflows/MLFlow are no exceptions
- Pilot exploitation possible thanks to interTwin AI Workflow management activities
  - CERN group in particular has technically produced a demo container image for this



User story: Serverless

- Executing payload in response of an external trigger
  - Being it a storage event or a web server call
- Pilot exploitation with dCNiOS (dCache + Nifi + OSCAR) devs for a first round of validation with payload executed on a virtual node



Building an “offloading” prototype

- Extend the container orchestration de-facto standard (K8s) to support offloading under the hood
  - With little to no knowledge required from the end user perspective
- What about making a Kubernetes cluster send your containers/pod to a “virtual” node that seamlessly takes care of your application lifecycle on a remote server/hpc batch queue?
  - While exposing the very SAME experience of running a pod on the cloud resources
    - Meaning that the API layer exposed is the “normal” set of K8s apis

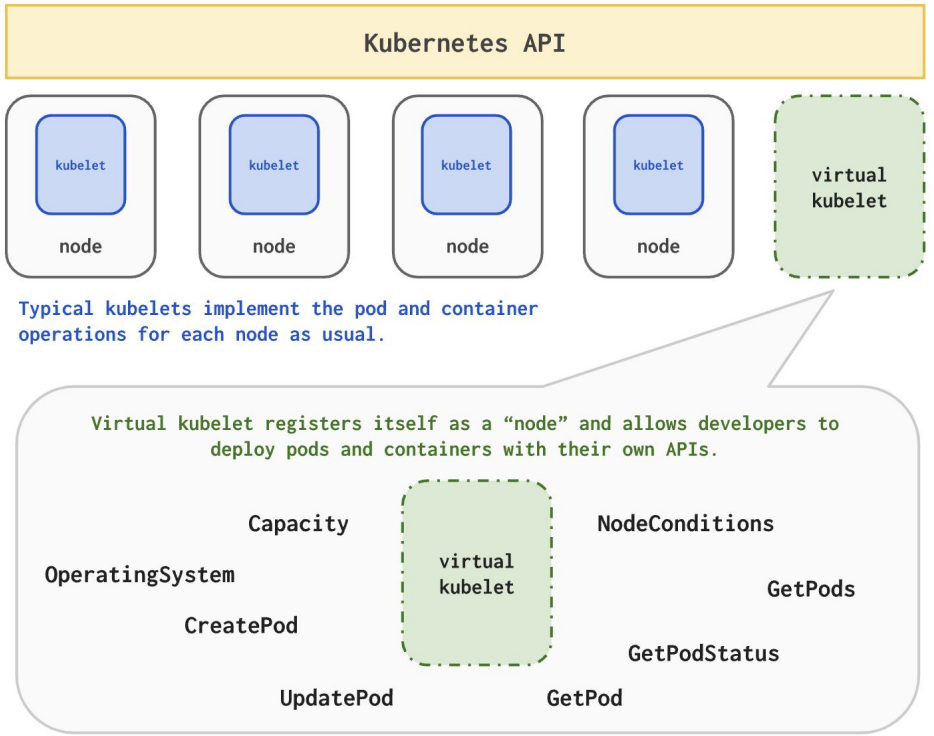
N.B. We aim to use Kubernetes as the workhorse for the “offloading”, NOT as the user interface though

The enabling technology

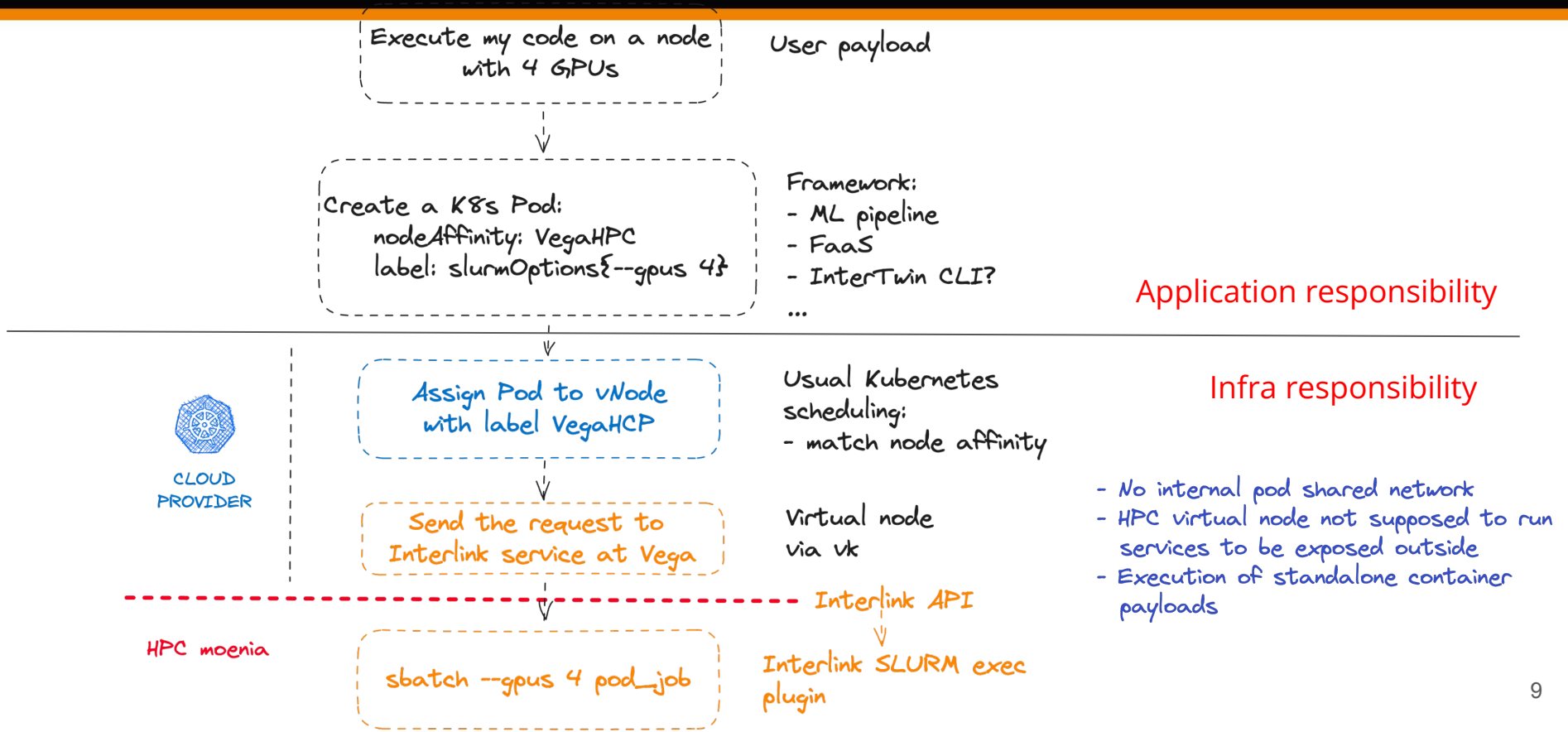
Virtual kubelet (VK):

"Open-source Kubernetes kubelet implementation that masquerades as a kubelet. This allows Kubernetes nodes to be backed by Virtual Kubelet providers"

Knoc project has been the trigger of our interest.



All in all the flow we want to achieve

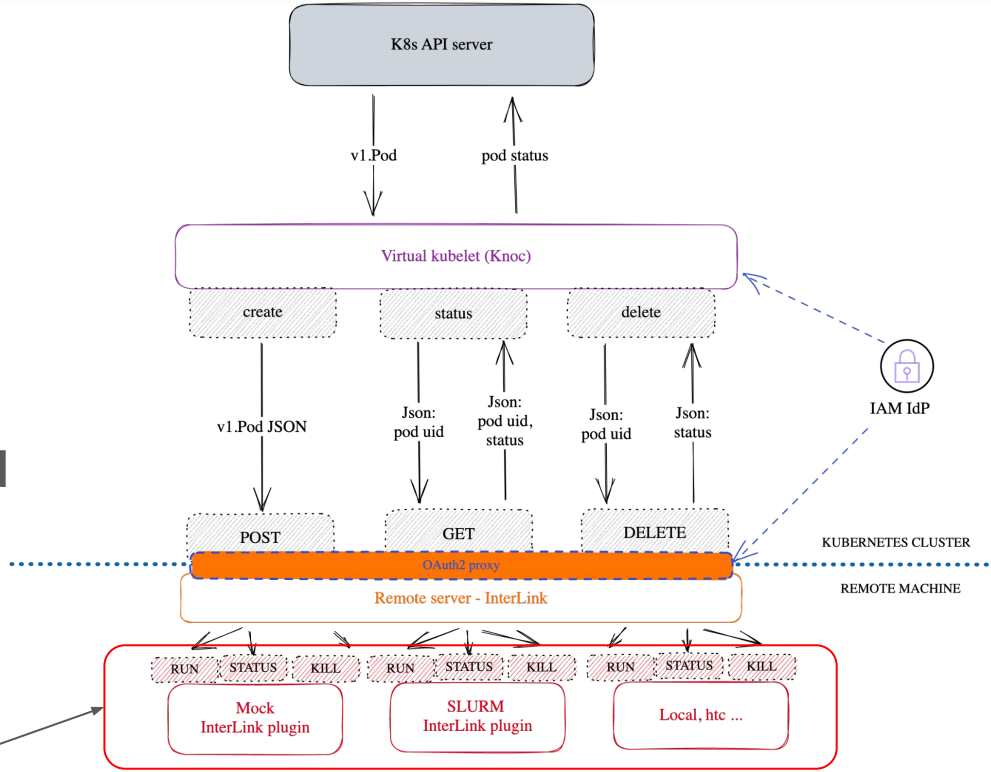


Currently working on

interLink

We extended the VK solutions with a first draft of a **generic API layer for delegating pod execution on ANY remote backend**

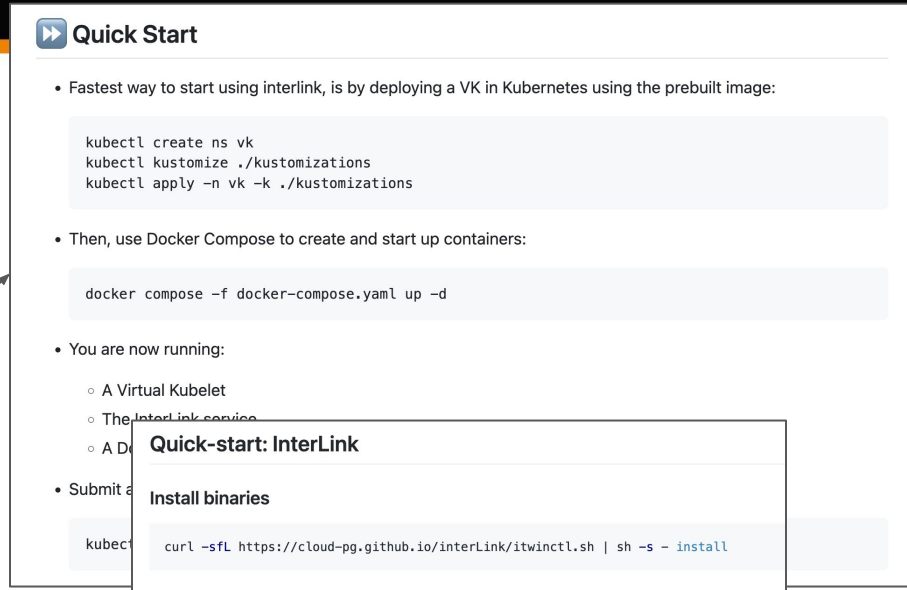
k8s POD requests are digested through the API layer (e.g. deployed on an HPC edge) **into batch job execution of a container**.



Building the first prototype

- We have a first implementation ready for a first round of validation
  - A standalone “remote docker run” implementation for testing and development
  - A SLURM quick start installation of interLink components
    - Validated already on different pilot systems (see later)

A development guide for creating custom plugin (like the slurm one) is available.



Quick Start

- Fastest way to start using interlink, is by deploying a VK in Kubernetes using the prebuilt image:

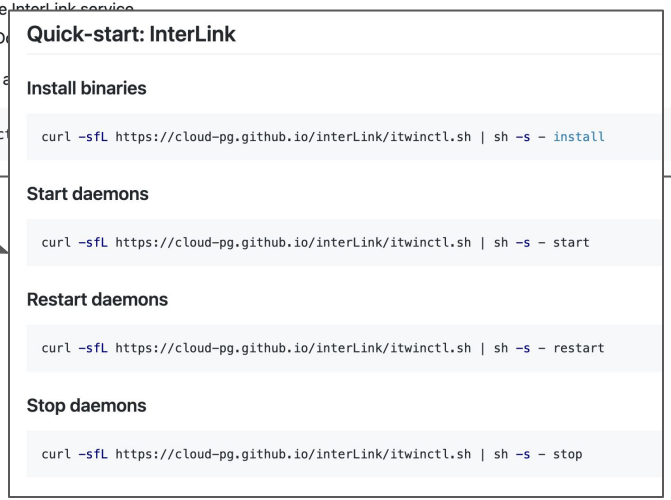
```
kubectl create ns vk
kubectl kustomize ./kustomizations
kubectl apply -n vk -k ./kustomizations
```

- Then, use Docker Compose to create and start up containers:

```
docker compose -f docker-compose.yaml up -d
```

- You are now running:
  - A Virtual Kubelet
  - The InterLink service
  - A D
- Submit a

kubect



Quick-start: InterLink

Install binaries

```
curl -sfL https://cloud-pg.github.io/interLink/itwinctl.sh | sh -s - install
```

Start daemons

```
curl -sfL https://cloud-pg.github.io/interLink/itwinctl.sh | sh -s - start
```

Restart daemons

```
curl -sfL https://cloud-pg.github.io/interLink/itwinctl.sh | sh -s - restart
```

Stop daemons

```
curl -sfL https://cloud-pg.github.io/interLink/itwinctl.sh | sh -s - stop
```

What it takes from user/framework POV

- Virtual nodes are available as normal nodes BUT
  - They are not included into the unconditioned scheduling
- Pods created by either users or frameworks have to EXPLICITLY indicate a k8s node affinity and toleration
  - This is the intended behavior since those are not general purpose resources, but rather computing specific ones
- NO shared FS are available:
  - Configmaps, secrets and empty dirs are the only available volume kind

```
test-vk          NotReady   agent                6m39s
vega-new-vk _    Ready      agent                17d
```

```
apiVersion: v1
kind: Pod
metadata:
  name: test-pod-cfg
  namespace: oscar
  annotations:
    slurm-job.knoc.io/flags: "--job-name=test-pod-cfg -t 2800  --ntasks=8 --nodes=1 --mem-per-cpu=2000"
spec:
  restartPolicy: OnFailure
  containers:
  - image: docker://busybox:latest
    volumeMounts:
    - name: foo
      mountPath: "/etc/foo"
      readOnly: true
    env:
      - name: BOOKKEEPING
        value: keepmehere
    command:
      - sleep
      - infinity
    imagePullPolicy: Always
    name: busyecho
  dnsPolicy: ClusterFirst
  nodeSelector:
    kubernetes.io/hostname: vega-new-vk
  tolerations:
  - key: virtual-node.interlink/no-schedule
    operator: Exists
  volumes:
  - name: foo
    emptyDir: {}
```

Status of pilot implementations

Infrastructure: the following sites were federated under a common K8s cluster deployed on cloud resources

- Vega - EuroHPC
  - Deployed interLink Slurm layer on a VM on the edge of the HPC
- Juelich
  - Deployed “remote docker run” interLink on a login node

```
(base) ➜  ~ kubectl get pod -n oscar test-pod-cfg-x-oscar-x-oscar -o wide
NAME                                READY   STATUS    RESTARTS   AGE   IP          NODE
test-pod-cfg-x-oscar-x-oscar        1/1     Running   1          24h   127.0.0.1   vega-new-vk
```

```
[ciangottinid@intertwin ~]$ squeue --me -i5
Tue Sep 26 09:41:12 2023
             JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
           4086045       cpu test-pod ciangott  R 1-00:41:14      1 cn0130
```

Applications

- Oscar at Vega
  - Serverless execution of a pod on Vega has been functionally tested
    - A lot of useful feedback already → looking forward to the next round of iteration
- MLFlow at Juelich
  - A ML container has been successfully delegated to docker on the login node at Juelich

```
ubuntu@virtual-kubelet:~/interLink/docs$ docker ps -a
CONTAINER ID   IMAGE                                          COMMAND                  CREATED         STATUS
826dbe86c9c4   ghcr.io/intertwin-eu/t6.5-ai-and-ml:containers   "sleep 10000000"         7 minutes ago   Up 6 minutes
ner
```

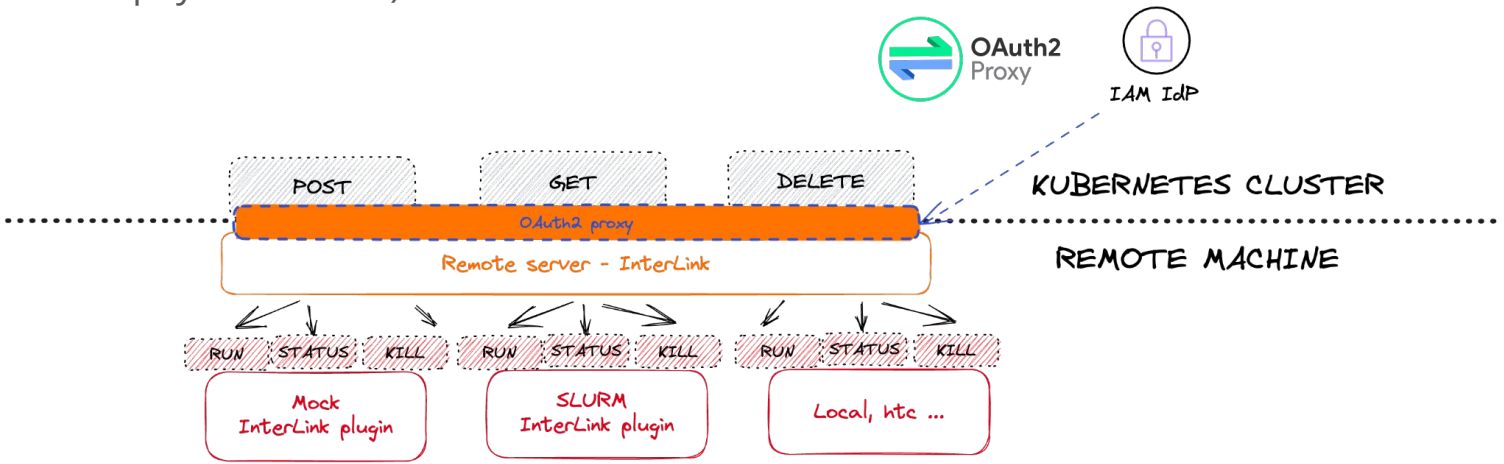
Conclusions

- **A first version of the interTwin offloading system has been prototyped and the fruitful collaboration with Vega and Juelich represented a key to the success for this early implementation**
  - **Along with the tests of real workflows**
- **Lessons learnt (technical):**
  - **Managing runtime environment in a transparent way is possible… but a challenge i.e.**
    - volume and Apptainer/Singularity
    - Site resources access policies
    - FS permission etc..
- **Next steps and priorities**
  - **Further integration and enhancements with Data-Lake (i.e. Rucio ecosystem)**
  - **Offloading integration with INDIGO-PaaS integration**
  - **Extend testbeds and pilots within interTwin communities**

BACKUP

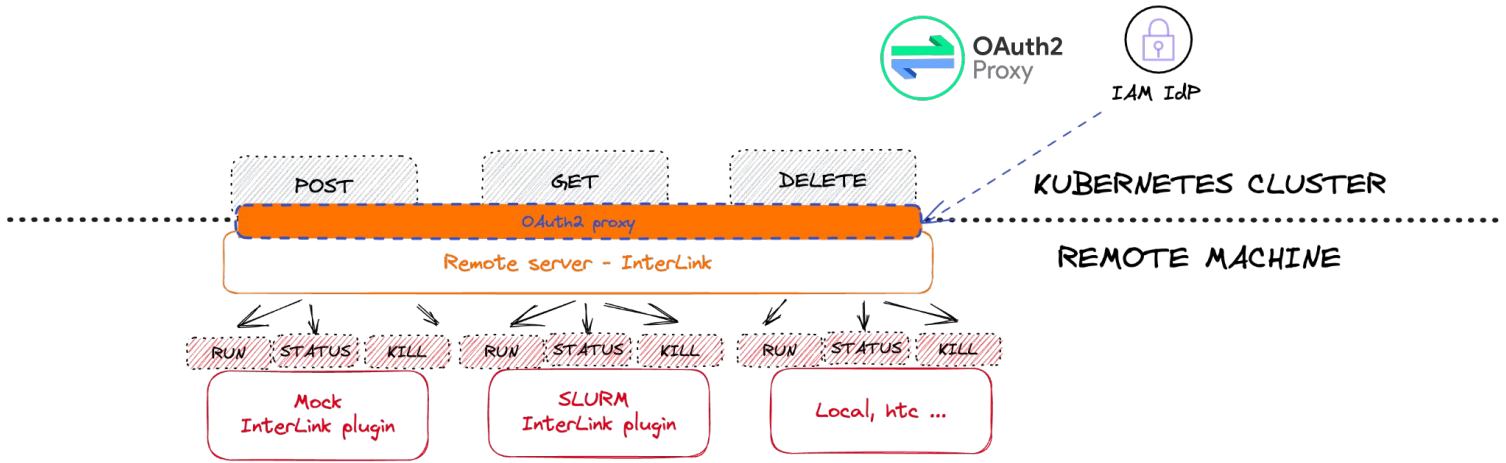
AuthN/Z 1/2

- **The initial idea is a model where each k8s cluster admin is responsible for authorizing access to the remote resources**
  - **On the remote side the HPC can allow connection only for “trusted” admins/VOs**
  - **Additional custom policies can be integrated at the plugin level**
    - e.g. contacting IdP for additional user info
    - Requesting additional information from pod annotation (e.g. the username of the payload owner)



AuthN/Z 2/2

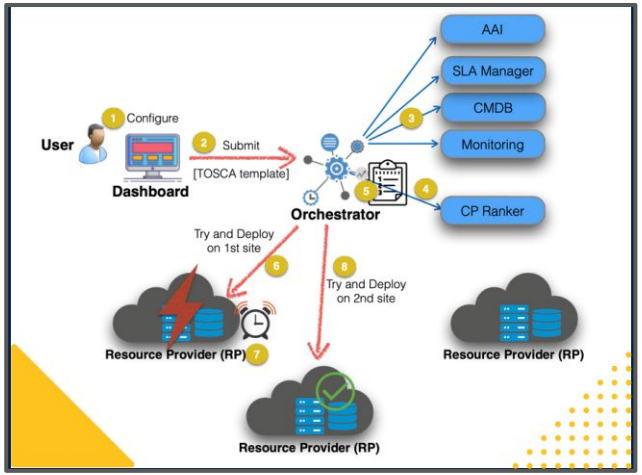
- Current implementation support generic OAuth2 access token via Authorization header
  - Thanks to [oauth2 proxy] instantiated automatically on the edge service
  - it can be configured to accept connection only from services presenting a token with a valid group claim (configurable) and audience



Orchestrating deployment on cloud

Compute Resource orchestration is a critical feature to **implement infrastructure federation functionality**

- to automate the deployment and setup of user-defined services
- make decisions regarding provider selection



Identification of the best Cloud provider by utilising a variety of features based on both static and dynamic metrics.

- The static metrics can be defined in advance
- Dynamic information collected at runtime and continuously updated.

Intertwin enhancement

A key aspect...

How the cloud dynamic orchestration and federation integrates with the offloading mechanism ?

- With interTwin we plan to **enhance the orchestration system for the workload offloading mechanism**
  - on-demand deployed services will be automatically configured to offload payloads.

For the system to successfully implement the workload offloading mechanism, it is essential to include and rely on a detailed information system that collects and publishes information about the different heterogeneous providers in the federation.

E.g. **we plan to use the interLink APIs layer to send telemetry/traces**

The overall configuration should be fully transparent to the end users. Keeping the possibility to go “advanced mode” and tweak manually the desired parameters.

Status and plans

The design has been discussed and agreed (see the deliverable)

The plan is to start with a semi-manual setup

- We deploy services via PaaS orchestrator
- We configure the services (by hands initially) to offload

This is a procedure to be adopted on the pilots (see later)

A key is that WP6 and users start defining requirements (i.e. TOSCA templates/Helm charts/whatever for the services to be deployed)