Running Kubernetes Workloads on HPC with HPK

HPC, Big Data & Data Science Devroom • FOSDEM 2025

Antony Chazapis • FORTH-ICS/CARV

FORTH INSTITUTE OF COMPUTER SCIENCE

Image source: https://www.bsc.es/marenostrum/marenostrum

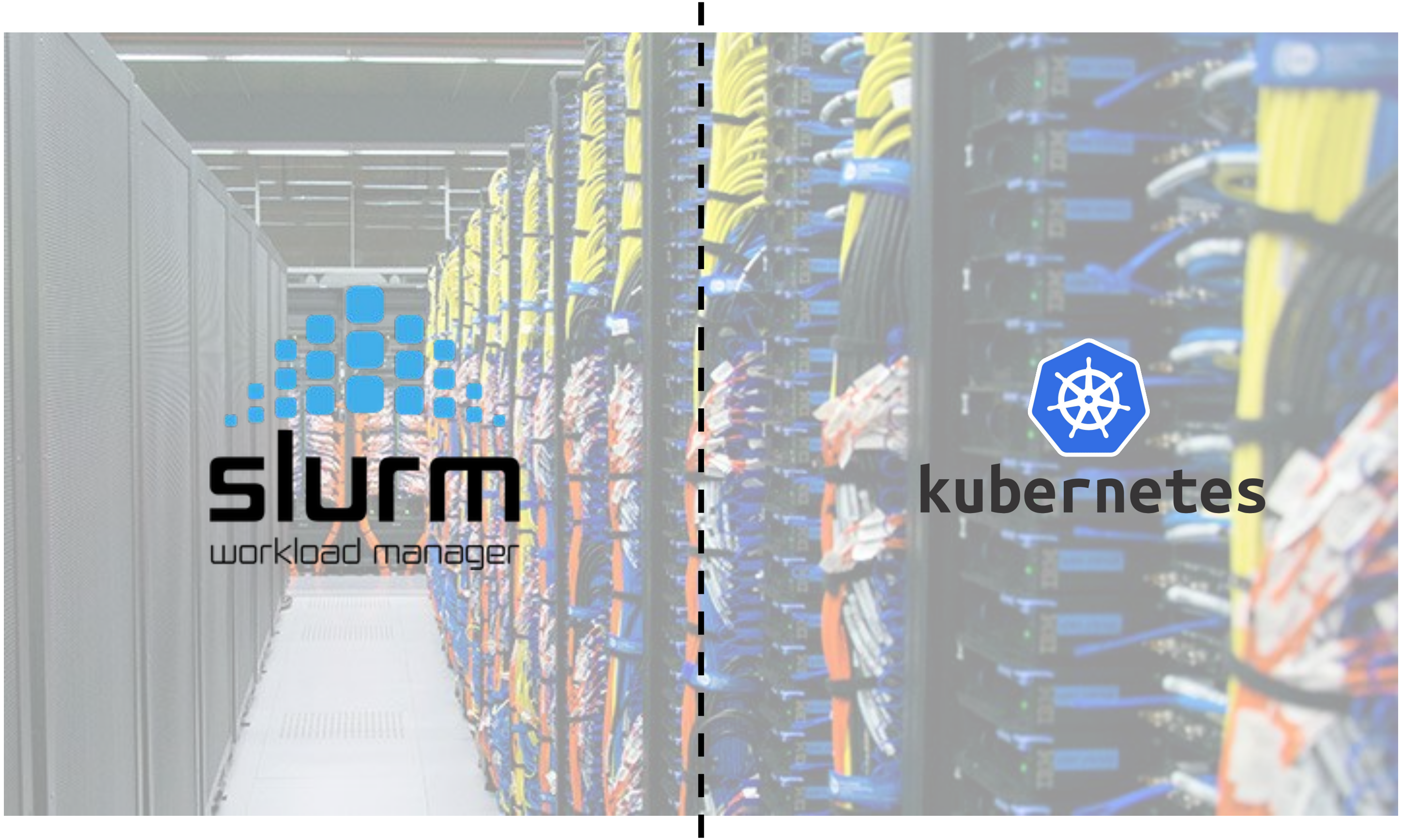
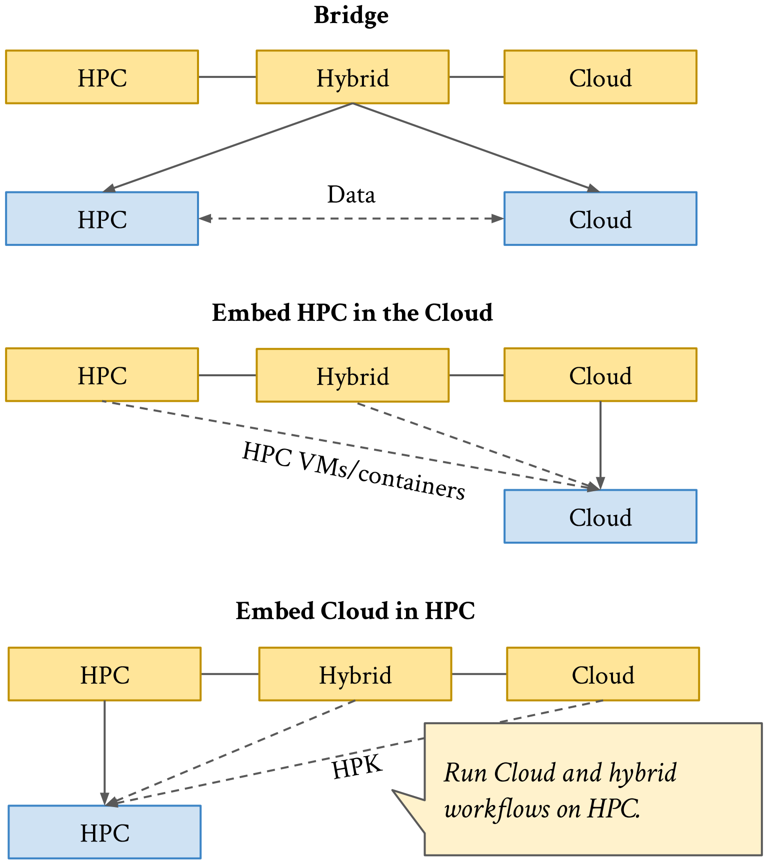


Image source: https://www.bsc.es/marenostrum/marenostrum

Running hybrid workloads

- Bridge two separate environments
  - Submit HPC jobs from the Cloud side or vice versa[1]
  - Deal with separate data and network contexts, unless running HPC in the Cloud (VM offerings available)
  - Hardware and maintenance costs
- Run in Kubernetes
  - Embed the HPC software stack in containers → Delegate scheduling to Kubernetes[2]
- Run on HPC with HPK!
  - Provide a way to run Cloud software in HPC
  - Create ephemeral Kubernetes mini-Clouds in HPC



[1] KNoC is a Kubernetes node to manage container lifecycle on HPC clusters: https://github.com/CARV-ICS-FORTH/knoc (InteractiveHPC 2022)
[2] Genisys is a Kubernetes scheduler for running HPC jobs inside Virtual Clusters alongside other services: https://github.com/CARV-ICS-FORTH/genisys (VHPC'22)

So... Kubernetes?

Kubernetes is a *container orchestration runtime*

- A cloud operating system?
    - Takes care of resource management, networking, failures
    - Interface to the container runtime → containerd
- The thin line between hardware and software—an abstraction
    - Primitives and conventions
    - DevOps compliant → Used for development and deployment
- Portability of operations across platforms (with exceptions, as always)
    - Local (minikube, MikroK8s, k3s, kind, ...)
    - Cloud (Amazon EKS, Azure AKS, Google GKE, DigitalOcea Kubernetes, ...)
- Many extensions and third-party tools

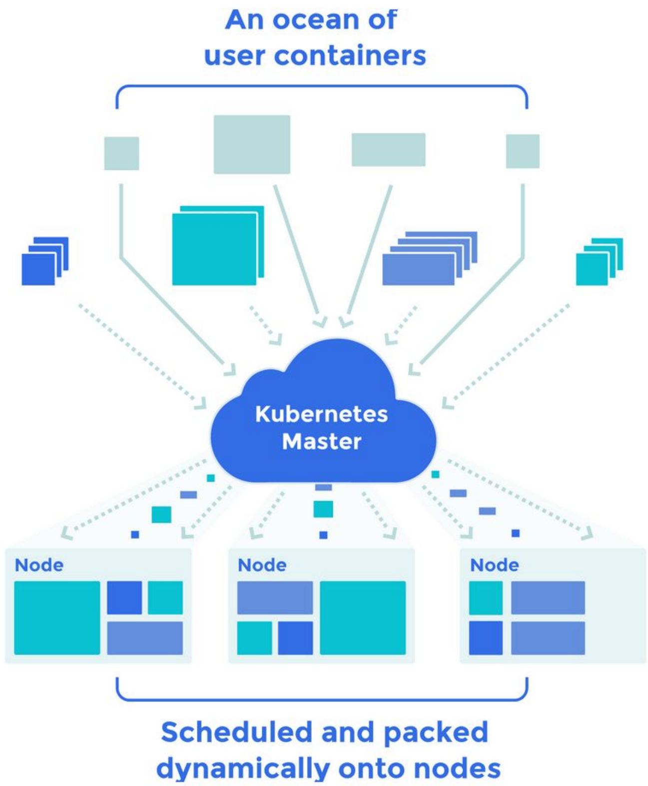
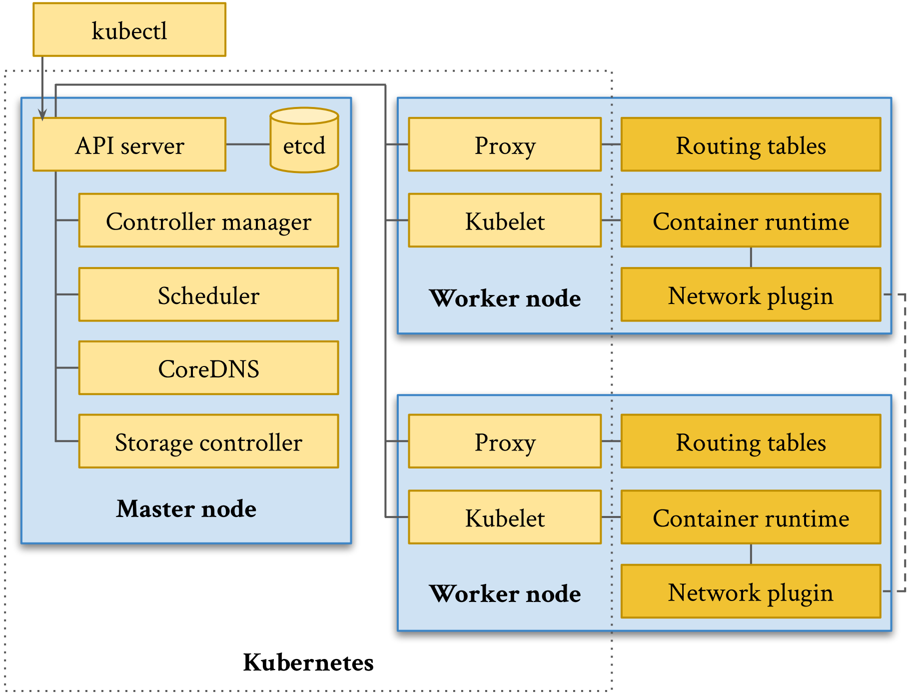


Image source: https://discuss.newrelic.com/t/relic-solution-what-you-need-to-know-about-new-relic-when-deploying-with-docker/52492

Kubernetes concepts

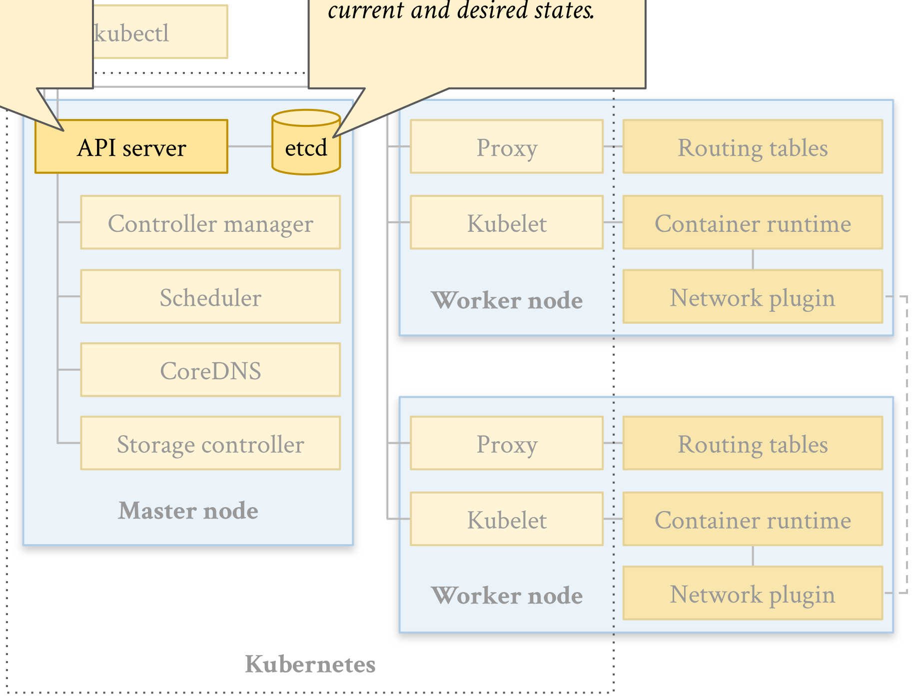
- Declarative vs imperative
- API endpoint & controllers
- Abstractions
  - Pods → Collection of containers
  - Deployments → Replicated pod groups
  - Services → Microservice naming
  - Jobs → Pods that run to completion
  - Volumes → Mountable file collections
  - Labels → Queryable metadata
- Typical distributed structure
  - "Control plane" → Scheduling and placement
  - Node agents → Handle execution
  - Monitoring and accounting infrastructure



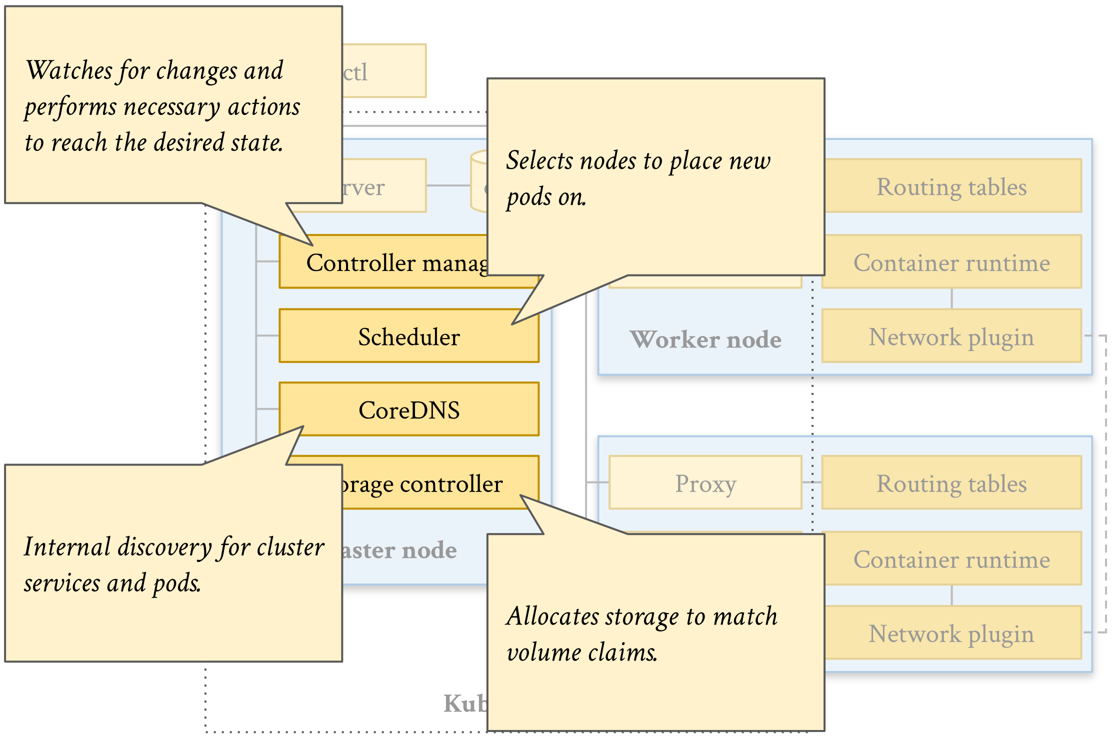
Kubernetes core components layout

The main entrypoint for all requests to the cluster, the point of sync for all controllers.

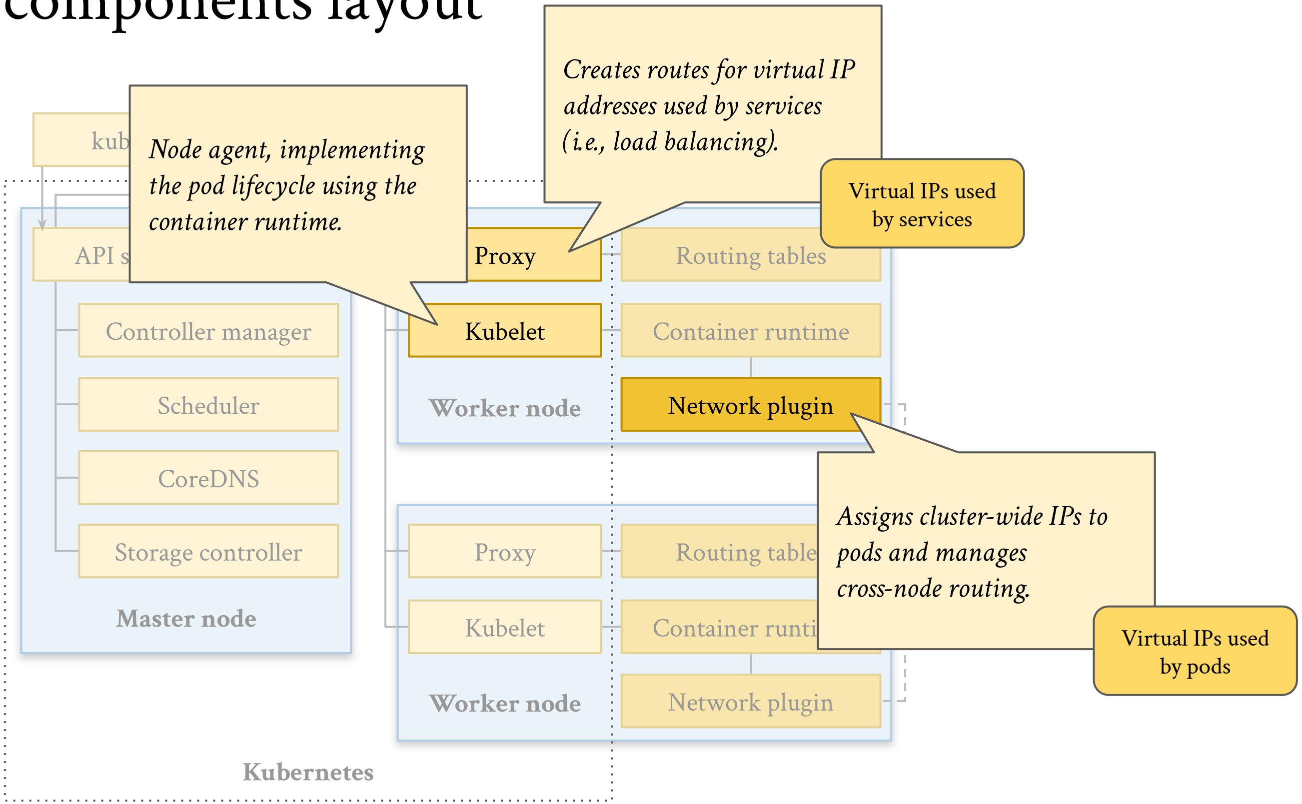
A key/value store holding current and desired states.



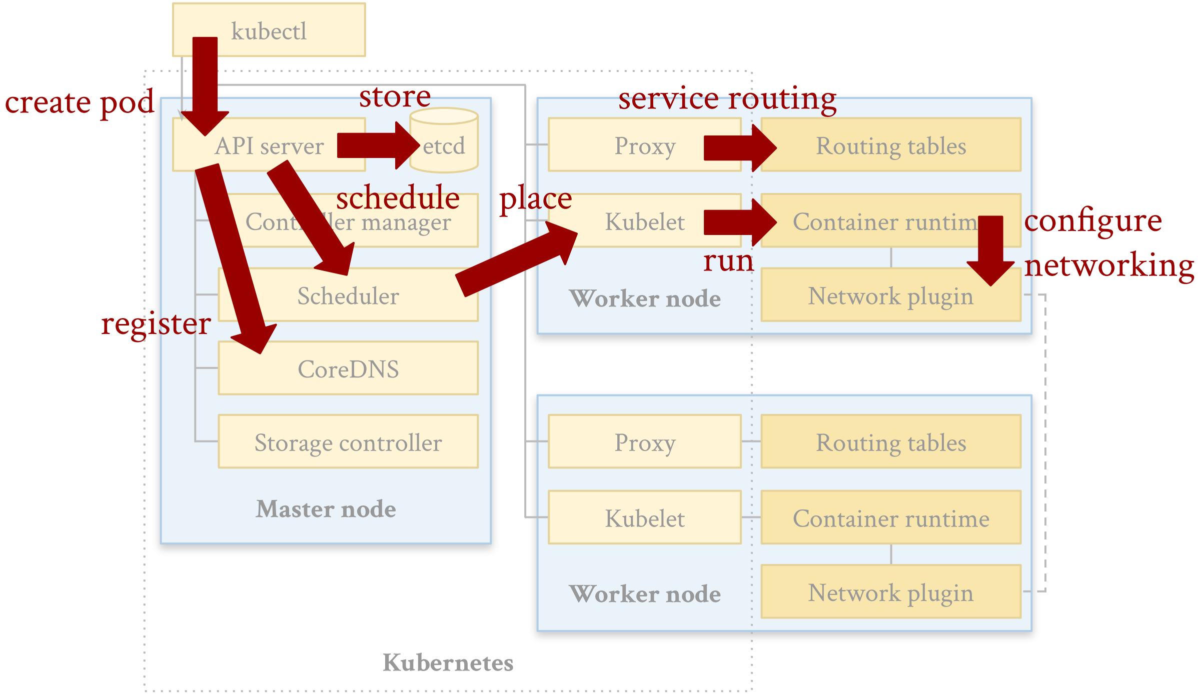
Kubernetes core components layout



Kubernetes core components layout



Kubernetes core components layout



Design goals

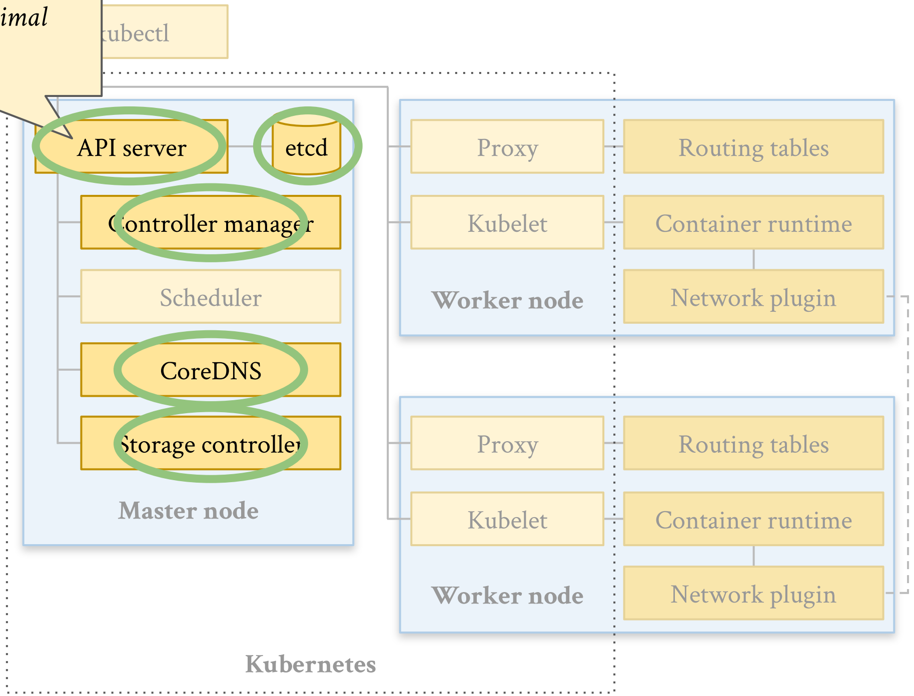
Run *Kubernetes in an HPC environment* as a user → High-Performance Kubernetes (HPK)

Requirements:

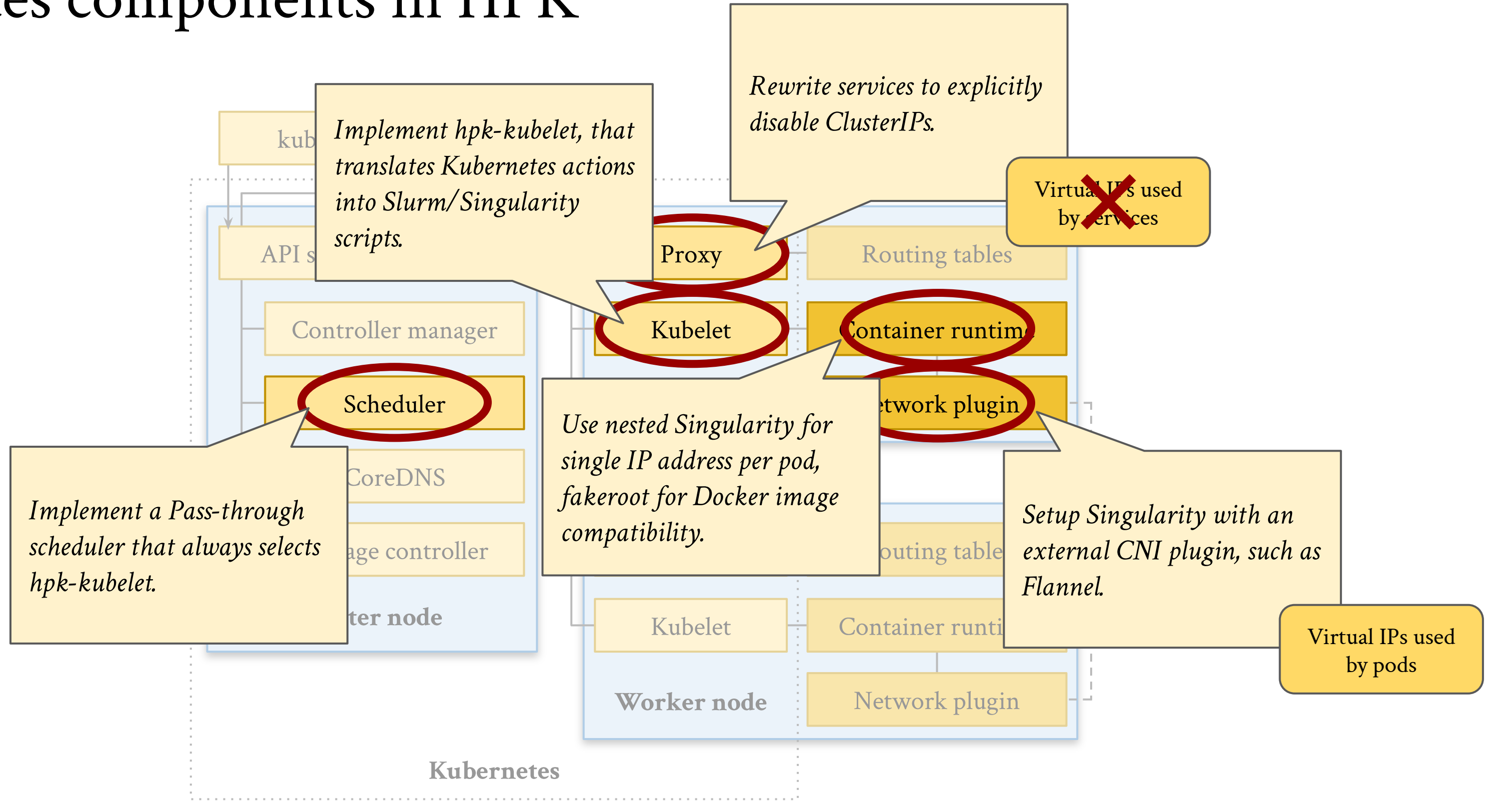
- All Kubernetes abstractions should be available and fully functional
  - Except those that affect physical hardware resources (like NodePort services)
  - Private, inter-container network and internal DNS should work as expected
- Delegate resource management to Slurm
  - Respect organization policies
  - Comply with established resource accounting mechanisms
  - Scale across all nodes of the cluster
- Use Singularity as the container runtime
  - Preinstalled in most HPC environments
- Make it easy for HPC administrators
  - No (or little) configuration changes should be required at the host level
  - No reliance on special libraries or binaries that execute with "elevated" permissions
- Make it easy for users
  - All neatly packaged up in one container
  - Simple, one-command deployment via Slurm
  - All relevant configuration and files should be in the user's home folder

Kubernetes components in HPK

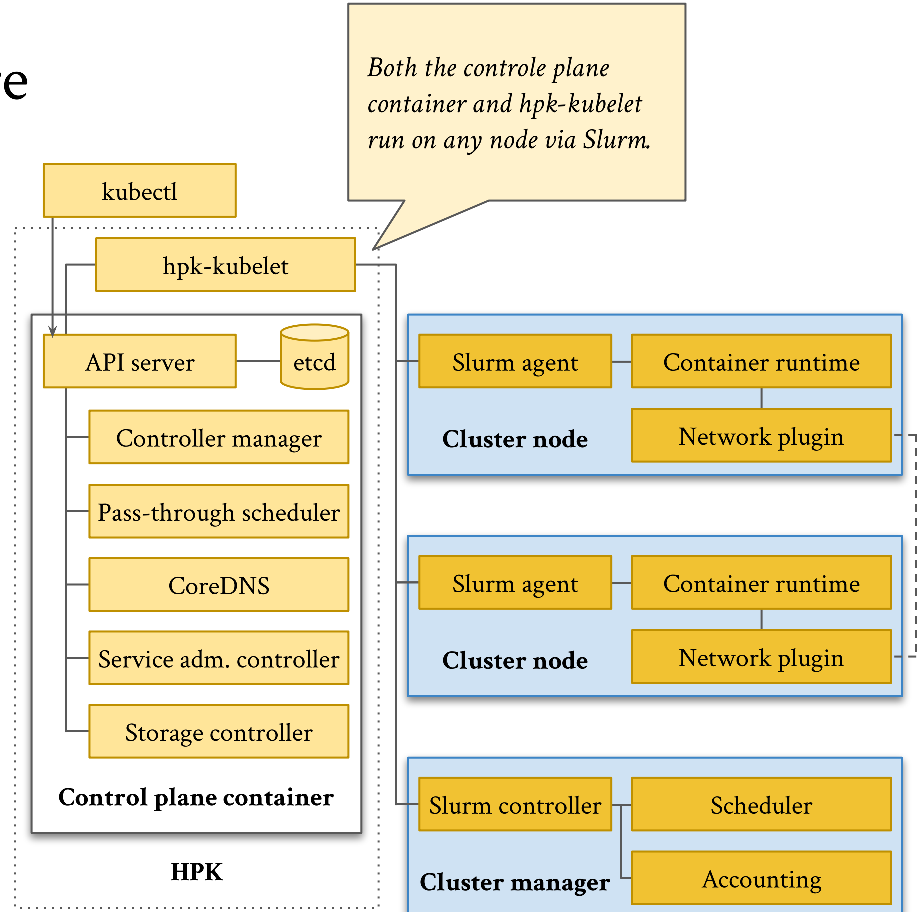
Services that run "out of the box", with no special dependencies, or minimal changes.



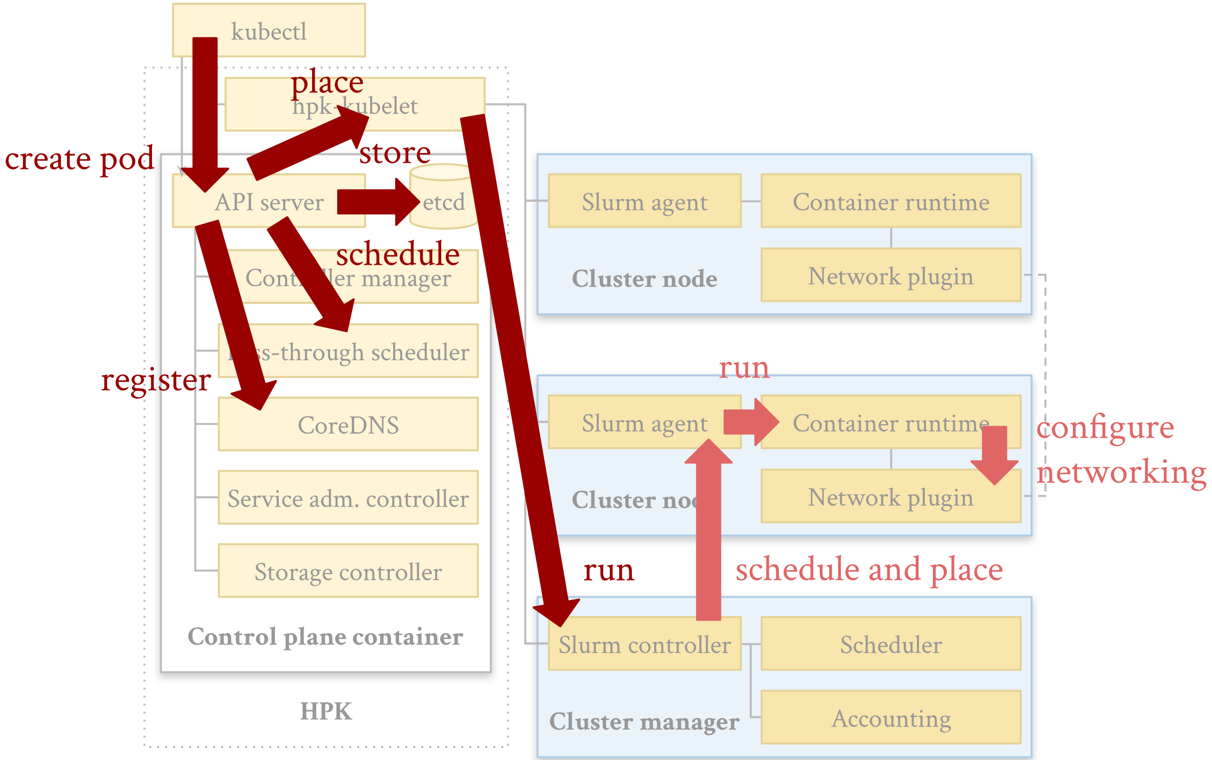
Kubernetes components in HPK



HPK architecture

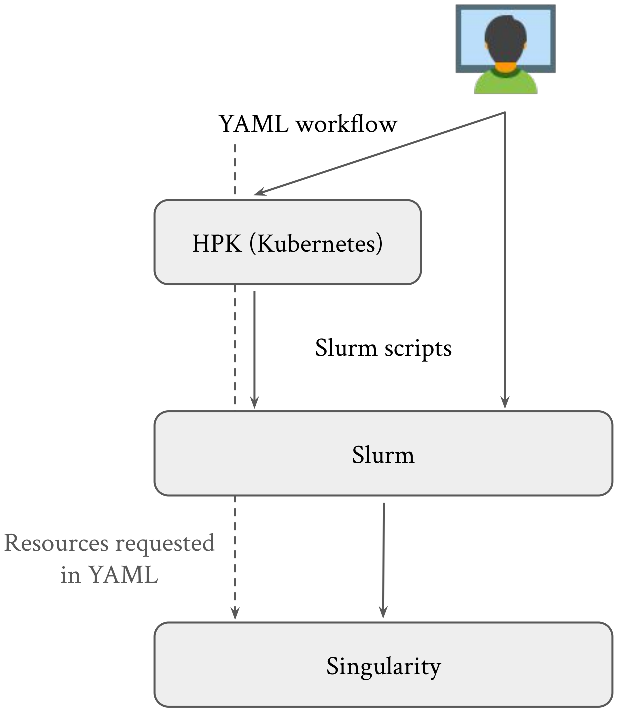


HPK architecture



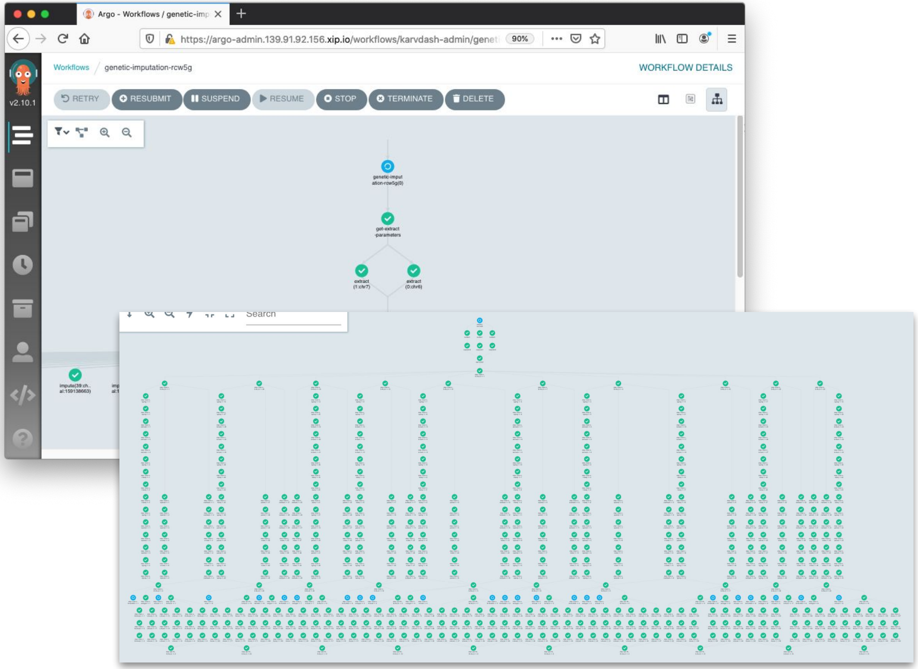
HPK implementation

- HPK translates Kubernetes to Slurm scripts
  - Pods/jobs show enter as YAML through the Kubernetes API, exit as Slurm scripts from hpk-kubelet → Pods show up as Slurm jobs
  - Slurm jobs run a hierarchy of Singularity containers (containers within pods)
  - Kubernetes resource requirements end up in Slurm allocation requests → No changes to accounting
- HPK runs as a user process via Slurm
  - User can run both Kubernetes and Slurm workloads at the same time
  - No "special" allocation needed for HPK → 1 CPU, few GBs of RAM should be enough
  - Little support needed by the environment → No Slurm modifications, some Singularity networking configuration (inc. Flannel or other CNI)



HPK for Cloud-native workflows

- Using Argo Workflows
  - All steps are containers → Containerized code is portable
  - Interactive UI → Monitor, control
- Language features
  - Loops, conditionals
  - High-level parameters (shared across all steps), workflow templates
  - Artifacts → Step I/O
- HPK extensions → Integrate MPI steps
  - Slurm passthrough flags via annotations → Control scalability, allocate GPUs
  - Still need to containerize code



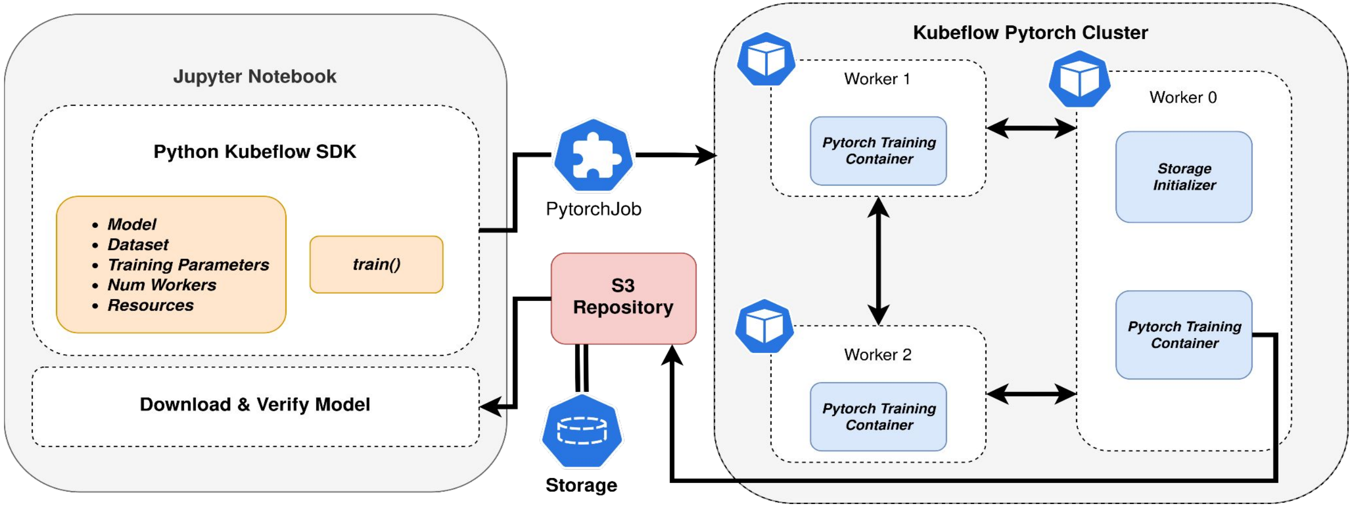
HPK for Cloud-native workflows

- Using Argo Workflows
  - All steps are containers → Containerized code is portable
  - Interactive UI → Monitor, control
- Language features
  - Loops, conditionals
  - High-level parameters (shared across all steps), workflow templates
  - Artifacts → Step I/O
- HPK extensions → Integrate MPI steps
  - Slurm passthrough flags via annotations → Control scalability, allocate GPUs
  - Still need to containerize code

```
kind: Workflow
metadata:
  ...
spec:
  entrypoint: npb-with-mpi
  templates:
  - name: npb-with-mpi
    dag:
      tasks:
      - name: A
        template: npb
        arguments:
          parameters:
          - {name: cpus, value: "{{item}}"}
        withItems:
        - 2
        - 4
        - 8
        - 16
  - name: npb
    metadata:
      annotations:
            slurm-job.hpk.io/flags: "--ntasks={{inputs.parameters.cpus}}"
            slurm-job.hpk.io/mpi-flags: "..."
    inputs:
      parameters:
      - name: cpus
    container:
      image: mpi-npb:latest
      command: ["ep.A.{{inputs.parameters.cpus}}"]
```

HPK for ML Workflows: Fine-Tune BERT with PyTorch

- Diverse microservices
  - Jupyter notebook → Coordinates tasks (data preprocessing, model fine-tuning, testing)
  - Kubeflow controller → Manages Pytorch jobs
  - MinIO → S3-compatible object store



HPK for Spark in HPC

- Evaluated HPK + Spark at Jülich (JSC)
  - TPC-DS benchmark
  - Spark Operator (EKS spark-benchmark) → Manages Spark applications on Kubernetes
  - MinIO → S3-compatible object store
- Challenges
  - Preparation of testbed → Apptainer (fakeroot support), Flannel (deploy on all nodes, configure subnets, requires etcd)
  - Integration with Slurm → Exclusive node policy does not align with container sizes

DEEP-SEA

HPK vision

- Cloud-user PoV → Run on HPC hardware
- HPC-user PoV → Exploit the Cloud software ecosystem
  - Combine HPC codes with backend services (database, queueing systems)
  - Interactive code execution → Jupyter
  - Workflow management → Argo Workflows, Apache Airflow, ...
  - Monitoring utilities → Grafana
  - Frameworks for automatically optimizing and scaling code → Spark, DASK, ...
- HPC centre PoV → Run Cloud workloads on the main HPC partition, attract users
  - The common practice is to have separate partitions for Cloud (analytics) and HPC

Get the code!

https://github.com/CARV-ICS-FORTH/HPK

Read the paper!

https://arxiv.org/abs/2409.16919

Acknowledgements