Running Kubernetes Workloads on HPC

24/2/2023 • DEEP-SEMINARS

A. Chazapis, F. Nikolaidis • FORTH-ICS/CARV

Motivation

User wants to write a scalable application on distributed hardware... *HPC or Cloud?*

Differences:

- Tight parallelization vs. data distribution
- MPI++ vs. large selection of frameworks
- Build binaries vs. microservices
- Deploy with scripts vs. YAML recipes
- Use console (ssh) vs GUIs (browser)
- ...

Similarities:

- Hardware (CPUs, GPUs, network)
- Use of a deployment runtime
- Containers possible
- ...

Convergence is about *seamless transition* between environments and *combining the best of both worlds*

Kubernetes on HPC

- Kubernetes is the mainstream runtime for using Cloud resources
- Kubernetes vs. the typical HPC software stack (Slurm)
  - Central "control plane" → Scheduling and placement decisions
  - Agents on every node → Handles execution
  - Monitoring and accounting infrastructure
- "Hybrid" solutions bridge the two environments[1]
  - Implement mechanisms for submitting HPC jobs from the Cloud side or vice versa
  - Two separate setups (hardware and maintenance costs)
- Accommodate both Cloud and HPC on the same hardware[2]
  - Possible because of similar hardware specifications, portability through containers
  - Embed one software stack within the other → Delegate core functions, as we can only have one cluster manager

[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*

- It manages the container lifecycle
  - Containers are lightweight (vs. VMs) and portable
  - Interface to the container runtime → Docker/containerd
- It is not only a scheduler
  - It handles networking between containers
  - It provides service discovery and load balancing mechanisms
  - It reacts to load (scaling) and failures
- It runs almost everywhere
  - Any scale → desktop to Cloud
  - Most architectures
  - Runs as a system service → Needs "elevated" permissions

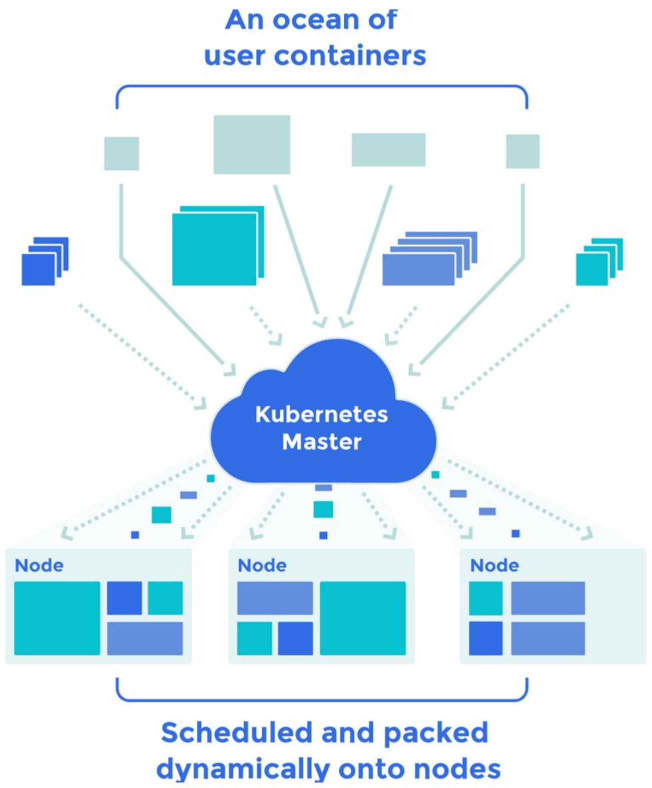
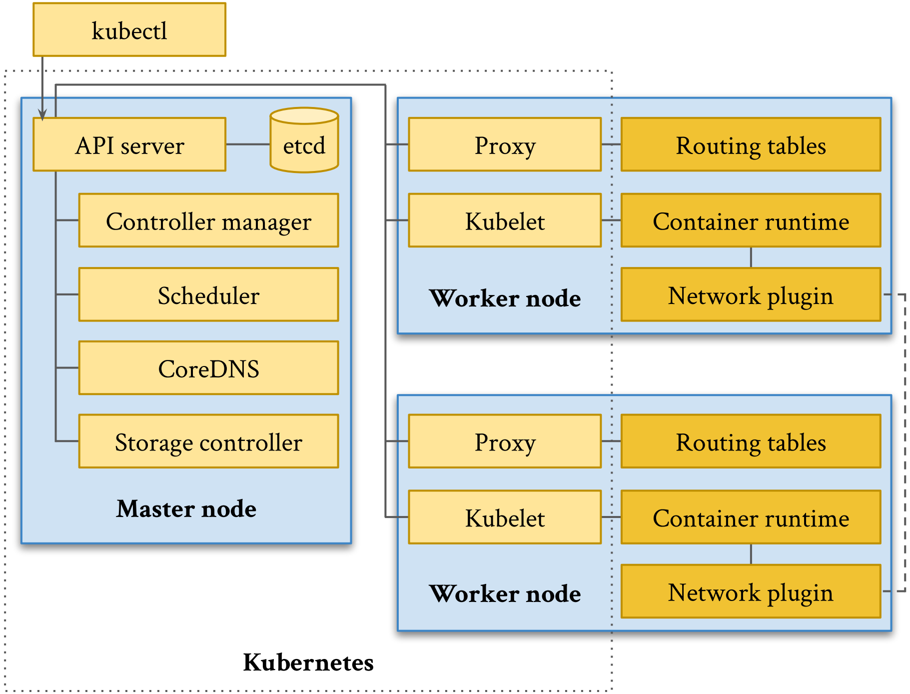


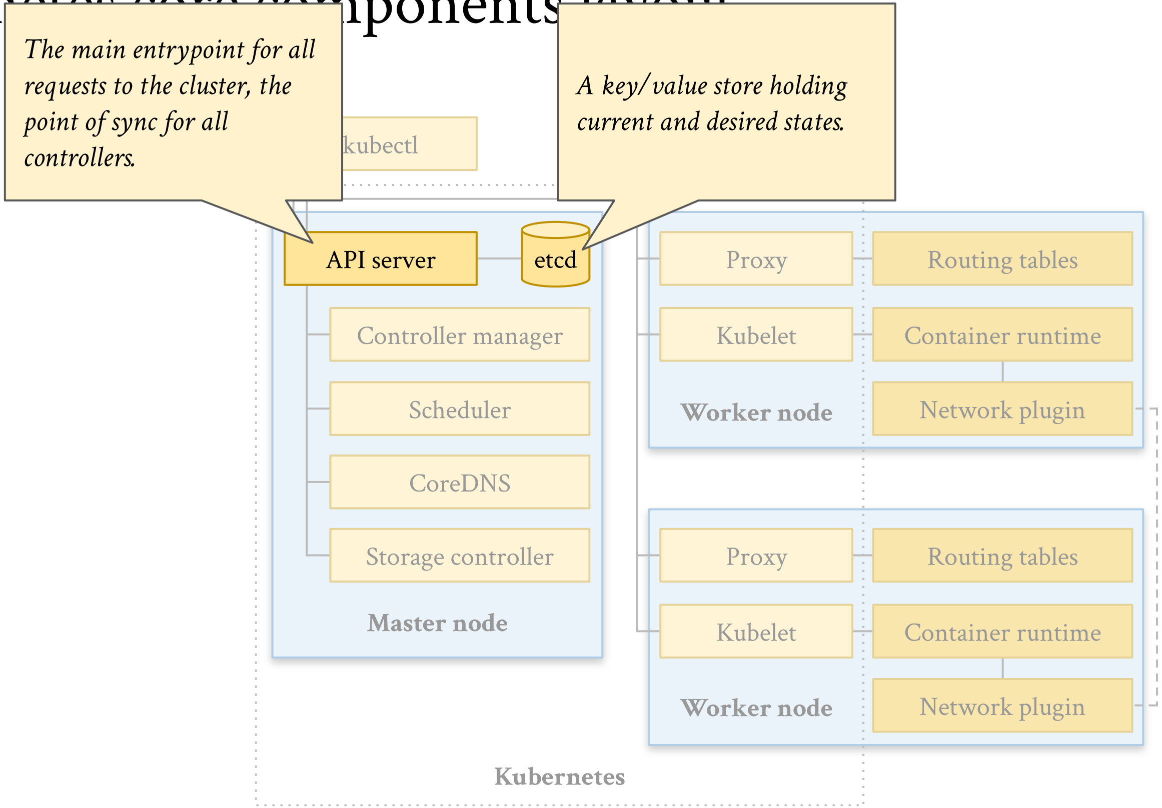
Image source: https://discuss.newrelic.com/t/relic-solution-what-you-need-to-know-about-new-relic-when-deploying-with-docker/52492

Kubernetes concepts

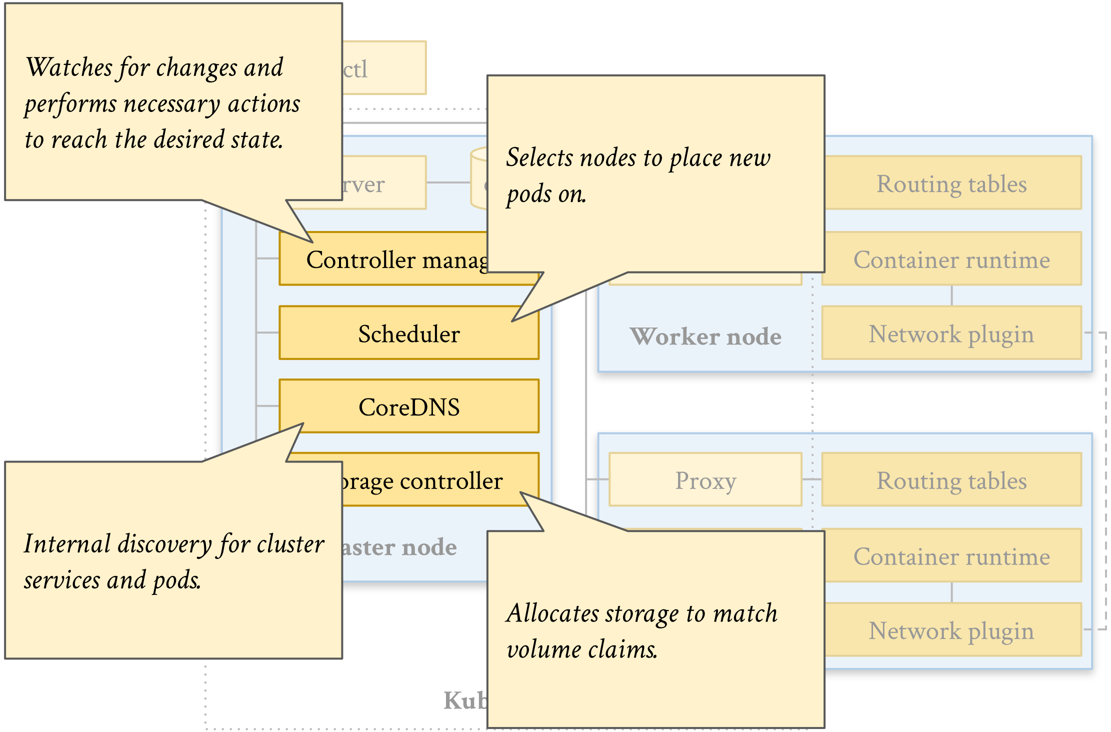
- Declarative vs imperative
- API endpoint & controllers
- Abstractions
  - Pods → Collection of containers
  - Deployments → Replicated pod groups
  - Services → Microservice naming
  - Jobs → Pods that run to completion
  - Volumes → Mountable file collections
  - Labels → Queryable metadata
  - ...
- DevOps compliant
  - Infrastructure as code
  - Version rollouts
  - CI/CD workflows
  - ...



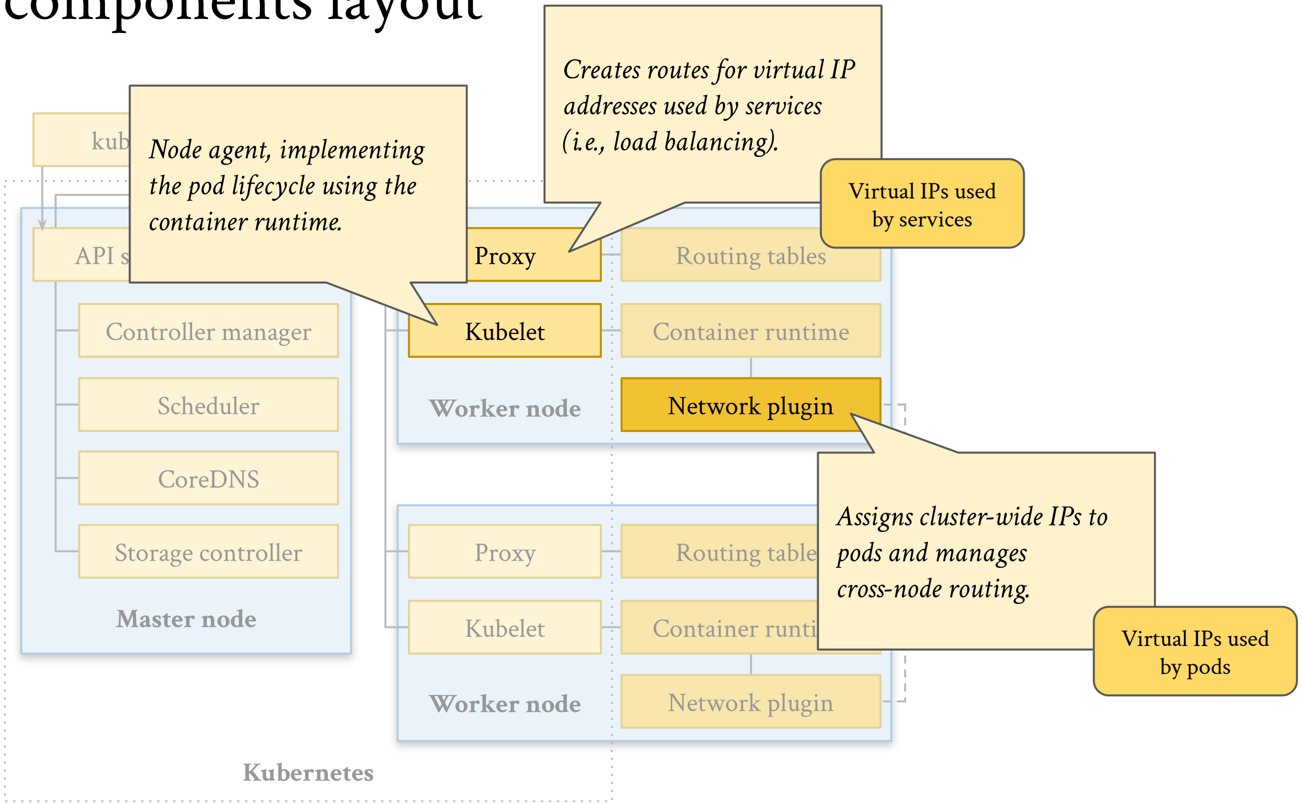
Kubernetes core components layout



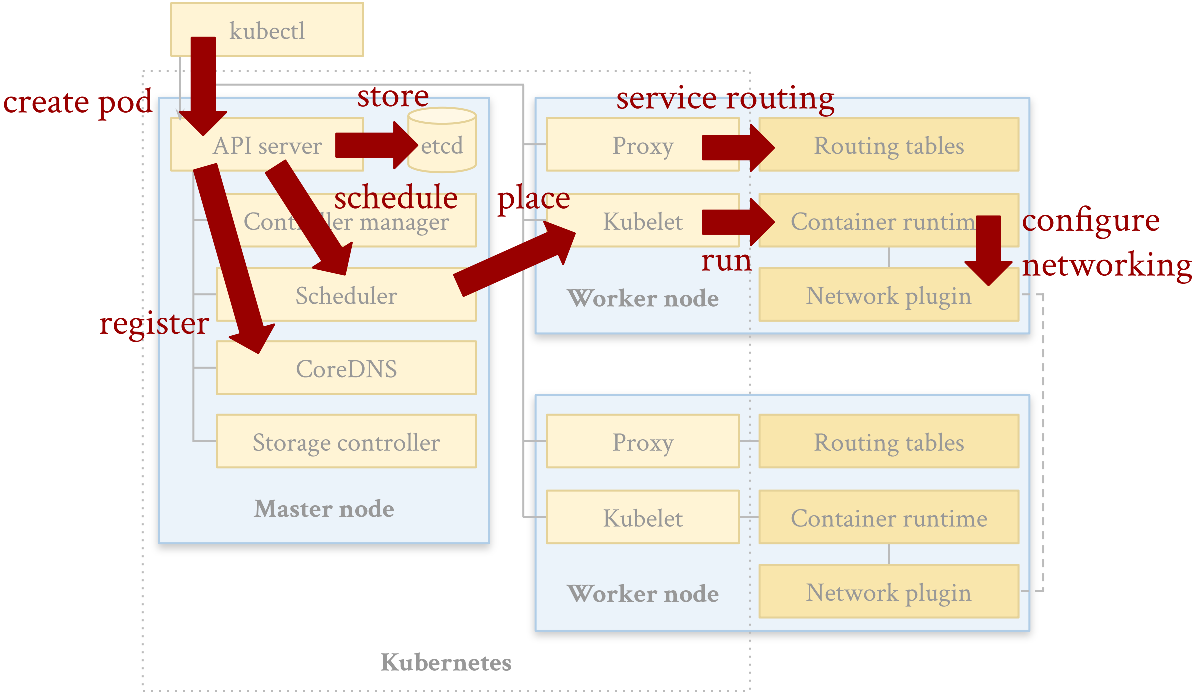
Kubernetes core components layout



Kubernetes core components layout



Kubernetes core components layout



Design goals

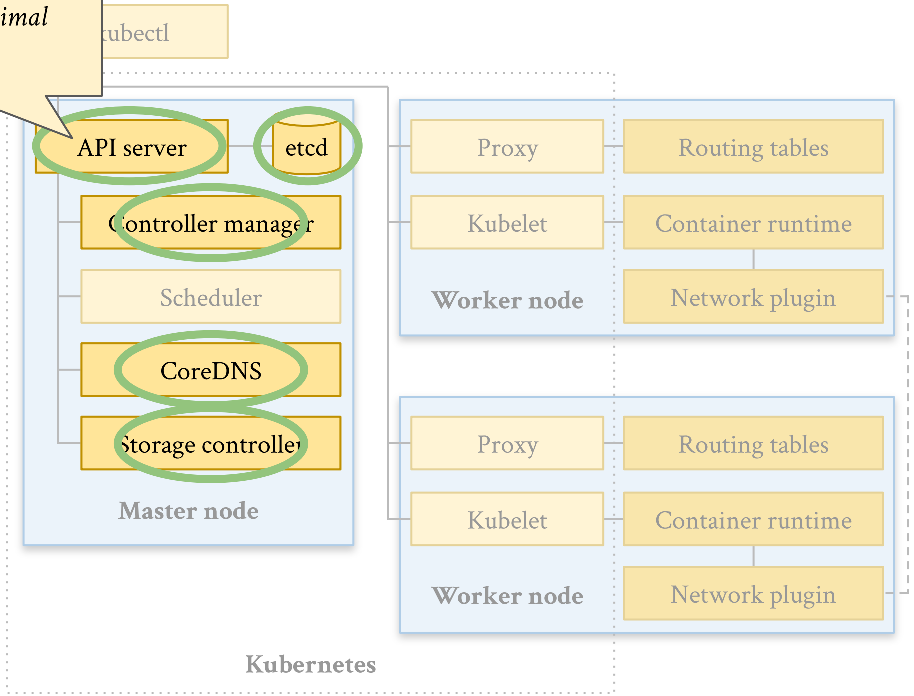
Run *Kubernetes in an HPC environment* as a user → High-Performance Kubernetes (HPK)

Requirements:

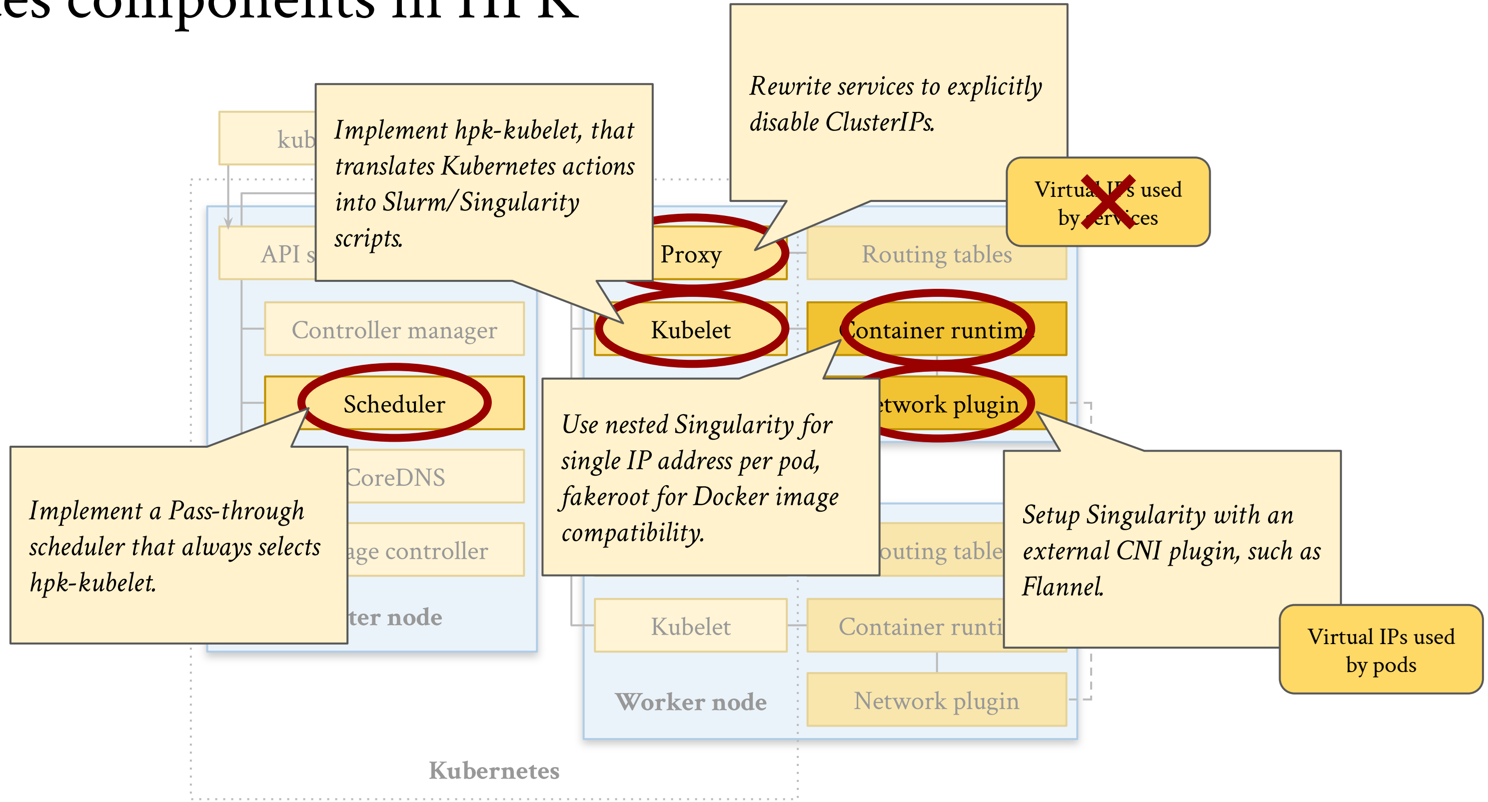
- All Kubernetes abstractions should be available and fully functional
  - Except those that affect physical hardware resources (like NodePort services)
  - Private, inter-container network and internal DNS should work as expected
- Delegate resource management to Slurm
  - Respect organization policies
  - 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

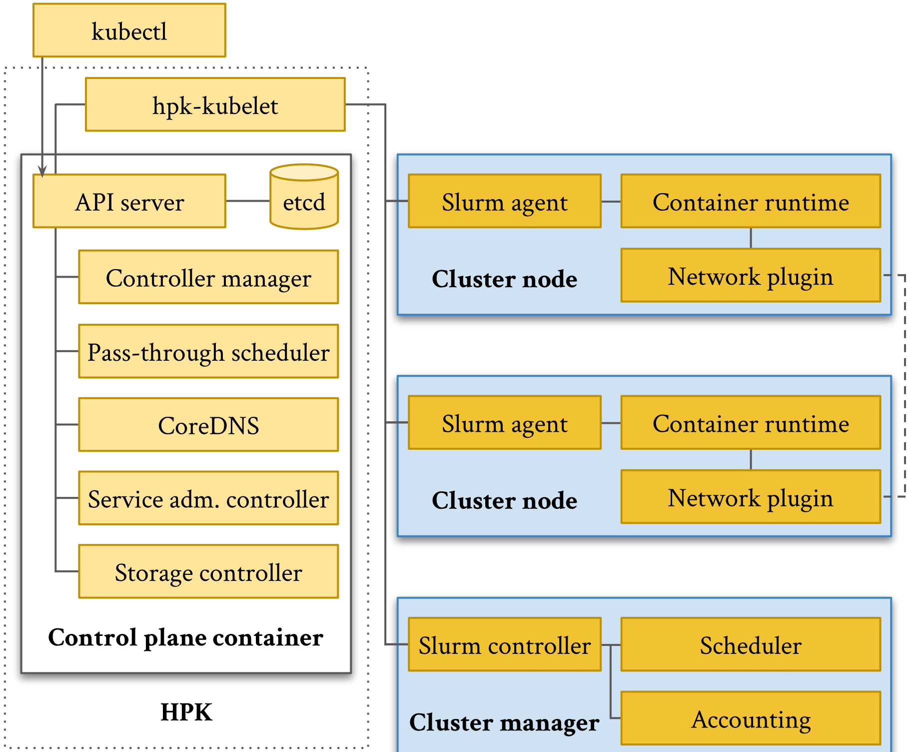
Services that run "out of the box", with no special dependencies, or minimal changes.



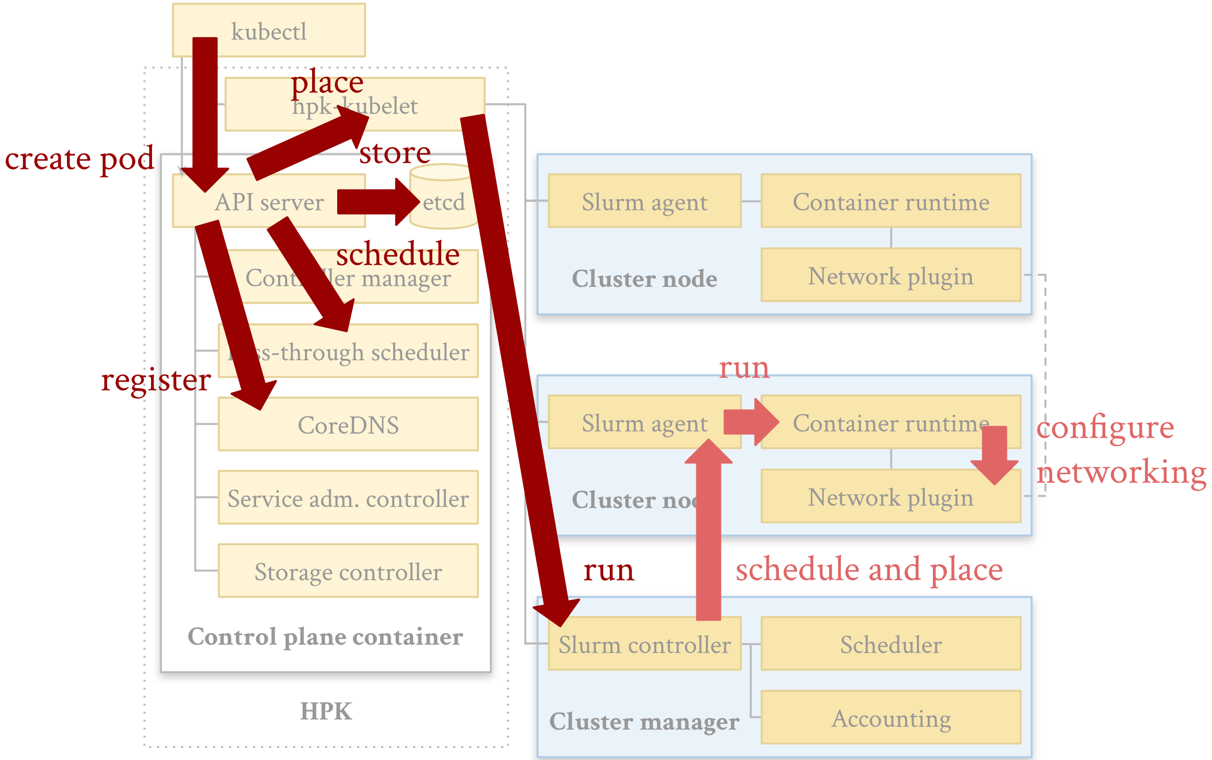
Kubernetes components in HPK



HPK architecture

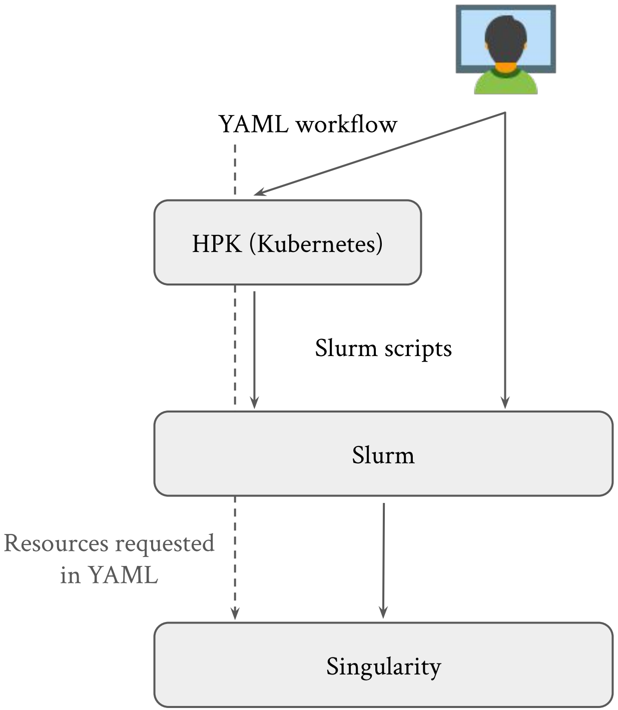


HPK architecture



HPK implementation

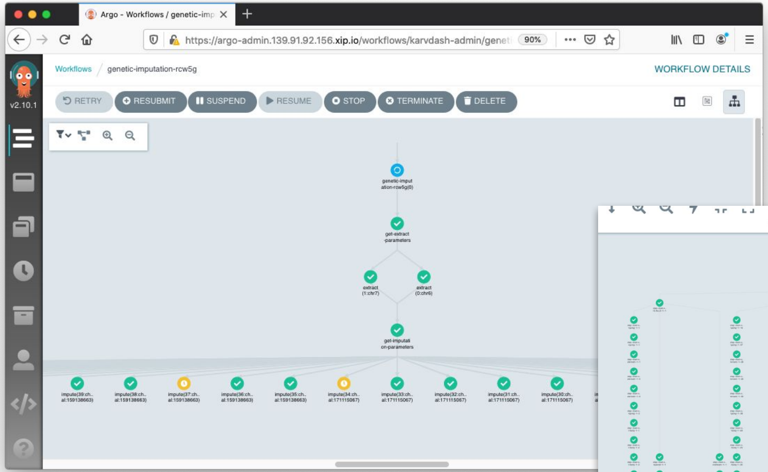
- HPK is Kubernetes-in-a-box
  - Custom kubelet for the execution of containers
  - Changes in various other components to enable the integration
- HPK runs as a user process via Slurm
  - User can run both Kubernetes and Slurm workloads at the same time
  - No "special" allocation needed for HPK → 1 CPU, few GBs of RAM should be enough
  - Little support needed by the environment → No Slurm modifications, some Singularity configuration
- 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
  - Kubernetes resource requirements end up in Slurm allocation requests → No changes to accounting

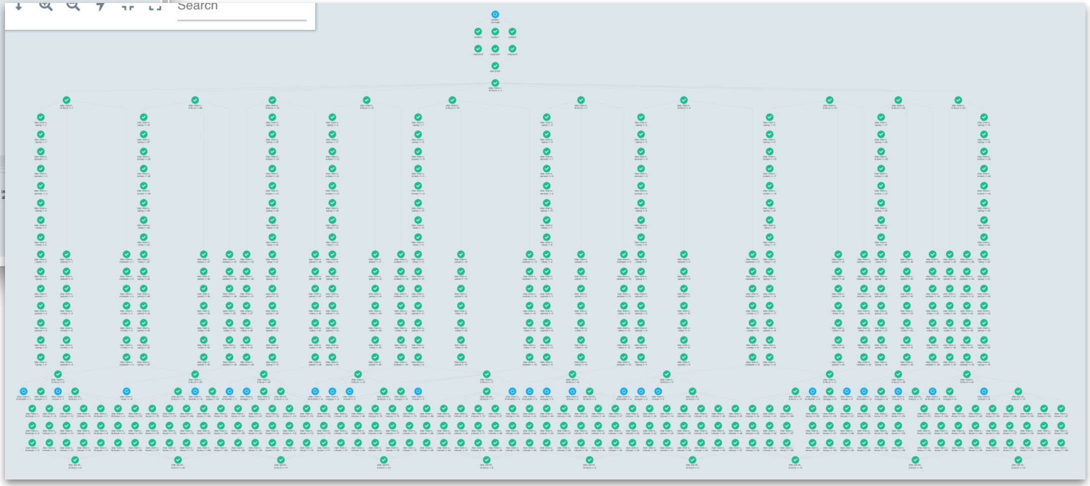


Current status

- System requirements identified for inter-container networking, Docker image compatibility
  - Singularity should be configured to use Flannel (or other CNI) for assigning cluster-wide IPs
  - Singularity should allow running as fakeroot
- Summary of Kubernetes changes
  - Kubernetes-from-scratch recipe for bootstrapping Kubernetes (generate keys, start basic services)
  - API server, etcd, controller manager, CoreDNS working out of the box
  - Custom pass-through scheduler always selects the first node → Effectively no scheduling from Kubernetes
  - Custom controller disables ClusterIP services → Effectively no need for kube-proxy (no IP range for services)
  - Custom kubelet, which we call hpk-kubelet → Kubernetes-to-Slurm/Singularity translator
  - OpenEBS storage provisioner integrated → Maps Kubernetes volume requests to local storage (in the user's home folder)
- Proof-of-concept applications
  - Argo Workflows with artifacts on MinIO (S3 service) → Can also be used for MPI steps
  - Spark operator
  - TensorFlow Serving
  - Several examples

Workflow frontend





Workflow example

- Containerize code for portability
- Define pass-through flags for Slurm via annotations
  - Control scalability
  - Allocate GPUs
- Combine with other tasks in a single workflow
- Use high-level parameters, shared across all steps
- Integrate other Cloud-native tools

```
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-flag : "..."
    inputs: s
      parameters:
      - name: cpus
    container:
      image: mpi-npb:latest
      command: ["ep.A.{{inputs.parameters.cpus}}"]
```

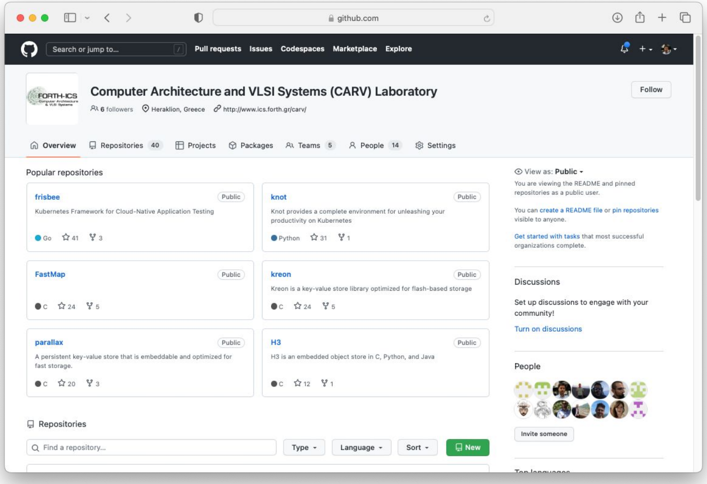
Benefits

- 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
  - The common practice is to have separate partitions for Cloud (analytics) and HPC

Next steps

- Large-scale applications in real clusters → Get feedback from users
- Packaging for easier deployment
  - User runs one script and everything else is downloaded and started automatically
- Future development tasks
  - Exploit GPUs, fast networking
  - Port forwarding to the login node using kubectl (?)

HPK will soon be available at https://github.com/CARV-ICS-FORTH



Acknowledgements

DEEP-SEA EUPEX