Kubenettes ↔ HTCSS

Understanding the possibilities

Brian Bockelman, 25 September 2024

MORGRIDGE
INSTITUTE FOR RESEARCH

Kubernetes

Kubernetes is an open-source container orchestration system.

What does this mean?

- **Open-source**: Started by Google engineers, now developed by a large community and run under the auspices of the Cloud Native Computing Foundation.
- **Container**: Atomic unit of functionality, the “pod”, is composed of one or more containers working together.
- **Orchestration**: A single object can tie together multiple aspects of a service – including dependencies (e.g., database) and network requirements (firewall, DNS).

Kubernetes originates from the Greek κυβερνήτης (kubernḗtēs), meaning governor, 'helmsman' or 'pilot'.

Let’s skip the boring parts

Kubernetes Lingo:

- A **custom resource definition** (CRD) is an custom object type that can extend Kubernetes.
- An **operator** is an extension to Kubernetes that manages custom objects for applications.

- You can run a static HTCondor pool within a Kubernetes cluster:
  - HTCSS team publishes a reference central manager, AP, and EP container image.
  - Going from container image to a deployment is left as an exercise for the user.
- The OSPool runs its central manager inside two Kubernetes clusters.
  - We use a “GitOps” methodology, using a shared base deployment inside a git repository. When changes are committed to the base, they are synchronized by the Flux **operator** to the production clusters.
- However, a statically-sized pool is rather boring: it’s a fairly “vanilla” deployment of an application on Kubernetes.
  - *Still, it’d be nice if we packaged this in a Helm chart.* ☹

Boring Case Study: OSPool Backfill

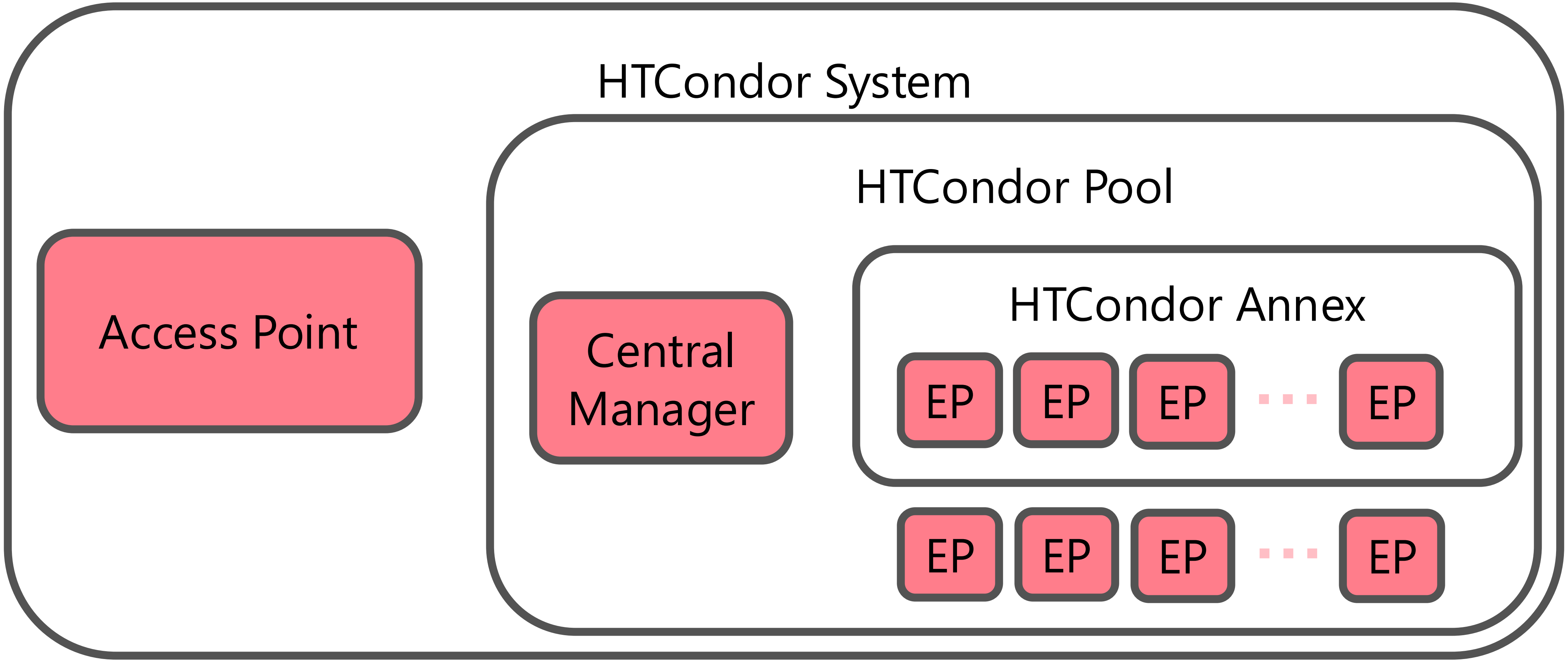
- The OSPool operators publish a “backfill container” image.
  - Given a token, the container will start an EP that connects to the OSPool.
- Cluster administrators can setup a Kubernetes deployment that launches enough pods to saturate the cluster.
  - Set the priority low enough so these are always preempted if another pod needs to run.
- Great for otherwise-idle resources!
  - Not great if you care that OSPool may run out of jobs for your EPs.
  - Not great if you want to balance resources across multiple pools.

Backfill is boring.

What about dynamic workloads?

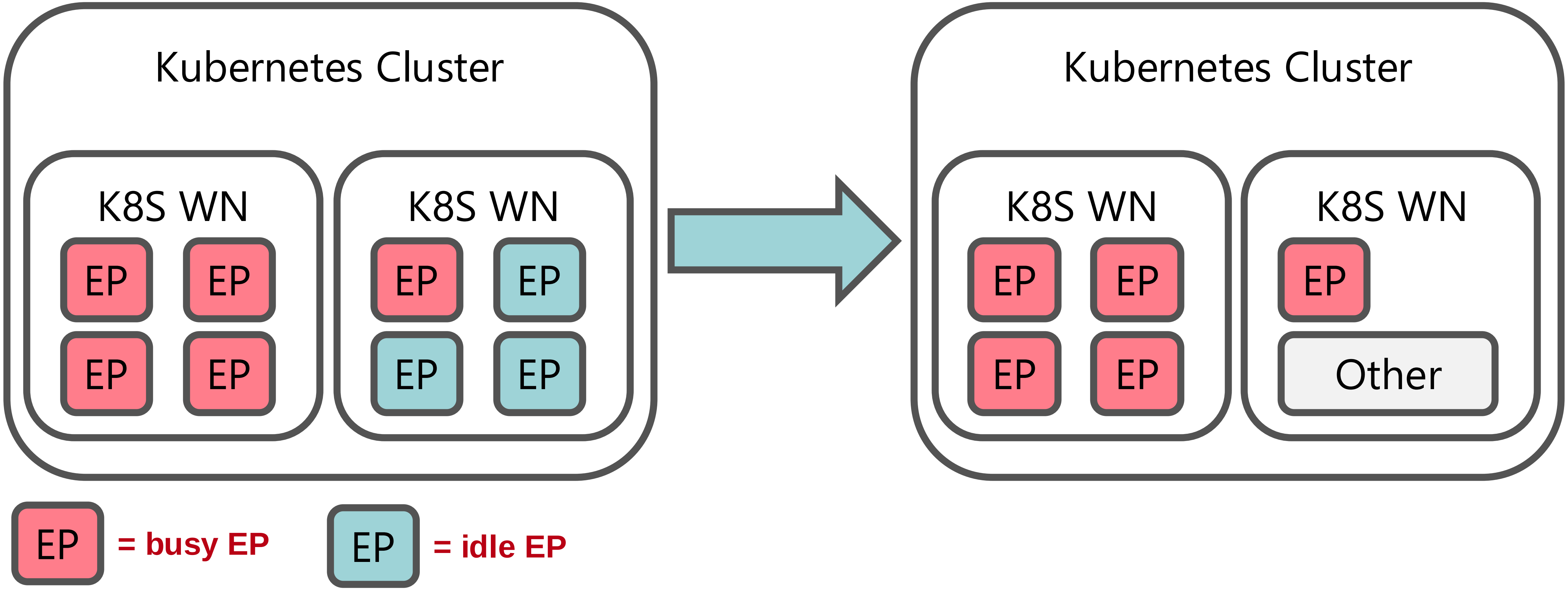
Something interesting – dynamic pools and annexes

- We want the HTCondor pool to grow and shrink based on demand or scheduling policy.



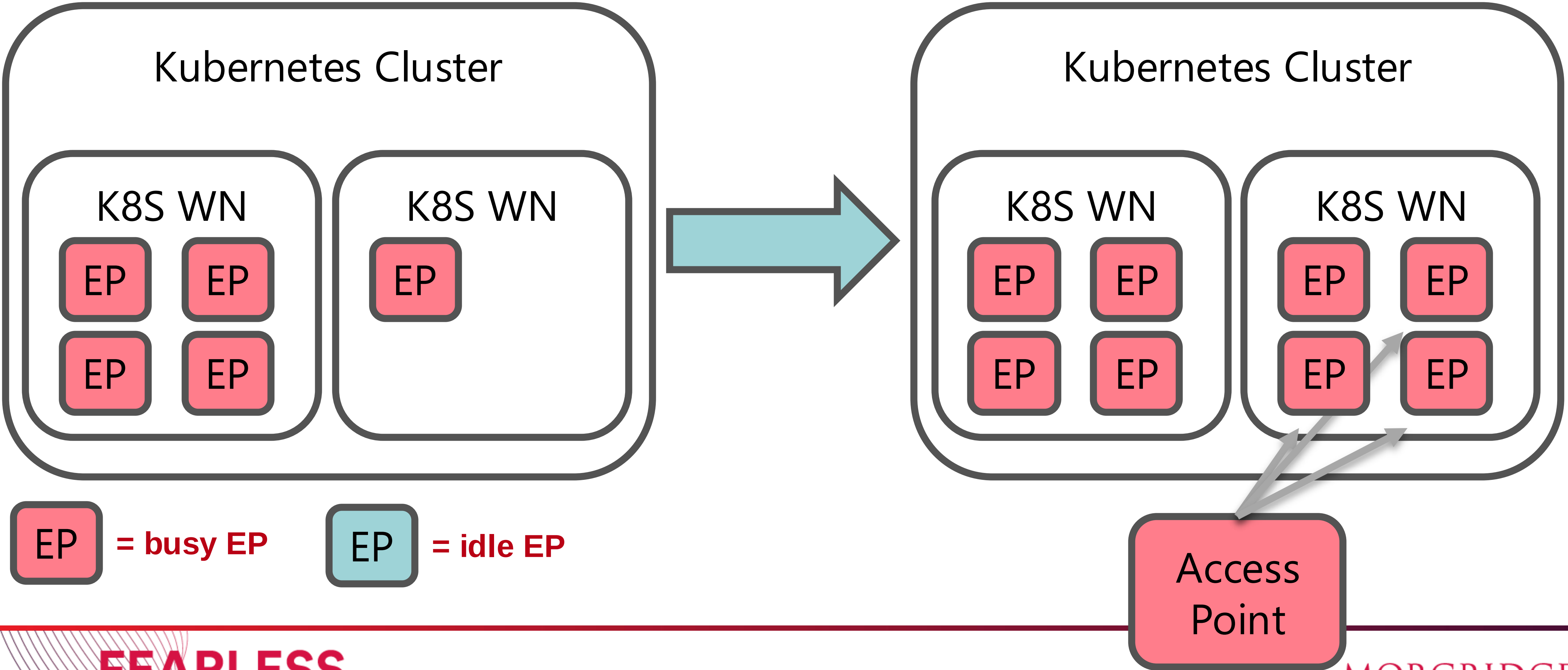
Desired Behavior

Shrink when EPs are idle, grow when there is demand – give Kubernetes scheduler a chance to make decisions!



Desired Behavior

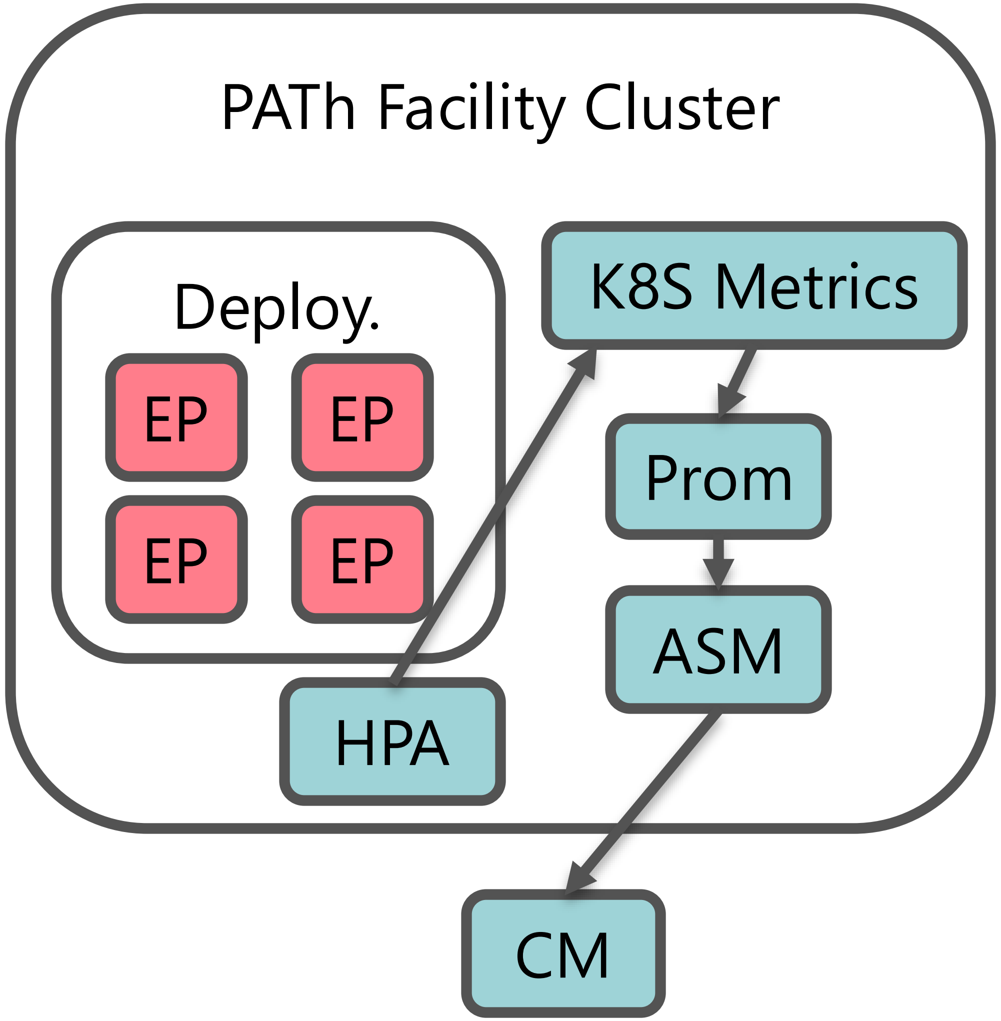
Shrink when EPs are idle, **grow when there is demand** – give Kubernetes scheduler a chance to make decisions!



Two starting models for Kubernetes

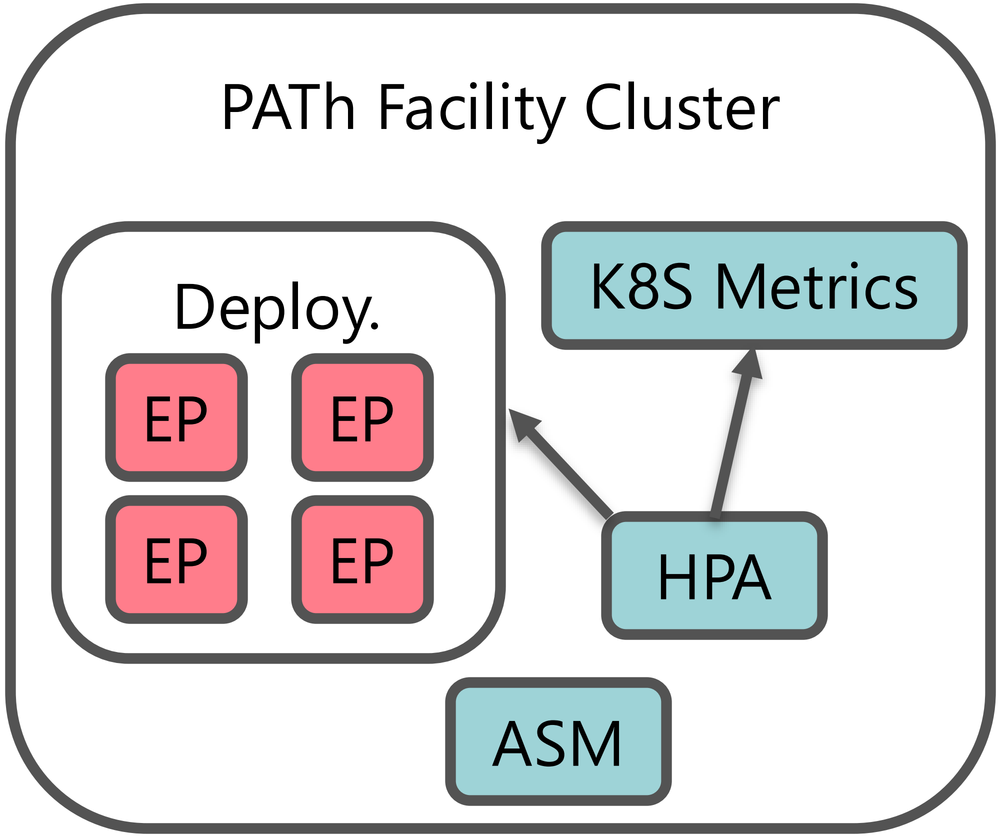
- "**Dynamic deployment**": Use the Kubernetes "**deployment**" object which manages a configured set of identical EP pods.
  - Kubernetes will automatically restart any pod in a deployment that dies.
  - The "Horizontal Pod Autoscaler" (HPA) component will scale the deployment up and down based on need.
- "**Glidein model**": Have a standalone service create a Kubernetes "**job**" / pod.
  - The service creates according to the need it detects.
  - Ephemeral: when the K8S job finishes (or errors), the pod goes away.

Case Study: PATh Facility



- At the PATh Facility cluster, we took the “dynamic deployment” route.
- A central service, the ‘htcondor-autoscale-manager’ (ASM), queries the collector for the state of the EPs and creates an “occupancy metric”:
  - <1 indicates EPs are idle and need to be shut down
  - >1 indicates
- The Prometheus operator scrapes the ASM’s metrics.
  - The Prometheus adapter converts the Prometheus metric into the Kubernetes metric system.

PATh Facility – Scaling up and down



- The Horizontal Pod Autoscaler (HPA) will adjust the number of pods in the deployment to bring the occupancy metric to 1.
- For scale-down, **how does Kubernetes know which pod to preempt?**
  - Every cycle, the ASM calculates a ‘preemption cost’ based on the work that would be lost on preemption.
  - The ASM annotates each pod with the preemption cost.
  - The Kubernetes scheduler will select the lower-cost pod, preempting the idle one.
    - Otherwise, it selects randomly!

PATh Facility - Scaling up

- **Q**: How does the ASM know when a new EP is needed?

  A: Offline ads!
- The ASM will take a snapshot of an EP's slot ad and advertise it to the collector as a "fake" offline slot.
  - Assumption: all EPs in the same deployment are "the same".
- When the negotiator has a match for the offline ad, it will annotate the ad.
  - This annotation says "**I could have used this slot if it was online**"
- During next ASM cycle, it will raise the occupancy metric and a new pod will launch.

Impact: No configuration, no ClassAd expressions; <u>the negotiator does all the work!</u>

PATh Facility – Upsides & Downsides

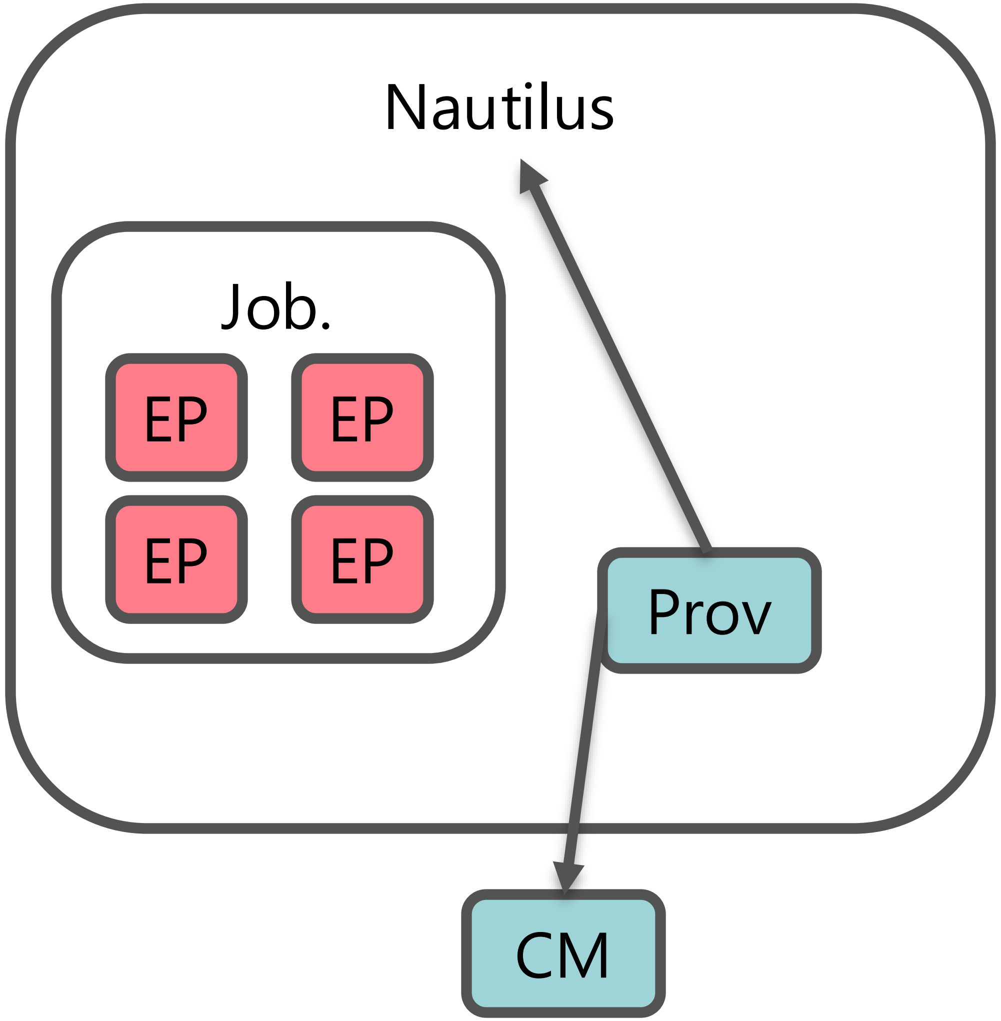
What's wrong with this model?

- **Pod auto-restarts when idle**: bad at releasing resources when they are incorrectly requested.
  - Prevents K8S from pulling updates automatically
- **Requires additional operators**, even in simple configuration:
  - Functionality requires a Prometheus setup and the Prometheus Adapter (latter is installed at the cluster level).
- "Control loop" grows or shrinks a single node at a time. Slow ramp-up for shifting workloads.

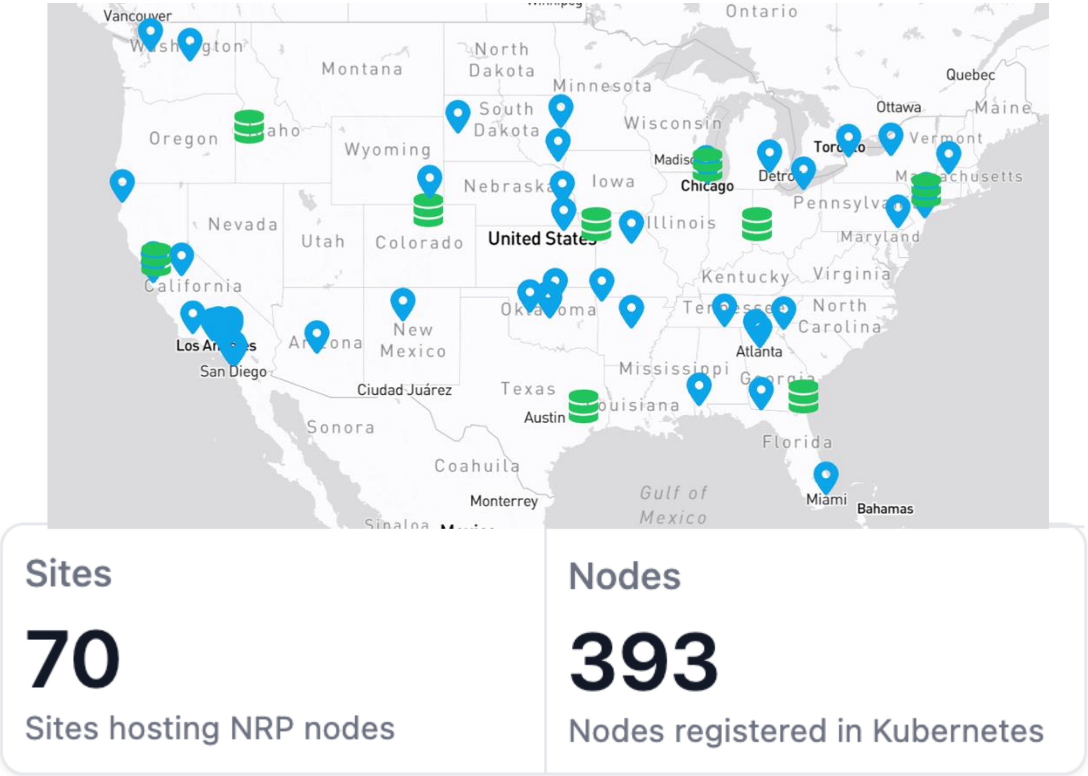
Advantages?

- Minimal ClassAd configuration: relies solely on the negotiator to indicate load.

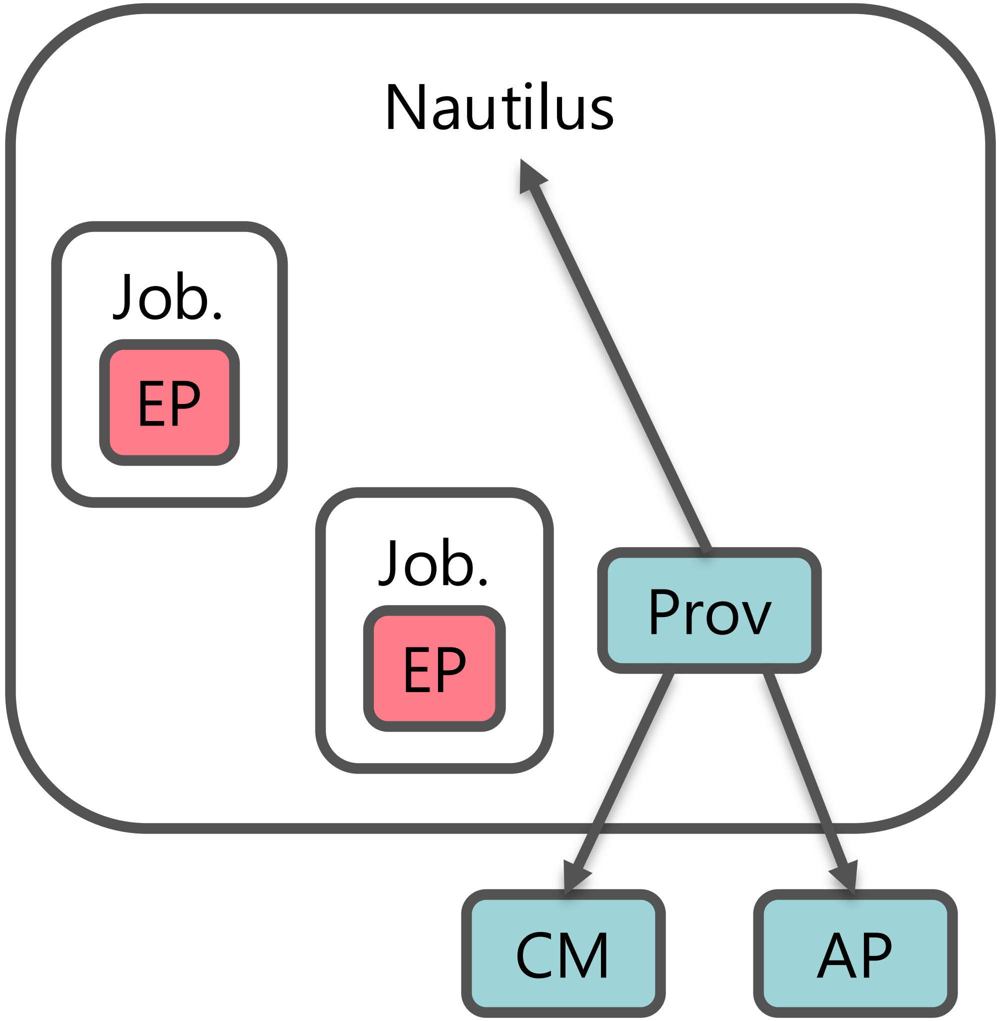
Case Study: National Research Platform



- The National Research Platform (NRP) project operates a stretched Kubernetes cluster, Nautilus, with hosts spanning the nation.



Case Study: National Research Platform



- The NRP has a *[provisioner component]()* that will create new K8S jobs containing an EP pod.
- The provisioner runs a periodic cycle where it (a) determines the current EP states, (b) determines the load, and (c) creates new jobs accordingly.
- When an EP is idle for a fixed period, it'll exit.
  - **Returns resources back to the cluster**.
- Provisioner needs to be configured with the EP image to use and the "needed resource" logic.

NRP: Scaling Up, Scaling Down

- Scaling down is natural:
  - When no job has matched for **X** minutes, the EP shuts off. All resources are cleaned up.
  - Analogous to a pilot / glidein-based system.
- Scaling up is complicated:
  - Provisioner must be configured to query for specific jobs.
  - Based on # of idle jobs, decides to launch new EPs.
  - **Problem**: provisioner query != negotiation. Relies on administrator to hand-write the expressions.
    - If administrator “gets it wrong”, then EP will idle and the resources will be wasted.

Comparisons: PATh Facility vs NRP

- The PATh Facility model is limited by deployment model: difficult to update the container image.
- The NRP model requires the administrator to write expressions matching jobs.
  - Quite difficult to get correct: near-impossible for GPU jobs.

What condor_q query do you write to count jobs if the job requirements are like this:

```
Requirements =
  ((Target.CUDADriverVersion >= 12.1) &&
   (Target.GPUs_GlobalMemoryMb > 45000) &&
   (Target.GPUs_GlobalMemoryMb < 60000)) &&
   …
```

E.g., NRP needs to query for all jobs that could utilize a host with 48GB of GPU memory but the job's GPU memory request is embedded in the Requirements expression.

Looking into the future: A HTCondor Operator

- The HTCSS team is working on its own Kubernetes operator, the “**glidein manager**”.
- Serves as a “CE”: an aggregation point for all EPs within a cluster.
- Purpose-built: aiming to tackle authorization models for the annex.
- Will provide an opportunity to combine the PATh Facility and the NRP models:
  - No penalties of deployment as in PATh Facility.
  - Can use offline ads / negotiatior, avoiding the expressions in the NRP model.
- Glidein Manager will serve as a CE, also managing the creation of the EP and any necessary credentials.

Matt Westphall

Lead developer for the “glidein manager”.

Questions?

FEARLESS SCIENCE

This project is supported by the National Science Foundation under Cooperative Agreements OAC-2030508. Any opinions, findings, conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

MORGRIDGE INSTITUTE FOR RESEARCH