A separated model for running rootless, unprivileged PMIx-enabled HPC applications in Kubernetes

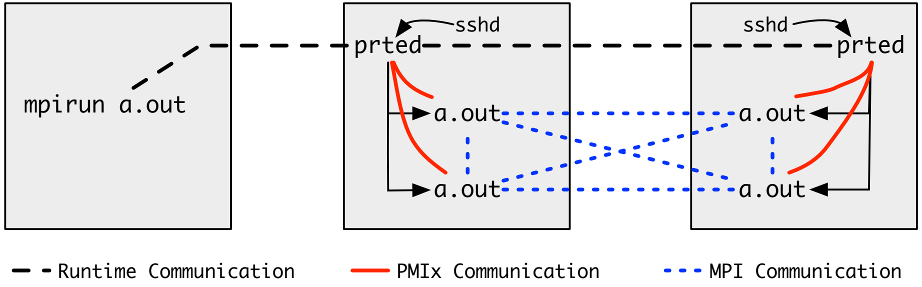
Joshua Hursey
Spectrum MPI Developer
jhursey@us.ibm.com

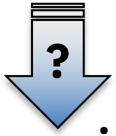
IBM

PMIx ASC 1Q 2023

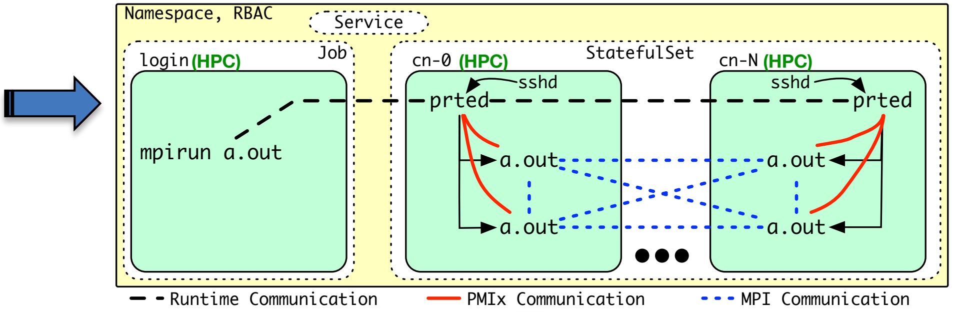
Goal: Moving towards a microservice model for MPI jobs in Kubernetes

Traditional HPC

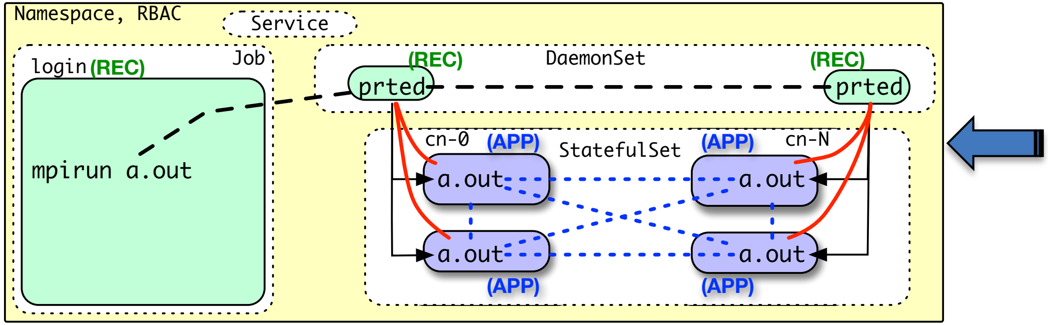




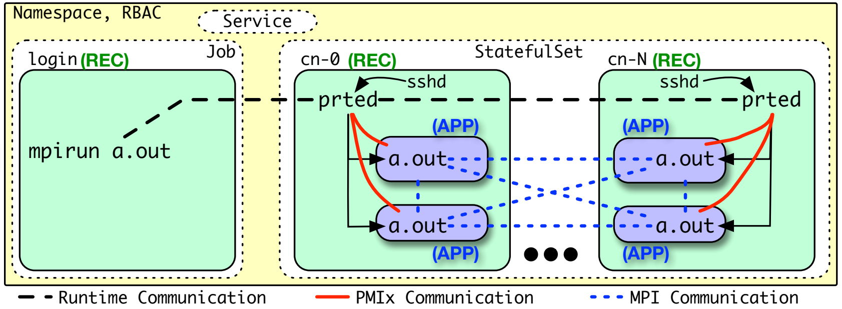
Traditional mapping of HPC to K8s



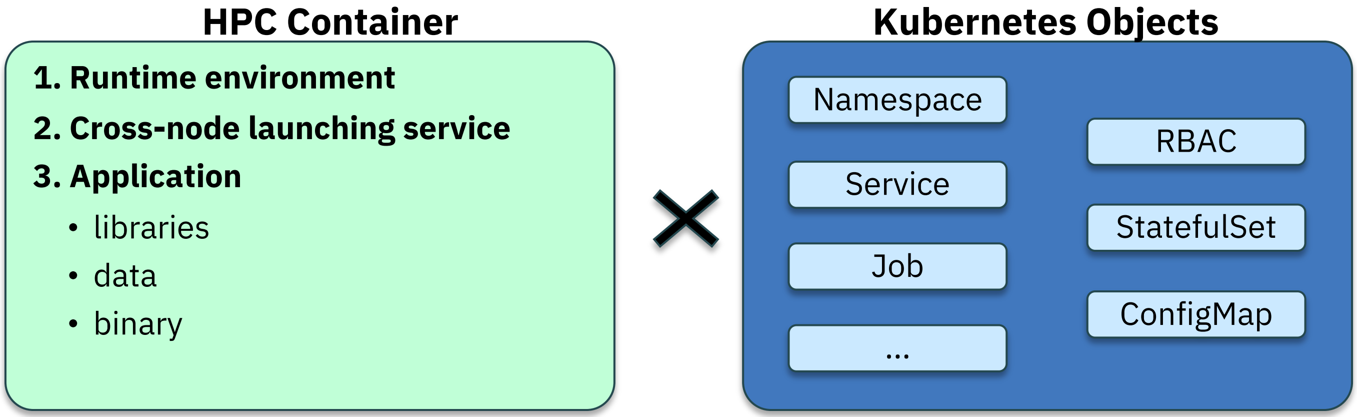
Goal: Microservice(ish) K8s



Separated mapping of HPC to K8s



High Performance Computing In Kubernetes (K8s): Traditional Model



HPC Container

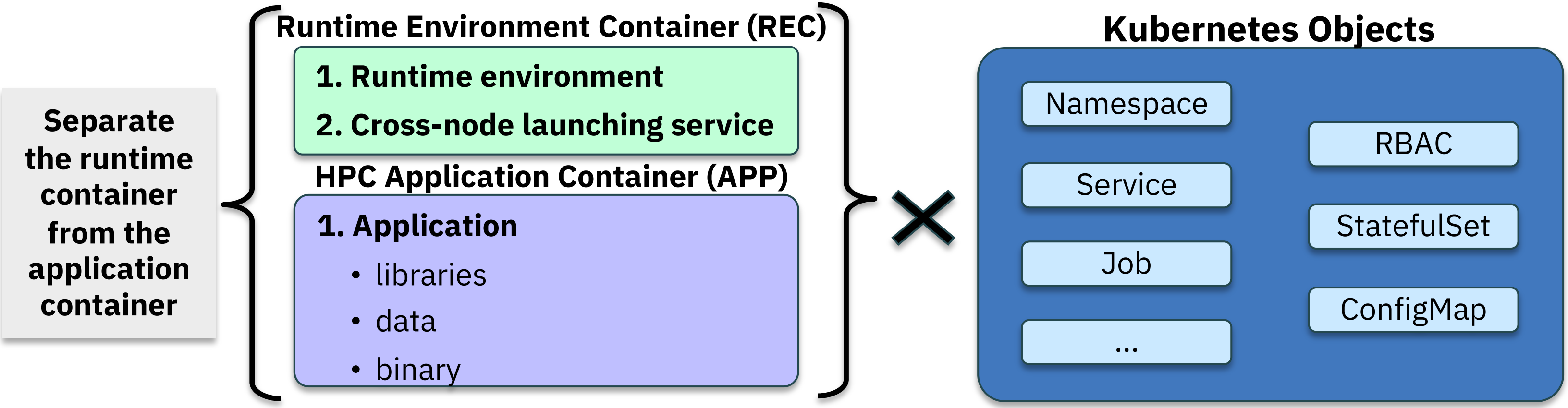
1. **Runtime environment**
2. **Cross-node launching service**
3. **Application**
   - libraries
   - data
   - binary

×

Kubernetes Objects

- **Containerized applications are necessary when running in Kubernetes (K8s)**
- **HPC container must bring a runtime environment in addition to the application**
- **Problem:**
  - Runtime environment and cross-node launching service may need to be customized per K8s system leading to maintaining many similar containers.
  - Container makers must secure the cross-node launching service (often SSH).
  - Running **`sshd`** in a container exposes an attack vector.

High Performance Computing In Kubernetes (K8s): Separated Model



Separate the runtime container from the application container

Runtime Environment Container (REC)

1. Runtime environment
2. Cross-node launching service

HPC Application Container (APP)

1. Application
   - libraries
   - data
   - binary

×

Kubernetes Objects

Namespace

Service

Job

...

RBAC

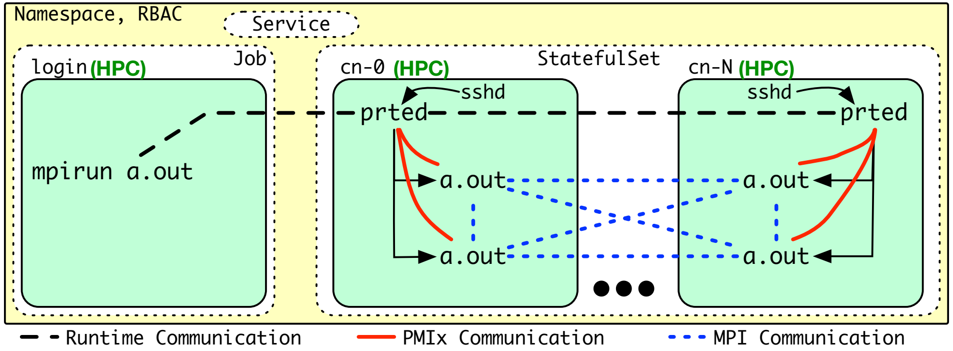
StatefulSet

ConfigMap

- **Separation of concerns**
  - Runtime container is maintained by sysadmins.
  - Application container is runtime agnostic, and smaller.
- **Improved security**
  - Remove ssh from the application container.
- **Improved performance**
  - Smaller application containers, persistent daemons.

The critical point is that the APP creator is no longer responsible for maintaining and configuring the REC portion of the environment which is most prone to security vulnerabilities that could put the host system at risk.

K8s Virtual Cluster



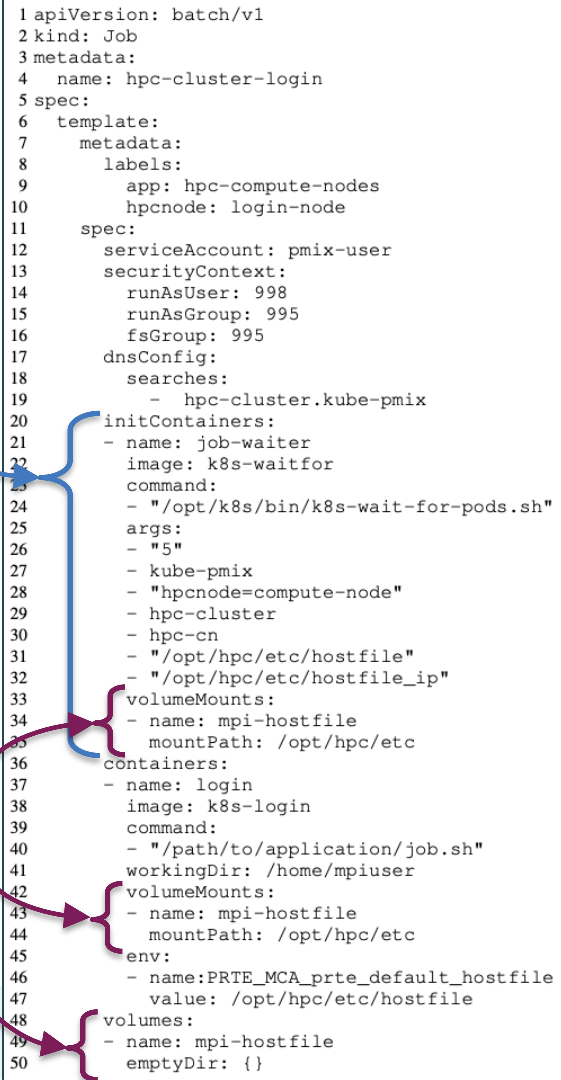
HPC Container

1. Runtime environment
2. Cross-node launching service
3. Application
- libraries
- data
- binary

- **K8s objects establish a *virtual cluster* to support traditional HPC applications**
  - **`Namespace`**: A context used to group the K8s objects
  - **Role-Based Access Control (RBAC):** Access permissions
    - Often consists of the combination of: `ServiceAccount, ClusterRole, ClusterRoleBinding`
  - **Headless `Service`**: Provides DNS services
  - **`StatefulSet`**: Pod set representing compute nodes
  - **`Job`**: A Pod representing a login/launch node from which the app. is launched
    - Not strictly necessary, but is convenient to have as a point of entry

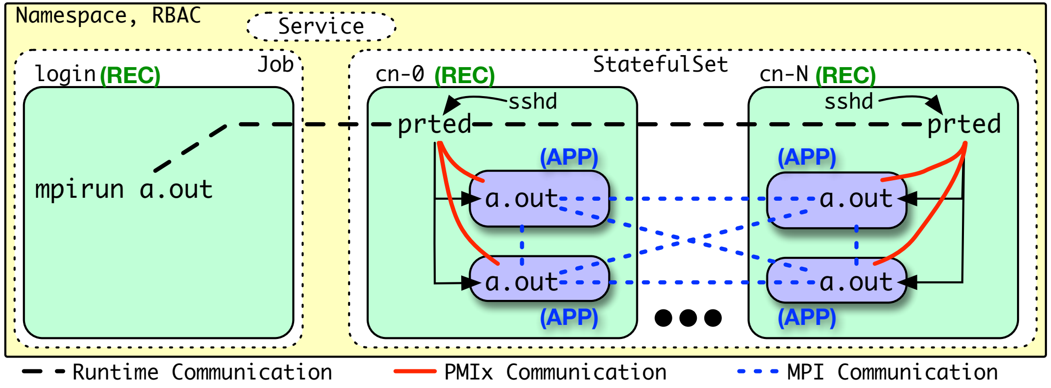
K8s Virtual Cluster: Job

- **Waiting for Pods to be ready with an `initContainers`**
  - A race between when the **`Job`** starts and the **`StatefulSet`** Pods are ready, causing launch failure.
  - Define an `initContainers` that waits for the virtual cluster to be ready before starting the **`Job`** script.
- **Hostfile generation**
  - The runtime needs a hostfile to launch its daemons.
  - A **`ConfigMap`** can be used but it must be created statically before starting the virtual cluster.
  - Alternatively, the `initContainers` can dynamically create the hostfile in a shared mount with the **`Job`**.
  - Hostfile can be a list of hostnames or IP addresses of the **`Pods`** in the overlay network.



```
apiVersion: batch/v1
kind: Job
metadata:
  name: hpc-cluster-login
spec:
  template:
    metadata:
      labels:
        app: hpc-compute-nodes
        hpcnode: login-node
    spec:
      serviceAccount: pmix-user
      securityContext:
        runAsUser: 998
        runAsGroup: 995
        fsGroup: 995
      dnsConfig:
        searches:
          -  hpc-cluster.kube-pmix
      initContainers:
      - name: job-waiter
        image: k8s-waitfor
        command:
        - "/opt/k8s/bin/k8s-wait-for-pods.sh"
        args:
        - "5"
        - kube-pmix
        - "hpcnode=compute-node"
        - hpc-cluster
        - hpc-cn
        - "/opt/hpc/etc/hostfile"
        - "/opt/hpc/etc/hostfile_ip"
        volumeMounts:
        - name: mpi-hostfile
          mountPath: /opt/hpc/etc
      containers:
      - name: login
        image: k8s-login
        command:
        - "/path/to/application/job.sh"
        workingDir: /home/mpiuser
        volumeMounts:
        - name: mpi-hostfile
          mountPath: /opt/hpc/etc
        env:
        - name:PRTE_MCA_prte_default_hostfile
          value: /opt/hpc/etc/hostfile
      volumes:
      - name: mpi-hostfile
        emptyDir: {}
```

Separated Model: Nested Containers



Runtime Environment Container (REC)

1. Runtime environment

2. Cross-node launching service

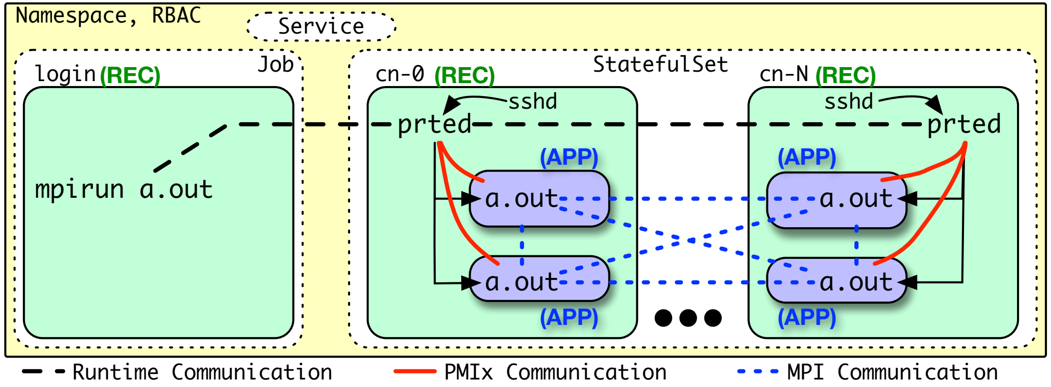
HPC Application Container (APP)

1. Application

- libraries
- data
- binary

- **The REC is started in each Pod. The REC starts the APP nested within itself.**
- **Two approaches for nesting containers**
  1. Expose the host container runtime socket to the REC (may have `root`).
     - Potential attack vector from the APP -> REC -> Host via the exposed socket
  2. Install a rootless container runtime inside the REC (e.g., Podman).
- Rootless container runtimes require a form of **privileged access** to mount file systems when starting a container.

Separated Model: Nested Containers



Runtime Environment Container (REC)

1. Runtime environment

2. Cross-node launching service

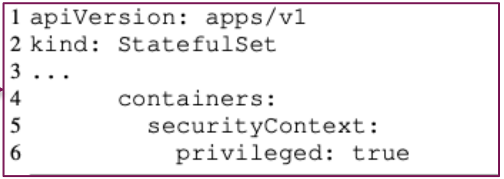
HPC Application Container (APP)

1. Application

- libraries
- data
- binary

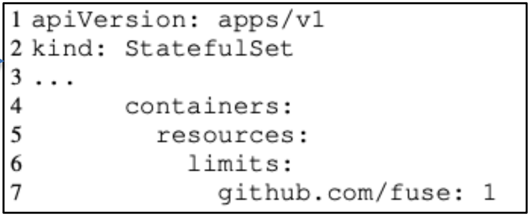
– **Two techniques for handling privileged access in K8s for rootless containers.**

1. Make the REC **`Pod`** fully privileged.
   - Potential attack vector from APP -> REC -> Host



```
apiVersion: apps/v1
kind: StatefulSet
...
      containers:
        securityContext:
          privileged: true
```

2. Use a **`DevicePlugin`** to expose the **`/dev/fuse`** file system into the REC **`Pod`**.
   - https://github.com/jjhursey/fuse-device-plugin/tree/ppc64le-support

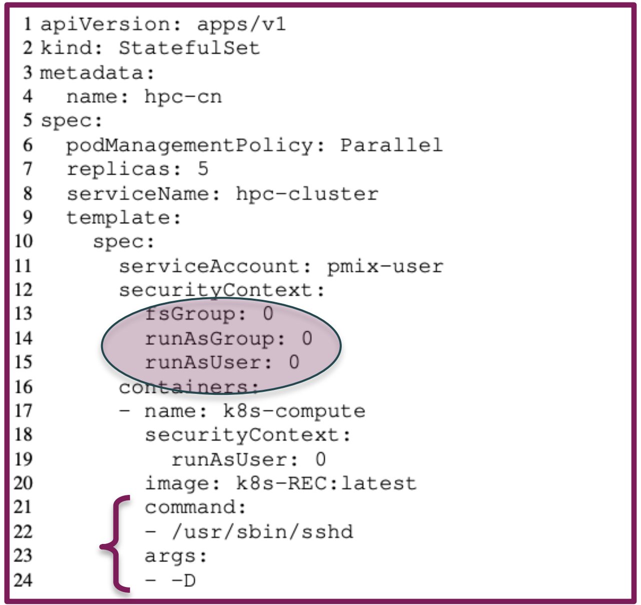


```
apiVersion: apps/v1
kind: StatefulSet
...
      containers:
        resources:
          limits:
            github.com/fuse: 1
```

Separated Model: Cross-Node Launching

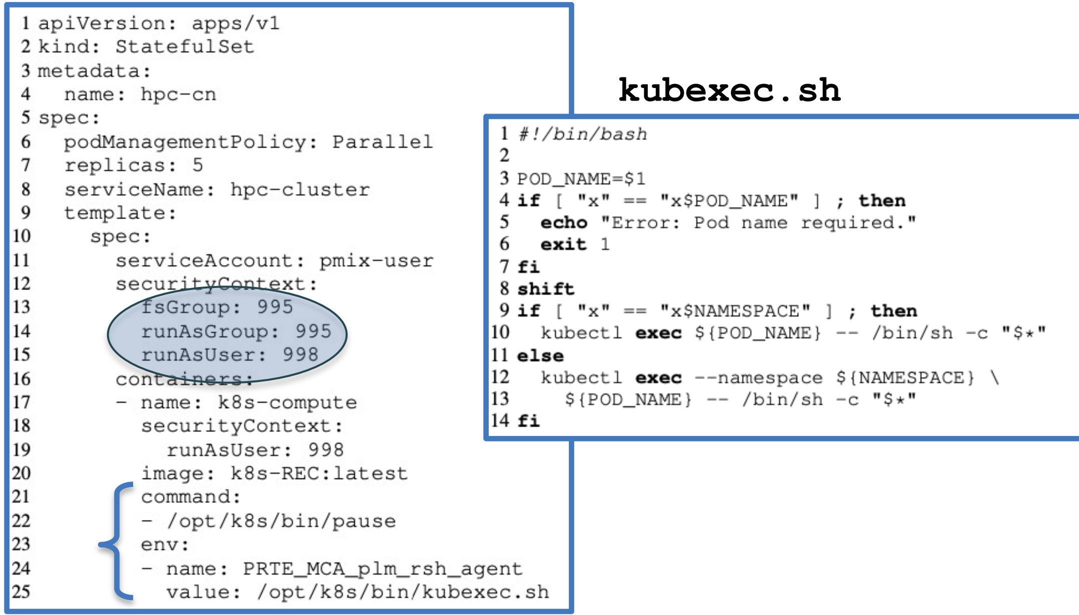
– **The REC needs to start a runtime daemon in each Pod to establish the runtime environment for the application.**

SSH Daemon



```
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: hpc-cn
spec:
  podManagementPolicy: Parallel
  replicas: 5
  serviceName: hpc-cluster
  template:
    spec:
      serviceAccount: pmix-user
      securityContext:
        fsGroup: 0
        runAsGroup: 0
        runAsUser: 0
      containers:
      - name: k8s-compute
        securityContext:
          runAsUser: 0
        image: k8s-REC:latest
        command:
        - /usr/sbin/sshd
        args:
        - -D
```

kubectl exec



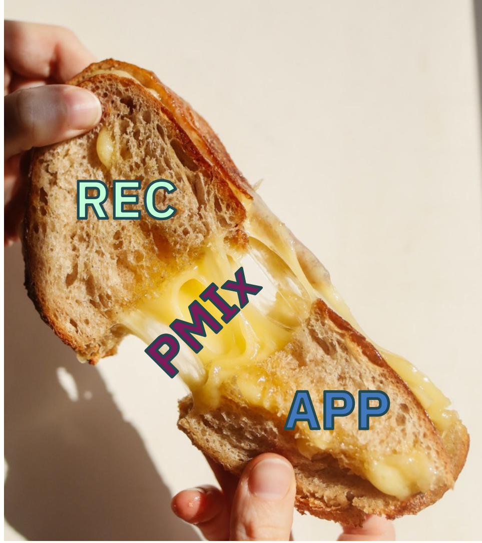
```
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: hpc-cn
spec:
  podManagementPolicy: Parallel
  replicas: 5
  serviceName: hpc-cluster
  template:
    spec:
      serviceAccount: pmix-user
      securityContext:
        fsGroup: 995
        runAsGroup: 995
        runAsUser: 998
      containers:
      - name: k8s-compute
        securityContext:
          runAsUser: 998
        image: k8s-REC:latest
        command:
        - /opt/k8s/bin/pause
        env:
        - name: PRTE_MCA_plm_rsh_agent
          value: /opt/k8s/bin/kubexec.sh
```

kubexec.sh

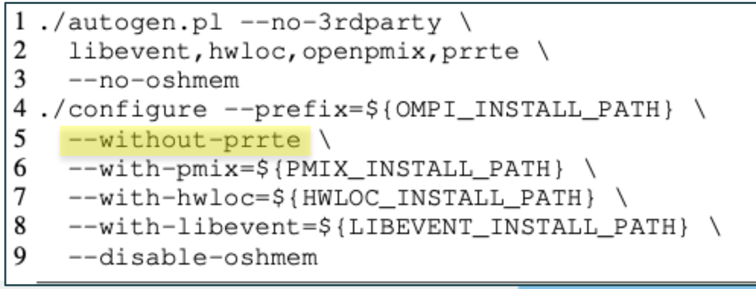
```
#!/bin/bash

POD_NAME=$1
if [ "x" == "x$POD_NAME" ] ; then
  echo "Error: Pod name required."
  exit 1
fi
shift
if [ "x" == "x$NAMESPACE" ] ; then
  kubectl exec ${POD_NAME} -- /bin/sh -c "$*"
else
  kubectl exec --namespace ${NAMESPACE} \
    ${POD_NAME} -- /bin/sh -c "$*"
fi
```

Separated Model: PMIx

- **The REC and APP need to communicate to support operations like inter-process wire-up and dynamics.**
- **PMIx library serves this purpose over a socket.**
  - REC contains an instance of the **PMIx Server** library
  - APP contains an instance of the **PMIx Client** library
- **The PMIx library must be either the same or cross-version compatible in the REC and APP.**
  - OpenPMIx implementation is committed to cross-version compatibility.



- **Open MPI "main," and "v5.0.x" branches allow for users to build Open MPI without a runtime environment.**

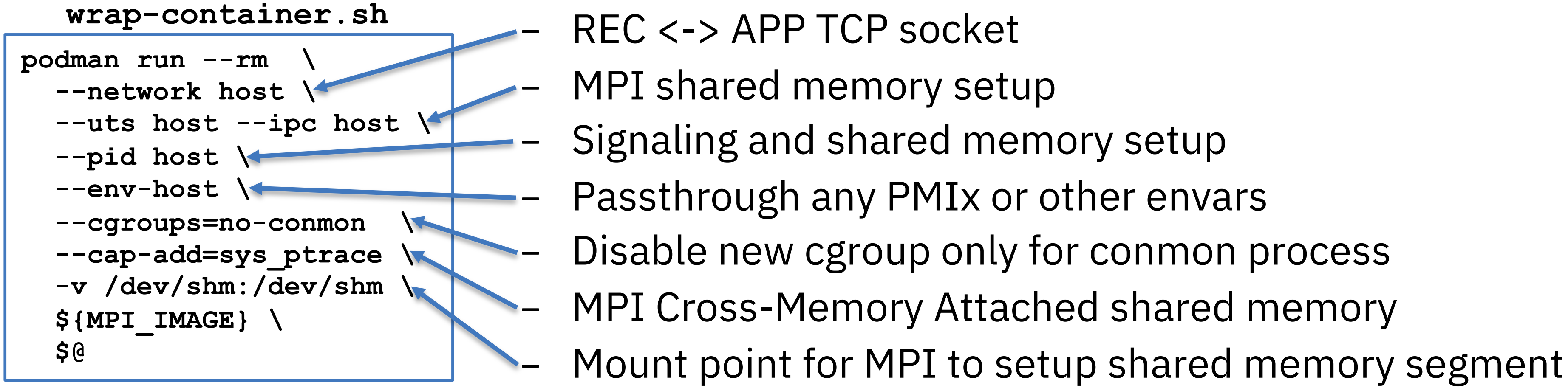


```
./autogen.pl --no-3rdparty \
  libevent,hwloc,openpmix,prrte \
  --no-oshmem
./configure --prefix=${OMPI_INSTALL_PATH} \
  --without-prrte \
  --with-pmix=${PMIX_INSTALL_PATH} \
  --with-hwloc=${HWLOC_INSTALL_PATH} \
  --with-libevent=${LIBEVENT_INSTALL_PATH} \
  --disable-oshmem
```

Separated Model: Launching the APP container

- **PMIx Reference Runtime Environment (PRRTE) is installed in the REC and the prterun launch command is used to start the APP containers.**
  - A wrapper script hides the Podman complexities from the user and PRRTE.
  - No changes were needed in PRRTE to support this model.

```
REC$ prterun --map-by ppr:2:node -x MPI_IMAGE wrap-container.sh /examples/init_finalize
```



wrap-container.sh

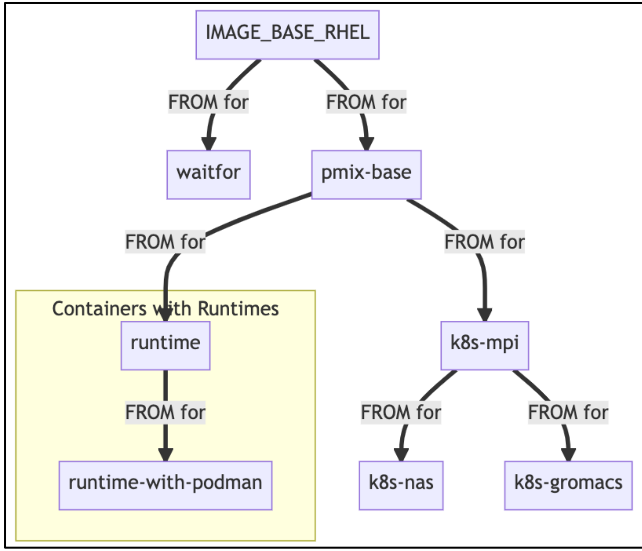
```
podman run --rm  \
  --network host \
  --uts host --ipc host \
  --pid host \
  --env-host \
  --cgroups=no-conmon  \
  --cap-add=sys_ptrace \
  -v /dev/shm:/dev/shm \
  ${MPI_IMAGE} \
  $@
```

- REC <-> APP TCP socket
- MPI shared memory setup
- Signaling and shared memory setup
- Passthrough any PMIx or other envars
- Disable new cgroup only for conmon process
- MPI Cross-Memory Attached shared memory
- Mount point for MPI to setup shared memory segment

Demo: Environment

https://github.com/jjhursey/pub-2022-CANOPIE-HPC

- **System setup**
  - 18 IBM Power8 machines with Infiniband CX-5 and 2x NVIDIA P100 GPUs
  - Kubernetes 1.23.4 (installed via kubeadm)
  - Device Plugins: Mellanox RDMA [1], NVIDIA [2], FUSE [3]
  - **Host**: RHEL 8.4, Docker 20.10.13
  - **REC**: RHEL 8.6, Podman 4.0.2, PRRTE 'master'
- **Containers**: All with OpenPMIx 'master'
  - **REC**: **`runtime-with-podman`**
  - **APP**: All with Open MPI 'main'
    - **`k8s-mpi`**: Basic MPI examples
    - **`k8s-nas`**: NAS Parallel Benchmarks
    - **`k8s-gromacs`**: Gromacs molecular dynamics



[1] https://github.com/mellanox/k8s-rdma-shared-dev-plugin
[2] https://github.com/NVIDIA/k8s-device-plugin
[3] https://github.com/jjhursey/fuse-device-plugin/tree/ppc64le-support

Demo: Launching the virtual cluster

https://github.com/jjhursey/pub-2022-CANOPIE-HPC

```
shell$ make deploy-ssh-with-podman-unpriv
...
```

```
shell$ kubectl get all
NAME                              READY   STATUS    RESTARTS   AGE
pod/hpc-cluster-login-cnt9r       1/1     Running   0          41s
pod/hpc-cn-0                      1/1     Running   0          41s
pod/hpc-cn-1                      1/1     Running   0          41s
pod/hpc-cn-2                      1/1     Running   0          41s
pod/hpc-cn-3                      1/1     Running   0          41s
pod/hpc-cn-4                      1/1     Running   0          41s

NAME                  TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)   AGE
service/hpc-cluster   ClusterIP   None         <none>        <none>    42s

NAME                          READY   AGE
statefulset.apps/hpc-cn       5/5     42s

NAME                              COMPLETIONS   DURATION   AGE
job.batch/hpc-cluster-login       0/1           41s        41s
```

```
shell$ make login-ssh-with-podman-unpriv
REC$ id
uid=998(mpiuser) gid=995(mpiuser) groups=995(mpiuser)
```

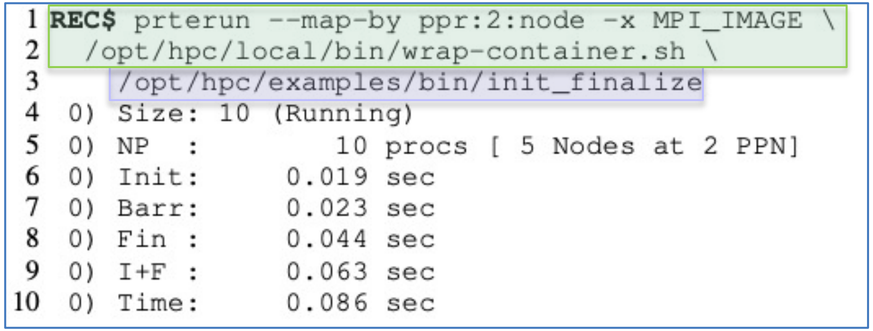
```
shell$ make undeploy-ssh-with-podman-unpriv
...
```

Demo: Applications

https://github.com/jjhursey/pub-2022-CANOPIE-HPC

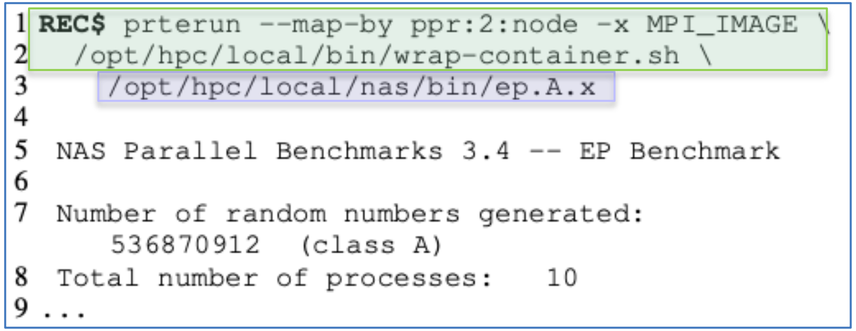
```
shell$ make login-ssh-with-podman-unpriv
REC$ id
uid=998(mpiuser) gid=995(mpiuser) groups=995(mpiuser)
REC$ cat $PRTE_MCA_prte_default_hostfile
192.168.85.141
192.168.85.42
192.168.135.156
192.168.53.138
192.168.95.120
```

k8s-mpi



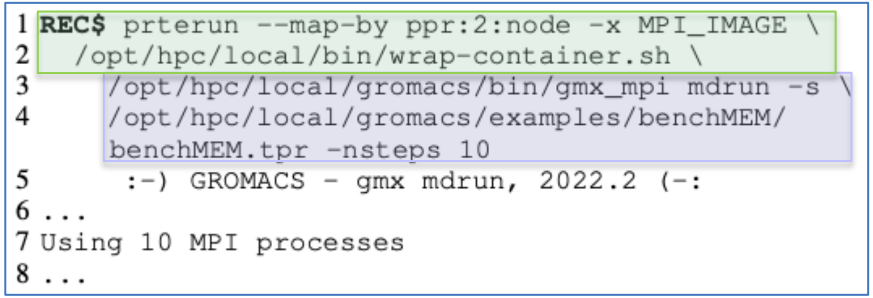
```
REC$ prterun --map-by ppr:2:node -x MPI_IMAGE \
  /opt/hpc/local/bin/wrap-container.sh \
     /opt/hpc/examples/bin/init_finalize
  0) Size: 10 (Running)
  0) NP   :          10 procs [ 5 Nodes at 2 PPN]
  0) Init:       0.019 sec
  0) Barr:       0.023 sec
  0) Fin :       0.044 sec
  0) I+F :       0.063 sec
  0) Time:       0.086 sec
```

k8s-nas



```
REC$ prterun --map-by ppr:2:node -x MPI_IMAGE \
  /opt/hpc/local/bin/wrap-container.sh \
    /opt/hpc/local/nas/bin/ep.A.x

 NAS Parallel Benchmarks 3.4 -- EP Benchmark

 Number of random numbers generated:
      536870912  (class A)
 Total number of processes:    10
...
```

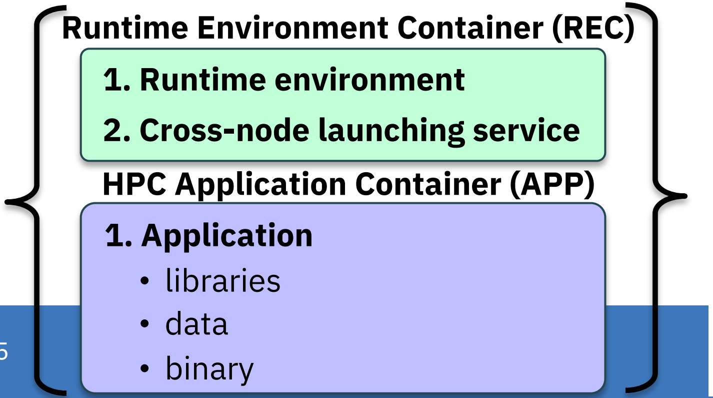
k8s-gromacs



```
REC$ prterun --map-by ppr:2:node -x MPI_IMAGE \
  /opt/hpc/local/bin/wrap-container.sh \
    /opt/hpc/local/gromacs/bin/gmx_mpi mdrun -s \
    /opt/hpc/local/gromacs/examples/benchMEM/
    benchMEM.tpr -nsteps 10
     :-) GROMACS - gmx mdrun, 2022.2 (-:
...
Using 10 MPI processes
...
```

Conclusions

- **In Kubernetes, HPC containers must include a runtime environment and cross-node launching service in addition to their application.**
  - Increases maintenance burden on the container maker.
  - Introduces security vulnerabilities into the k8s environment.
- **Separated model separates the runtime portion from the application portion**
  - Separation of concerns: Sysadmin maintains REC, User maintains APP.
  - Improved security: Removing ssh from the APP, maybe REC as well.
  - Improved performance: Smaller APP, persistent daemons



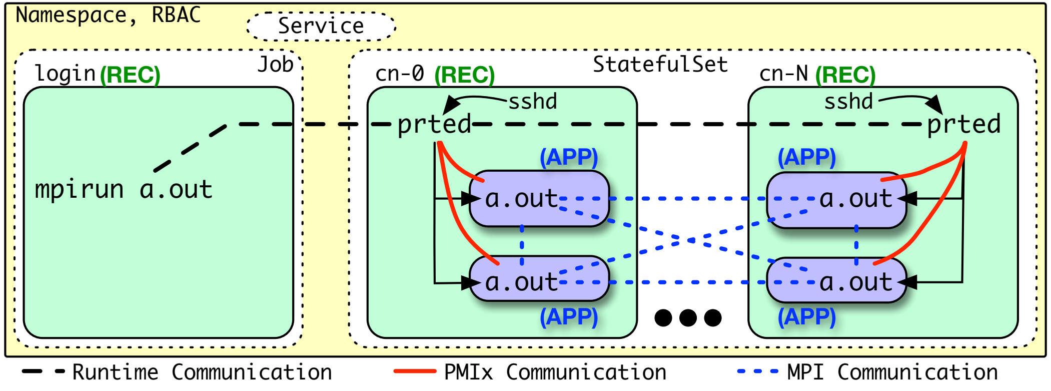
Runtime Environment Container (REC)

1. Runtime environment

2. Cross-node launching service

HPC Application Container (APP)

1. Application
- libraries
- data
- binary



Future work

- **Consolidate this effort into a K8s operator**
  - Advancement on current CRDs (Kubeflow, Volcano) since it uses the separated model and would support more than just MPI applications.
- **Alternative to ssh and kubectl for cross-node launching**
  - Interact more natively with the K8s environment
  - Start the runtime daemons in each **`Pod`** as soon as the **`Pod`** starts running.
  - Have the daemon "phone home" instead of being launched by prterun.
- **Runtime independent application launcher command**
  - Currently, the job script must know the launcher command (**`prterun, mpirun, srun, jsrun`**) installed in the REC to launch their job.
  - Develop a runtime independent tool that uses **`PMIx_spawn`** to start the application against whatever runtime is installed in the REC.
  - Will allow the application job script to be runtime launcher agnostic.

Thank you.