Kubernetes

Below are the yaml documents which describe the bare minimum infrastructure needed for a functional Flow server. It is best to start with a single-node proof of concept deployment. Once that works, the deployment can be extended to multi-node with elastic worker allocation. Each section is explained below.

The Flow headnode pod

```
apiVersion: v1
kind: Pod
metadata:
  name: flowheadnode
  namespace: partek-flow
  labels:
    app.kubernetes.io/name: flowheadnode
    deployment: dev
spec:
  securityContext:
    fsGroup: 1000
  containers:
    - name: flowheadnode
      image: xxxxxxxxxxxx.dkr.ecr.us-west-2.amazonaws.com/partek-flow:current-23.0809.22
      resources:
        requests:
          memory: "16Gi"
          cpu: 8
      env:
        - name: PARTEKLM_LICENSE_FILE
          value: "@flexlmserver"
        - name: PARTEK_COMMON_NO_TOTAL_LIMITS
          value: "1"
        - name: CATALINA_OPTS
          value: "-DFLOW_WORKER_MEMORY_MB=1024 -DFLOW_WORKER_CORES=2 -Djavax.net.ssl.trustStore=/etc/flowconfig
/cacerts -Xmx14g"
      volumeMounts:
        - name: home-flow
          mountPath: /home/flow
        - name: flowconfig
          readOnly: true
          mountPath: "/etc/flowconfig"
  volumes:
    - name: home-flow
      persistentVolumeClaim:
        claimName: partek-flow-pvc
    - name: flowconfig
      secret:
        secretName: flowconfig
```

Pod metadata

On a kubernetes cluster, all Flow deployments are placed in their own namespace, for example *namespace: partek-flow*. The label *app.kubernetes.io/name : flowheadnode* allows binding of a service or used to target other kubernetes infrastructure to this headnode pod. The label *deployment: dev* allows running multiple Flow instances in this namespace (dev, tst, uat, prd, etc) if needed and allows workers to connect to the correct headnode. For stronger isolation, running each Flow instance in its own namespace is optimal.

Data storage

The Flow docker image requires 1) a writable volume mounted to */home/flow* 2) This volume needs to be readable and writable by *UID:GID 1000:1000* 3) For a multi-node setup, this volume needs to be cross mounted to all worker pods. In this case, the persistent volume would be backed by some network storage device such as EFS, NFS, or a mounted FileGateway.

This section achieves goal 2)

```
spec:
  securityContext:
    fsGroup: 1000
```

The *flowconfig* volume is used to override behavior for custom Flow builds and custom integrations. It is generally not needed for vanilla deployments.

The Flow docker image:

Partek Flow is shipped as a single docker image containing all necessary dependencies. The same image is used for worker nodes. Most deployment-related configuration is set as environment variables. Auxiliary images are available for additional supporting infrastructure, such as flexlm and worker allocator images.

Official Partek Flow images can be found on our release notes page: Release Notes
The image tags assume the format: *registry.partek.com/rtw:YY.MMMM.build*
New rtw images are generally released several times a month.
The image in the example above references a *private* ECR. It is highly recommended that the target image from registry.partek.com be loaded into your ECR. Image pulls will be much faster from AWS - this reduces the time to dynamically allocate workers. It also removes a single point of failure - if registry.partek.com were down it would impact your ability to launch new workers on demand.

Flow headnode resource request

Partek Flow uses the head node to handle all interactive data visualization. Additional CPU resources are needed for this, the more the better and 8 is a good place to start. As for memory, we recommend 8 to 16 GiB. Resource limits are not included here, but are set to large values globally:

```
# This allows us to create pods with only a request set, but not a limit set. Further tuning is recommended.
apiVersion: v1
kind: LimitRange
metadata:
  name: partek-flow-limit-range
spec:
  limits:
    - max:
        memory: 512Gi
        cpu: 64
      default:
        memory: 512Gi
        cpu: 64
      defaultRequest:
        memory: 4Gi
        cpu: 2
      type: Container
```

Relevant Flow headnode environment variables

`PARTEKLM_LICENSE_FILE`

Partek Flow uses FlexLM for licensing. Currently we do not offer or have implemented any alternative. Values for this environment variable can be:

@flexlmserveraddress - an external flexlm server. We provide a Partek specific container image and detail a kubernetes deployment for this below. This license server can also live outside the kubernetes cluster - the only requirement being that it is network accessible.
/home/flow/.partekflow/license/Partek.lic - Use this path exactly. This path is internal to the headnode container and is persisted on a mounted PVC.

Unfortunately, FlexLM is MAC address based and does not quite fit in with modern containerized deployments. There is no straightforward or native way for kubernetes to set the MAC address upon pod/container creation, so using a license file on the flowheadnode pod (*/home/flow/.partekflow/license/Partek.lic*) could be problematic (but not impossible). In further examples below, we provide a custom FlexLM container that can be instantiated as a pod/service. This works by creating a new network interface with the requested MAC address inside the FlexLM pod.

`PARTEK_COMMON_NO_TOTAL_LIMITS`

Please leave this set at "1". Partek Flow need not enforce any limits as that is the responsibility of kubernetes. Setting this to anything else may result in Partek executables hanging.

`CATALINA_OPTS`

This is a hodgepodge of Java/Tomcat options. Parts of interest:

-DFLOW_WORKER_MEMORY_MB=1024 -DFLOW_WORKER_CORES=2 - It is possible for the Flow headnode to execute jobs locally in addition to dispatching them to remote workers. These two options set resource limits on the Flow internal worker to prevent resource contention with the Flow server. If remote workers are not used and this remains a single-node deployment, meaning ALL jobs will execute on the internal worker, then it is best to remove the CPU limit (*-DFLOW_WORKER_CORES)* and only set *-DFLOW_WORKER_MEMORY_MB* equal to the kubernetes memory resource request.

-Djavax.net.ssl.trustStore=/etc/flowconfig/cacerts - If Flow connects to a corporate LDAP server for authentication, it will need to trust the LDAP certificates.

-Xmx14g - JVM heap size. If the internal worker is not used, set this to be a little less than the kubernetes memory resource request. If the internal worker is an use, and the intent is to stay with a single-node deployment, then set this to be ~ 25% of the kubernetes memory resource request, but no less than ~ 4 GiB.

The Flow headnode service definition

```
apiVersion: v1
kind: Service
metadata:
  name: flowheadnode
spec:
  type: ClusterIP
  ports:
    - port: 80
      targetPort: 8080
      protocol: TCP
      name: http
    - port: 2552
      targetPort: 2552
      protocol: TCP
      name: akka
    - port: 8443
      targetPort: 8443
      protocol: TCP
      name: licensing
  selector:
    app.kubernetes.io/name: flowheadnode
```

The flowheadnode service is needed 1) so that workers have a DNS name (*flowheadnode*) to connect to when they start and 2) so that we can attach an ingress route to make the Flow web interface accessible to end users. The *app.kubernetes.io/name: flowheadnode* selector is what binds this to the flowheadnode pod.

80:8080 - Users interact with Flow entirely over a web browser
2552:2552 - Workers communicate with the Flow server over port 2552
8443:8443 - Partek executed binaries connect back to the Flow server over port 8443 to do license checks

Ingress to flowheadnode

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: flowheadnode
  annotations:
    kubernetes.io/ingress.class: "nginx"
    nginx.ingress.kubernetes.io/force-ssl-redirect: "true"
spec:
  rules:
    - host: flow.dev-devsvc.domain.com
      http:
        paths:
          - path: /
            pathType: Prefix
            backend:
              service:
                name: flowheadnode
                port:
                  number: 80
```

This provides external users HTTPS access to Flow at *host: flow.dev-devsvc.domain.com* Your details will vary. This is where we bind to the *flowheadnode* service.

The flexlm service pod

```
# On a NEW deployment, you need to exec into this pod and add the license file
# to /usr/local/flexlm/licenses
# After a license file is present, the flexlm daemon will start automatically

apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: flexlmserver-pvc
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 10Gi     # flex.log is the only thing that slowly grows here
  storageClassName: gp2-ebs-sc
  volumeMode: Filesystem
---
apiVersion: v1
kind: Service
metadata:
  name: flexlmserver
spec:
  type: ClusterIP
  ports:
    - port: 27000
      targetPort: 27000
      protocol: TCP
      name: flexmain
    - port: 27001
      targetPort: 27001
      protocol: TCP
      name: flexvendor
  selector:
    app.kubernetes.io/name: flexlmserver
---
apiVersion: v1
kind: Pod
metadata:
  name: flexlmserver
  namespace: partek-flow
  labels:
    app.kubernetes.io/name: flexlmserver
spec:
  containers:
    - name: flexlmserver
      image: public.ecr.aws/partek-flow/kube-flexlm-server
      ports:
        - containerPort: 27000
        - containerPort: 27001
      resources:
        limits:
          memory: "256Mi"
          cpu: 1
      securityContext:
        capabilities:
          add: ["NET_ADMIN"]
      volumeMounts:
        - name: flexlmserver-pvc
          mountPath: /usr/local/flexlm/licenses
  volumes:
    - name: flexlmserver-pvc
      persistentVolumeClaim:
        claimName: flexlmserver-pvc
```

The yaml documents above will bring up a complete Partek-specific license server.

Note that the service name is *flexlmserver*. The flowheadnode pod connects to this license server via the PARTEKLM_LICENSE_FILE="@flexlmserver" en vironment variable.

You should deploy this flexlmserver first, since the flowheadnode will need it available in order to start in a licensed state.

Partek will send a Partek.lic file licensed to some random MAC address. When this license is (manually) written to /usr/local/flexlm/licenses, the pod will continue execution by creating a new network interface using the MAC address in Partek.lic, then it will start the licensing service. This is why the *NET_AD MIN* capability is added to this pod.

The license from Partek must contain *VENDOR parteklm PORT=27001* so the vendor port remains at 27001 in order to match the service definition above. Without this, this port is randomly set by FlexLM.

This image is currently available from *public.ecr.aws/partek-flow/kube-flexlm-server* but this may change in the future.