Hewlett Packard Enterprise

CRAY SYSTEM MANAGEMENT FOR HPE CRAY EX SYSTEMS

Harold Longley

CSM User Experience Solutions Architect

CUG 2022, May 2-5, 2022

AGENDA

HPE CRAY EX SYSTEM OVERVIEW

MANAGEMENT SERVICES

WHAT IS HAPPENING ON MY SYSTEM?

MANAGING USER ENVIRONMENTS

RESOURCES

HPE CRAY EX SYSTEM OVERVIEW

- CSM Architecture
- HPE Cray EX Hardware
- Networks
- Continuous Operations
- Kubernetes
- Ceph
- Etcd
- Istio Service Mesh and API gateway
- Authentication and Authorization

CSM ARCHITECTURE

HPE CRAY SYSTEM MANAGEMENT FOR EXASCALE SUPERCOMPUTERS

Manage and extend Exascale supercomputer system management capabilities

Resilient, elastic, scalable systems management solution designed using extensible microservices cloud stack

Powerful
Comprehensive set of tools you need to manage all aspects of your Cray EX Supercomputer

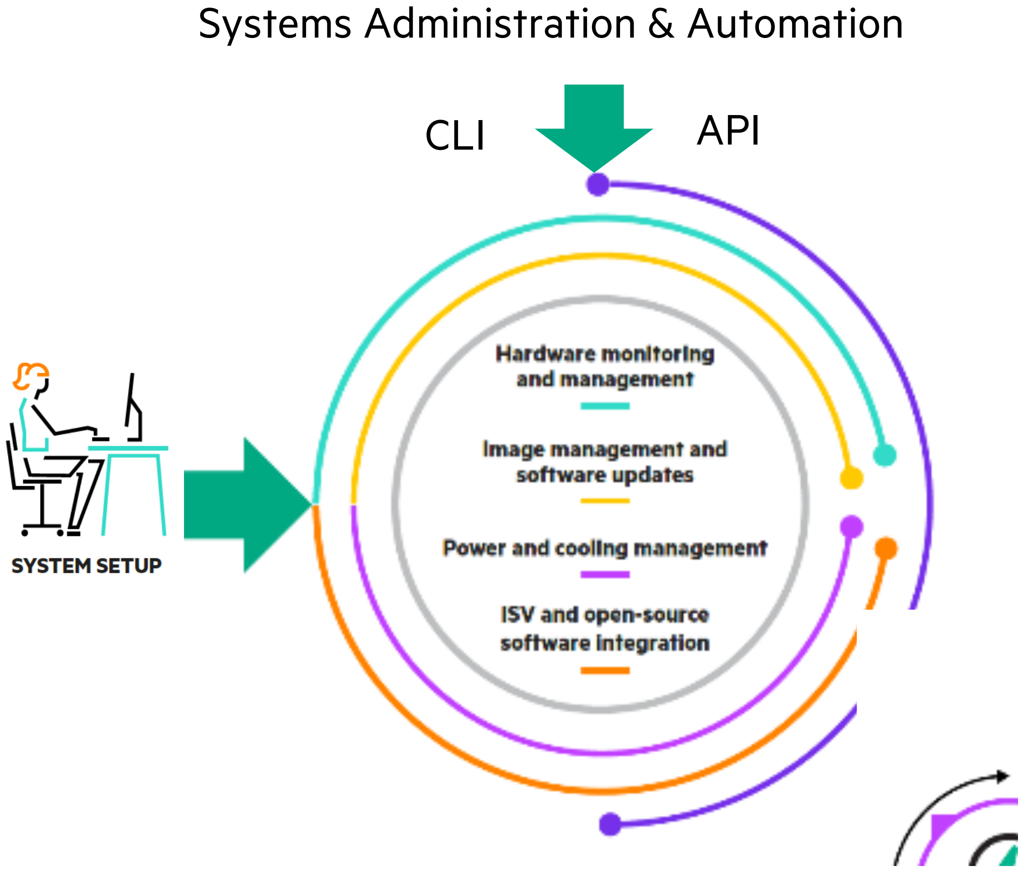
Productive
Designed to maximize productivity of your HPC system, automate actions, and optimize running costs

Secure
Support customizable role-based access control for systems management administration

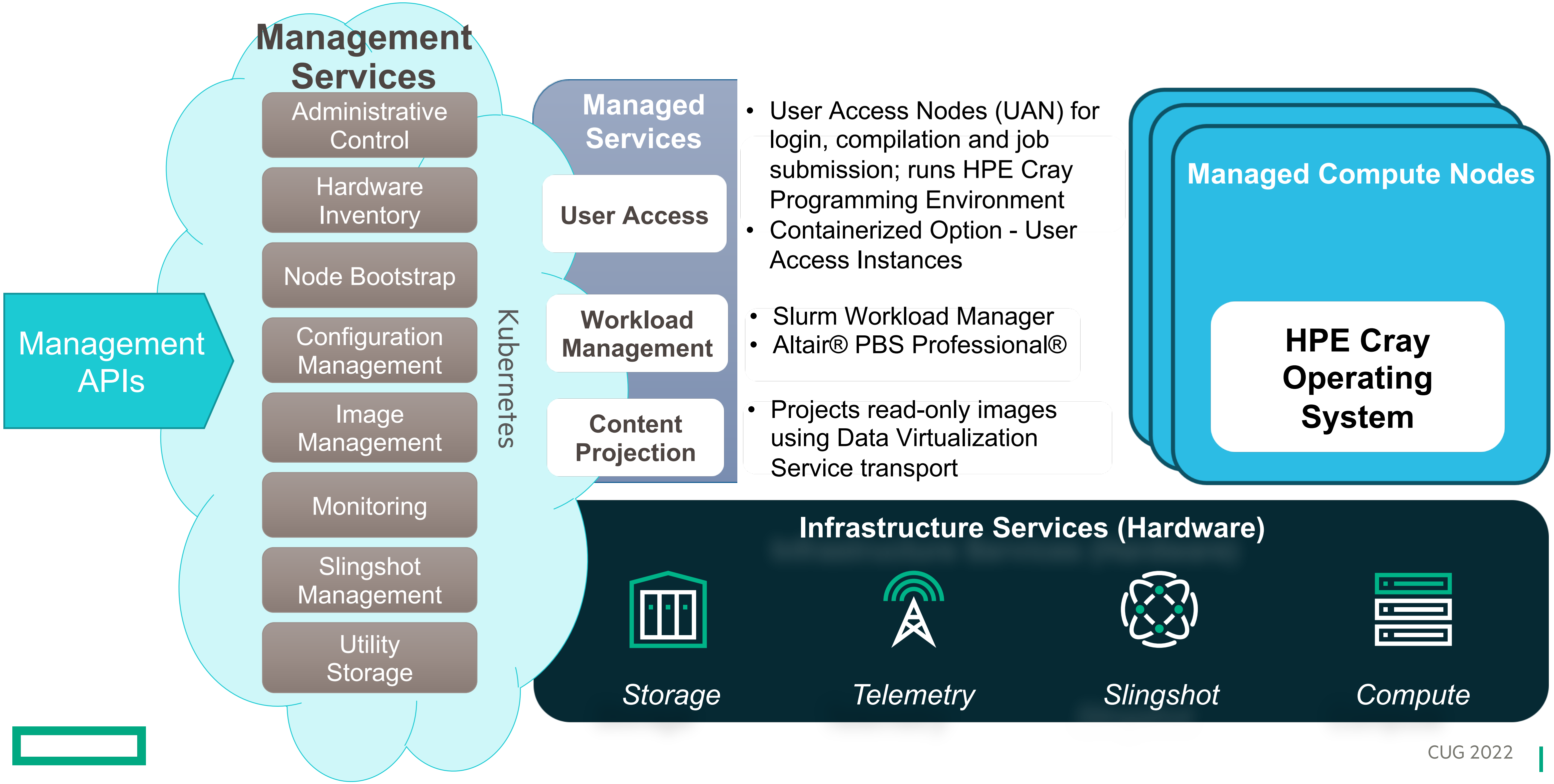
Scalable
Manage Exascale systems with thousands of nodes

Flexible
Enable cloud-like secure multitenant operations with extensible microservices APIs

Proven
Used by customers globally with large supercomputing systems

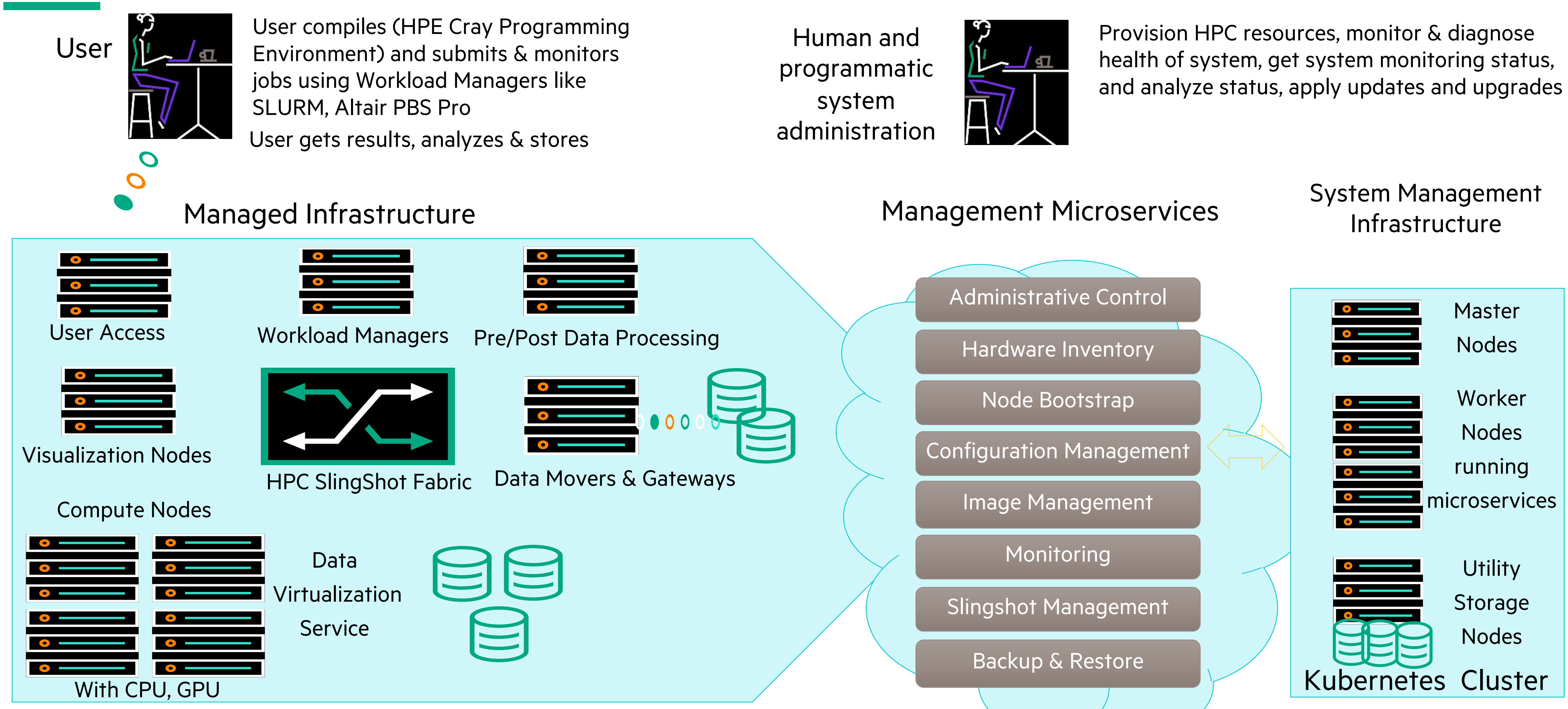


HPE CRAY SYSTEM MANAGEMENT SOLUTION OVERVIEW



HPE CRAY SYSTEMS MANAGEMENT COMPONENTS

Manage Exascale Supercomputers to deliver optimal performance for HPC workloads



HPE CRAY SYSTEM MANAGEMENT UNIQUE ATTRIBUTES

System management software designed for Exascale HPC and beyond

Key Capabilities

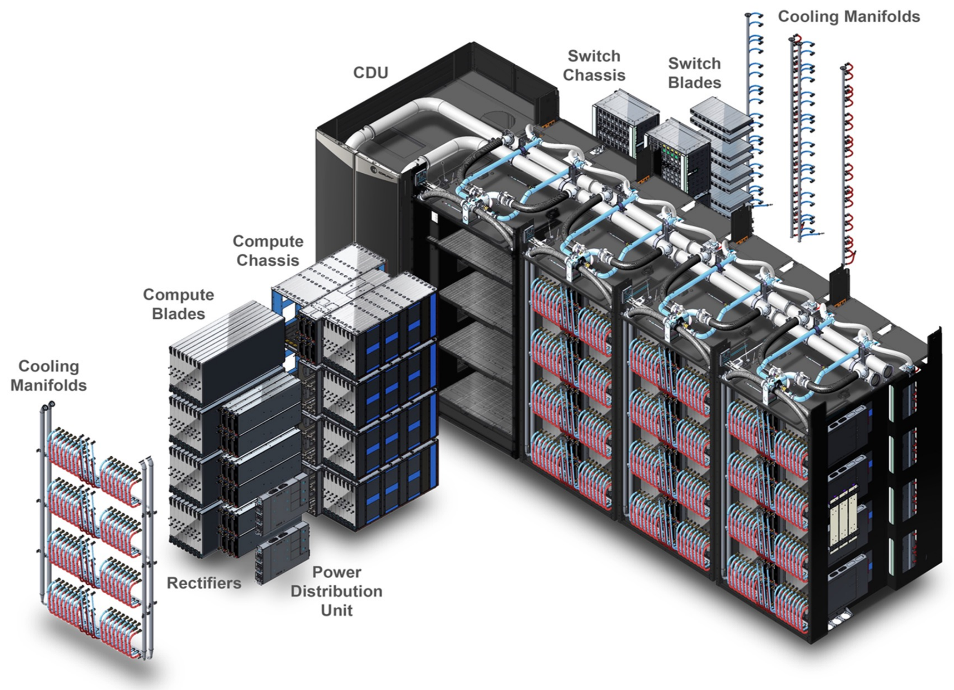
- Comprehensive monitoring and management of all aspects of the system: CPU/GPU, network (integrated Cray Slingshot Fabric Manager), power management and monitoring combined with provisioning for operational efficiency
- REST APIs & standard systems management protocols enable full interoperability and extensibility of monitoring, management, and automation capabilities
- Infrastructure-as-code: Login nodes as dynamic containers (User Access Instances), workload managers as containerized services
- Built from open-source software components, is open-source software

Unique Attributes

- Kubernetes platform for running system management and sysadmin tooling enabling infrastructure-as-code & CI/CD for jobs, tenants, and environments
- Declarative and dynamic inventory and state management represents single source of truth (configurations and artifacts), continuous delivery
- aaS Security with auditable access to all APIs
- Supports scalable deployment with massive system extensibility

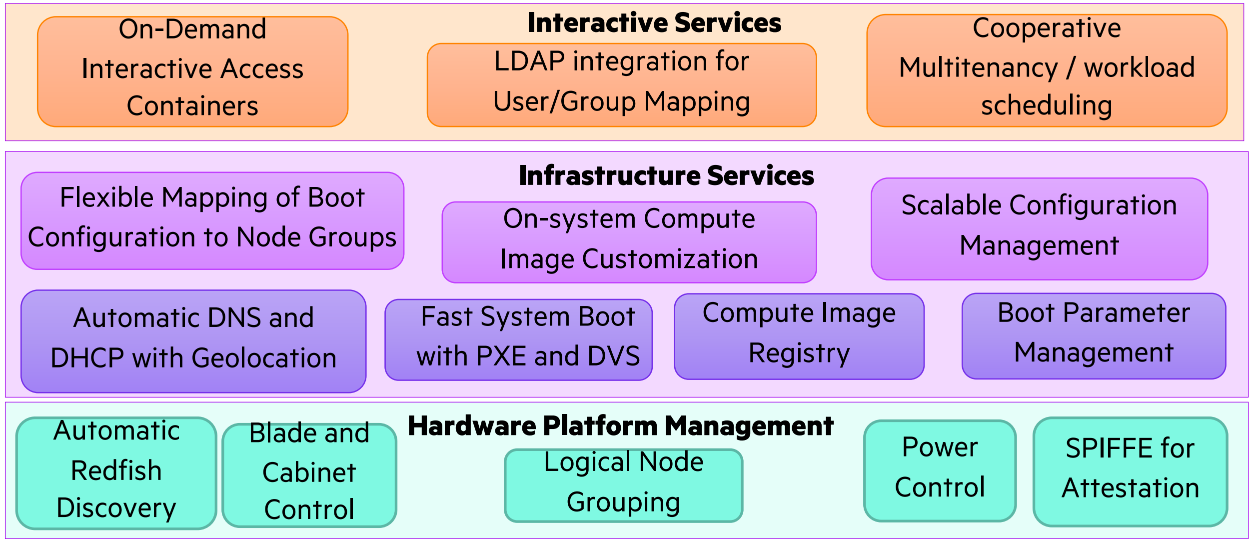
HPE Cray EX Supercomputer &
HPE Cray Supercomputer with HPE Slingshot

Exascale and beyond scalable hardware architecture and infrastructure

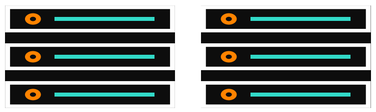


HPE CRAY SYSTEM MANAGEMENT IS ELASTIC AND RESILIENT

- Flexible Deployment Options
  - Management Kubernetes cluster scales with more nodes, CPUs, memory, network, and storage
    - Proven to scale from small number of nodes to more than 50 worker nodes for very large customer deployments
- Elasticity
  - Services are continuously checked and updated to match state
  - When nodes are added or subtracted or the load suddenly changes, configuration is automatically modified
    - Autoscale Horizontally and Vertically within constraints
    - When the system is under-scaled, microservices fail according to defined priorities
- Resiliency
  - Microservices are active/active HA
    - Separate gateways and individual load balancers
    - Multiple Pods
    - Rolling deployments and rollbacks
  - Managed nodes running custom app services have HA



K8s Master Nodes



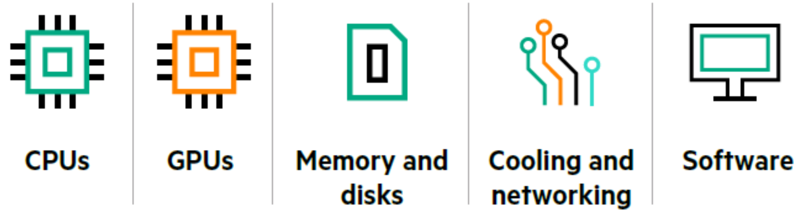
K8s Worker Nodes

Common footprint

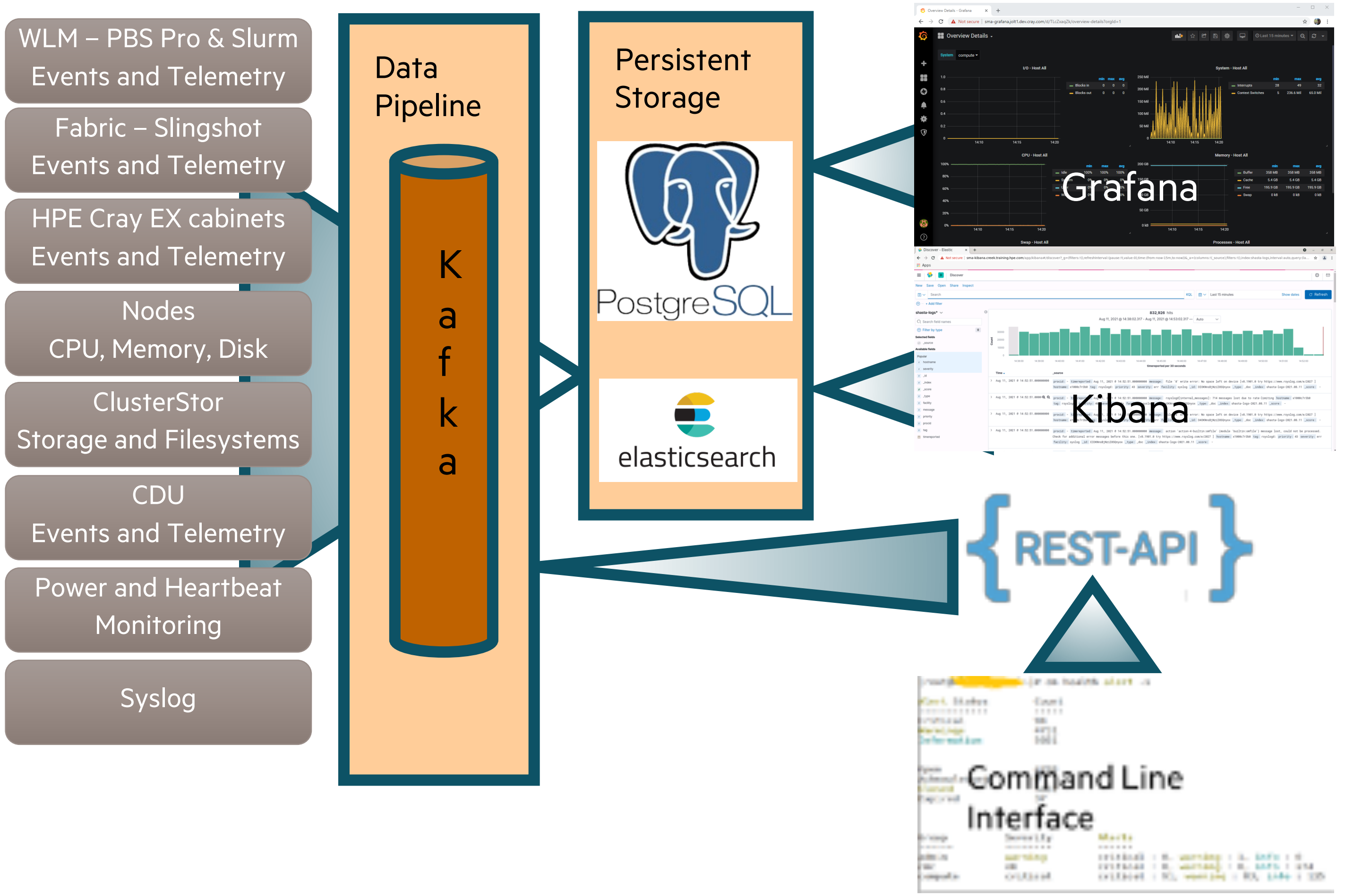
- 3 Kubernetes Master nodes for active failover
- 4+ K8s Worker nodes
- 3+ Utility storage nodes for state abstraction

SCALABLE MONITORING AND MANAGEMENT

HPE Cray Systems management offers fine-grained centralized monitoring and management of your Exascale HPC systems to keep it performing at its best

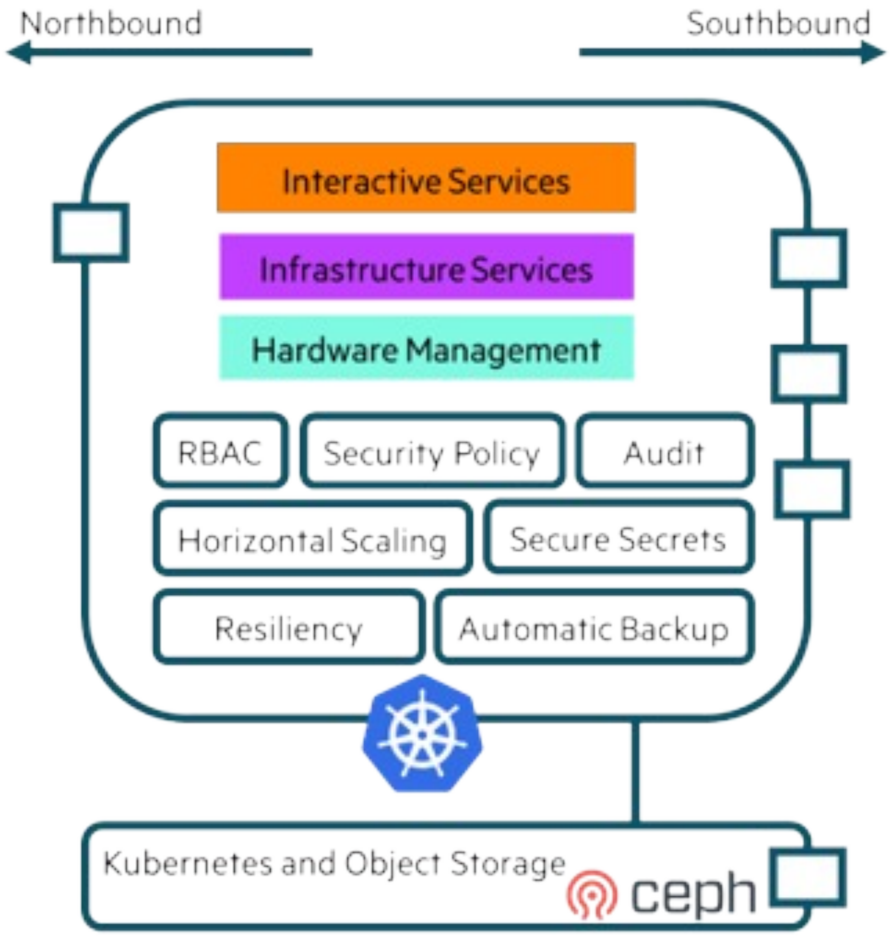


- In-band (LDMS) and out of band telemetry
- Access metrics and alerts via GUI, CLI, REST APIs
- Customize system telemetry and alerts to best suit your needs
- Set up automatic reactions to events to prevent failures

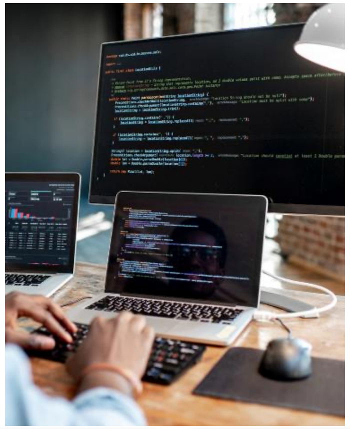


HPE CRAY SYSTEM MANAGEMENT DESIGNED FOR AS-A-SERVICE SECURITY

- CSM supports human and non-human IAM (Identity and Access Management)
- Fully supported custom RBAC (Role Based Access Control)
  - No limits to the group or role structure, infinite customization
  - Control managed entities with a URL
  - Programmatic interface for change control after upgrades, patches, etc.
- Multiple identity providers
- Credentials management
- Certificate management
- Mesh network encryption (TLS) and access policies
- DNS and external zone transfers
- Non-root users
- User traffic isolation - necessary for multitenancy
- Node attestation
  - SPIFFE (Secure Production Identity Framework For Everyone) provides a secure identity with X.509 certificate to every workload
  - SPIRE (SPIFFE Runtime Environment) manages platform and workload attestation, has API, and handles certificate issuance and rotation



CRAY SYSTEM MANAGEMENT EXTENSIBILITY FOR SYSTEM OPERATIONS



CLI Access | Extended Microservices

Loosely-coupled Microservices

API-First Development

HPE Cray System Management

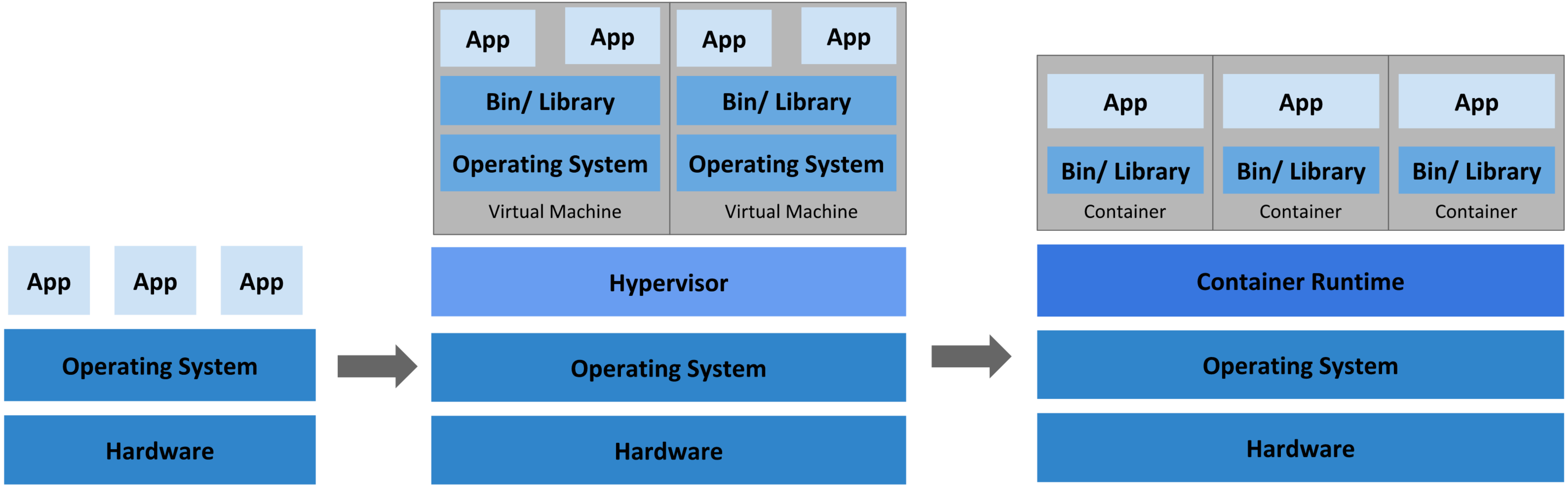
API-First Development

- Nearly 100% of the systems management functionality is exposed via API
- Machine readable Swagger API definitions are available for all
- Cray CLI– a tool for discovering and implementing the APIs
- System Administration Toolkit (SAT) – a CLI tool covering more common workflows spanning APIs

Loosely-coupled Microservices

- Customers are developing their own APIs to extend functionality
- Customers can pick and choose which HPE provided aspects to use or replace
- Enables granular deployment elasticity
  - Not limited because of a monolithic application design
  - “[this] functionality should scale and failover in [these] ways”
- Can be updated continuously with high confidence

KUBERNETES IS EVOLUTION FOR MANAGING SCALE

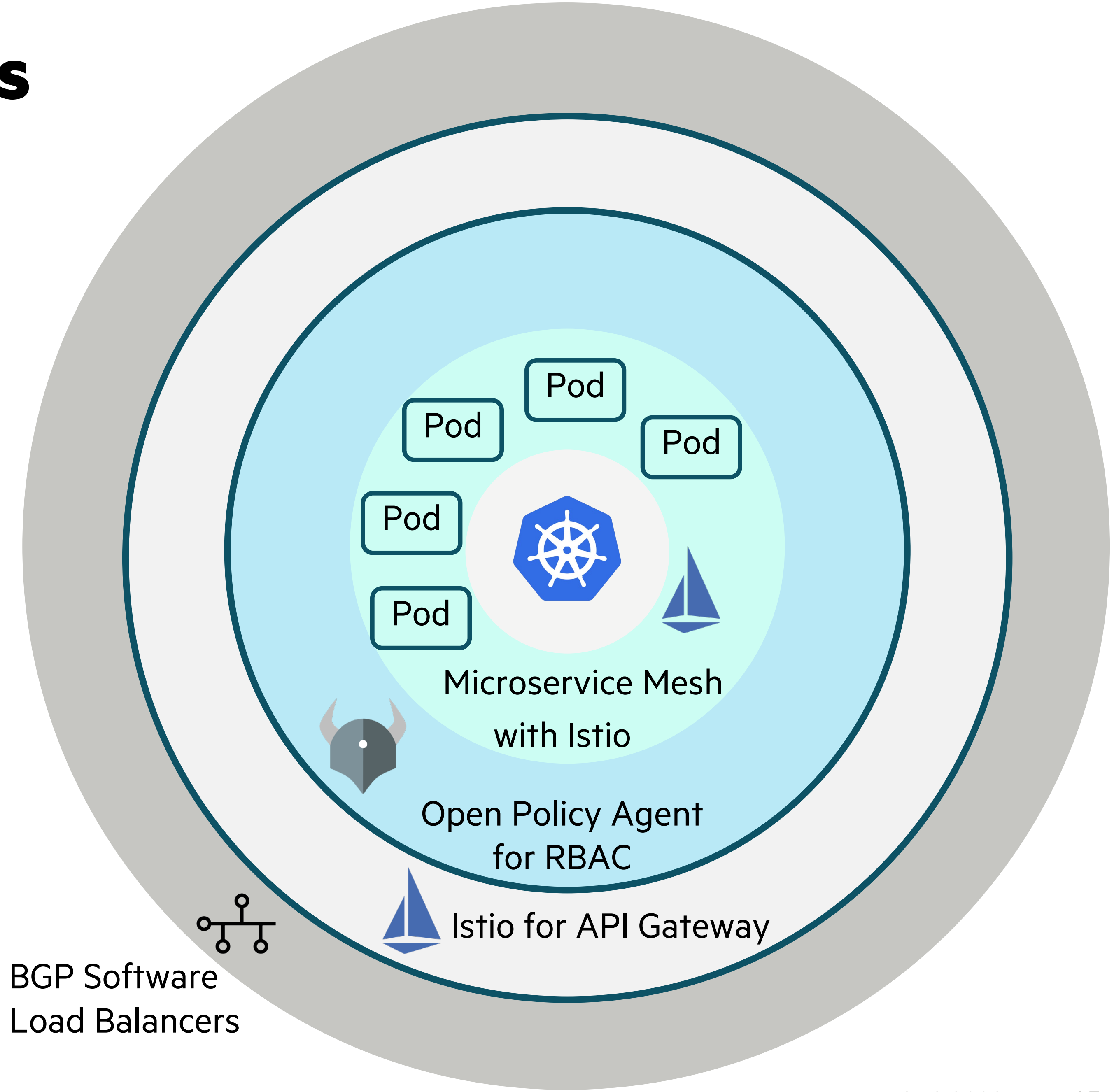


Traditional Deployment **Virtualized Deployment** **Container Deployment**

https://kubernetes.io/docs/concepts/overview/what-is-kubernetes/

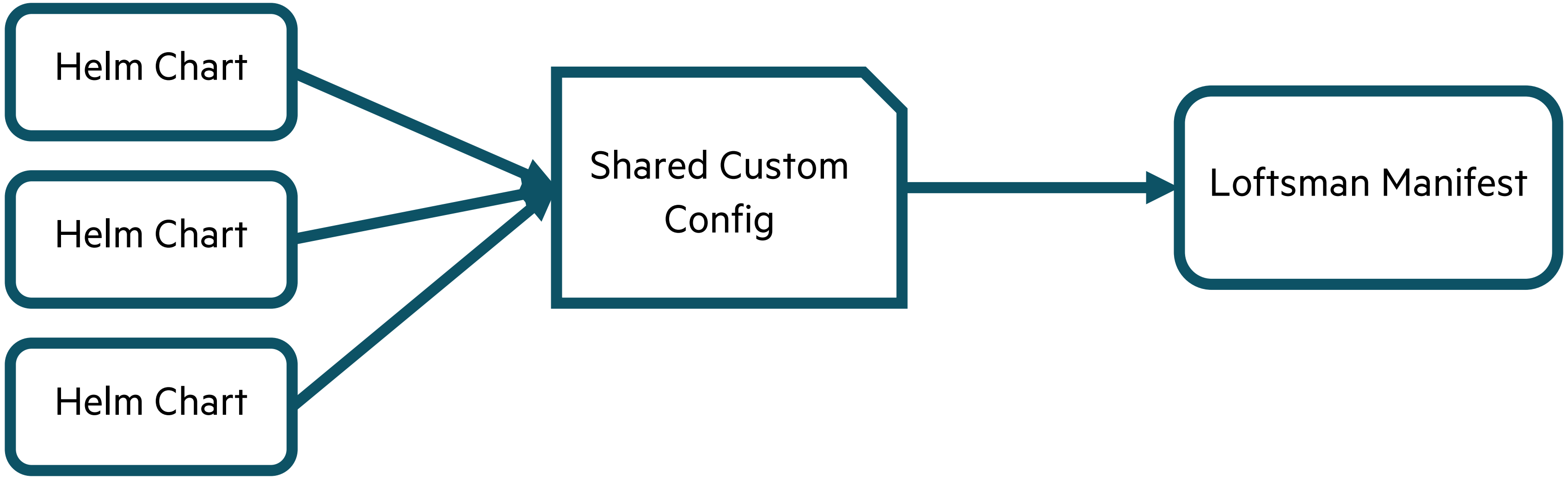
MICROSERVICE SECURITY LAYERS

- Pod to Pod Traffic is secured by Istio with mTLS and Kubernetes Policy
- Ingress and Egress traffic is regulated by Open Policy Agent (OPA)
- Istio provides API Gateway services to expose collections of services
- MetalLB allocates Virtual IP addresses that pass traffic to Istio API Gateways
- Keycloak handles authentication and issues refreshable bearer tokens, required for API Access
- Keycloak federates with upstream LDAP or Kerberos for user directories



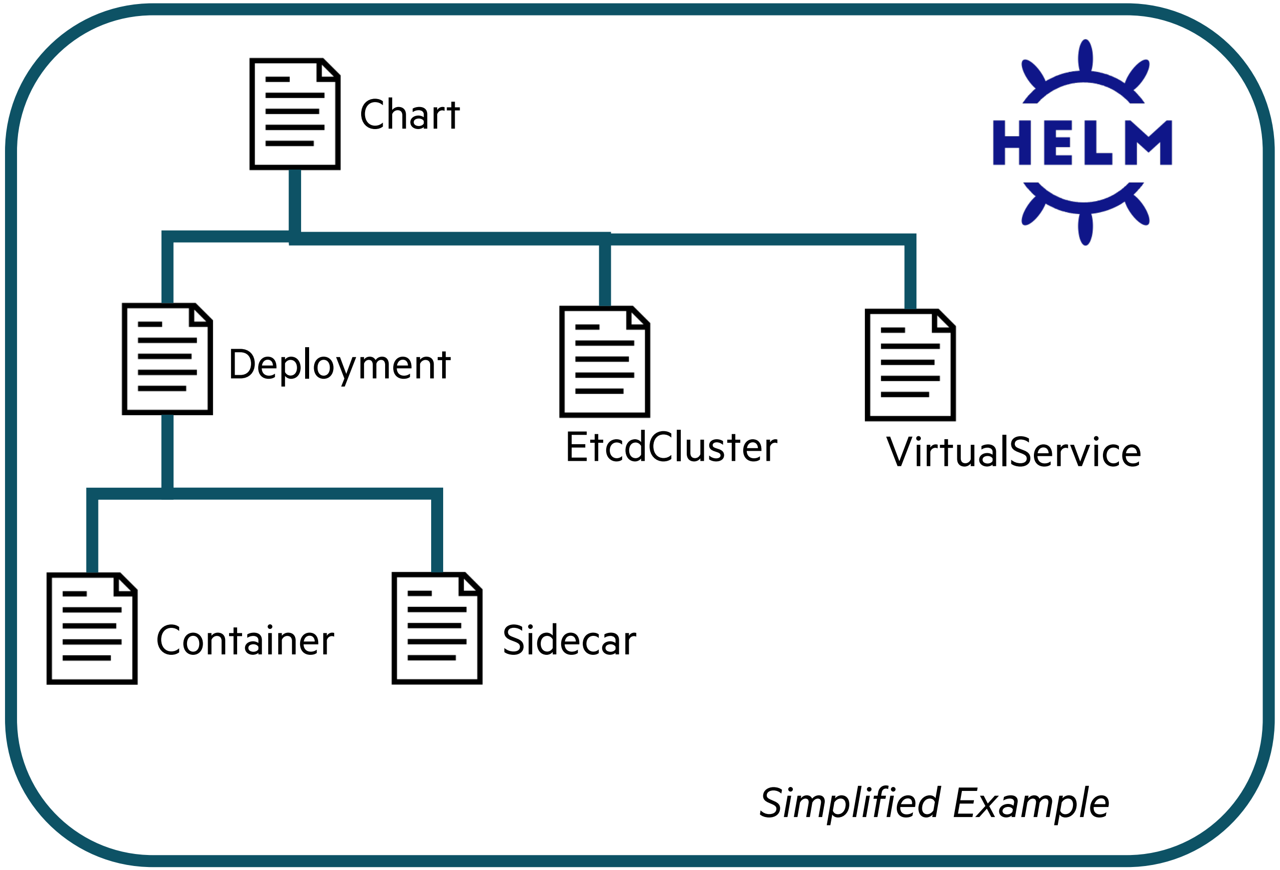
KUBERNETES PACKAGE MANAGEMENT WITH LOFTSMAN AND HELM

- HELM v3 is a packaging standard for Kubernetes applications
- Loftsman supports a single “manifest” for a collection of Helm applications
- Loftsman manifests are merged at runtime with “sealed secrets” and Kubernetes cluster parameters
- Loftsman manifests are suitable for GitOps Operations

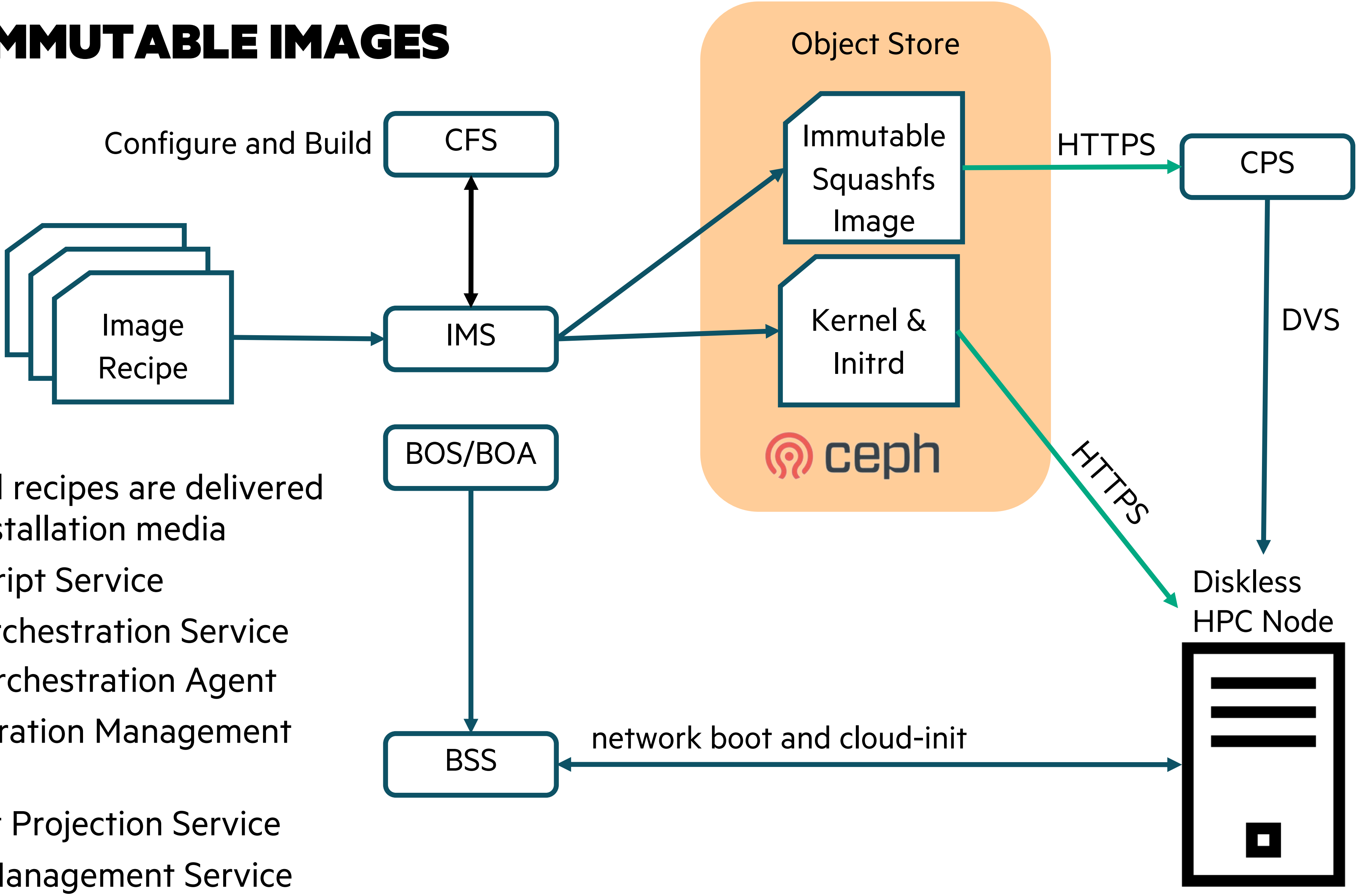


DEPENDENCY TRACKING WITH HELM

- Resources needed by the service are named in the helm chart
- Every resource is managed by an operator
- Kubernetes supports Pods/Services/Deployments natively
- Third Party Operators add Resources and manage them
- EtcdCluster is managed by the etcd-operator
- VirtualService is managed by istio
- LoadBalancer is managed by MetalLB
- The Resource is important
- The operator is incidental



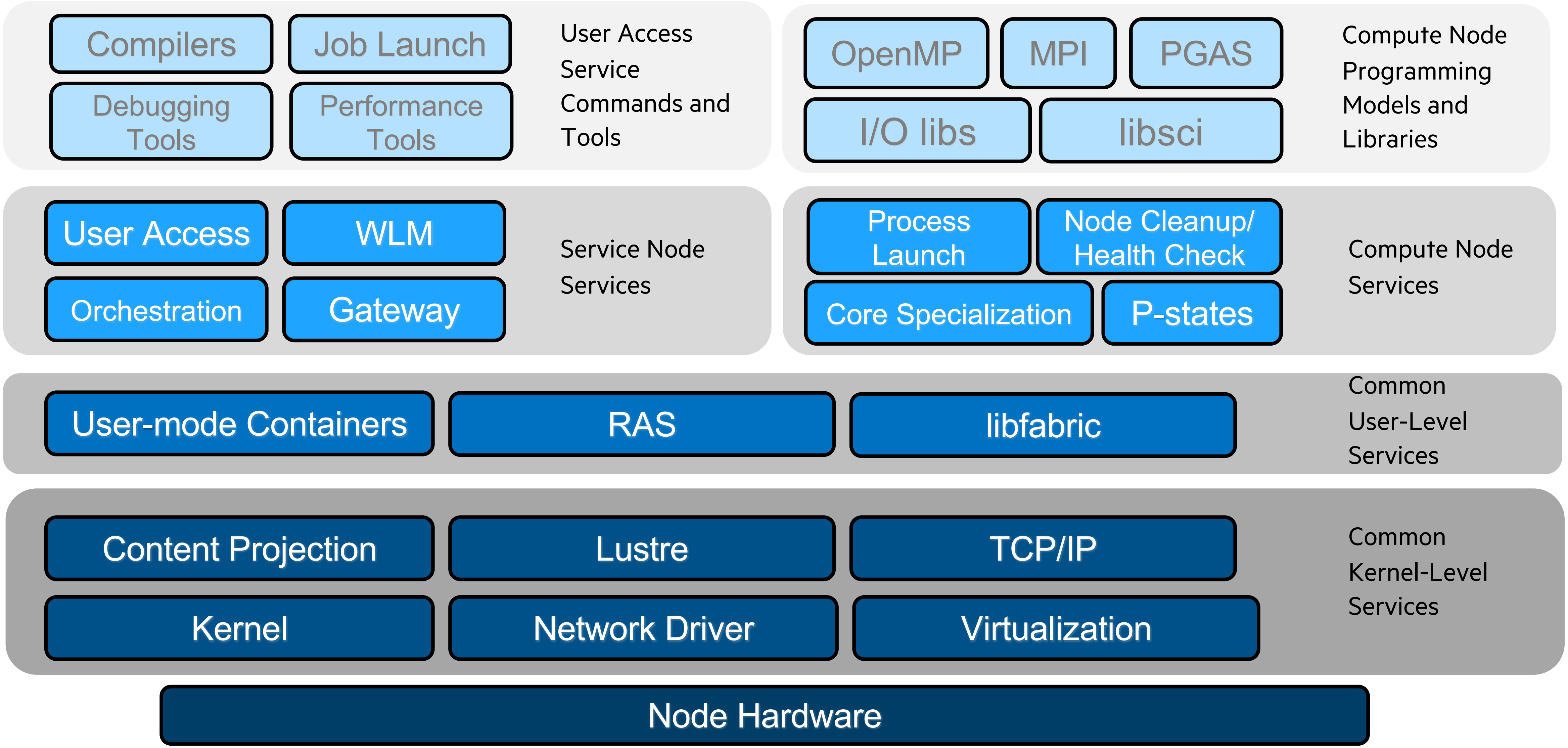
BOOTING IMMUTABLE IMAGES



Both images and recipes are delivered as part of the installation media

- BSS: Boot Script Service
- BOS: Boot Orchestration Service
- BOA: Boot Orchestration Agent
- CFS: Configuration Management Service
- CPS: Content Projection Service
- IMS: Image Management Service

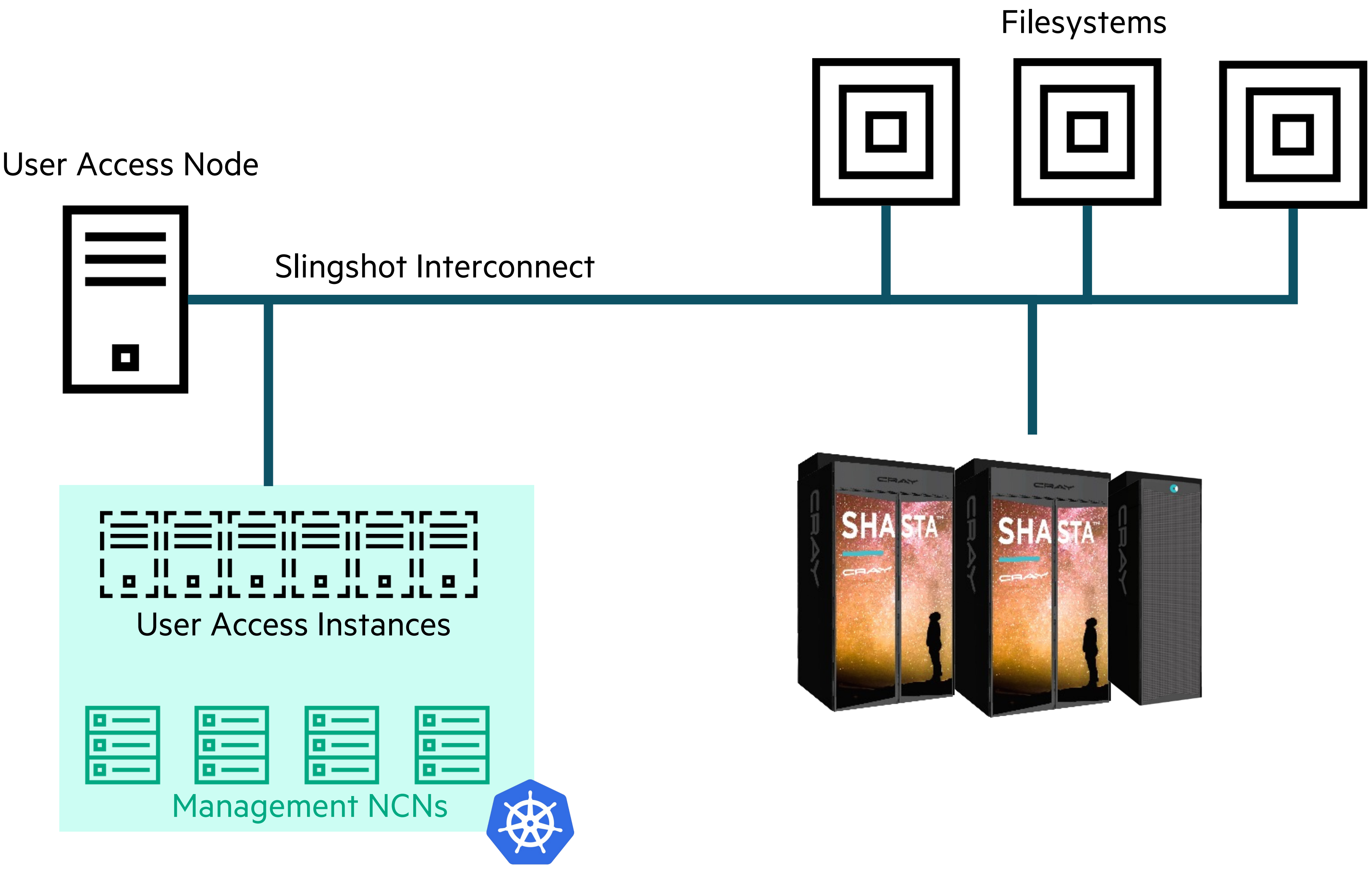
ECOSYSTEM COMPONENTS



USER ACCESS OPTIONS

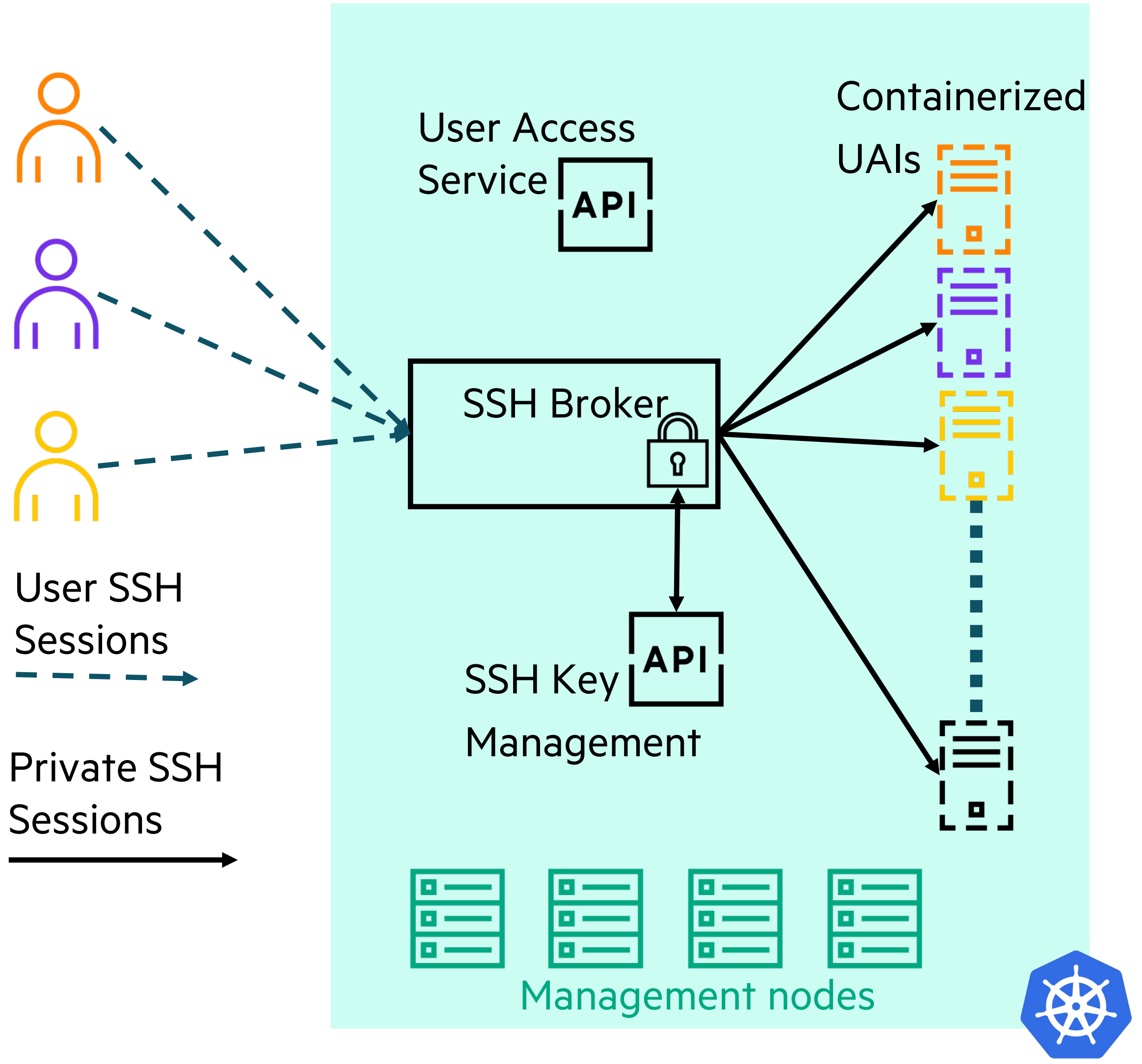
Power Users
Compile and Run

Standard Users
Run and Monitor

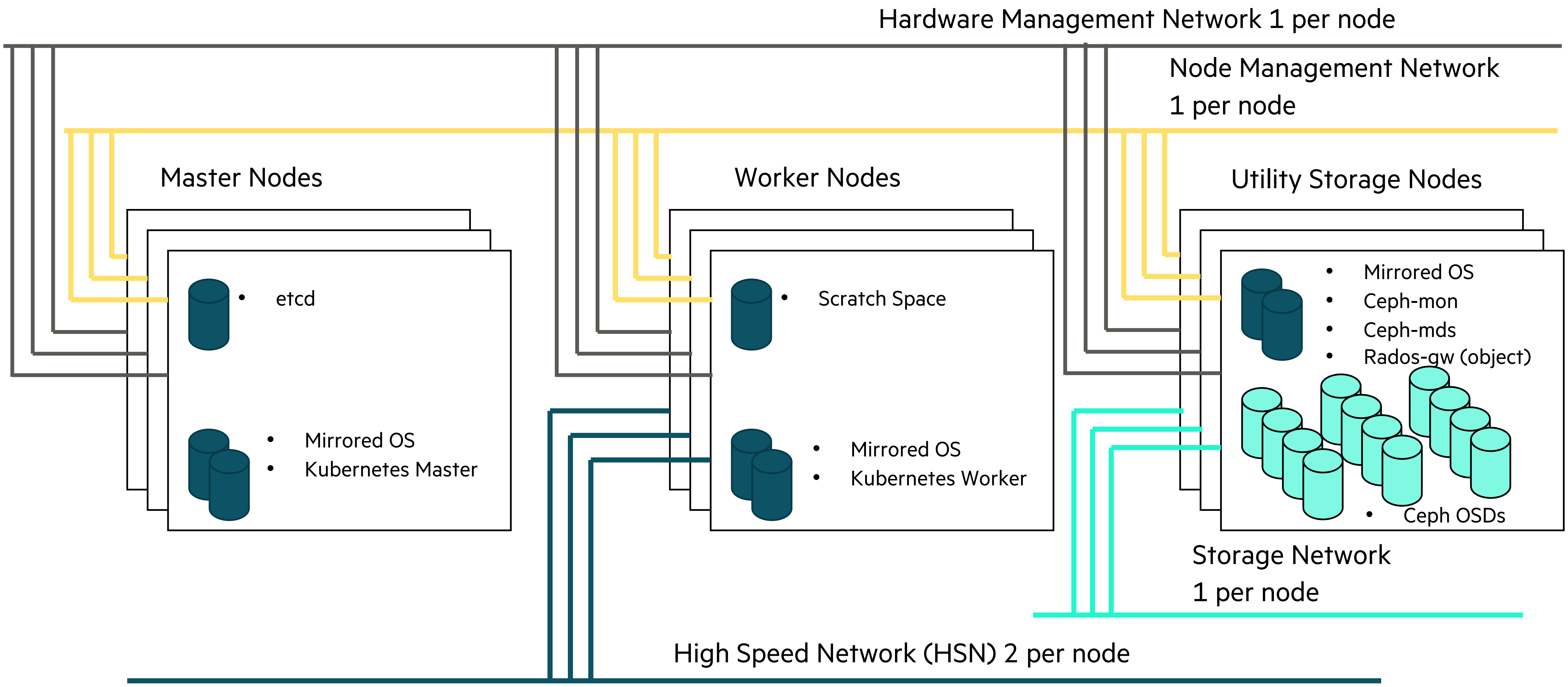


USER ACCESS SERVICE AND BROKER

- On-Demand containerized SSH environment “serverless”
- SSH is the only User-Facing API
- Templated UAI Pods launched and destroyed as-needed
- User state persisted only in mounted filesystems
  - Home, Lustre, SpectrumScale (GPFS), etc.
- Internal SSH relies only on single-use SSH keys
- Broker consumes a single IP regardless of how many users
- Multiple brokers can be used to handle different user types and user groups



MANAGEMENT NODES



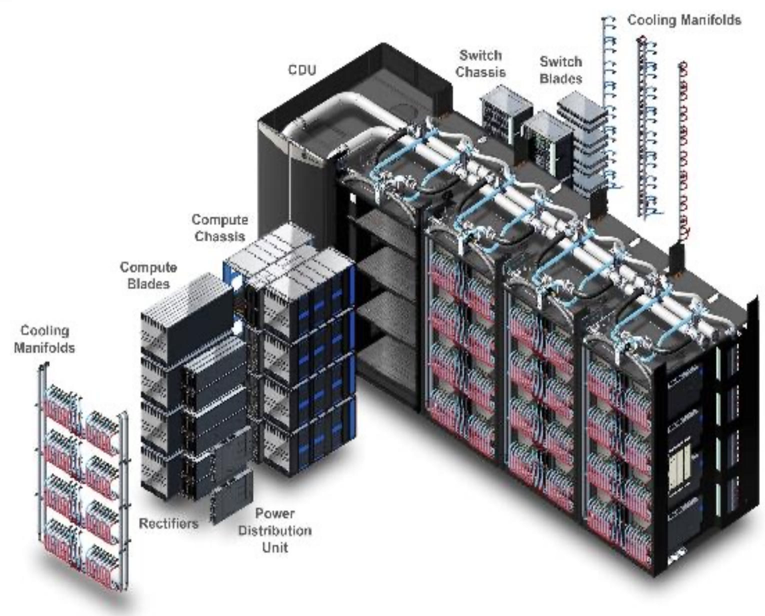
HPE CRAY EX HARDWARE

FLEXIBLE COMPUTE INFRASTRUCTURE

HPE Cray EX liquid-cooled optimized cabinet (Olympus)

- Up to 64 compute blades, and 512 processors per rack
- Flexible bladed architecture supports multiple generations of CPUs, GPUs, and interconnect
- Cableless interconnect between switches and nodes inside chassis
- 100% direct liquid-cooling – no fans
- Up to 400KW capability per rack
- Designed to provide an optimal solution for tens to hundreds of thousands of nodes, scales to hundreds of cabinets
- CEC (Cabinet Environment Controller)
- CMC (Chassis Management Controller)
- CDU (Coolant Distribution Unit) supports up to 4 cabinets

Scaling building block



Choice of blade types for optimal density, efficiency, and cost per compute node

AIR-COOLED CABINETS

HPE Cray standard air-cooled cabinet (River)

- Standard 19” cabinet
- Air-cooled, but with optional liquid-cooled door
- One or more cabinets with Management infrastructure nodes
- One or more cabinets with high-performance and capacity Storage
- One or more cabinets with commodity compute nodes (CPU and GPU)
- PDU
- Management network switches
- Slingshot network switches

Management infrastructure, high-performance parallel filesystem, commodity compute nodes

HPE CRAY COMPONENT NAMES (XNAMES)

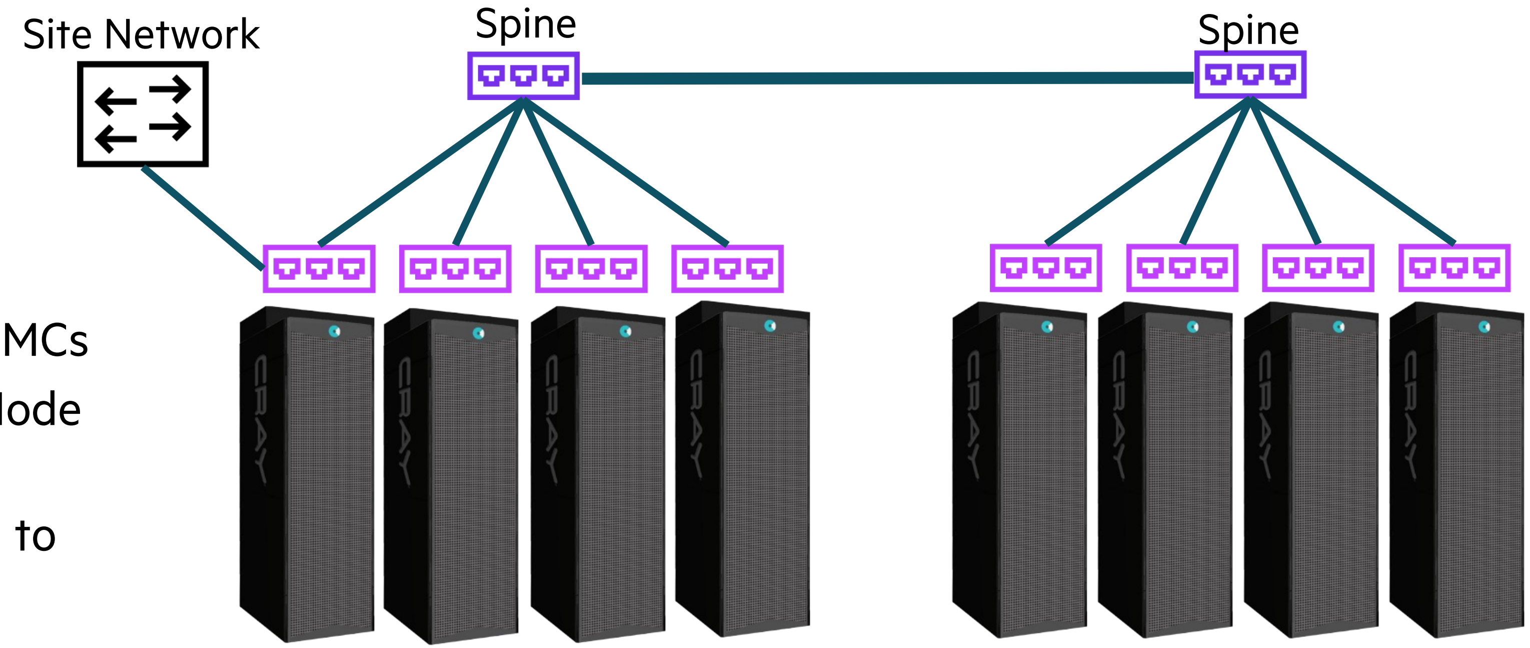
Component	Xname Scheme	Examples	Note
Cabinet	**x#**	`x1000 , x3000`	Cabinets don't have an X-Y grid
CDU	**d#**	`d0`	Up to 4 liquid-cooled cabinets per CDU
Chassis	**x#c#**	`x1000c3, x3000c0`	Air-cooled cabinets don't have chassis but for consistency always use c0 for chassis 0
Compute Blade Slot	**x#c#s#**	`X1000c3s4, X3000c0s22`	In air-cooled cabinets the slot is the lowest rack U height occupied by a server
Node card controller	**x#c#s#b#**	`x1000c3s4b1, X3000c0s22b2`	1st example - Node card 1 of blade 4 in chassis 3 2nd example - BMC in air-cooled 4 node server
Node	**x#c#s#b#n#**	`x1000c3s4b1n1, x3000c0s22b2n0`	Nodes are dependent on their BMCs. BMCS are always zero-based
Processor	**x#c#s#b#n#p#**	`x1000c3s4b1n1p0, x3000c0s22b2n0p1`	Processor sockets are zero-based in xnames
Slingshot Switch	**x#c#r#**	`X1000c3r7, x3000c0r42`	Air-cooled Slingshot switches use rack "U" height just like air-cooled servers
Ethernet Switch	**x#c#w#**	`d0w1, x3000c0w38`	Leaf switches in CDUs extend SMNet to the cooling group

NETWORKS

- Management
- Customer Access
- Slingshot

MANAGEMENT NETWORKING ARCHITECTURE

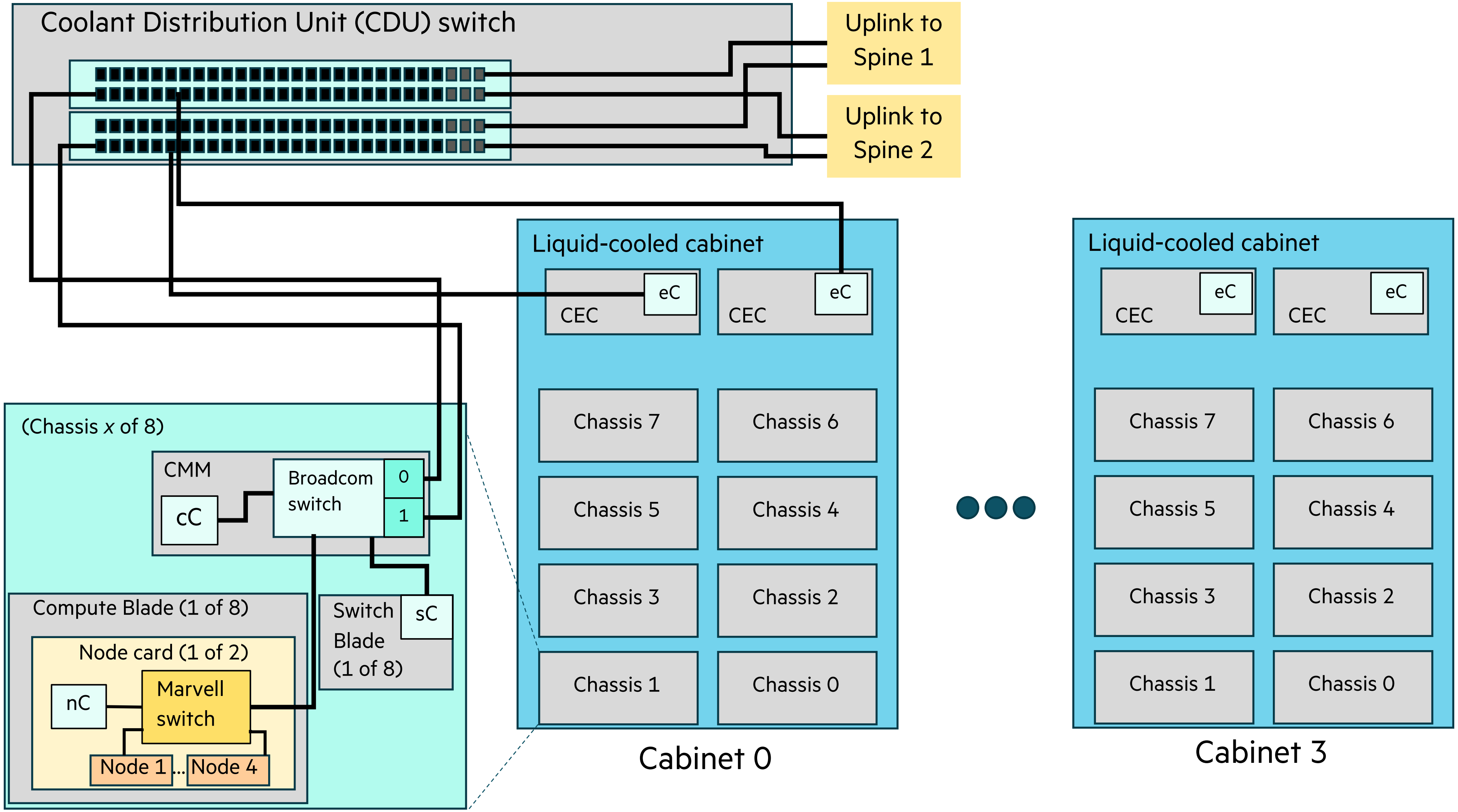
- Spine switches are mainly for Layer 3 Routing between Subnets
  - A pair of switches are used for redundancy - configured as a Layer2 / MLAG pair
- Leaf switches connect directly to nodes and node controllers
- Leaf switches used for physical connection(s) to site networks
- Olympus/Mountain Cabinets have embedded network switches that connect to CDU switches
- Aggregation switches are used on systems with many leaf switches in air-cooled cabinets or many CDU switches



Per Cabinet Subnets

- /22 of private ipv4 space for BMCs
- /22 of private ipv4 space for Node Management
- VLANs limit broadcast domain to individual cabinets

SMNet LIQUID-COOLED TOPOLOGY



SYSTEM MANAGEMENT NETWORK (SMNET) OVERVIEW

- Standard Ethernet fabric directly connected to every node and controller in the system
  - Leaf/Spine topology implemented with commodity switches
  - Divided into multiple “Virtual Networks”
    - Implemented with VLANs and Access Control Lists

Virtual Network	Connections
Node Management Network (NMN)	• All Non-Compute Nodes (NCNs) • Air-cooled Compute Nodes • Liquid-cooled Compute Nodes
Hardware Management Network (HMN)	• Air-cooled Nodes (Compute and NCN) BMCs • All Slingshot Switch Controllers (sC) • Liquid-cooled Node Controllers (nC) • Liquid-cooled Chassis Controllers (cC) • Air-cooled Hardware Controllers (smart PDUs, CMCs, etc) • SMNet switch management ports
Customer Access Network (CAN)	• All NCNs

CUSTOMER ACCESS NETWORK (CAN)

- The Customer Access Network (CAN) allows users and administrators to access the system
  - The CAN is used to:
    - Directly login to each of the NCNs or UANs
    - Access web-based user interfaces within the system (Kibana, Grafana, etc.)
    - Access the API gateway for services using:
      - Direct REST API calls from custom applications and scripts
      - Cray CLI commands from outside the system
    - Login to User Access Instances (UAI)
    - Access systems outside of the system (LDAP servers, license servers, etc.)

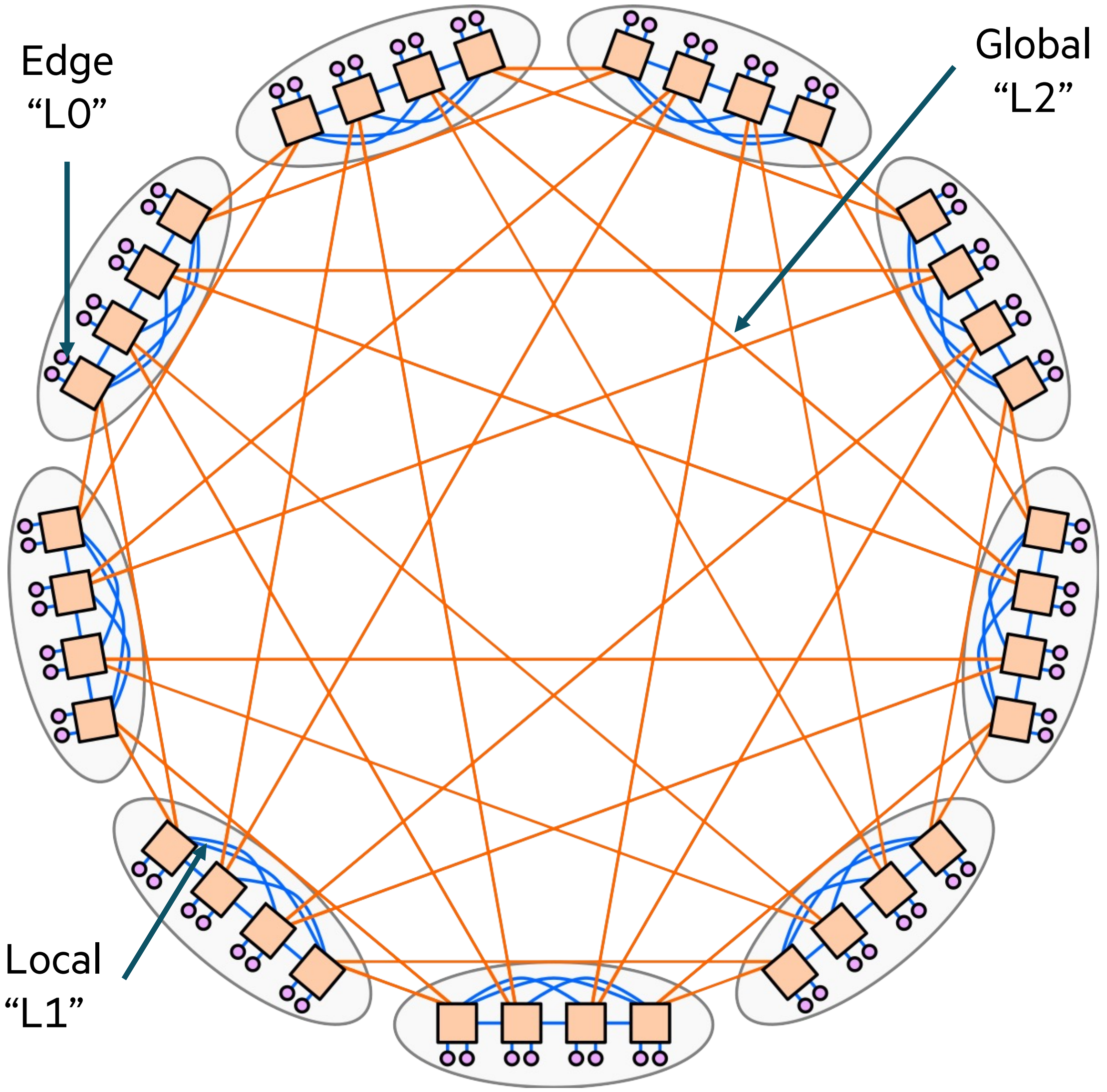
SLINGSHOT DRAGONFLY TOPOLOGY

Dragonfly Topology

- Provides All-to-All connectivity across the fabric
- Reduces costs of network hardware
- Efficient and consistent connectivity

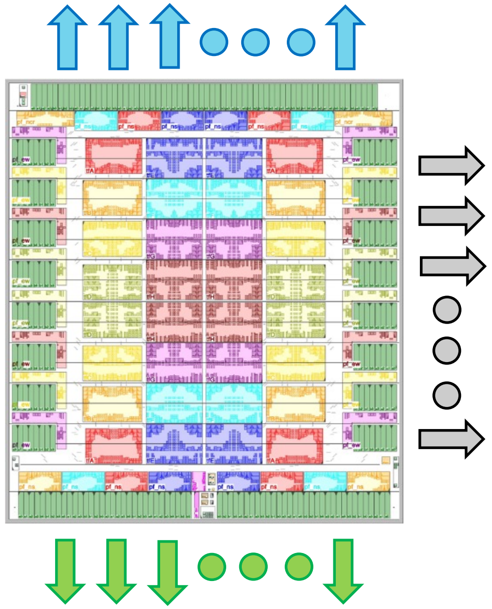
Link Types

- Edge
  - Nodes are connected directly to Switches
  - *These are called "Edge" or "L0" Links*
- Local
  - Groups of Switches connected all-to-all
  - *All switches within a group have links between them*
  - *These are called "Local" , "Group" or "L1" Links*
- *Global*
  - *Links connect different groups together*
  - *These are called "Global" or "L2" Links*



CLASSES OF SLINGSHOT DRAGONFLY TOPOLOGY

Global Links (L2)
other switch groups



Group Links (L1) switches in the same group

Host Links (L0)
nodes and edge devices

HPE Cray EX Slingshot Dragonfly Topology Classes

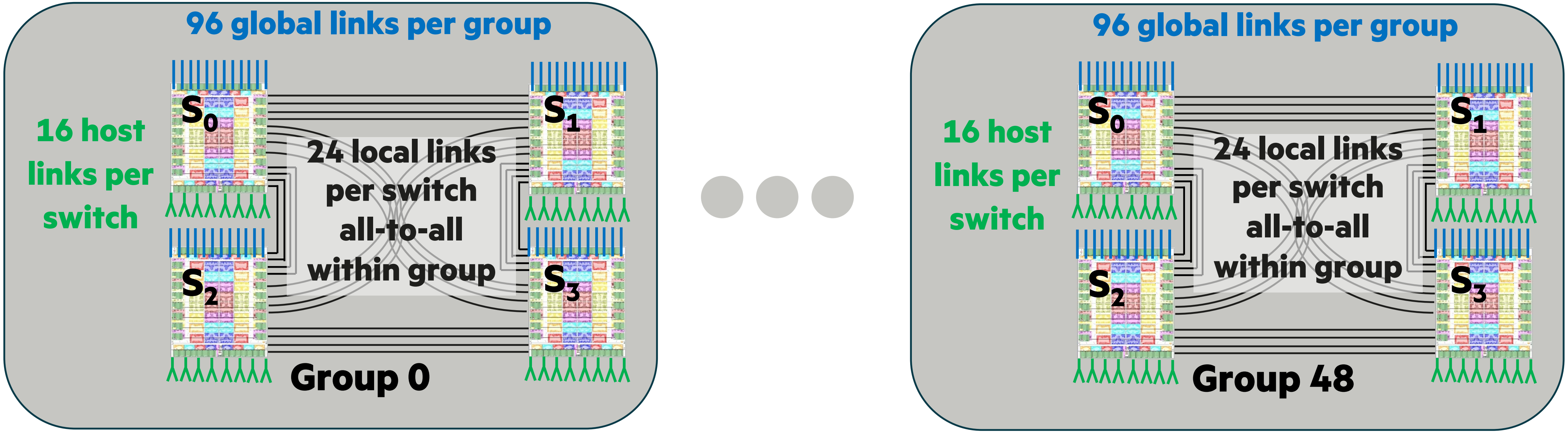
Network Class	L0 Links (Host)	L1 Links (Local)	L2 Links (Global)	Switches per Group	Max Edge Devices
0	64	0	0	n/a	64
1	32	0	32	1	528
2	16	24	24	4	3,136
3	16	28	20	8	10,368
4	16	30	18	16	73,984
5	16	31*	15*	32	262,656

Class 0 and class 1 are used in Manufacturing

*A Class 5 Slingshot network is theoretical and would require bifurcated cables for L1 and L2 links

3-HOP DRAGONFLY TOPOLOGY – CLASS 2 EXAMPLE – AIR-COOLED

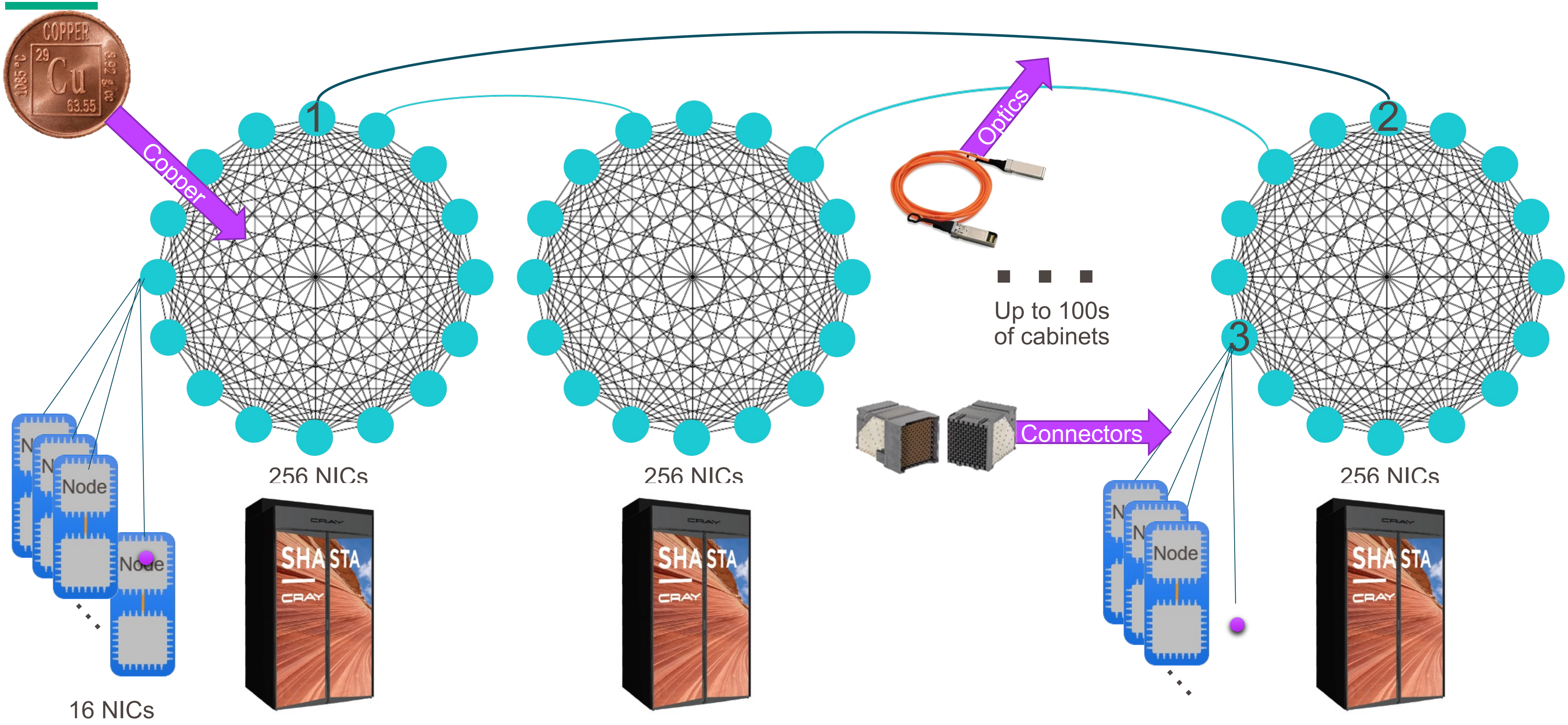
Example of a class 2 Slingshot Dragonfly network commonly used in air-cooled compute cabinets



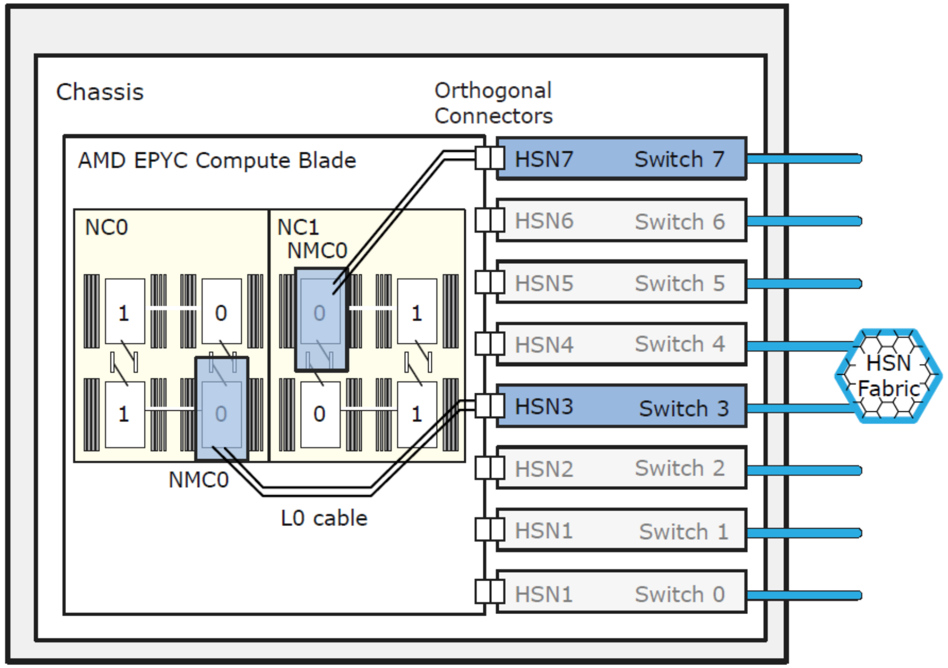
Global links can be fully populated to provide additional bandwidth on non maximal networks

A Class 2 network with nine cabinets (9 groups) could use 12 global links from each group to every other group

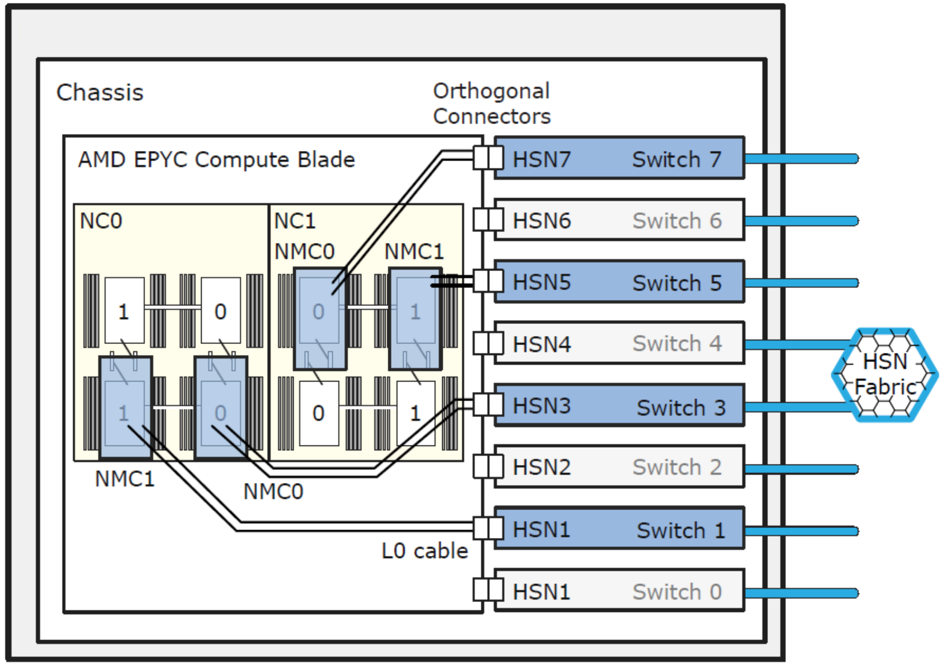
3-HOP DRAGONFLY TOPOLOGY – CLASS 4 EXAMPLE – LIQUID-COOLED



SLINGSHOT DRAGONFLY TOPOLOGY - NODE LOCALITY



Single Injection per node
16 switches per cabinet (2 per chassis)



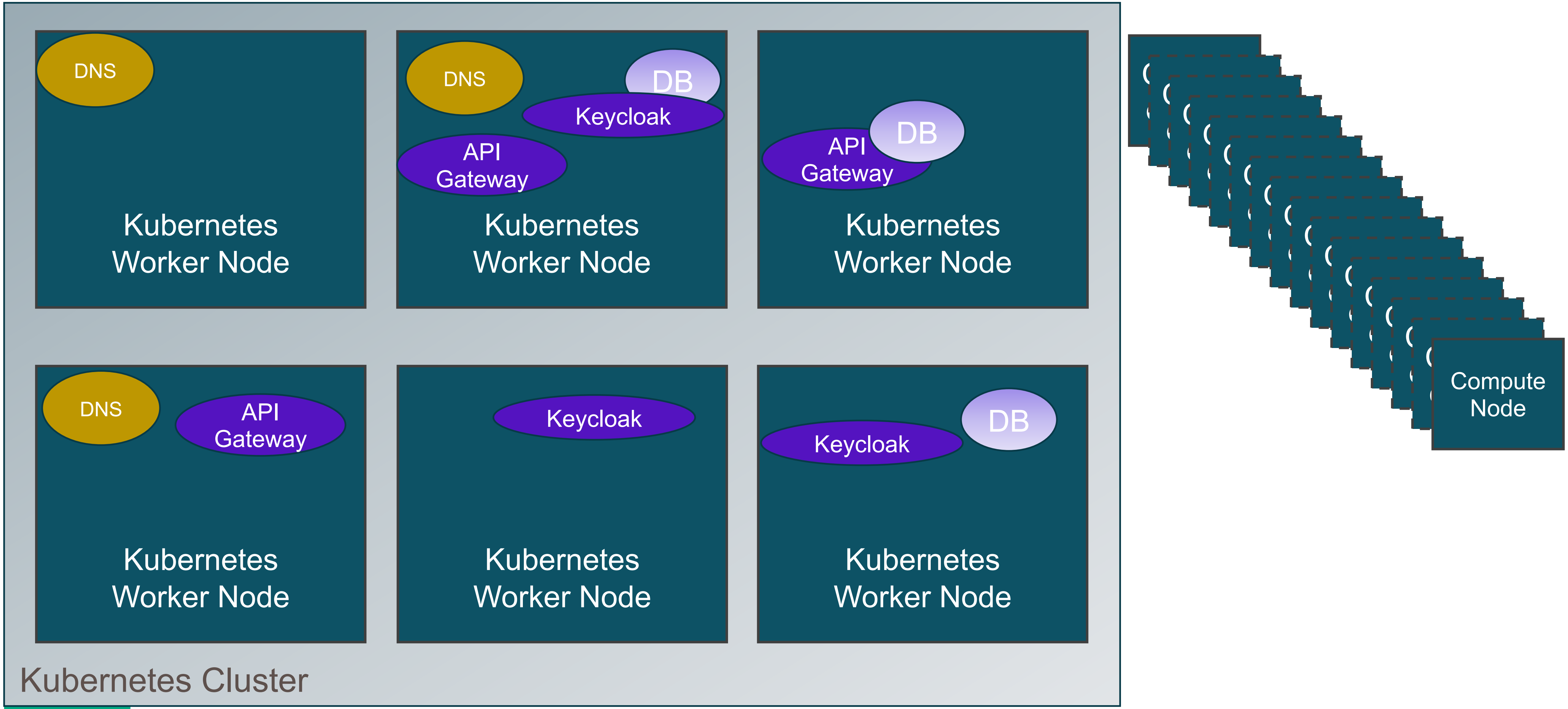
Dual Injection per node
32 switches per cabinet (4 per chassis)

- All nodes on a compute blade are **not** connected to the same switch
  - For example, `x1000c0s7b0n1` is closer to `x1000c0s4b0n0` than to `x1000c0s7b1n1`
- Nodes on an individual node card are connected to the same switch
  - Each node is connected to a distinct switch port

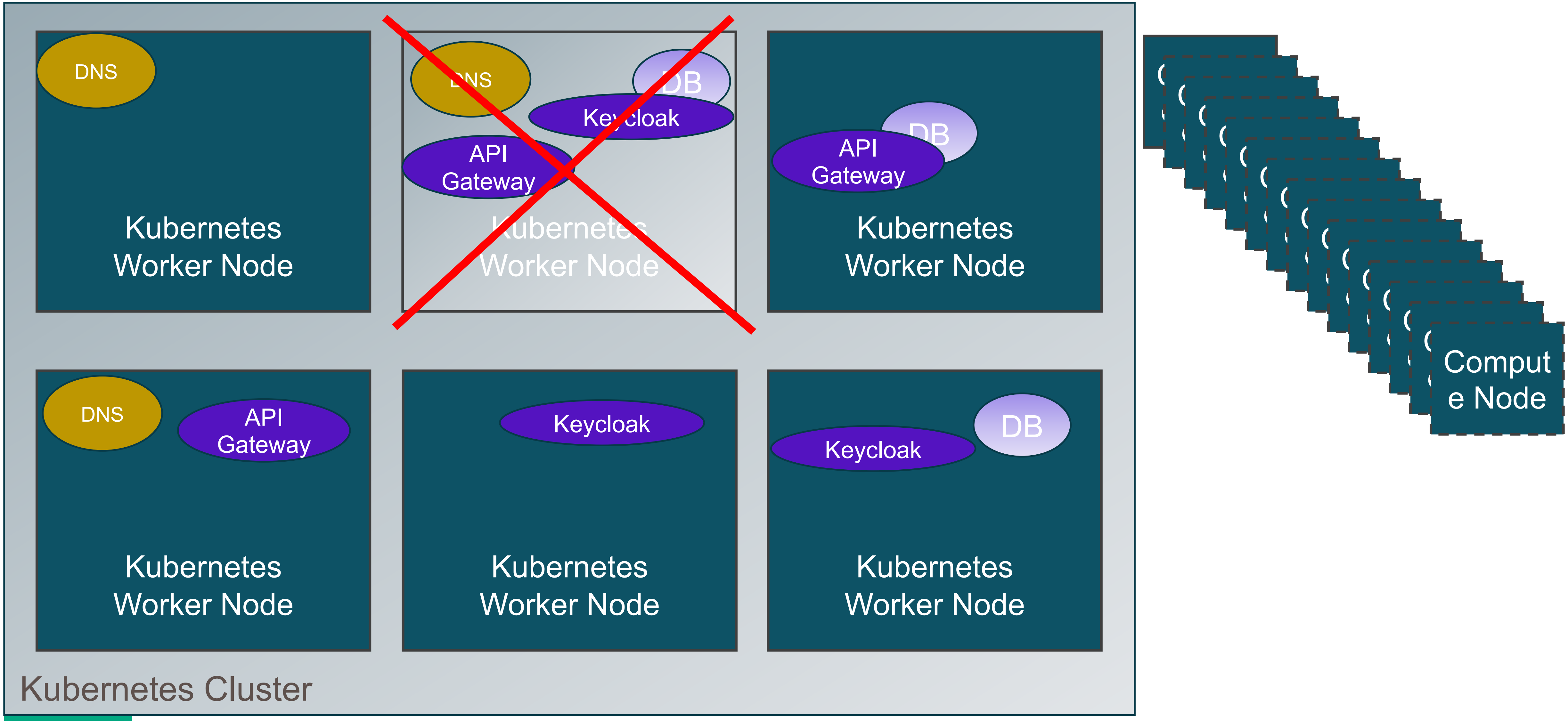
CONTINUOUS OPERATIONS

- Management service resiliency
- Rolling reboot of management nodes
- Rolling rebuild of management nodes
- Rolling upgrade of compute nodes

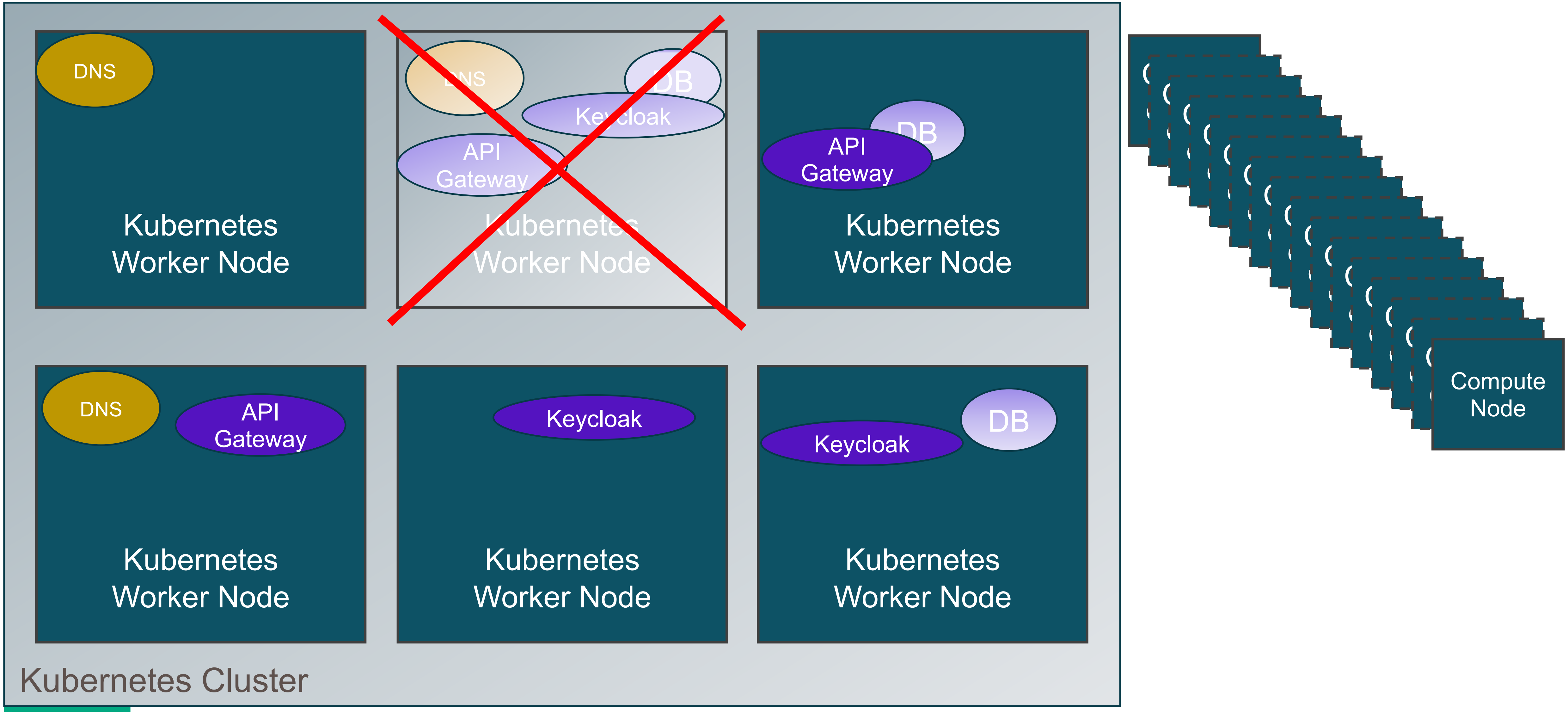
RESILIENCY WITH KUBERNETES – SERVICES RUNNING



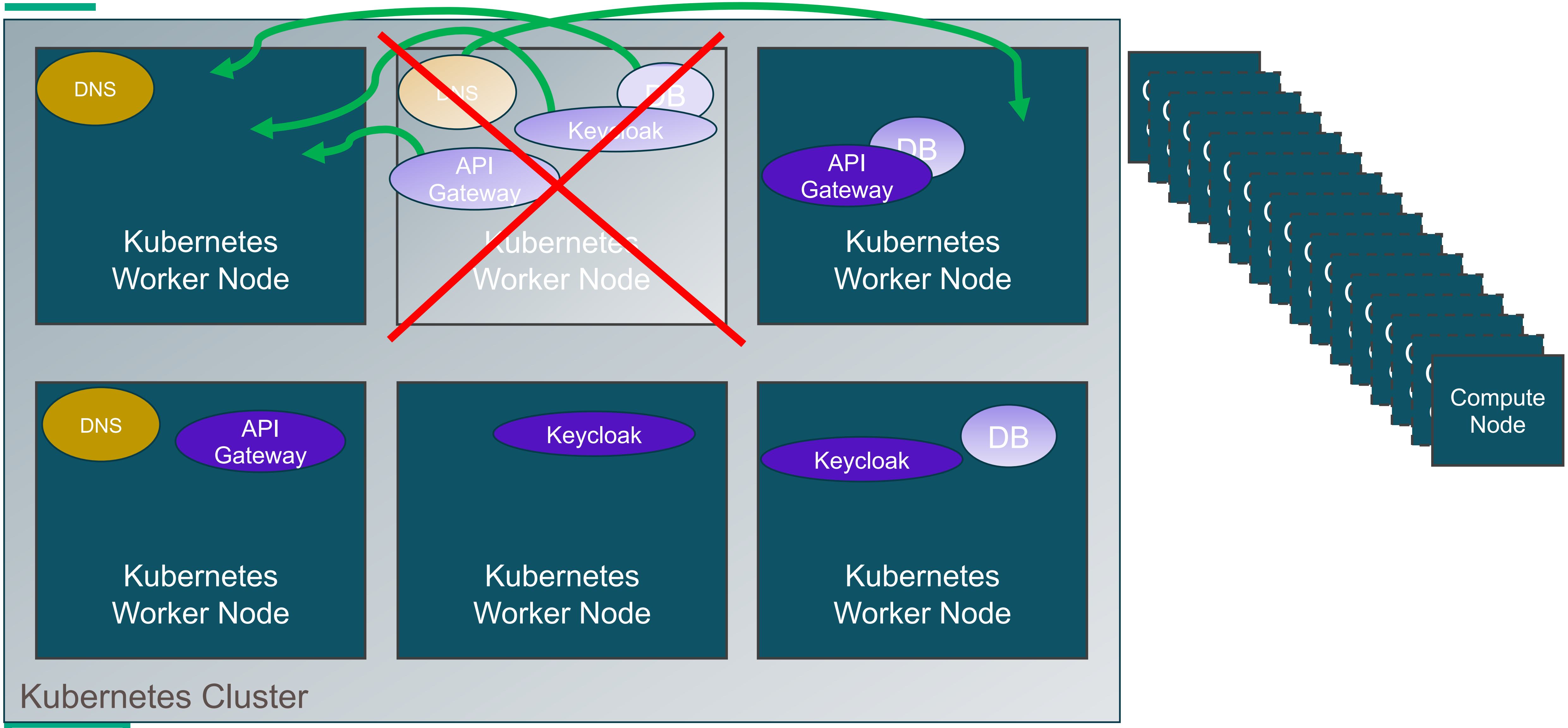
NODE GOES DOWN



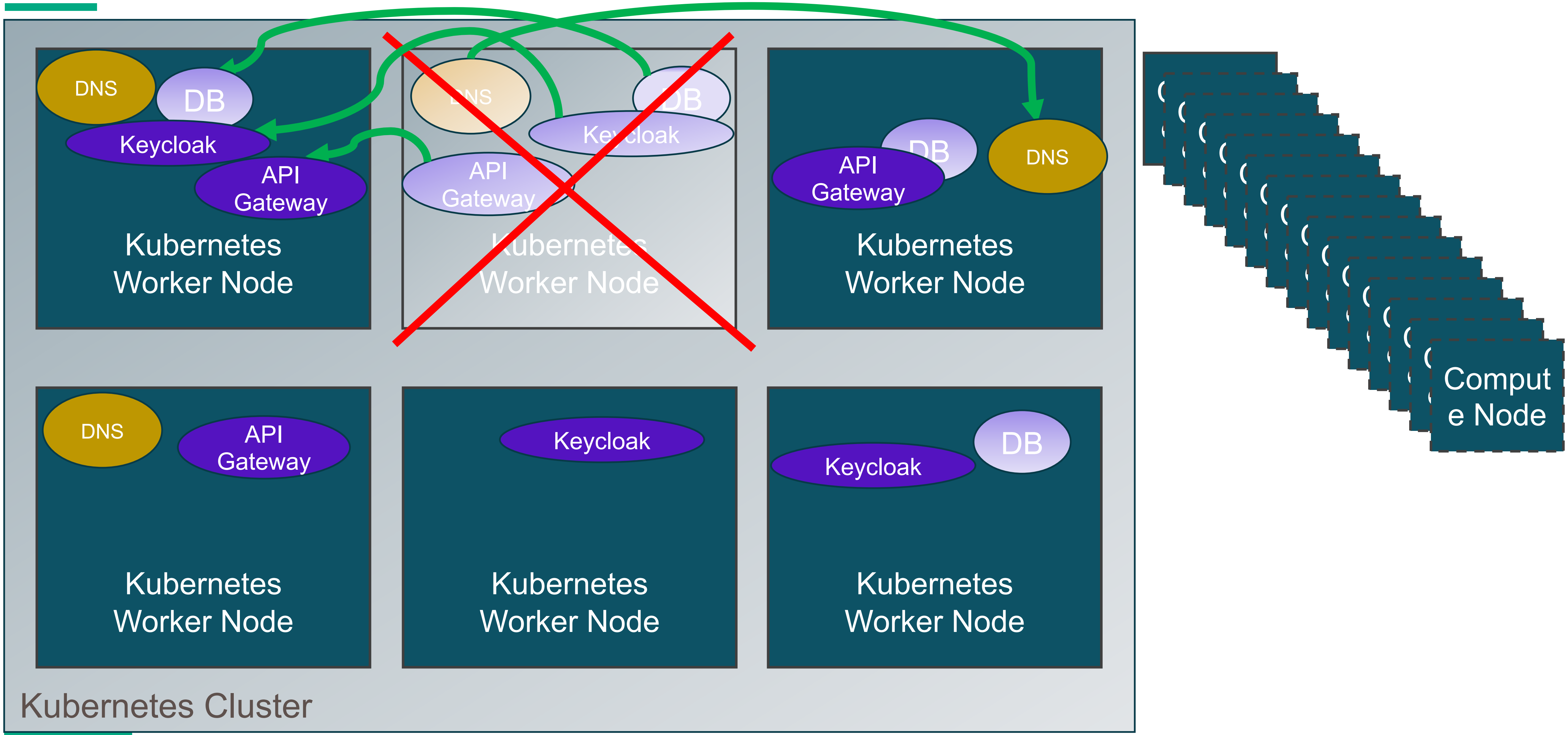
NODE AND SERVICES STOP RESPONDING



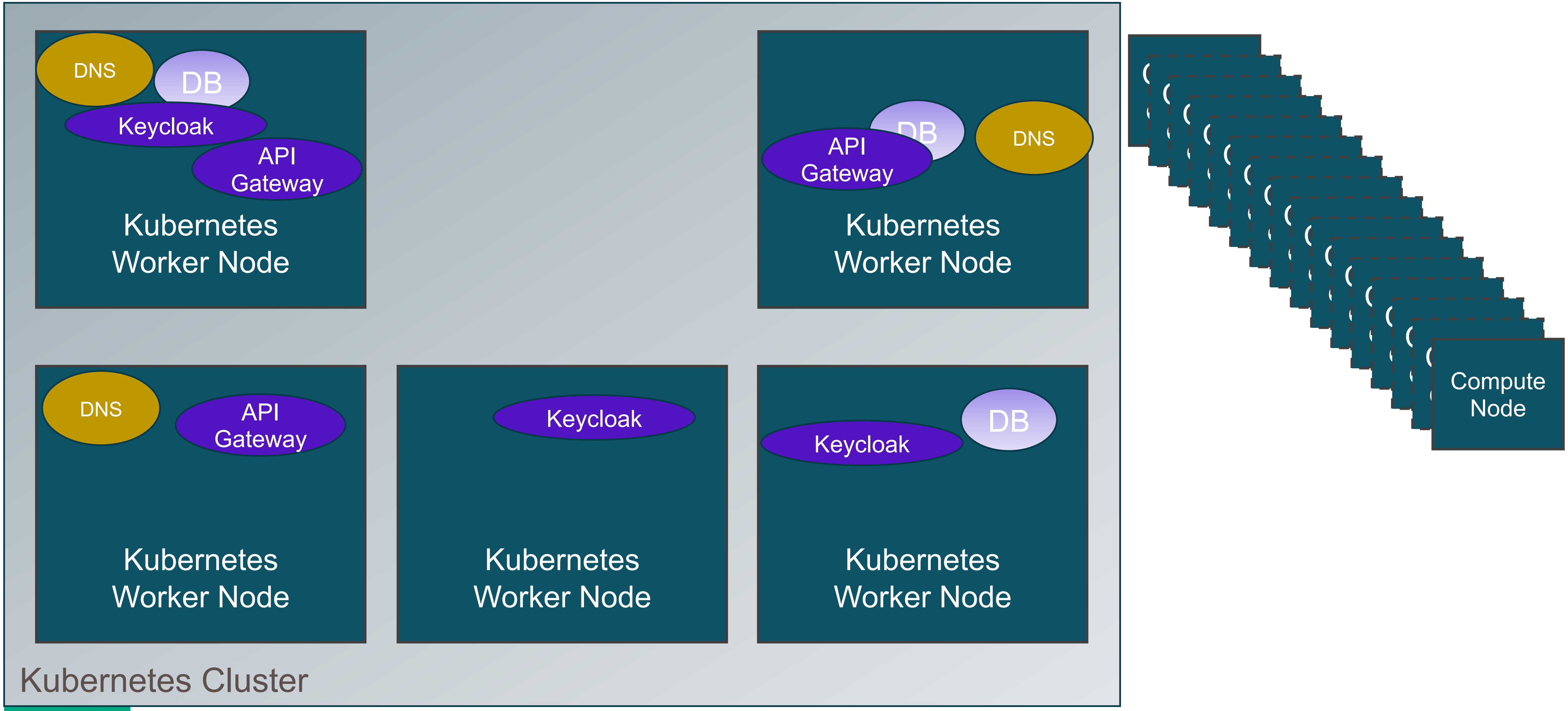
KUBERNETES SELECTS NEW NODES FOR PODS



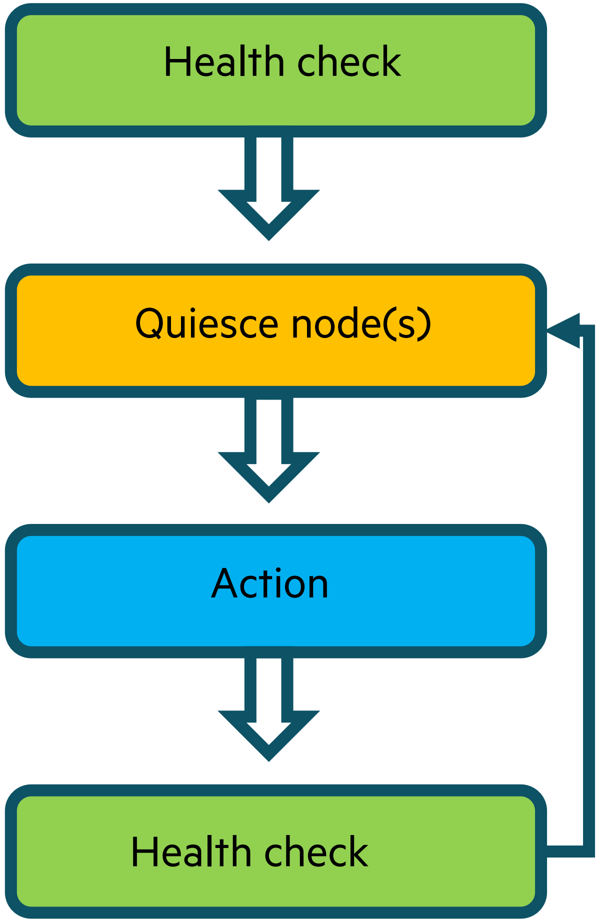
OLD PODS TERMINATE, NEW PODS SPIN UP



NODE REMOVED FROM KUBERNETES CLUSTER



ROLLING NCN MAINTENANCE



- **Rolling** process will remove one or more nodes from service while the rest still support management services needed by the compute nodes and application nodes
- System health check of management nodes and services
  - Only quiesce node(s) if passing
- One or a few nodes may be quiesced from service
  - Master nodes
    - One at a time to ensure quorum for Kubernetes and etcd is maintained
  - Worker nodes
    - One (or a few) at a time to ensure remaining worker nodes are not overloaded
  - Utility Storage nodes
    - One at a time to ensure data consistency for Ceph storage which has 3-way replication
- Perform action on quiesced node(s)
  - Node returns to service after the action
- System health check of management nodes and services
  - Only continue rolling to next node(s) if passing

ROLLING REBOOT/REBUILD FOR MANAGEMENT NODES

- **Rolling reboot**
  - Local **disk** on management node is **not wiped** before rebooting the node from the on-disk operating system
  - Kubernetes (k8s) master and worker nodes are drained from workload, removed from k8s, and then rejoin k8s after reboot
  - Might be used during upgrade of software products like COS when a change is needed for kernel modules (Lnet, Lustre, DVS) on worker nodes that support CPS or UAI
- **Rolling rebuild**
  - Local **disk** on management node is **wiped** before booting from new operating system image
  - Kubernetes (k8s) master and worker nodes are drained from workload, removed from k8s, and then rejoin k8s after rebuild
  - Might be used during an upgrade of CSM software which might include new images for management nodes
- Other actions could be done while the node is quiesced, such as firmware update or hardware component replacement
  - Replacing a disk in the node requires the rebuild procedure, not the reboot procedure
- Documentation and improving automation of procedures with every CSM release

ROLLING UPGRADE OF COMPUTE NODES

- New image and configuration should be prepared or staged in advance
- Compute Rolling Upgrade Service (CRUS) handles the following steps by calls to WLM, Boot Orchestration Service (BOS), and Hardware State Manager (HSM)
  - Admin selects a group of nodes to be changed
  - Identify the individual Boot Artifacts for the new software
  - Identify the desired configuration
  - Quiesce each node before taking the node out of service
  - Reboot the node into the upgraded state
  - Return the node to service within its respective WLM
- Without CRUS
  - Set the desired boot artifacts in Boot Script Service (BSS) and desired configuration in Configuration Framework Service (CFS)
    - Identify the individual Boot Artifacts for the new software
    - Identify the desired configuration
    - Update the nodes in BSS with the new Boot Artifacts
    - Ensure that the 'enable' attribute is set to 'False' for each node, so that CFS will only configure them upon reboot
    - Update the nodes in CFS with the new configuration
- Configure WLM to call something to do power shepherding on the nodes
  - CAPMC doesn't have a single call to guarantee shutdown (off) and boot (on)
  - Nothing in CSM 1.0 has this BOS logic to power shepherd the node (Coming soon)
    - Try graceful power off
    - Check power state
    - If graceful power off failed, do force power off
    - Check power state
    - If node still on, report error
    - Power on node
- Command so WLM (Slurm) starts reboot when current job is done on the node

```
scontrol reboot ASAP nextstate=resume
reason="reboot message”
```

KUBERNETES

WHAT IS KUBERNETES?

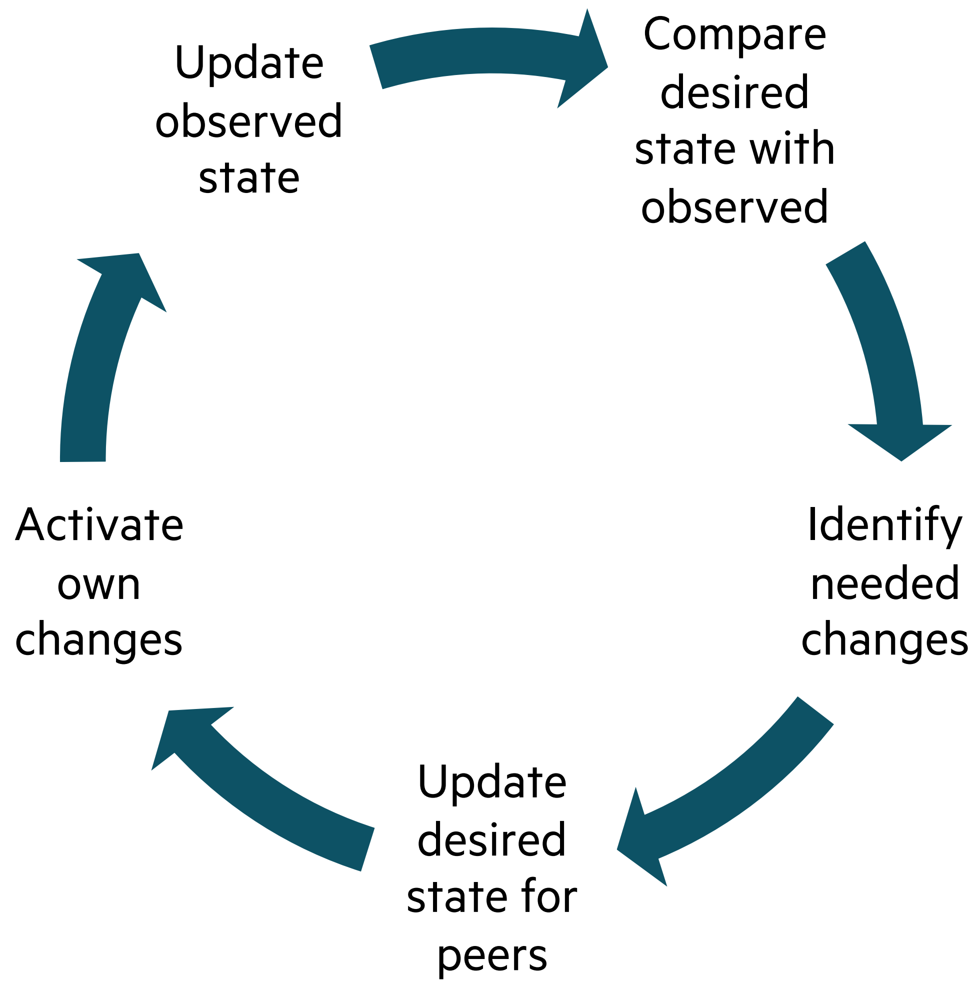
- Kubernetes (k8s)
  - Portable and extensible platform for managing containerized workloads and services
    - Application deployment
    - Scaling
    - Management
  - Resiliency feature
    - Desired number of deployments of a microservice are always running on one or more nodes
    - If one node becomes unresponsive, microservices are recreated on another node
- Each microservice is
  - Modular
  - Resilient
  - Fine-grained
  - Uses lightweight protocols
  - Can be updated independently

KUBERNETES RESOURCE ORCHESTRATION

How can this set of physical nodes fulfill the requested allocation of resources to ensure that all the desired resources are healthy and available?

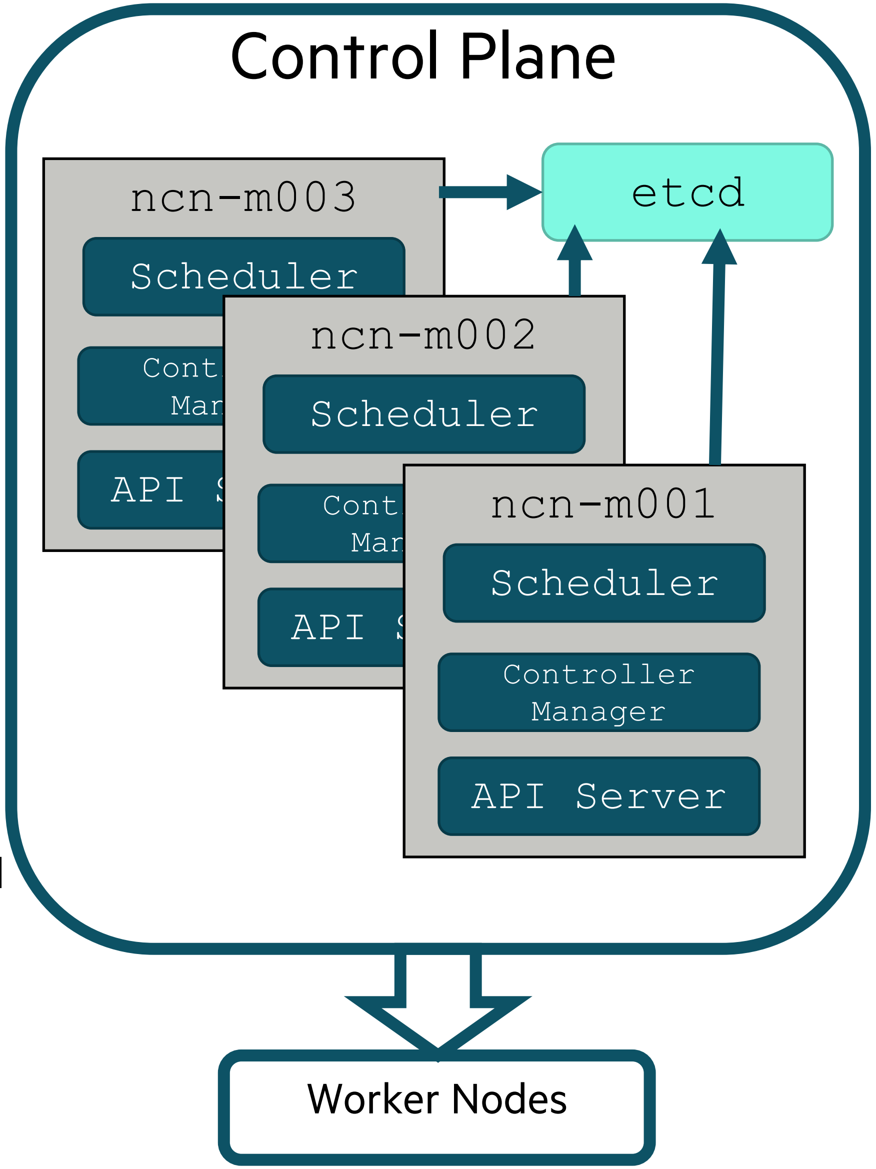
- There is a controller for each resource type that uses the scheduler to iteratively solve resource allocation in a constantly changing environment

- Pods
- Services
- ConfigMaps
- Secrets
- Volumes
- VirtualServices
- etcdClusters
- KafkaTopics



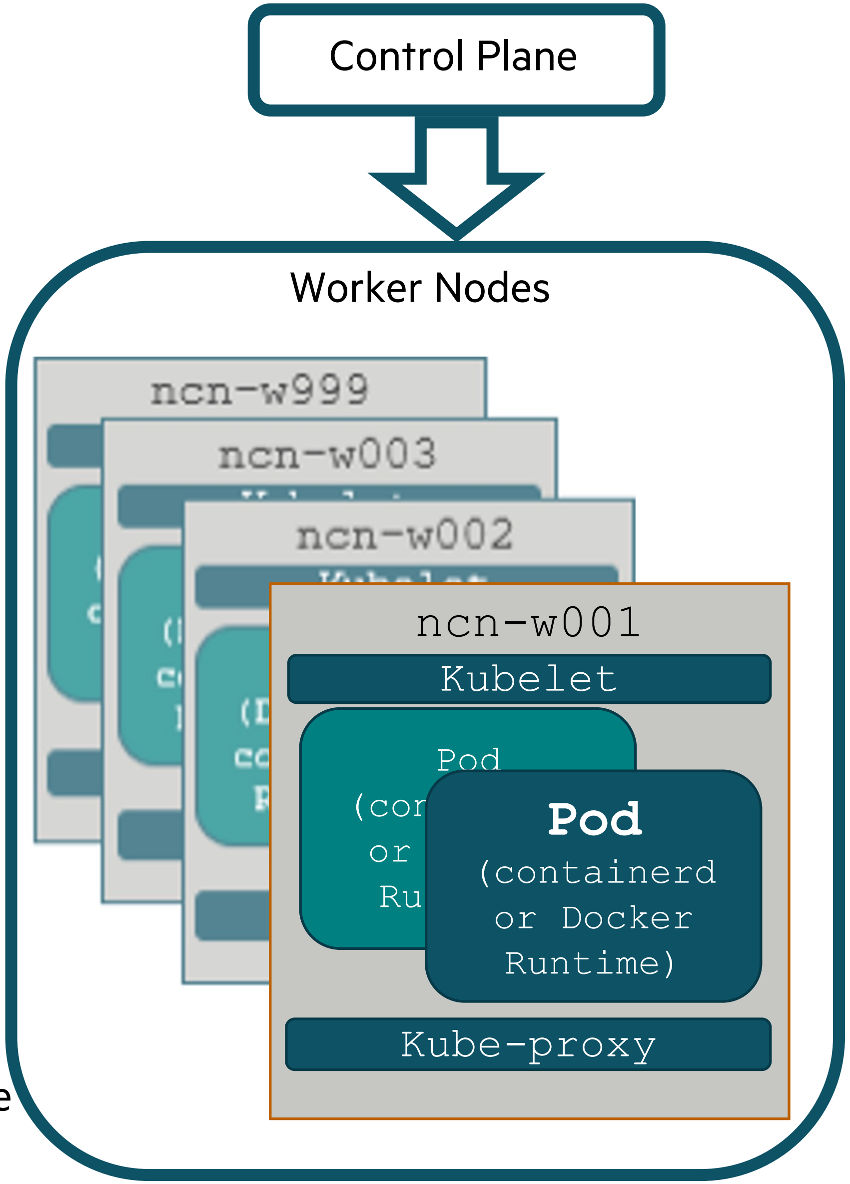
MASTER NODES

- The Kubernetes master nodes are grouped into a highly available cluster called the Kubernetes Control Plane, which manages the worker nodes and the pods in the Kubernetes cluster
- Systems will always have at least three Kubernetes master nodes called
  ncn-m001, ncn-m002 and ncn-m003
- Master node components:
  - **API server**
    - Serves the Kubernetes API and acts as the gateway to the Kubernetes cluster
    - Not to be confused with the System Management Services API Gateway
  - **Cluster state store (etcd)**
    - All persistent cluster state information is stored in a distributed storage system
    - On HPE Cray EX this is an instance of `etcd`
  - **Controller-Manager**
    - Performs many cluster-level functions including namespace creation, terminated pod garbage collection & scaling of pods controlled by a ‘ReplicaSet‘
  - **Scheduler**
    - Watches for unscheduled pods and binds them to worker nodes



WORKER NODES

- A worker node is a physical or virtual server that has the services necessary to run application containers and be managed from the master node(s)
- Worker node components
  - **Kubelet**
    - The Kubelet is the most important controller in Kubernetes
    - It is the primary implementer of the pod and node APIs
  - **Pod**
    - Runs on a node and is the smallest unit of replication on a Kubernetes cluster
    - One or more containers run logically in a Pod
  - **Container Runtime**
    - Responsible for downloading images and running containers
    - *containerd* is the container runtime used for system management
      - *Docker* is still being used in some pods, but development is moving to *containerd*
  - **Kube-proxy**
    - Provides a HA load-balancing solution for groups of replicated pods by creating a virtual IP accessible by clients and transparently proxied to the pods in a Service



VIEWING & CHANGING RESOURCES WITH `kubectl`

Command	Description and Options
kubectl get RESOURCE_TYPE	List instances of the specified resource type. Common resource types include pods, nodes, namespaces (ns), jobs, ReplicaSets (rs), deployments (deploy), PersistentVolumeClaims (pvc) & events
kubectl get RESOURCE_TYPE -n NAMESPACE	List resources of the given type in the specified namespace
kubectl get RESOURCE_TYPE -A	List resources of the given type for all namespaces
kubectl get RESOURCE_TYPE -o wide	List resources with wide output which typically provides extra information
kubectl get pod –o yaml POD	List very detailed information about a pod
kubectl describe RESOURCE_TYPE NAME	Show detailed information about the specified instance
kubectl logs POD_NAME	Show logs from the named pod (and any containers inside the pod with –c containername)
kubectl logs --since TIME POD_NAME	Show all logs from the named pod in the last specified time. Time is specified in terms of a number of time units e.g. 2h == two hours 10m == ten minutes
kubectl delete pods NAME	Delete the specified pod. This will generally result in restarting (recreating) the pod
kubectl exec –it POD_NAME -- COMMAND	Run the specified command interactively on the specified pod
kubectl apply –f FILE	Apply the specified file to create or update the item specified in the file. Files are typically YAML
kubectl scale –-replicas=NUM_OF_REPS APP	Scale the application (deployment, ReplicaSet, etc.) to the specified number of reps

OTHER NOTABLE ELEMENTS OF KUBERNETES ON CRAY EX

- **ReplicaSet** (`kubectl get rs`)
  - A ReplicaSet is used to maintain a stable set of replica pods running at any given time
- **Deployment** (`kubectl get deploy`)
  - A deployment provides declarative updates for pods and ReplicaSets
  - Deployments are used to create, modify, and remove ReplicaSets and, by extension, pods
- **Services** (`kubectl get svc`)
  - A service is an abstraction which defines a logical set of pods and a policy by which to access them
  - A service is used to expose a pod's IP address outside of the Kubernetes cluster
- **Jobs (`kubectl get jobs`)**
  - A job in Kubernetes is a supervisor for pods carrying out batch processes
    - A process that runs for a certain time to completion
- **CustomResourceDefinitions (`kubectl get crd`)**
  - A CustomResourceDefinition is used to create a new custom resource which is an extension of the Kubenetes API
  - Custom Resources are used in CFS as well as for security and monitoring

THE UNDERCLOUD – KUBERNETES SERVICES

Operators and Platform Integrations

- velero
- etcD operator
- postgres operator
- istio-operator
- cert-manager
- metallb
- spire
- hashicorp vault
- ceph storage classes
- Keycloak w/Keycloak Gatekeeper
- strimzi kafka
- Grafana and Prometheus

Conventions and Standards

- Keycloak provides JWT which is checked at Gateway
- All Gateways have TLS managed by cert-manager and published through external-dns
- Open Policy Agent applies unique policy for each gateway
- Services do not provide their own security
- OpenAPI 2.0 standards allow automatic documentation and CLI generation
- Loftsman manifests for templated helm installation

CSM KUBERNETES PLATFORM SERVICES

- Platform services are deployed in dedicated namespaces
- Per-namespace security and resource limits
- Service accounts required for any cross-namespace communication
- Plans to template namespace permissions with Hierarchical Namespace Controller

- Collections of services are deployed together with loftsman
- Loftsman offers GitOps-style management of collections with history
- CSM services use platform services for infrastructure

Namespaces

velero	backups	ceph	cert-manager	istio
gatekeeper	loftsman	metallb	opa	nexus
pki	spire	operators	sysmgmt-health	vault

VIEWING NAMESPACES

```
ncn# kubectl get namespaces
NAME                  STATUS   AGE
backups               Active   90d
ceph-cephfs           Active   90d
ceph-rbd              Active   90d
ceph-rgw              Active   90d
cert-manager          Active   90d
cert-manager-init     Active   90d
default               Active   90d
gatekeeper-system     Active   90d
ims                   Active   90d
istio-operator        Active   90d
istio-system          Active   90d
kube-node-lease       Active   90d
kube-public           Active   90d
kube-system           Active   90d
loftsman              Active   90d
metallb-system        Active   90d
nexus                 Active   90d
opa                   Active   90d
operators             Active   90d
pki-operator          Active   90d
services              Active   90d
sma                   Active   90d
spire                 Active   90d
sysmgmt-health        Active   90d
uas                   Active   90d
user                  Active   90d
vault                 Active   90d
velero                Active   90d
```

```
ncn# for NS in $(kubectl get namespaces --no-headers | awk '{print $1}'); \
do echo $NS "has" $(kubectl get pods -n $NS --no-headers 2>/dev/null |wc -l) "pods" ; done
backups has 0 pods
ceph-cephfs has 0 pods
ceph-rbd has 0 pods
ceph-rgw has 0 pods
cert-manager has 9 pods
cert-manager-init has 0 pods
default has 12 pods
gatekeeper-system has 5 pods
ims has 13 pods
istio-operator has 1 pods
istio-system has 13 pods
kube-node-lease has 0 pods
kube-public has 0 pods
kube-system has 38 pods
loftsman has 0 pods
metallb-system has 4 pods
nexus has 4 pods
opa has 3 pods
operators has 10 pods
pki-operator has 3 pods
services has 239 pods
sma has 53 pods
spire has 21 pods
sysmgmt-health has 11 pods
uas has 0 pods
user has 3 pods
vault has 5 pods
velero has 4 pods
```

A **namespace** is a way to subdivide Kubernetes clusters into virtual sub clusters

When listing resources with `kubectl`, the `-n NAMESPACE` tag is needed for resources not in the `default` namespace

The namespaces used and the number of pods in each is subject to change

VIEWING POD CHARACTERISTICS 1 OF 5

- A pod is the basic execution unit of a Kubernetes application
  - The smallest and simplest unit in the Kubernetes object model that can be created or deployed
  - A pod encapsulates an application's container(s), storage resources, a unique network IP, and options that govern how the container(s) should run

```
ncn# kubectl get pods -A |grep slingshot
services      slingshot-fabric-manager-599979fd6c-w9wbj     2/2     Running            0              60d
ncn# kubectl describe pod -n services slingshot-fabric-manager-599979fd6c-w9wbj
Name:           slingshot-fabric-manager-599979fd6c-w9wbj
Namespace:      services
Priority:       0
Node:           ncn-w001/10.252.1.12
Start Time:     Fri, 05 Nov 2021 15:15:57 +0000
Labels:         app.kubernetes.io/instance=slingshot-fabric-manager
                app.kubernetes.io/name=slingshot-fabric-manager
<< snip >>
Annotations:    k8s.v1.cni.cncf.io/networks-status:
                  [{
                      "name": "weave",
                      "ips": [
                          "10.44.0.53"
                      ],
                      "default": true,
                      "dns": {}
                  }]
```

Name: The unique name of the pod.

Namespace: The namespace of which the pod is a member.

Node: The Kubernetes node on which the pod is scheduled

Labels: Key/value pairs identify attributes of the pod that are meaningful to users.

Annotations: Key/value pairs used to attach non-identifying attributes to a pod. Annotations are available to clients of the pod. For example, an annotation could be used as a pointer to a logging repository

VIEWING POD CHARACTERISTICS 2 OF 5

```
Status:         Running
IP:           10.44.0.53
IPs:
  IP:           10.44.0.53
Controlled By:  ReplicaSet/slingshot-fabric-manager-599979fd6c
Init Containers:
  istio-init:
    Container ID:  containerd://075af9f171157c63f22e133d3eb9a79b3ad4d8c9b04a4fdbbebb003639bcc082
    Image:         dtr.dev.cray.com/cray/istio/proxyv2:1.7.8-cray2-distroless
    Image ID:
dtr.dev.cray.com/cray/istio/proxyv2@sha256:3aa788734a525077bba128f140b2871a53b4c13245618932dc04be6a70ca2a2f
...<snip>...
 State:          Terminated
      Reason:       Completed
      Exit Code:    0
      Started:      Fri, 05 Nov 2021 15:16:16 +0000
      Finished:     Fri, 05 Nov 2021 15:16:16 +0000
    Ready:          True
    Restart Count:  0
    Limits:
      cpu:     2
      memory:  1Gi
    Requests:
      cpu:     10m
      memory:  10Mi
```

`Status:` Summary of where the pod is in its lifecycle. Possible values include “Pending”, “Running”, “Succeeded”, “Failed”, and “Unknown”

`IP:` IP address assigned to the pod from CIDR range of its host node

`Controlled By:` Controller responsible for creating & managing the pod. Common controllers include DaemonSets, ReplicaSets, deployments, and jobs

`Init Containers:` Specialized containers that run before app containers in a pod. Init containers can contain utilities or setup scripts not present in an app image

`State:` The state of the container(s) inside the pod.
`waiting, running, terminated`

`Limits & Requests:` Control resource utilization and limit how much of a resource the container is allowed to use. Requests are also used by the scheduler to determine which node would be best suited to place the pod

VIEWING POD CHARACTERISTICS 3 OF 5

Containers: Kubernetes pods exist to encapsulate application containers. Most HPE Cray EX containers are `containerd` or Docker containers but that is not a strict requirement

```
Containers:
  slingshot-fabric-manager:
    Container ID:   containerd://9ff6a5fa182cc6d757886e0ccbefc1b814dddf312043c96972414bb75b0c73df
    Image:          dtr.dev.cray.com/cray/slingshot-fabric-manager:1.6.0-2153-20211104145551_298bfa3
    Image ID:       dtr.dev.cray.com/cray/slingshot-fabric-
                      manager@sha256:8b5a38eb5944cfc017fcf7daf6290abe050dfd47647b061ea6d649cba4e193b5
    Port:           8000/TCP
    Host Port:      0/TCP
    State:          Running
      Started:      Fri, 05 Nov 2021 15:17:17 +0000
    Ready:          True
...<snip>...
```

Mounts: Containers' mounted filesystems

```
 Mounts:
      /opt/cray from data (rw,path="cray-data")
      /opt/slingshot/config from data (rw,path="fmn-config")
      /opt/slingshot/data from data (rw,path="fmn-data")
      /var/run/configmap/ca-public-key from ca-public-key (rw)
      /var/run/secrets/kubernetes.io/serviceaccount from default-token-sbbhv (ro)
      /var/run/secrets/kubernetes.io/serviceaccount/admin-client-auth from admin-client-auth (rw)
...<snip>...

Conditions:
  Type              Status
  Initialized       True
  Ready             True
  ContainersReady   True
  PodScheduled      True
```

Conditions: An array of pod conditions (or states) through which the pod has or has not passed "PodScheduled", "Ready", "Initialized", "ContainersReady", and "Unschedulable"

VIEWING POD CHARACTERISTICS 4 OF 5

`Volumes:` Shared storage available to all containers within the pod. There are many different types of volumes including `PersistentVolumeClaim, configMap, secret & emptyDir` as well as `local` for local volumes.
Once a volume is identified it can be described with: **`kubectl describe VOLUME_TYPE VOLUME_NAME`**

```
Volumes:
  admin-client-auth:
    Type:         Secret (a volume populated by a Secret)
    SecretName:   admin-client-auth
    Optional:     false
  ca-public-key:
    Type:       ConfigMap (a volume populated by a ConfigMap)
    Name:       cray-configmap-ca-public-key
    Optional:   false
  data:
    Type:        PersistentVolumeClaim (a reference to a PersistentVolumeClaim in the same namespace)
    ClaimName:   slingshot-fabric-manager-data-claim
    ReadOnly:    false
```

`ConfigMap:` see upcoming slide

`PersistentVolume(pv):` A Persistent Volume is a chunk of storage in the Kubernetes cluster. PVs are volume plugins like volumes but have a lifecycle independent of any individual pod that uses the PV

`PersistentVolumeClaims(pvc):` A Persistent Volume Claim (PVC) is a request for storage (PersistentVolume resources) by a user. Claims can request specific size and access modes (e.g., RW, RO)

VIEWING POD CHARACTERISTICS 5 OF 5

```
 default-token-sbbhv:
    Type:         Secret (a volume populated by a Secret)
    SecretName:   default-token-sbbhv
    Optional:     false
...<snip>...

Node-Selectors:   <none>

Tolerations:      node.kubernetes.io/not-ready:NoExecute op=Exists for 300s
                  node.kubernetes.io/unreachable:NoExecute op=Exists for 300s

Events:           <none>
```

Secrets : see upcoming slide

Node-Selectors: Limits pod to run only on a specific node whose label matches the nodeselector label specified, e.g., only run on a node with SSD attached to it

Tolerations: Tolerations allow a pod to be scheduled on a node with a particular **taint.** Taints are assigned to nodes and tolerations are assigned to pods. Both taints and tolerations are assigned in terms of key/value pairs

Events: Kubernetes events are automatically created when other resources have state changes, errors, or other messages that should be broadcast to the system

SECRETS

- A secret is an object that contains sensitive data such as a password, a token, or a key
- The contents of a secret could be put in a pod specification or in an image as opposed to a secret
  - Using a secret allows for more control over how the sensitive data is presented and consumed
- Example of retrieving and decoding a secret

```
ncn# kubectl get secret admin-client-auth
NAME                TYPE     DATA   AGE
admin-client-auth   Opaque   3      47d
ncn# kubectl get secret admin-client-auth -o yaml | head -6
apiVersion: v1
data:
  client-id: YWRtaW4tY2xpZW50
  client-secret: Y2U5N2UyODYtYjIyNy00MjY5LTljODYtNzYzOGJhOWJlNDRj
  endpoint:
aHR0cHM6Ly9hcGktZ3ctc2VydmljZS1ubW4ubG9jYWwva2V5Y2xvYWsvcmVhbG1zL3NoYXN0YS9wcm90b2NvbC9vcGVuaWQtY29ubmVjdC90
b2tlbg==
kind: Secret
ncn# kubectl get secrets admin-client-auth -ojsonpath='{.data.client-secret}'
Y2U5N2UyODYtYjIyNy00MjY5LTljODYtNzYzOGJhOWJlNDRj
ncn# echo "$(kubectl get secrets admin-client-auth -ojsonpath='{.data.client-secret}' | base64 -d)"
ce97e286-b226-4269-9c86-7638ba9be44c
```

CONFIGMAPS

- A ConfigMap holds key-value pairs of configuration data that can be consumed in pods
- ConfigMaps, unlike secrets, are designed to support working with strings that do not contain sensitive information (ConfigMaps don't provide secrecy or encryption)

Example: Several Slurm configuration files are available via a ConfigMap:

```
ncn# kubectl describe configmap slurm-map
Name:          slurm-map
Namespace:     default
Labels:        <none>
Annotations:   <none>
Data
====
cgroup.conf:
…
gres.conf:
…
plugstack.conf:
…
slurm.conf:
# slurm.conf file generated by configurator easy.html.
# Put this file on all nodes of your cluster.
# See the slurm.conf man page for more information.
#
SlurmctldHost=slurm-host(10.252.2.4)
#
#LaunchParameters=enable_nss_slurm
#MailProg=/bin/mail
MpiDefault=cray_shasta
MpiParams=ports=20000-32767
…
```

CEPH

CEPH UTILITY STORAGE

- Ceph is the utility storage platform used with the System Management Services (SMS)
- Used in conjunction with Kubernetes and `etcd` to run the SMS
  - Used only for the SMS-supported services
  - Not intended to be storage (`/home`) for system users
- Enables pods to store persistent data and provides block, object, and file storage to the SMS
  - Data in use by a Kubernetes node that goes down is still accessible to other nodes
  - Used only for supported services such as pod log files, repositories, and images
    - Ceph does not store "user" level files (like applications and datasets)
- Stores large amounts of telemetry and log data
- Runs as an external process native on the utility storage nodes
  - The Storage nodes and Ceph filesystem are started prior to bringing up the Kubernetes pods
  - The Kubernetes pods require the storage provided by Ceph
- Administrative commands
  - ceph
  - cephadm

CEPH CONCEPTS AND COMPONENTS

- **Replicas** – Objects are replicated to protect against data loss
- **Pools** – Logical partitions of the Ceph cluster for storing objects
  - Replicated pools –All objects in a pool are replicated on multiple OSDs (like RAID1)
  - Erasure Coded (EC) pools – Objects are not replicated but can tolerate the loss of an OSD (like RAID5 or RAID6)
- **Placement Group (PG)** – An internal implementation detail of how Ceph distributes data
- **RADOS Block Device (RBD)** – The block storage component of Ceph
- **CephFS** – The POSIX file system components of Ceph
- **RADOS Gateway (RGW)** – The S3/Swift gateway component of Ceph
  - Simple Storage Service (S3) is HTTP REST API to get, put, post, and delete data
    - HPE Cray EX system is not using AWS (Amazon Web Services) S3
- **CephX** – The Ceph authentication protocol, it operates like Kerberos, but it has no single point of failure
- **BlueStore** – Ceph-specific backing for OSDs that improves performance 2-3 times over the previous FileStore implementation used

CEPH DAEMONS

- Object storage daemon (`ceph-osd`)
  - The OSD is both the block device (disk) and the daemon on top of it
  - Every disk is an OSD and there is no limit to how many OSDs are supported
- Monitors (`ceph-mon`)
  - Maintains maps of the Ceph cluster state, including: MONs, OSDs, Managers, and the CRUSH (Controlled Replication Under Scalable Hashing) map
  - Manages authentication (using CephX) between Ceph daemons and clients
- Managers (`ceph-mgr`)
  - Tracks runtime metrics (e.g., storage utilization, system load)
  - Hosts modules for the Ceph Dashboard and the Ceph REST API
- Metadata servers (`ceph-mds`)
  - Stores metadata for the Ceph file system to support POSIX file system commands (e.g., ls, find)
- Crash module (`ceph-crash`)
  - Collects information about daemon crash dumps and stores it in the Ceph cluster for later analysis
  - Dumps are stored in `/var/lib/ceph/crash` by default

CEPH STATUS

- Ceph status shows health, expected and running services, storage information

```
ncn-s# ceph -s
  cluster:
    id:     b1781806-9370-43af-96aa-61447a4d9411
    health: HEALTH_OK

  services:
    mon: 3 daemons, quorum ncn-s003,ncn-s002,ncn-s001 (age 6w)
    mgr: ncn-s001(active, since 6w), standbys: ncn-s003, ncn-s002
    mds: cephfs:1 {0=ncn-s001=up:active} 2 up:standby
    osd: 24 osds: 24 up (since 6w), 24 in (since 6w)
    rgw: 3 daemons active (ncn-s001.rgw0, ncn-s002.rgw0, ncn-s003.rgw0)

  data:
    pools:   11 pools, 816 pgs
    objects: 357.05k objects, 786 GiB
    usage:   1.2 TiB used, 41 TiB / 42 TiB avail
    pgs:     816 active+clean

  io:
    client:   75 KiB/s rd, 10 MiB/s wr, 24 op/s rd, 1.07k op/s wr
```

OTHER CEPH COMMANDS

```
ncn-s# ceph node ls mon
{
    "ncn-s001": [
        "ncn-s001"
    ],
    "ncn-s002": [
        "ncn-s002"
    ],
    "ncn-s003": [
        "ncn-s003"
    ]
}
ncn-s# ceph
alerts             df                 log                progress           telemetry
auth               features           mds                prometheus         tell
balancer           fs                 mgr                quorum_status      test_orchestrator
cephadm            fsid               mon                rbd                time-sync-status
config             health             nfs                report             versions
config-key         influx             node               restful            zabbix
crash              insights           orch               service
dashboard          iostat             osd                status
device             k8sevents          pg                 telegraf
```

Ceph commands support tab completion and tab tab to get a list of available options

STORAGE UTILIZATION

```
ncn-s# ceph df
--- RAW STORAGE ---
CLASS   SIZE    AVAIL   USED     RAW USED  %RAW USED
ssd     63 TiB  60 TiB  2.8 TiB   2.9 TiB       4.55
TOTAL   63 TiB  60 TiB  2.8 TiB   2.9 TiB       4.55

--- POOLS ---
POOL                        ID  PGS  STORED   OBJECTS  USED     %USED  MAX AVAIL
cephfs_data                  1  256  385 GiB  311.95k  1.1 TiB   1.96     19 TiB
cephfs_metadata              2  256  405 MiB   19.83k  1.2 GiB      0     19 TiB
default.rgw.buckets.data     3  256  103 GiB   27.96k  309 GiB   0.53     19 TiB
default.rgw.buckets.index    4   32  3.1 MiB      704  9.2 MiB      0     19 TiB
.rgw.root                    5   16  5.2 KiB       18  204 KiB      0     19 TiB
default.rgw.control          6   16      0 B        8      0 B      0     19 TiB
default.rgw.meta             7   16  788 KiB      171  3.9 MiB      0     19 TiB
default.rgw.log              8   16   30 KiB      210  624 KiB      0     19 TiB
kube                         9  256   36 GiB   18.30k   76 GiB   0.13     19 TiB
smf                         10  512  1.1 TiB  488.25k  1.3 TiB   2.28     28 TiB
default.rgw.buckets.non-ec  11   16      0 B        0      0 B      0     19 TiB
device_health_metrics       12    1   48 MiB       39  145 MiB      0     19 TiB
```

ETCD

ETCD

- etcd is a distributed reliable key-value store for the most critical data of a distributed system, with a focus on being:
  - *Simple*: well-defined, user-facing API (gRPC)
  - *Secure*: automatic TLS with optional client cert authentication
  - *Fast*: benchmarked 10,000 writes/sec
  - *Reliable*: properly distributed using Raft
  - etcd is written in Go and uses the Raft consensus algorithm to manage a highly-available replicated log
  - General documentation - https://github.com/etcd-io/etcd
- CSM utilizes etcd in two major ways:
  - etcd running on Kubernetes master nodes
  - etcd running via a Kubernetes operator as data store for a specific service

ETCD FOR KUBERNETES

- etcd running on Kubernetes master nodes
  - Supports only Kubernetes datastore needs
    - Failures in the etcd cluster at the heart of Kubernetes will cause a failure of Kubernetes
    - To mitigate this risk, the system is deployed with etcd on dedicated disks and with a specific configuration to optimize Kubernetes workloads
  - Includes a dedicated partition to provide the best throughput and scalability
    - Enables the Kubernetes services to be scaled, as well as the physical nodes running those services
  - Run on the Kubernetes master nodes and will not relocate
    - Handles replication and instance re-election in the event of a node failure
    - Scaling to more nodes will provide more resiliency, but it will not provide more speed
      - For example, one write to the cluster is actually three writes, so one to each instance
      - Scaling to five or more instances in a cluster would mean that one write will actually equal five writes to the cluster
  - Backed up to a Ceph Rados Gateway (S3 compatible) bucket

ETCD FOR SERVICES

- etcd running via a Kubernetes operator
  - Services utilize this to deploy an etcd cluster on the worker nodes
  - These additional clusters do not interact with the core Kubernetes etcd service
  - The etcd pods are mobile and will relocate to another worker node in the event of a pod or node failure
  - Each etcd cluster can be backed up to a Ceph Rados Gateway (S3 compatible) bucket
    - This option is decided by the service owner or developer as some information has an extremely short lifespan, and by the time the restore could be performed, the data would be invalid

ETCD CLUSTERS

```
ncn# kubectl get pods -A -l app=etcd -o wide |head -19
NAMESPACE NAME                       READY  STATUS   RESTARTS  AGE  IP           NODE      NOMINATED NODE  READINESS GATES
Services  cray-bos-etcd-b4xhqkrsxd   1/1    Running  0         62d  10.36.0.30   ncn-w002  <none>          <none>
services  cray-bos-etcd-d7ffv7pc7v   1/1    Running  0         56d  10.32.0.31   ncn-w003  <none>          <none>
services  cray-bos-etcd-scqbx8rvqf   1/1    Running  0         62d  10.44.0.14   ncn-w001  <none>          <none>
services  cray-bss-etcd-brp85brbnd   1/1    Running  0         56d  10.32.0.44   ncn-w003  <none>          <none>
services  cray-bss-etcd-h7pbzq9k5d   1/1    Running  0         62d  10.36.0.23   ncn-w002  <none>          <none>
services  cray-bss-etcd-hx7dv2gt9n   1/1    Running  0         62d  10.44.0.24   ncn-w001  <none>          <none>
services  cray-cps-etcd-lwfwq6xr8s   1/1    Running  0         62d  10.36.0.31   ncn-w002  <none>          <none>
services  cray-cps-etcd-q5ggc7b2gz   1/1    Running  0         62d  10.44.0.39   ncn-w001  <none>          <none>
services  cray-cps-etcd-snp2kpm2pp   1/1    Running  0         56d  10.32.0.27   ncn-w003  <none>          <none>
services  cray-crus-etcd-9gfz9wb2bn  1/1    Running  0         62d  10.36.0.24   ncn-w002  <none>          <none>
services  cray-crus-etcd-kzl9kh9b52  1/1    Running  0         56d  10.32.0.37   ncn-w003  <none>          <none>
services  cray-crus-etcd-nxhbk79hx6  1/1    Running  0         62d  10.44.0.34   ncn-w001  <none>          <none>
services  cray-externaldns-etcd-c9…  1/1    Running  0         62d  10.36.0.16   ncn-w002  <none>          <none>
services  cray-externaldns-etcd-h5…  1/1    Running  0         62d  10.44.0.15   ncn-w001  <none>          <none>
services  cray-externaldns-etcd-jp…  1/1    Running  0         56d  10.32.0.46   ncn-w003  <none>          <none>
services  cray-fas-etcd-6z825mv7b6   1/1    Running  0         56d  10.32.0.43   ncn-w003  <none>          <none>
services  cray-fas-etcd-9hs8d56nsp   1/1    Running  0         62d  10.36.0.18   ncn-w002  <none>          <none>
services  cray-fas-etcd-lt2rcvnwws   1/1    Running  0         62d  10.44.0.23   ncn-w001  <none>          <none>
```

When only 2 pods in a cluster are running that indicates an unbalanced etcd cluster

RESTORE AN ETCD CLUSTER FROM AUTOMATIC BACKUP – PART 1

```
ncn# cray bos sessiontemplate list
results = []
```

Cray service is missing data after an `etcd` cluster restart
In this example for BOS (Boot Orchestration Service)

```
ncn# kubectl exec -it -n operators \
$(kubectl get pod -n operators | grep etcd-backup-restore | head -1 | awk '{print $1}') \
-c boto3 -- list_backups cray-bos

cray-bos/etcd.backup_v2821522_2021-12-28-19:34:57
cray-bos/etcd.backup_v2825842_2021-12-29-19:34:57
cray-bos/etcd.backup_v2830162_2021-12-30-19:34:57
cray-bos/etcd.backup_v2834482_2021-12-31-19:34:57
cray-bos/etcd.backup_v2838802_2022-01-01-19:34:57
cray-bos/etcd.backup_v2843122_2022-01-02-19:34:57
cray-bos/etcd.backup_v2847442_2022-01-03-19:34:57
```

Step 1) List Backups for the desired etcd cluster (e.g., BOS)

`etcd` cluster backups & restores are explained in the CSM documentaiton

Step 2) Restore the cluster using a backup

```
ncn# kubectl exec -it -n operators \
$(kubectl get pod -n operators | grep etcd-backup-restore | head -1 | awk '{print $1}') \
-c util -- restore_from_backup cray-bos etcd.backup_v2847442_2022-01-03-19:34:57

etcdrestore.etcd.database.coreos.com/cray-bos-etcd created
```

Custom resource created during restore operation

RESTORE AN ETCD CLUSTER FROM AUTOMATIC BACKUP – PART 2

```
ncn# kubectl -n services get pod | grep bos
cray-bos-5d886cc78d-f72h6  2/2     Running     0     7d15h
cray-bos-5d886cc78d-v2rd2  2/2     Running     0     7d17h
cray-bos-5d886cc78d-vzl4p  2/2     Running     0     7d13h
cray-bos-etcd-wdcvldzlxt   0/1     Init:0/3    0     26s
ncn# kubectl -n services get pod | grep bos-etcd
cray-bos-etcd-bctpgwl5l5   0/1     Init:0/1    0      11s
cray-bos-etcd-wdcvldzlxt   1/1     Running     0      52s
ncn# kubectl -n services get pod -o wide | grep bos-etcd
cray-bos-etcd-bctpgwl5l5   0/1     Init:0/1    0      26s     10.39.0.138   ncn-w003   <none>     <none>
cray-bos-etcd-wdcvldzlxt   1/1     Running     0      67s     10.47.0.228   ncn-w002   <none>     <none>
ncn# kubectl -n services get pod -o wide | grep bos-etcd
cray-bos-etcd-bctpgwl5l5   1/1     Running     0      2m      10.39.0.138   ncn-w003   <none>     <none>
cray-bos-etcd-wdcvldzlxt   1/1     Running     0      2m41s   10.47.0.228   ncn-w002   <none>     <none>
cray-bos-etcd-x48gq2nh25   1/1     Running     0      71s     10.42.1.2     ncn-w001   <none>     <none>
```

Step 3) Watch the pods come back online

```
ncn# cray bos sessiontemplate list | grep name
name = "cos-sessiontemplate-2.0.27"
name = "uan-sessiontemplate-2.0.1"
```

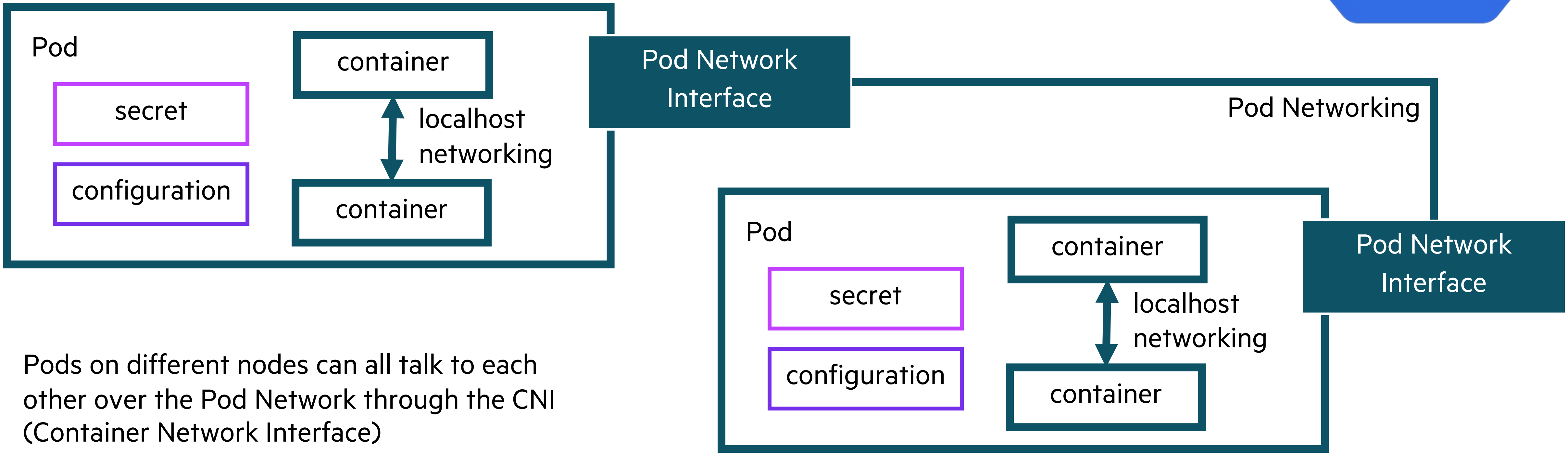
Step 4) Delete the `etcdrestore` custom resource to allow for future restores to occur

```
ncn# kubectl -n services delete etcdrestore.etcd.database.coreos.com/cray-bos-etcd
etcdrestore.etcd.database.coreos.com "cray-bos-etcd" deleted
```

ISTIO SERVICE MESH AND API GATEWAY

KUBERNETES IN SYSTEM MANAGEMENT

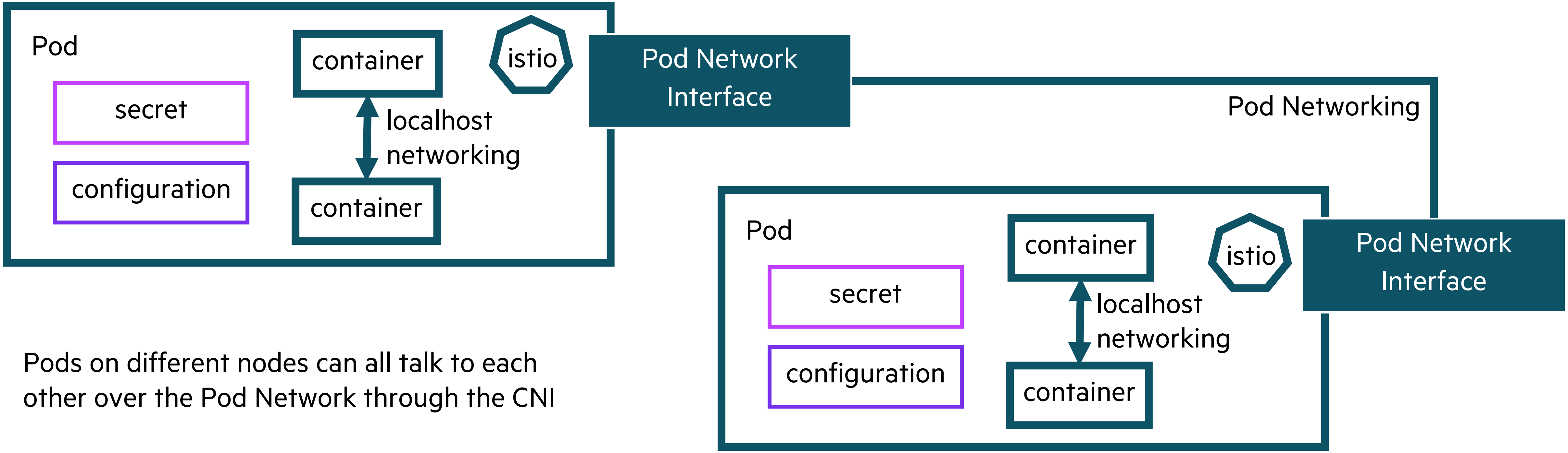
- Kubernetes is a tool for building platforms
- Higher level abstractions for grouping containerized services with their proprietary networking, data, and secrets and scheduling them for concurrent execution
- Declarative policy languages to express complex interactions
- Extensible with custom resources and reconciliation loops



Pods on different nodes can all talk to each other over the Pod Network through the CNI (Container Network Interface)

ISTIO EXTENDS KUBERNETES

- Istio attaches to the Pod Network Interface and intercepts all network traffic entering or leaving each Pod
- Istio containers in each pod are coordinated through the Istio control plane to intercept, upgrade, and redirect HTTP/gRPC traffic on the Pod Network



Pods on different nodes can all talk to each other over the Pod Network through the CNI

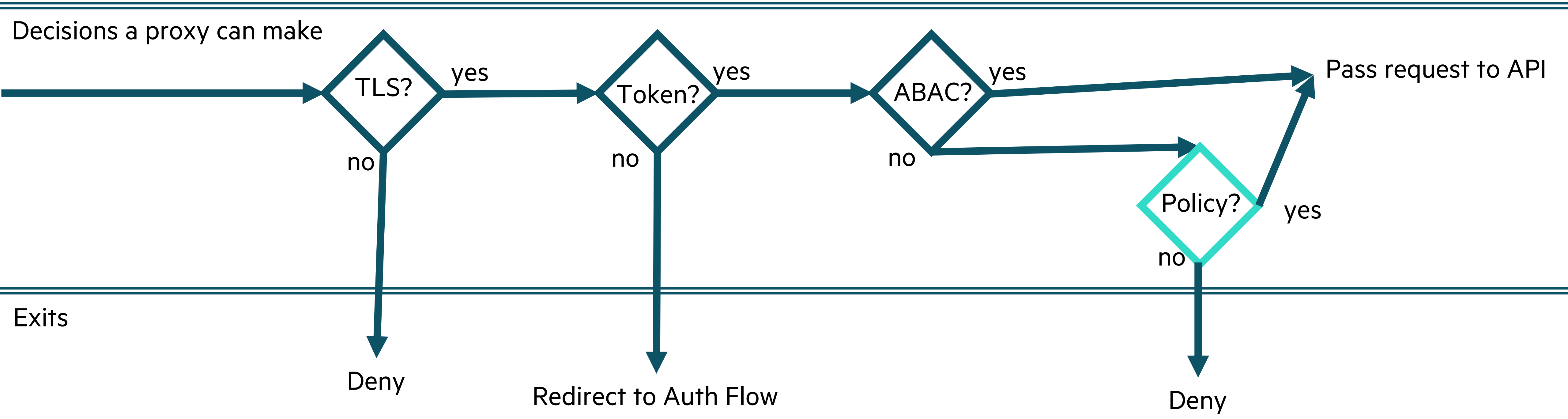
WHAT DOES AN API PROXY DO?

Is this connection properly encrypted?

Does this request include a **secure token** that is **valid for this action**?

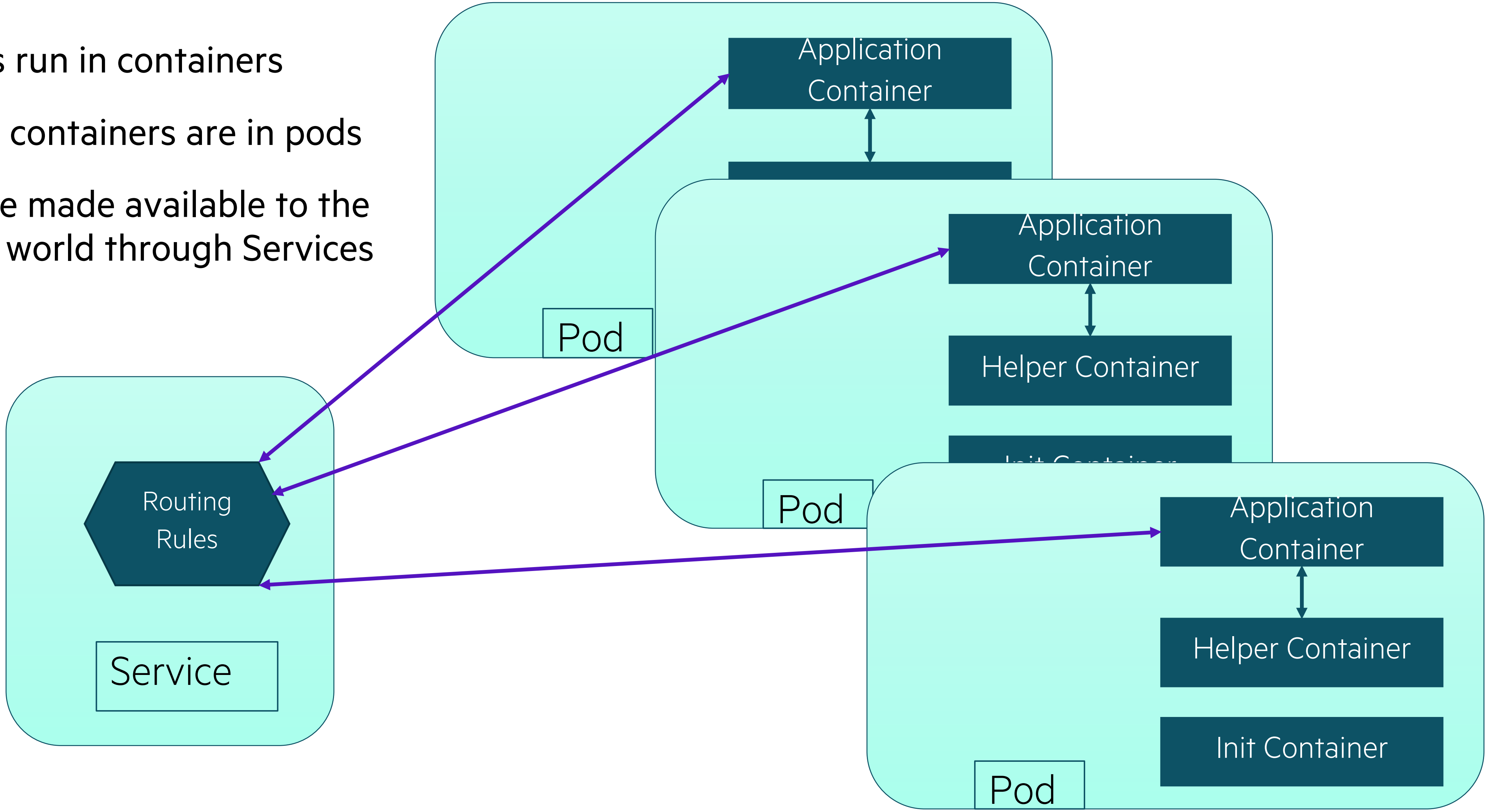
Does an **existing policy** decision exist permitting this action?

Do **all policies agree** that this action can proceed?



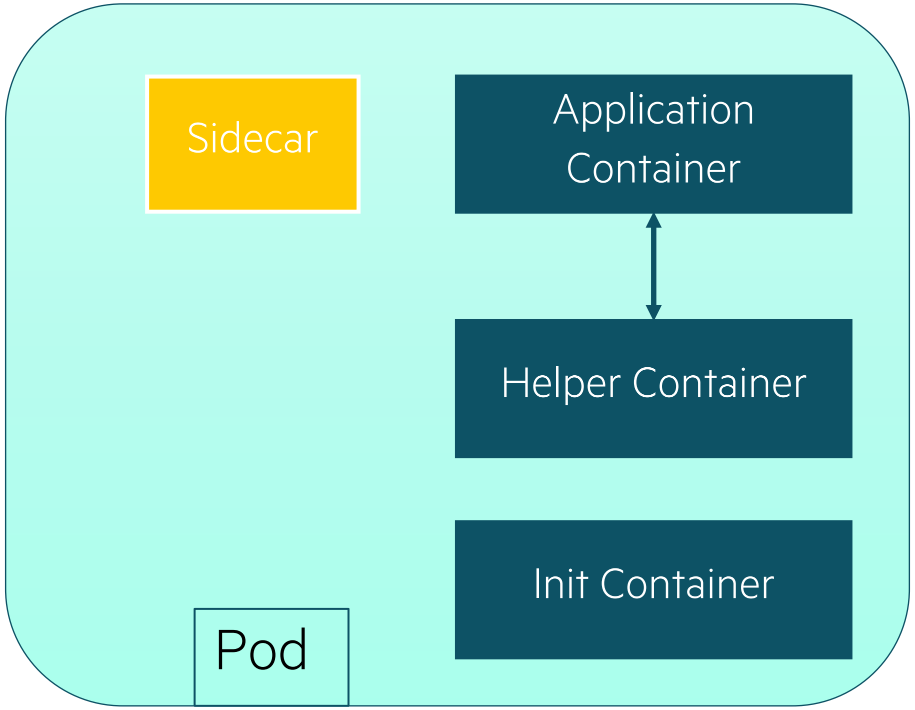
WHAT'S A SERVICE MESH: BASIC PODS/SERVICES

- Binaries run in containers
- Related containers are in pods
- Pods are made available to the outside world through Services



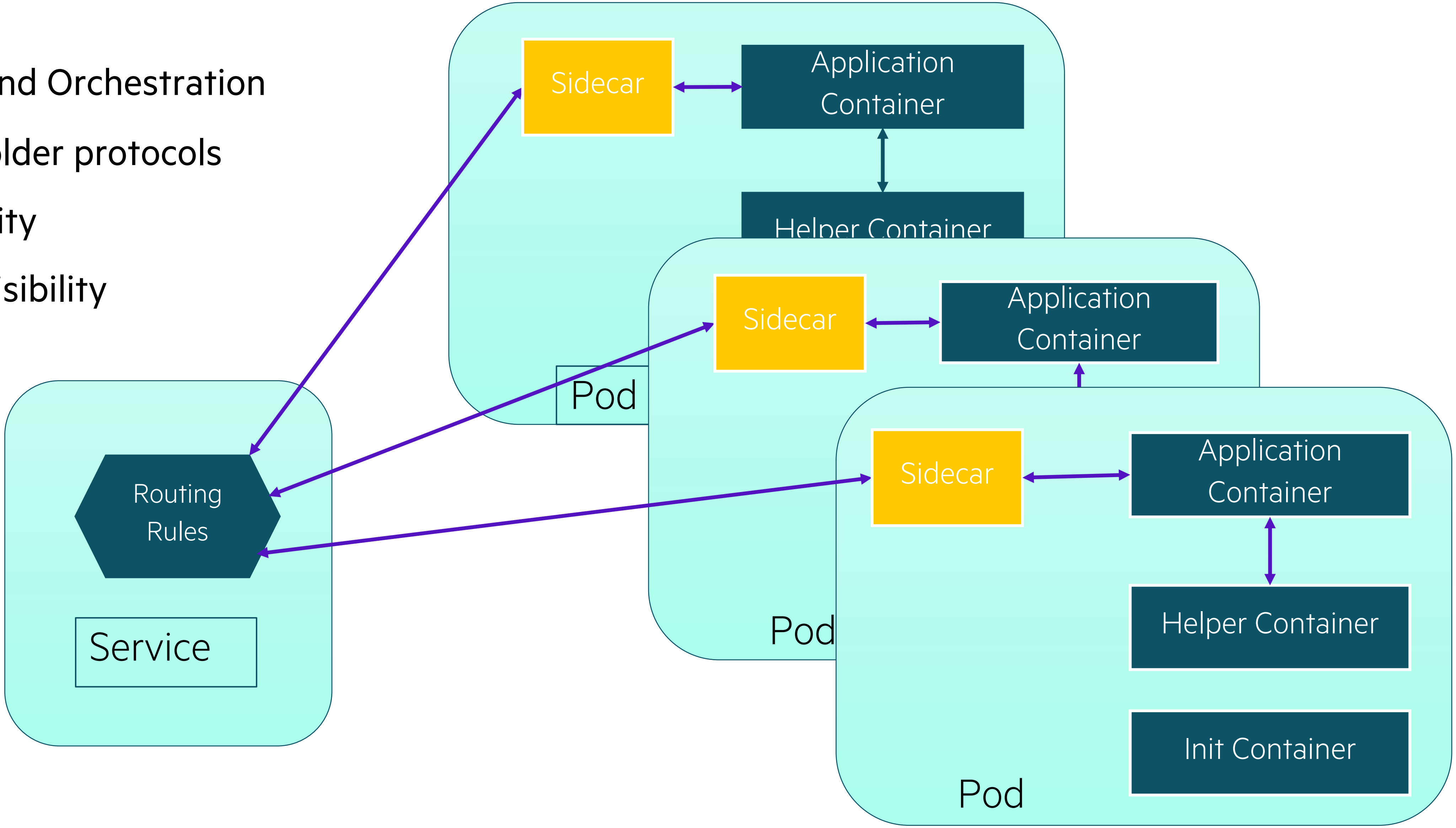
WHAT'S A SERVICE MESH: SIDECARS

- Sidecars are containers
- Share the Pod
- Affect traffic in and out of the pod
- Add and Standardize
  - Encryption
  - Logging
  - Tracing
  - Transparent mTLS between services
  - Managed TLS for API Clients
- Upgrades existing applications
- New applications can focus on business logic



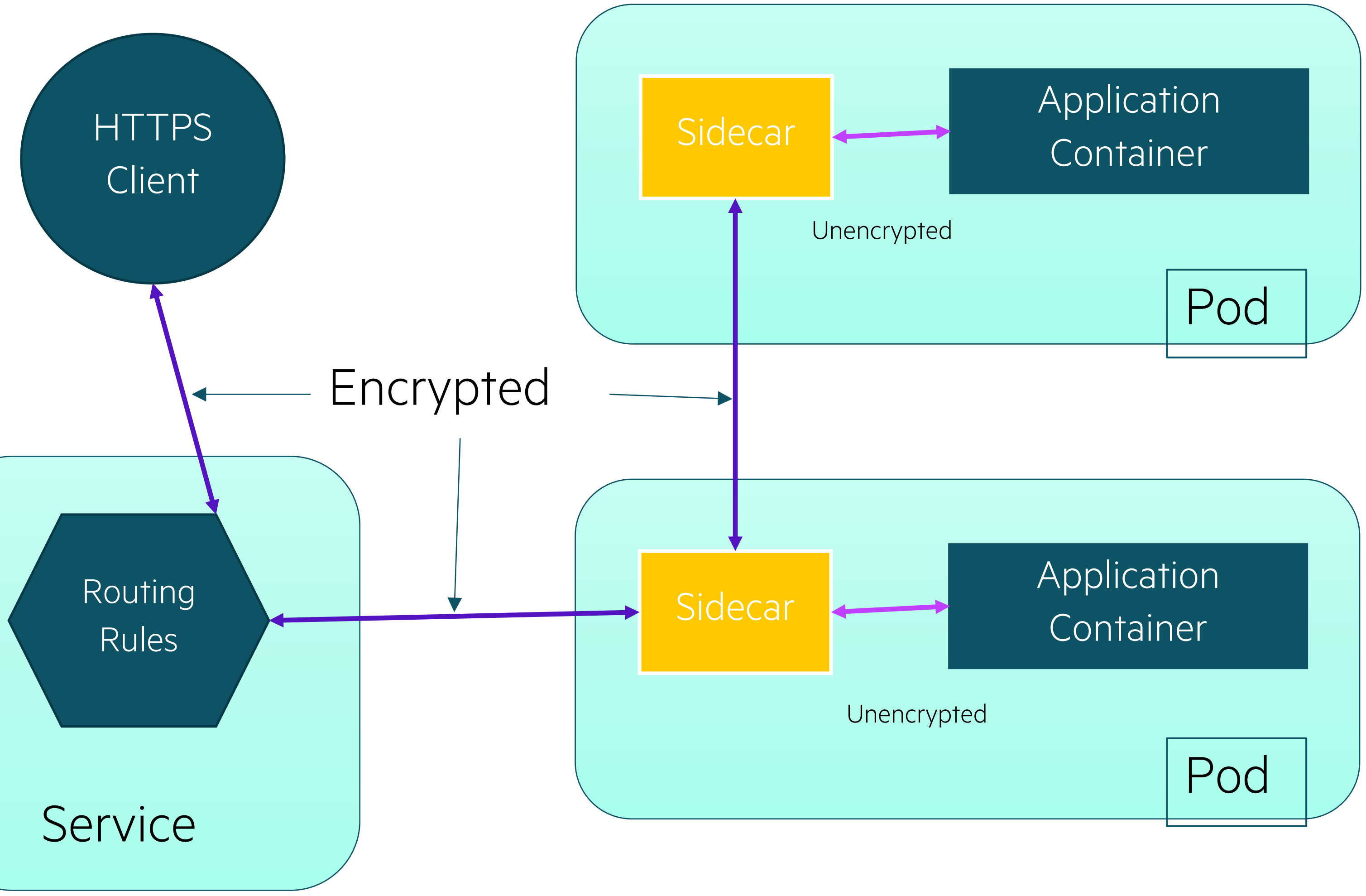
WHAT'S A SERVICE MESH?

- Sidecars and Orchestration
- Upgrade older protocols
- Add security
- Improve visibility



USING A SERVICE MESH FOR TLS

- mTLS negotiation between sidecars
  - Client authenticates the server
  - Server authenticates the client
- TLS negotiation between sidecars and external clients
- Applications are unaware of TLS

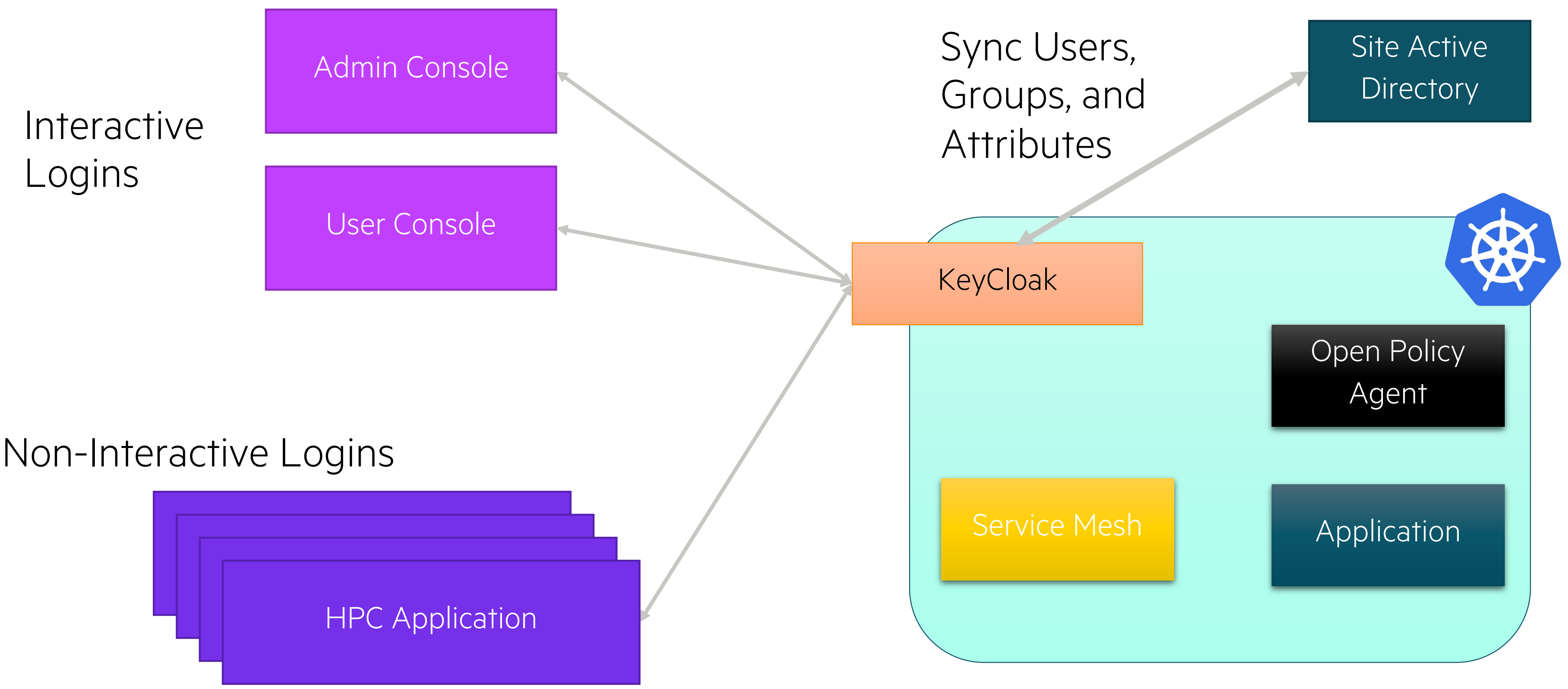


AUTHENTICATION AND AUTHORIZATION

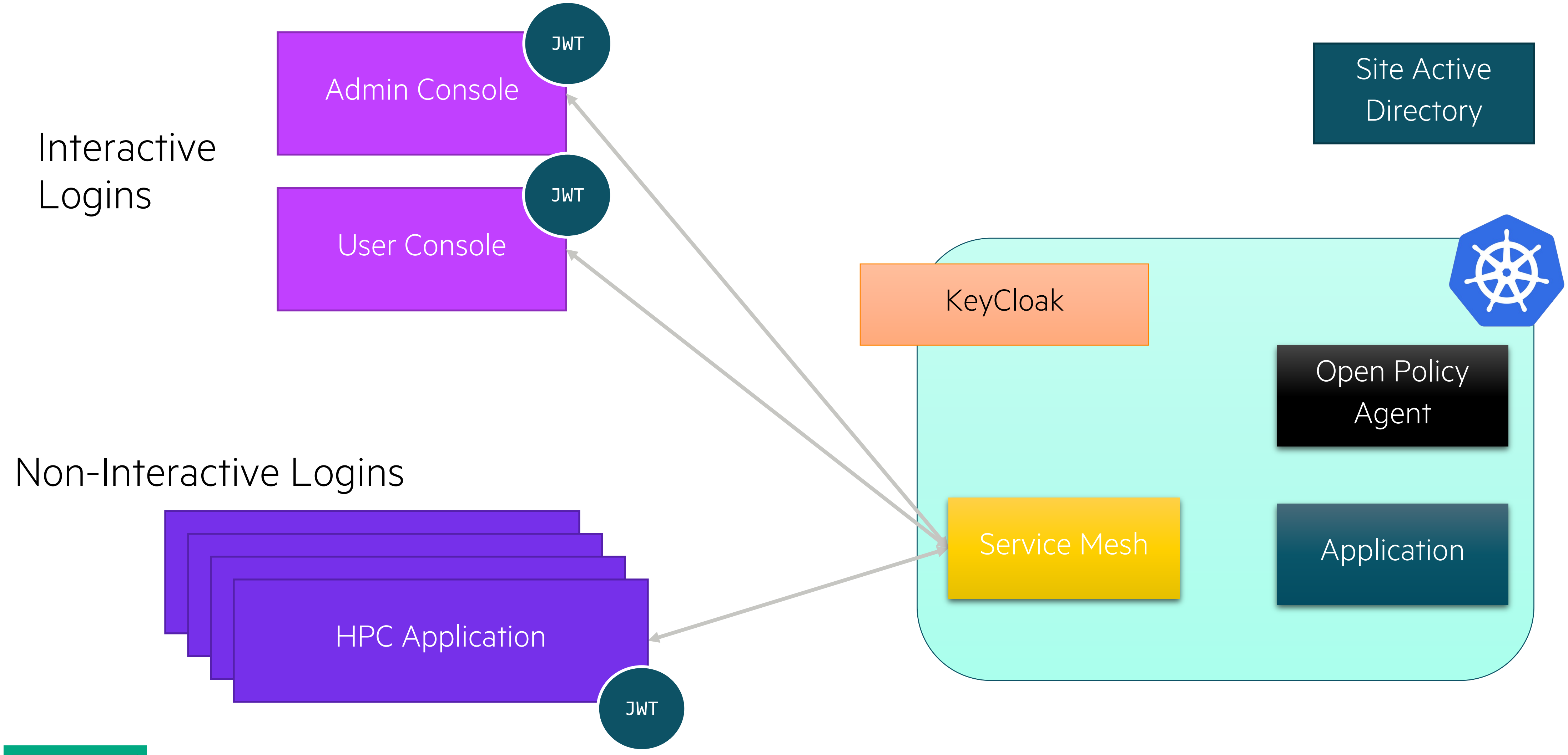
STRONG AND UBIQUITOUS AUTHENTICATION

- Strong Authentication is a prerequisite for Authorization
- Narrow Authorization scope goes beyond POSIX permissions and ACLS
- Role Based Access Control (RBAC)
  - What access do users like me have?
- Aspect Based Access Control (ABAC)
  - Expressive policies that include more than users and groups
  - Permission to access an API can be limited based on other recent access, connection origin, and many other things
- Every Authentication token will expire
- Every Authorization decision will expire
- Recheck only as often as necessary
- Applications and Hardware Devices need to authenticate too

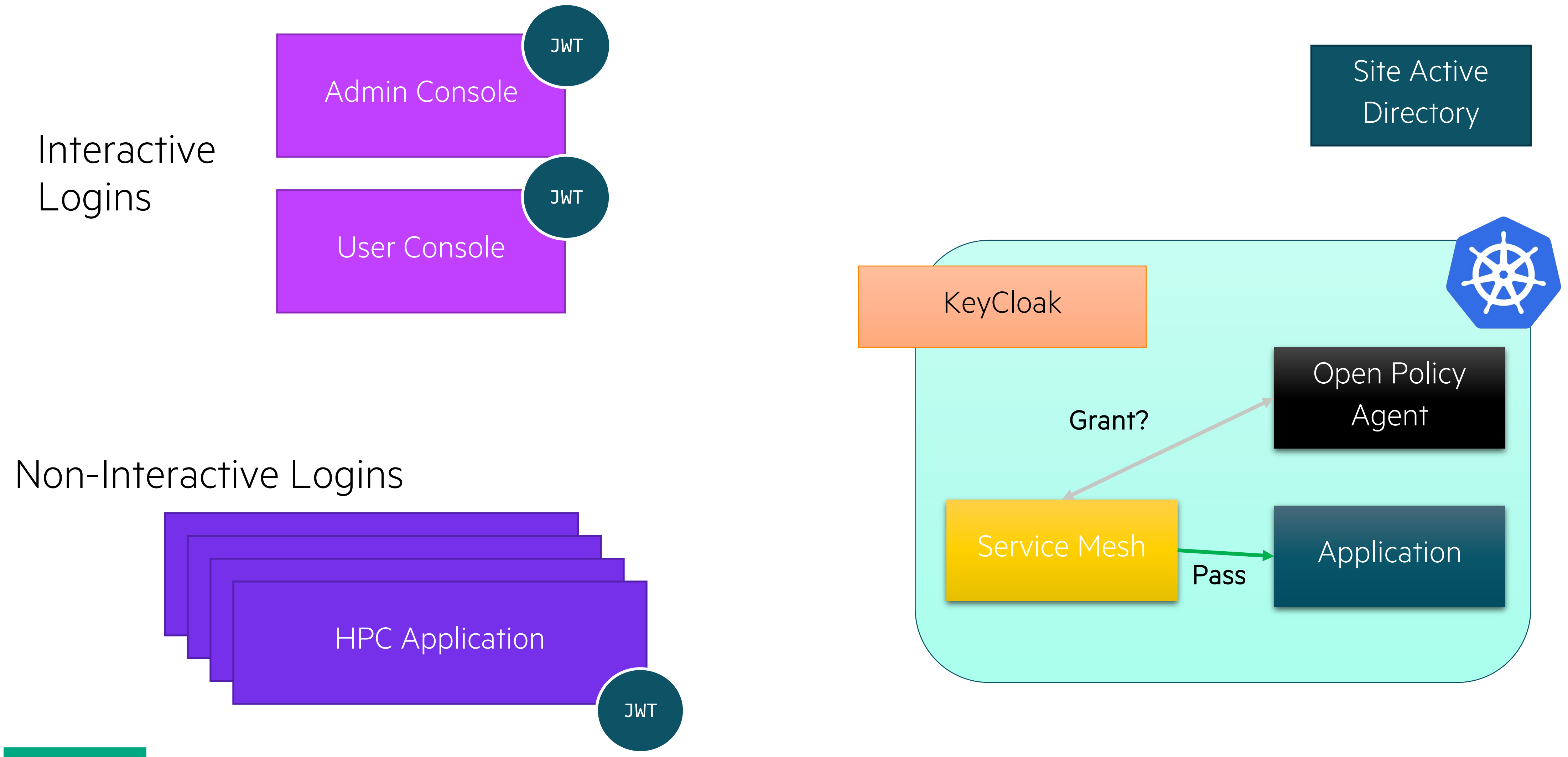
SHASTA IAM WITH KUBERNETES: LOGGING IN



SHASTA IAM WITH KUBERNETES: ACCESS SERVICE MESH



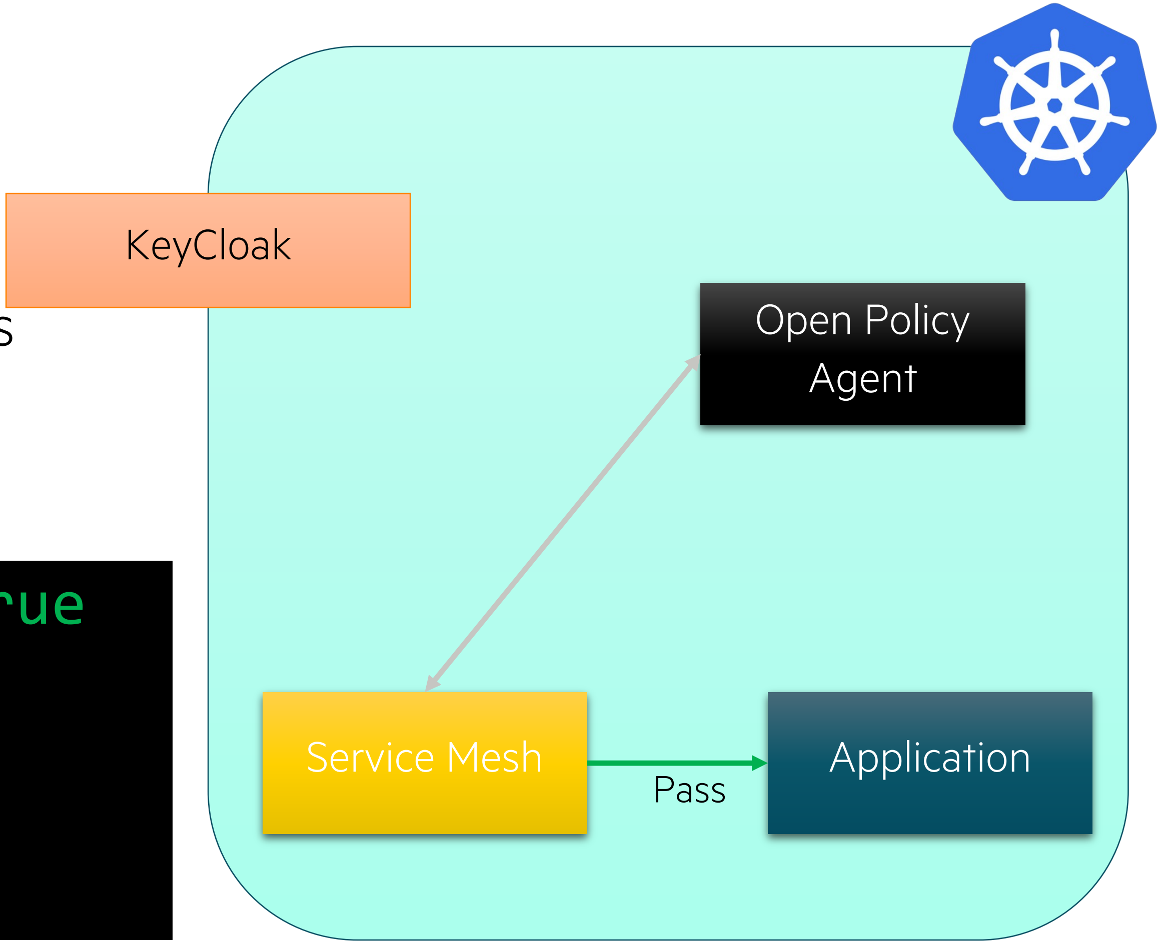
SHASTA IAM WITH KUBERNETES: ACCESS CONTROL



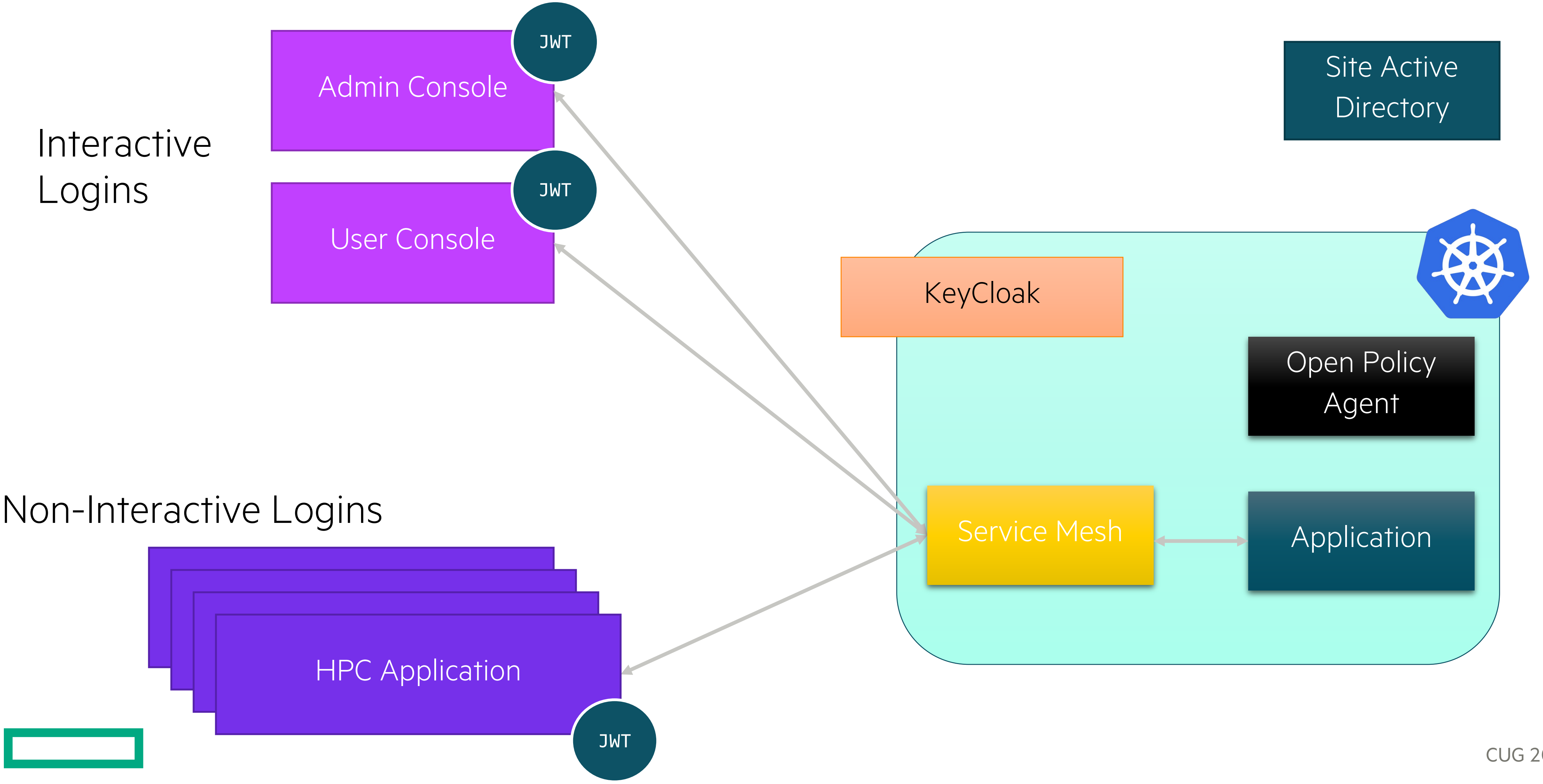
SCALABLE ACCESS POLICY WITH OPA

- Open Policy Agent (OPA)
- Stores and processes policies
- Cacheable answers
- Decisions based on arbitrary attributes (ABAC)

```
Origin_Network_is_Admin: True
User_on_duty: True
User_Active_in_WLM: True
User_MFA_Recent: True
```



SHASTA IAM WITH KUBERNETES: ACCESS SERVICE



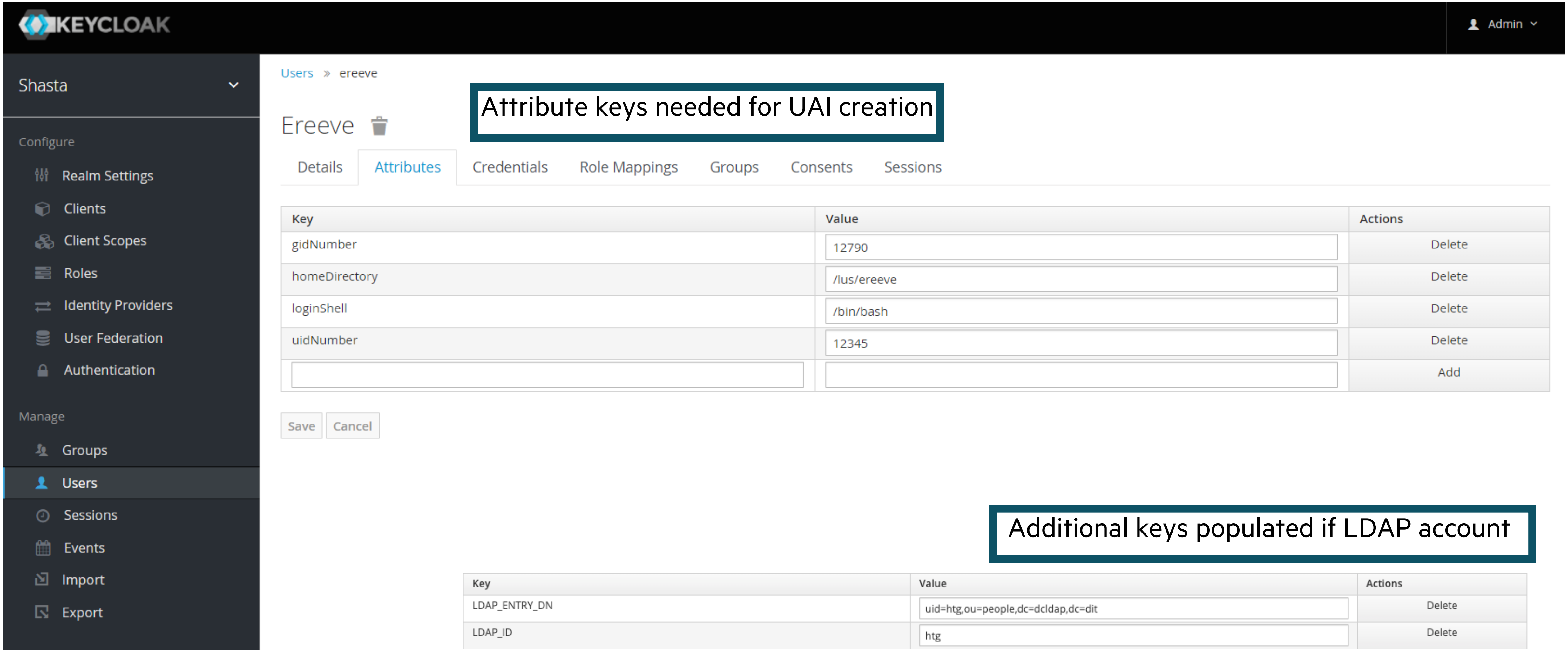
ADDING AUTHORIZED USERS TO KEYCLOAK

- A user must exist in Keycloak to use the `cray` CLI or `sat` CLI

```
user@ncn> cray auth login --username MYNAME
Password:
Usage: cray auth login [OPTIONS]
Try "cray auth login --help" for help.
Error: Invalid Credentials
```

- System administrator must set up the users in Keycloak
  - Local accounts in Keycloak
    - Use the Keycloak User Management UI in a browser
      - https://auth.SYSTEM_DOMAIN_NAME/keycloak/
    - Users can also be set up from the command line
  - Federation to LDAP or other identity provider

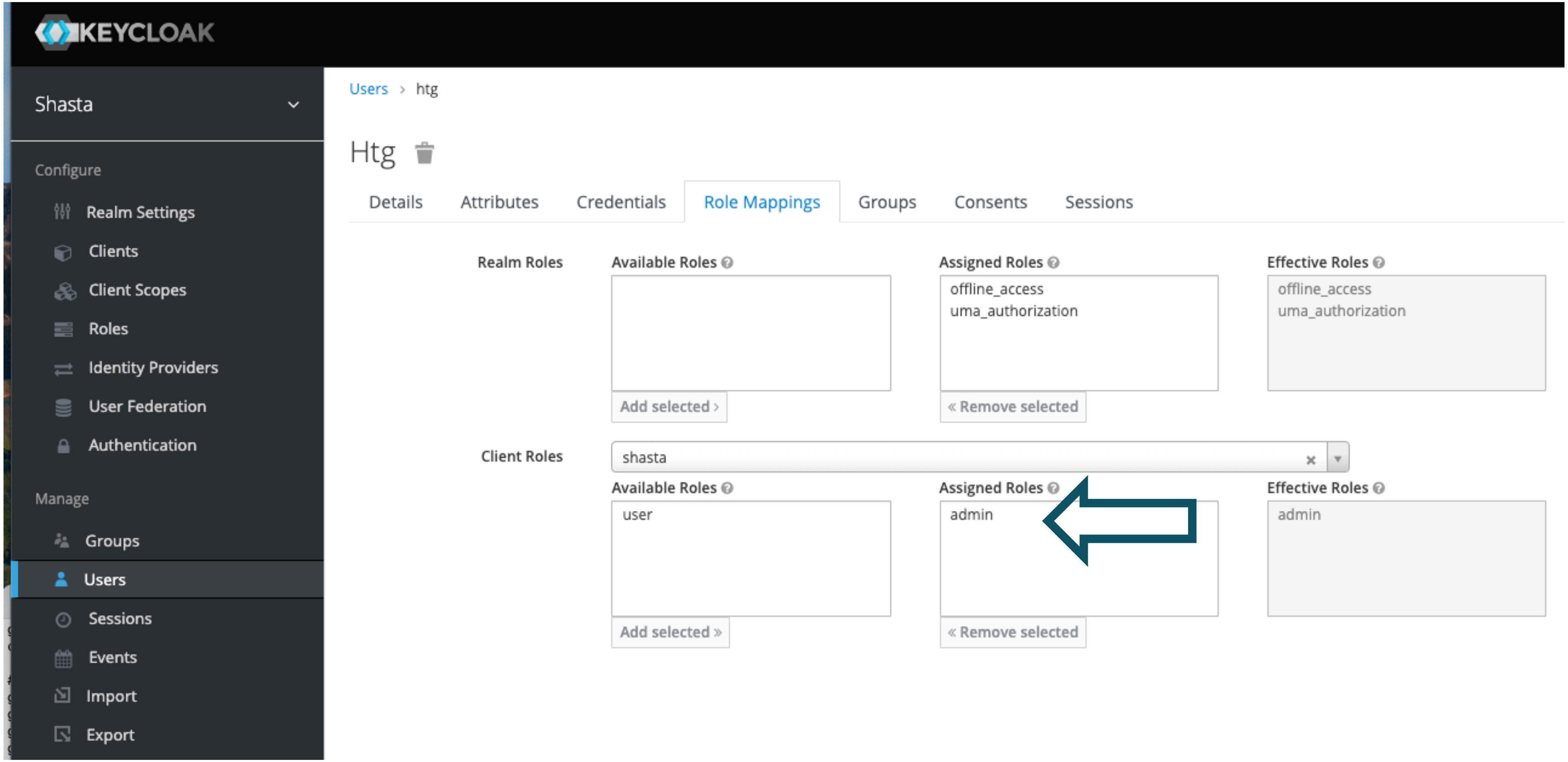
EXAMPLE KEYCLOAK USER ACCOUNT: ATTRIBUTES



Attribute keys needed for UAI creation

Additional keys populated if LDAP account

EXAMPLE KEYCLOAK USER ACCOUNT: ROLE MAPPINGS



HPE CRAY EX SYSTEM OVERVIEW

MANAGEMENT SERVICES

WHAT IS HAPPENING ON MY SYSTEM?

MANAGING USER ENVIRONMENTS

RESOURCES

MANAGEMENT SERVICES

- Common Commands and RESTful APIs
- Hardware Management
- Network Management
- Image Management
- Configuration Management
- Ansible Primer

COMMON COMMANDS AND RESTFUL APIS

COMMON COMMANDS

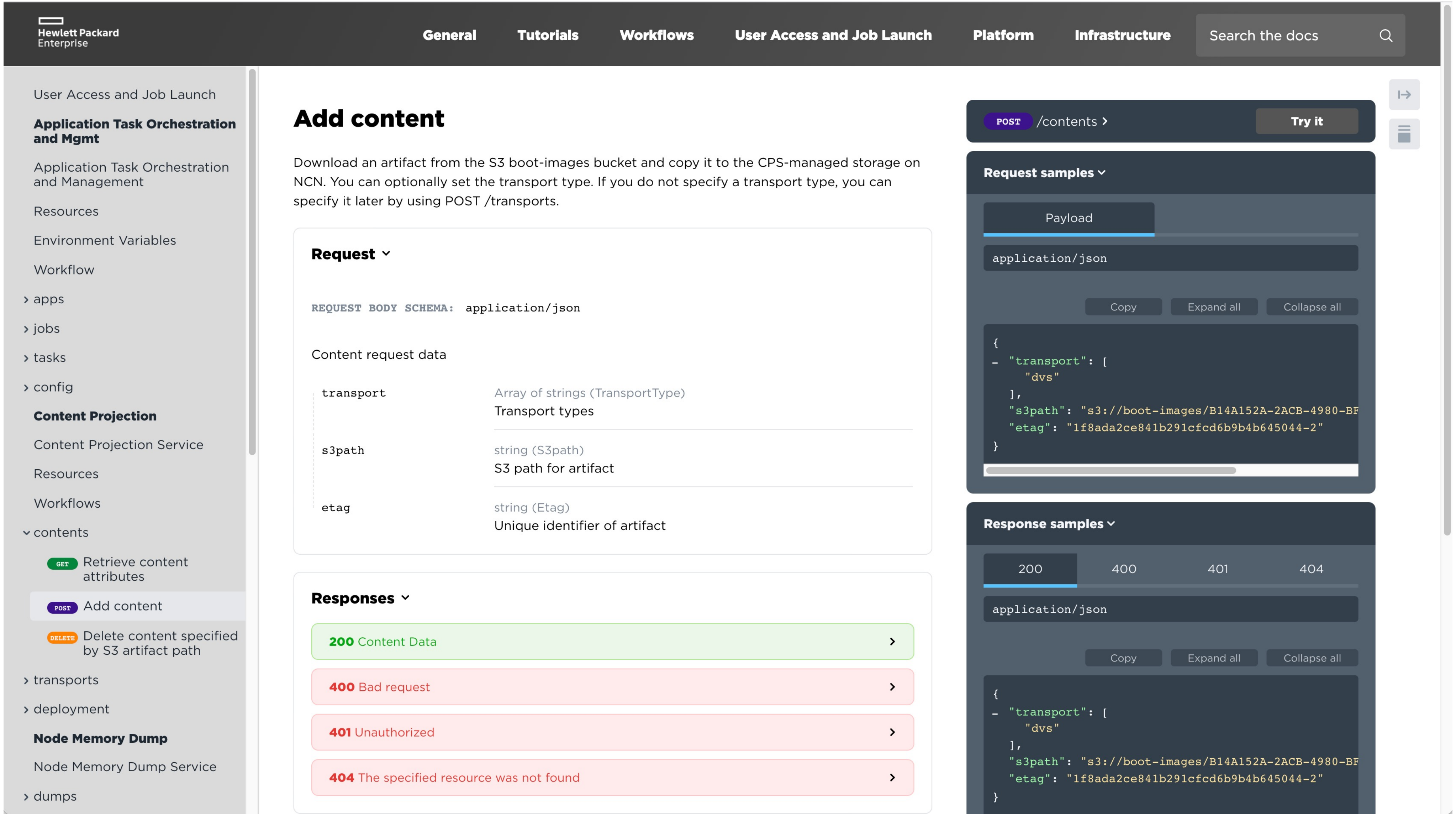
Command	Description
kubectl	CLI for Kubernetes cluster's control plane, using the Kubernetes API • jsonpath - kubectl uses JSONPath expressions to filter on specific fields in the JSON object and format the output
ceph	Control utility for manual deployment and maintenance of a Ceph cluster
cephadm	cephadm - deploys and manages a Ceph cluster
cray	CLI framework integrates system management REST APIs into easily usable commands • Outputs data in JSON, YAML, TOML
sat	CLI interacts with the REST APIs of many services to perform more complex system management tasks • Outputs data in JSON, YAML, TOML
fmctl	CLI for Slingshot fabric management
stt	CLI for Slingshot Topology Tool
jq	command works on JSON data to slice and filter and map and transform structured data like sed, awk, grep and friends let you play with text
Linux tools	systemctl, journalctl, pdsh/dshbak, curl

REST API

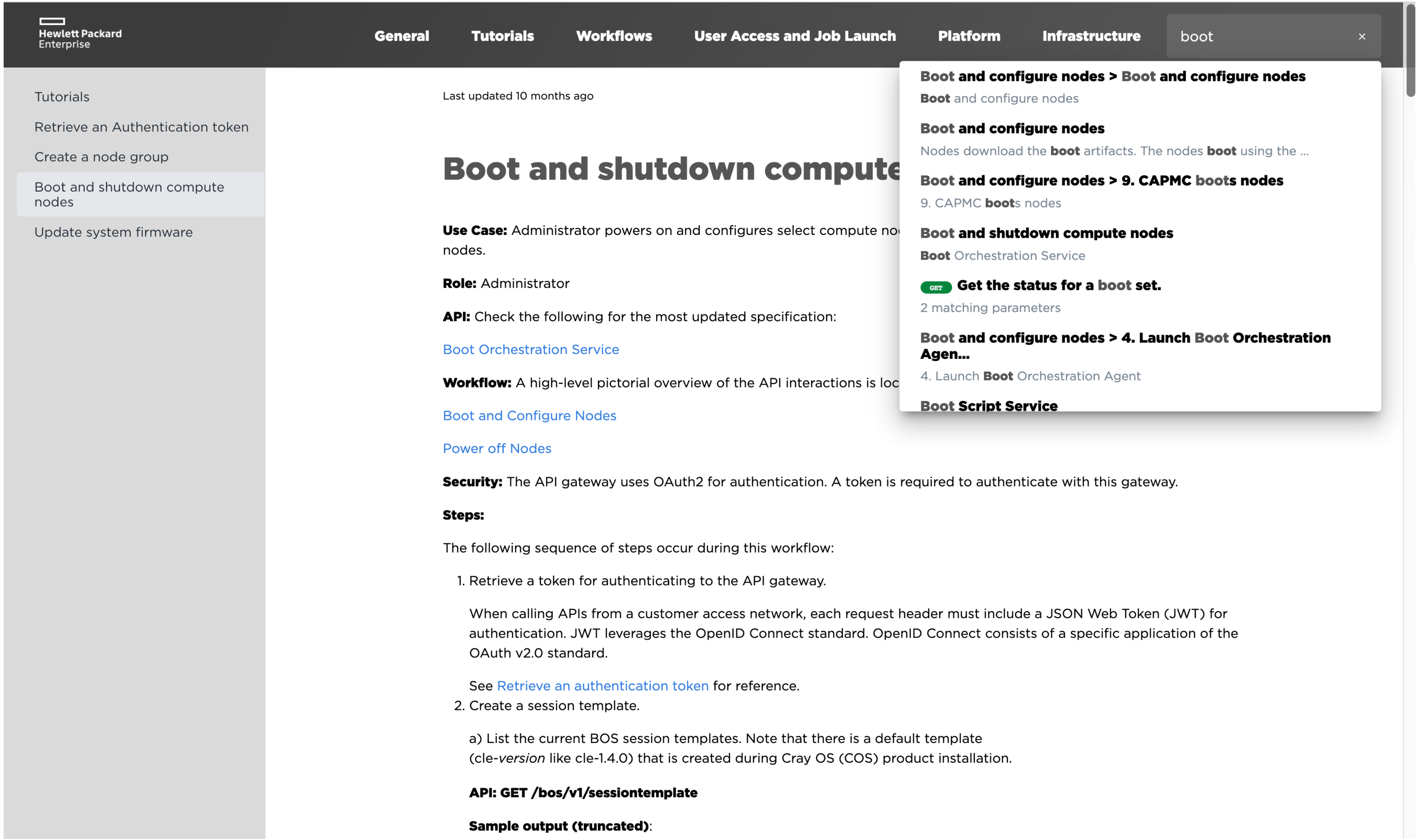
- A RESTful API is an application program interface (API) that uses HTTP requests
  - GET, DELETE, PUT, PATCH, POST
- REST API specification (Swagger/OpenAPI 3.0) for microservices used to generate
  - API documentation
    - Provided in docker image and in tarball for webserver
  - API server stubs for the microservice
  - API client code for the cray CLI framework

- The entire system could be managed by calling the APIs
  - This presentation has a few places using direct access to the API for services or Redfish
  - Most of this presentation shows use of the Kubernetes CLI or cray CLI or sat CLI
    - Every command shown using the cray CLI has a direct mapping to the REST API of the related service
  - Anything done with a CLI could be done programmatically with API calls instead

API DOCUMENTATION FROM REST API SPECIFICATION



API DOCUMENTATION SEARCH



Hewlett Packard Enterprise

General | Tutorials | Workflows | User Access and Job Launch | Platform | Infrastructure | boot ×

- Tutorials
- Retrieve an Authentication token
- Create a node group
- Boot and shutdown compute nodes
- Update system firmware

Boot and configure nodes > Boot and configure nodes
Boot and configure nodes

Boot and configure nodes
Nodes download the boot artifacts. The nodes boot using the ...

Boot and configure nodes > 9. CAPMC boots nodes
9. CAPMC boots nodes

Boot and shutdown compute nodes
Boot Orchestration Service

GET **Get the status for a boot set.**
2 matching parameters

Boot and configure nodes > 4. Launch Boot Orchestration Agen...
4. Launch Boot Orchestration Agent

Boot Script Service

Last updated 10 months ago

Boot and shutdown compute

Use Case: Administrator powers on and configures select compute no
nodes.

Role: Administrator

API: Check the following for the most updated specification:

Boot Orchestration Service

Workflow: A high-level pictorial overview of the API interactions is loc

Boot and Configure Nodes

Power off Nodes

Security: The API gateway uses OAuth2 for authentication. A token is required to authenticate with this gateway.

Steps:

The following sequence of steps occur during this workflow:

1. Retrieve a token for authenticating to the API gateway.

   When calling APIs from a customer access network, each request header must include a JSON Web Token (JWT) for authentication. JWT leverages the OpenID Connect standard. OpenID Connect consists of a specific application of the OAuth v2.0 standard.

   See Retrieve an authentication token for reference.
2. Create a session template.

   a) List the current BOS session templates. Note that there is a default template (cle-*version* like cle-1.4.0) that is created during Cray OS (COS) product installation.

   API: GET /bos/v1/sessiontemplate

   Sample output (truncated):

CRAY CLI FRAMEWORK FROM REST API SPECIFICATION

user@ncn> **cray auth login --username UserWithAdminRole**
Password:
user@ncn> **cray --help**
Usage: cray [OPTIONS] COMMAND [ARGS]...
  Cray management and workflow tool
Options:
  --version Show the version and exit.
  --help   Show this message and exit.
Commands:
  aprun   Run an application using the Parallel Application Launch...
  init   Initialize/reinitialize the Cray CLI
  mpiexec Run an application using the Parallel Application Launch...

- Documentation convention is that if the admin role is required for cray CLI or sat CLI, then the command prompt will use `hostname#` rather than `user@hostname>`
- Linux account and Keycloak authentication are different credentials

Management services which have API specifications

Groups:
  artifacts Manage artifacts in S3
  auth   Manage OAuth2 credentials for the Cray CLI
  badger   Badger Service API
  bos   Boot Orchestration Service
  bss   Boot Script Service API
  capmc   Cray Advanced Platform Monitoring and Control API
  cfs   Configuration Framework Service
  config   View and edit Cray configuration properties
  cps   Content Projection Service
  crus   Compute Rolling Upgrade Service
  fas   Firmware Action Service
  hsm   Hardware State Manager API
  ims   Image Management Service
  nmd   Node Memory Dump Service
  pals   Parallel Application Launch Service
  scsd   System Configuration Service
  sls   System Layout Service
  uas   User Access Service

WHAT IS JQ AND WHY WOULD AN ADMINISTRATOR USE IT?

- Problem:
  - Some commands generate a lot of output

```
ncn# cray hsm inventory hardware list --format json | wc -l
14930
```

- Solutions:
  - Shrink font size impossibly small, buy a magnifying glass
    – Do lots of scrolling

  - Liberally use standard Linux tools: `grep`, `awk`, `head`, and `tail`
    – Recreate monster commands every time output format changes.
  - Learn to use `jq` and parse the JSON output

```
ncn# cray hsm inventory hardware list --format json
[
  {
    "Ordinal": 0,
    "Status": "Populated",
    "HWInventoryByLocationType": "HWInvByLocNode",
    "NodeLocationInfo": {
      "Description": "System Self",
      "HostName": "",
      "MemorySummary": {
        "TotalSystemMemoryGiB": 61
      },
      "ProcessorSummary": {
        "Count": 1,
        "Model": "AMD EPYC 7402 24-Core Processor        "
      },
      "Id": "Self",
      "Name": "System"
    },
    "PopulatedFRU": {
      "Subtype": "",
      "FRUID": "Node.GJG7N8812A0064",
      "Type": "Node",
      "HWInventoryByFRUType": "HWInvByFRUNode",
      "NodeFRUInfo": {
        "BiosVersion": "C12",
        "SKU": "0123456789012345678 9AB",
        "UUID": "cd210000-3b17-11ea-8000-b42e99a23071",
        "AssetTag": "Free form asset tag",
        "SystemType": "Physical",
        "SerialNumber": "GJG7N8812A0064",
        "Model": "R272-Z30-00",
        "PartNumber": "000000000001",
        "Manufacturer": "Cray Inc."
      }
    },
    "Type": "Node",
    "ID": "x3000c0s6b0n0"
  },
...
```

USING JQ TO FILTER JSON OUTPUT

```
ncn# cray hsm inventory hardware list --format json \
| jq 'map(select(.ID == "x3000c0s23b2n0")) | .[].PopulatedFRU'
{
  "Subtype": "",
  "FRUID": "Node.GJG8U6712A004902",
  "Type": "Node",
  "HWInventoryByFRUType": "HWInvByFRUNode",
  "NodeFRUInfo": {
    "BiosVersion": "C10",
    "SKU": "0123456789012345678 9AB",
    "UUID": "cd210000-3b17-11ea-8000-b42e997f0d24",
    "AssetTag": "Free form asset tag",
    "SystemType": "Physical",
    "SerialNumber": "GJG8U6712A004902",
    "Model": "H262-Z63-00",
    "PartNumber": "000000000001",
    "Manufacturer": "Cray Inc."
  }
}
ncn# cray hsm inventory hardware list --format json \
| jq 'map(select(.ID == "x3000c0s23b2n0")) | .[].PopulatedFRU.NodeFRUInfo.BiosVersion'
"C10"
```

The **`map(select(.KEY == VALUE))`** function selects only qualifying objects from the array of results

In this case only the objects with the "**`ID`**" of "**`x3000c0s23b2n0`**" are included in the output

The pipe "**|**" operator, inside the single quotes of the `jq` query, filters output to only include objects with named keys. For sub-objects, dots can be used to show the path of the desired object

In these examples, the result of the **`cray`** command is an array, in "**`[]`**" brackets, of JSON objects. Each object has a "**`PopulatedFRU`**" object at the top level and within that some have a "**`NodeFRUInfo`**" object and within that a "**`BiosVersion`**" object

SAMPLE JQ COMMAND WITH FUNCTION CALLS

```
ncn# cray hsm state components list --format json \
| jq -j '.Components | map(select(.Type == "Node")) | map(select(.Role == "Compute")) \
| sort_by(.NID) | map({"NID": .NID, "State": .State, "Xname": .ID })'
[
  {
    "NID": 1,
    "State": "Ready",
    "Xname": "x3000c0s20b1n0"
  },
  {
    "NID": 2,
    "State": "Ready",
    "Xname": "x3000c0s20b2n0"
  },
...
  {
    "NID": 16,
    "State": "Ready",
    "Xname": "x3000c0s27b4n0"
  }
]
```

This more interesting example filters the result set by **`Type`** and then **`Role`** and then uses the **`sort_by`** function to sort the results by `NID` number. Finally, the results are filtered to only include **`NID`**, **`State`**, and **`Xname`**

SYSTEM ADMIN TOOLKIT (SAT)

- Assists administrators with common tasks
  - Troubleshooting and querying information about the HPE Cray EX System and its components
  - System boot and shutdown
  - Replacing hardware components
- SAT offers a command line utility which uses subcommands
  - Most commands require authentication to API gateway
  - Some commands require Kubernetes configuration and authentication
- Several Kibana dashboards provide organized output for system health information
- Some Grafana dashboards display messages that are generated by the HSN (High Speed Network) and reported through Redfish

SAT CLI

- Runs on master nodes in a container using podman, a daemonless container runtime
  - Using either sat or sat bash always launches a container
  - The SAT container does not have access to the NCN file system
- There are two ways to run sat
  - Interactive: Launching a container using sat bash, followed by sat commands

```
ncn-m# sat bash
(CONTAINER-ID)sat-container# source /sat/venv/bin/activate
(CONTAINER-ID)sat-container# sat status
(CONTAINER-ID)sat-container# sat hwinv
(CONTAINER-ID)sat-container# exit
```

  - Non-interactive: Running a sat command directly on a master node

```
ncn-m# sat status
```

- Authentication using Keycloak credentials
  - `sat auth` and use Keycloak username and password per session
  - Account used needs to have admin role in Keycloak
- Man pages exist for `sat` and subcommands
  - Use to get more information on how to use options for subcommands

SAT COMMANDS

sat auth	Authenticate to the API gateway and save the token	sat k8s	Report on Kubernetes replicasets that have co-located replicas
sat bmccreds	Set BMC Redfish access credentials	sat nid2xname	Translate node IDs to node xnames
sat bootprep	Prepare to boot nodes with images and configurations	sat sensors	Report current sensor data
sat bootsys	Boot or shutdown the system (compute nodes, application nodes, and management nodes)	sat setrev	Set HPE Cray EX system revision information
sat diag	Launch diagnostics on the HSN switches and generate a report	sat showrev	Print revision information for the HPE Cray EX system
sat firmware	Report firmware version	sat slscheck	Perform a cross-check between SLS and HSM
sat hwhist	Report hardware component history	sat status	Report node status across the HPE Cray EX system
sat hwinv	Give a listing of the hardware of the HPE Cray EX system	sat swap	Prepare HSN switch or cable for replacement and bring HSN switch or cable into service
sat hwmatch	Report hardware mismatches for processors and memory	sat xname2nid	Translate node and node BMC xnames to node IDs
sat init	Create a default SAT configuration file		

Newest SAT commands

SAT STATUS

- Shows current status of NCNs and CNs as reported by Hardware State Manager (HSM)
  - Information must be discovered by HSM
- Requires authentication to show any information

```
ncn-m# sat status --sort-by NID
+----------------+-----------+------+----------+-------+------+---------+------+----------+------------+----------+-----------+
| xname          | Aliases   | Type | NID      | State | Flag | Enabled | Arch | Class    | Role       | Subrole  | Net Type  |
+----------------+-----------+------+----------+-------+------+---------+------+----------+------------+----------+-----------+
| x3000c0s20b1n0 | nid000001 | Node | 1        | On    | OK   | True    | X86  | River    | Compute    | None     | Sling     |
| x3000c0s20b2n0 | nid000002 | Node | 2        | Ready | OK   | True    | X86  | River    | Compute    | None     | Sling     |
| x3000c0s20b3n0 | nid000003 | Node | 3        | On    | OK   | True    | X86  | River    | Compute    | None     | Sling     |
| x3000c0s20b4n0 | nid000004 | Node | 4        | Ready | OK   | True    | X86  | River    | Compute    | None     | Sling     |
| x3000c0s23b1n0 | nid000005 | Node | 5        | On    | OK   | True    | X86  | River    | Compute    | None     | Sling     |
| x3000c0s23b2n0 | nid000006 | Node | 6        | Ready | OK   | True    | X86  | River    | Compute    | None     | Sling     |
| x3000c0s23b3n0 | nid000007 | Node | 7        | On    | OK   | True    | X86  | River    | Compute    | None     | Sling     |
| x3000c0s23b4n0 | nid000008 | Node | 8        | On    | OK   | True    | X86  | River    | Compute    | None     | Sling     |
| x1000c0s1b0n0  | nid001004 | Node | 1004     | Ready | OK   | True    | X86  | Mountain | Compute    | None     | Sling     |
| x1000c0s1b0n1  | nid001005 | Node | 1005     | Ready | OK   | True    | X86  | Mountain | Compute    | None     | Sling     |
| x1000c0s1b1n0  | nid001006 | Node | 1006     | Ready | OK   | True    | X86  | Mountain | Compute    | None     | Sling     |
| x1000c0s1b1n1  | nid001007 | Node | 1007     | Ready | OK   | True    | X86  | Mountain | Compute    | None     | Sling     |
| x3000c0s1b0n0  | ncn-m001  | Node | 100001   | Ready | OK   | True    | X86  | River | Management    | Master   | Sling     |
| x3000c0s3b0n0  | ncn-m002  | Node | 100002   | Ready | OK   | True    | X86  | River | Management    | Master   | Sling     |
| x3000c0s5b0n0  | ncn-m003  | Node | 100003   | Ready | OK   | True    | X86  | River | Management    | Master   | Sling     |
| x3000c0s7b0n0  | ncn-w001  | Node | 100004   | Ready | OK   | True    | X86  | River | Management    | Worker   | Sling     |
| x3000c0s9b0n0  | ncn-w002  | Node | 100005   | Ready | OK   | True    | X86  | River | Management    | Worker   | Sling     |
| x3000c0s11b0n0 | ncn-w003  | Node | 100006   | Off   | OK   | True    | X86  | River | Management    | Worker   | Sling     |
| x3000c0s13b0n0 | ncn-s001  | Node | 100007   | Ready | OK   | True    | X86  | River | Management    | Storage  | Sling     |
| x3000c0s15b0n0 | ncn-s002  | Node | 100008   | Ready | OK   | True    | X86  | River | Management    | Storage  | Sling     |
| x3000c0s17b0n0 | ncn-s003  | Node | 100009   | Ready | OK   | True    | X86  | River | Management    | Storage  | Sling     |
| x3000c0s27b0n0 | uan01     | Node | 49169248 | Off   | OK   | True    | X86  | River | Application   | UAN      | Sling     |
+----------------+-----------+------+----------+-------+------+---------+------+-------+---------------+----------+-----------+
```

SAT STATUS FILTERED

- Can filter by any of the columns with both “equal to” and “not equal to”
- Can remove some of the pretty printing

```
ncn-m# sat status --no-borders --filter nid=1000
 xname          Aliases   Type  NID      State   Flag  Enabled Arch Class    Role        Subrole Net Type
 x1000c0s0b0n0 nid000004 Node  1000     Ready   OK    True    X86  Mountain Compute      None    Sling
ncn-m# sat status --no-borders --no-headings --filter role=compute --filter state!=ready \
--filter enabled=true
 x1000c1s2b0n1 nid001041 Node  1041     Standby Alert True    X86  Mountain Compute      None    Sling
 x1000c2s1b0n0 nid001068 Node  1068     Off     OK    True    X86  Mountain Compute      None    Sling
 x1000c7s5b1n0 nid001246 Node  1246     On      OK    True    X86  Mountain Compute      None    Sling
ncn-m# sat status --no-borders --no-headings --filter class=river --filter role=application
 x3000c0s23b0n0 uan01     Node  49169120 Ready   OK    True    X86  River    Application UAN     Sling
```

- Can change fields displayed

```
ncn-m# sat status --no-borders --filter class=river --filter role=management \
--fields xname,aliases,nid,subrole,state
 xname           Aliases  NID    Subrole State
 x3000c0s3b0n0  ncn-m002 100002 Master  Ready
 x3000c0s7b0n0  ncn-w001 100004 Worker  Ready
 x3000c0s17b0n0 ncn-s003 100008 Storage Ready
```

- Can report status on different types of components, but default is “Node”
  - all, Chassis, ChassisBMC, ComputeModule, HSNBoard, Node, NodeBMC, NodeEnclosure, RouterBMC, RouterModule

```
ncn-m# sat status --no-borders --types RouterBMC
 xname          Type       State Flag Enabled Arch Class Net Type
 x3000c0r21b0  RouterBMC  Ready OK   True    X86  River Sling
```

CHECKING SOFTWARE VERSIONS WITH KUBECTL

- Search for information in the product-catalog with jq filtering the output for only CSM

```
ncn# kubectl get cm cray-product-catalog -n services -o json | jq -r .data.csm
1.0.11:
  configuration:
    clone_url: https://vcs.groot.dev.cray.com/vcs/cray/csm-config-management.git
    commit: 8c09c934b3b7e3a4b085c50575442226a133eba7
    import_branch: cray/csm/1.6.28
    import_date: 2022-02-25 20:39:53.562810
    ssh_url: git@vcs.groot.dev.cray.com:cray/csm-config-management.git
  images:
    cray-shasta-csm-sles15sp2-barebones.x86_64-shasta-1.5:
      id: bc6351e1-429c-4859-a559-08b575fb8517
  recipes:
    cray-shasta-csm-sles15sp2-barebones.x86_64-shasta-1.5:
      id: 9677a6fb-e561-465c-b4ed-cafcdb919fc0
```

CHECKING SOFTWARE VERSIONS WITH SAT

- Display information for all software products installed

```
ncn-m# sat showrev --products
################################################################################
Product Revision Information
################################################################################
+-------------------------+-----------+--------+--------------------------------------------------------+--------------------------------------------------------+
| product_name            | product   | active | images                                                 | image_recipes                                          |
|                         |_version   |        |                                                        |                                                        +
+-------------------------+---------------+-------------------------------------------------------------+--------------------------------------------------------+
| analytics               | 1.1.24    | N/A    | Cray-Analytics.x86_64-base                             | -                                                      |
| cos                     | 2.2.101   | N/A    | cray-shasta-compute-sles15sp3.x86_64-2.2.38            | cray-shasta-compute-sles15sp3.x86_64-2.2.38            |
| cpe                     | 21.12.3   | N/A    | cpe-barebones-sles15sp3.x86_64-21.12.2                 | cpe-barebones-sles15sp3.x86_64-21.12.2                 |
| cpe                     | 22.3.1    | N/A    | cpe-barebones-sles15sp3.x86_64-22.03.0                 | cpe-barebones-sles15sp3.x86_64-22.03.0                 |
| cray-sdu-rda            | 1.2.9     | N/A    | -                                                      | -                                                      |
| csm                     | 1.0.11    | N/A    | cray-shasta-csm-sles15sp2-barebones.x86_64-shasta-1.5 | cray-shasta-csm-sles15sp2-barebones.x86_64-shasta-1.5 |
| hfp                     | 22.03.0   | N/A    | -                                                      | -                                                      |
| sat                     | 2.2.15    | False  | -                                                      | -                                                      |
| sat                     | 2.2.16    | True   | -                                                      | -                                                      |
| sle-os-backports-15-sp2 | 22.02.1   | N/A    | -                                                      | -                                                      |
| sle-os-backports-15-sp3 | 22.02.1   | N/A    | -                                                      | -                                                      |
| sle-os-products-15-sp2  | 22.02.1   | N/A    | -                                                      | -                                                      |
| sle-os-products-15-sp3  | 22.02.1   | N/A    | -                                                      | -                                                      |
| sle-os-ptf-15-sp2       | 22.02.1   | N/A    | -                                                      | -                                                      |
| sle-os-updates-15-sp2   | 22.02.1   | N/A    | -                                                      | -                                                      |
| sle-os-updates-15-sp3   | 22.02.1   | N/A    | -                                                      | -                                                      |
| slingshot               | 1.7.0-59  | N/A    | -                                                      | -                                                      |
| slingshot               | 1.7.1-407 | N/A    | -                                                      | -                                                      |
| slingshot-host-software | 1.7.1-22  | N/A    | -                                                      | -                                                      |
| slurm                   | 1.1.5     | N/A    | -                                                      | -                                                      |
| sma                     | 1.5.27    | N/A    | -                                                      | -                                                      |
| uan                     | 2.3.2     | N/A    | -                                                      | -                                                      |
```

QUERYING HARDWARE INVENTORY

`sat` supports tab completion! From the podman pod, `sat bash`, but not from the `sat` CLI. Hitting tab twice provides a list of options

```
ncn-m# sat bash
(cab2475ed202) sat-container:/sat # source /etc/bash_completion.d/sat-completion.bash
(cab2475ed202) sat-container:/sat # sat hwinv --list-
--list-all               --list-drives             --list-node-accels                     --list-nodes
--list-chassis           --list-hsn-boards         --list-node-enclosure-power-supplies   --list-procs
--list-cmm-rectifiers    --list-mems               --list-node-enclosures                 --list-router-modules
--list-compute-modules --list-node-accel-risers  --list-node-hsn-nics
(cab2475ed202) sat-container:/sat # sat hwinv --list-nodes --node-fields xname,serial_number,memory_size
################################################################################
Listing of all nodes
################################################################################
+-----------------+------------------+-------------------+
| xname           | Serial Number    | Memory Size (GiB) |
+-----------------+------------------+-------------------+
| x1000c0s1b0n0   | HR19380063       | 256.0             |
| x1000c0s1b0n1   | HR19380063       | 256.0             |
| x1000c0s5b0n0   | HR19380023       | 256.0             |
(cab2475ed202) sat-container:/sat # sat hwinv --list-router-modules
################################################################################
Listing of all router modules
################################################################################
+-----------+--------------+
| xname     | Manufacturer |
+-----------+--------------+
| x1000c0r3 | Cray Inc     |
| x1000c0r7 | Cray Inc     |
```

SLINGSHOT SWITCH OR CABLE REPLACEMENT

- Disable a Slingshot switch before maintenance or enable a switch after maintenance is complete.

```
ncn-m# sat swap switch --dry-run x1000c3r3
Ports: x1000c3r3j104p1 x1000c3r3j105p0 x1000c3r3j105p1 x1000c3r3j106p0 x1000c3r3j106p1
x1000c3r3j107p0 x1000c3r3j107p1 x1000c3r3j100p1 x1000c3r3j101p0 x1000c3r3j101p1
x1000c3r3j102p0 x1000c3r3j102p1 x1000c3r3j103p0 x1000c3r3j103p1 x1000c3r3j104p0
x1000c3r3j100p0 x1000c3r3j9p0 x1000c3r3j8p1 x1000c3r3j8p0 x1000c3r3j6p1 x1000c3r3j6p0
x1000c3r3j4p1 x1000c3r3j4p0 x1000c3r3j2p1 x1000c3r3j2p0 x1000c3r3j22p1 x1000c3r3j22p0
x1000c3r3j20p1 x1000c3r3j20p0 x1000c3r3j24p1 x1000c3r3j24p0 x1000c3r3j18p1
x1000c3r3j18p0 x1000c3r3j16p1 x1000c3r3j12p0 x1000c3r3j11p1 x1000c3r3j10p1
x1000c3r3j11p0 x1000c3r3j10p0 x1000c3r3j16p0 x1000c3r3j14p1 x1000c3r3j14p0
x1000c3r3j13p1 x1000c3r3j12p1 x1000c3r3j13p0 x1000c3r3j9p1
Dry run completed with no action to enable/disable switch.
```

- Determine all linked ports from a single jack

```
ncn-m# sat swap cable --dry-run x5000c1r3j16
Ports: x5000c1r3j16p0 x5000c3r7j18p0 x5000c1r3j16p1 x5000c3r7j18p1
Dry run completed with no action to enable/disable cable.
```

SAT BOOTPREP

- Create CFS configurations, build IMS images, customize IMS images with CFS configurations, and create BOS session templates using the customized IMS image and CFS configuration which can then be used to boot compute and application nodes

```
ncn-m# sat bootprep generate-example
ncn-m# cp example-bootprep-input.yaml \
 bootprep-input.yaml
ncn-m# vi bootprep-input.yaml
ncn-m# sat bootprep run bootprep_input.yaml
```

- This bootprep_input.yaml example has been trimmed to only show compute node information
  - configurations
  - images
  - session_templates


```
configurations:
- name: cos-config
  layers:
  - name: cos-integration-2.2.101
    playbook: site.yml
    product:
      name: cos
      version: 2.2.101
      branch: integration
  - name: cpe-integration-22.3.1
    playbook: pe_deploy.yml
    product:
      name: cpe
      version: 22.3.1
      branch: integration
  - name: slurm-master-1.1.5
    playbook: site.yml
    product:
      name: slurm
      version: 1.1.5
      branch: master
  - name: analytics-integration-
1.1.24
    playbook: site.yml
    product:
      name: analytics
      version: 1.1.24
      branch: integration
```

```
images:
- name: cray-shasta-compute-
sles15sp3.x86_64-2.2.38
  ims:
    is_recipe: true
    name: cray-shasta-compute-
sles15sp3.x86_64-2.2.38
  configuration: cos-config
  configuration_group_names:
  - Compute
session_templates:
- name: cray-shasta-compute-
sles15sp3.x86_64-2.2.38
  image: cray-shasta-compute-
sles15sp3.x86_64-2.2.38
  configuration: cos-config
  bos_parameters:
    boot_sets:
      compute:
        kernel_parameters: ip=dhcp
quiet
spire_join_token=${SPIRE_JOIN_TOKEN}
        node_roles_groups:
        - Compute
```

FIRMWARE REPORTING

Node controller (or BMC) for two liquid-cooled nodes

```
ncn-m# sat firmware -x x1000c0s0b0
+-------------+-------------------------+-------------------------+--------------------------------------------------------+
| xname       | name                    | target_name             | version                                                |
+-------------+-------------------------+-------------------------+--------------------------------------------------------+
| x1000c0s0b0 | Node0.ManagementEthernet | Node0.ManagementEthernet | wnc.i210-p2sn01                                       |
| x1000c0s0b0 | Bootloader              | Bootloader              | 1.10-wnc                                               |
| x1000c0s0b0 | FPGA2                   | mFPGA1                  | 1.05                                                   |
| x1000c0s0b0 | BMC                     | BMC                     | nc.1.5-31-shasta-release.arm.2021-11.-                 |
|                                                                                           03T03:49:30+00:00.b9ced71 |
| x1000c0s0b0 | FPGA1                   | mFPGA0                  | 1.05                                                   |
| x1000c0s0b0 | Node1.BIOS              | Node1.BIOS              | ex425.bios-1.6.1                                       |
| x1000c0s0b0 | Node0.BIOS              | Node0.BIOS              | ex425.bios-1.6.1                                       |
| x1000c0s0b0 | FPGA0                   | nFPGA                   | 5.02                                                   |
| x1000c0s0b0 | Recovery                | Recovery                | nc.1.5-31-shasta-release.arm.2021-11-                  |
|                                                                                           03T03:49:30+00:00.b9ced71 |
| x1000c0s0b0 | Node1.ManagementEthernet | Node1.ManagementEthernet | wnc.i210-p2sn01                                       |
+-------------+-------------------------+-------------------------+--------------------------------------------------------+
```

FIRMWARE REPORTING WITH XNAME LIST

List of xnames: cabinet controller and Slingshot switch

```
ncn-m# sat firmware -x x1003c6b0,x3001c0r11b0
+--------------+------------+--------------+-------------------------------------------------------------+
| xname        | name       | target_name  | version                                                     |
+--------------+------------+--------------+-------------------------------------------------------------+
| x1003c6b0    | Recovery   | Recovery     | cc.1.5-31-shasta-release.arm64.2021-11-03T03:50:18+00:00.b9ced71    |
| x1003c6b0    | Rectifier1 | Rectifier 1  | PFC_01.03-SEC_02.10                                         |
| x1003c6b0    | Bootloader | Bootloader   | 1.7-cc-pass4                                                |
| x1003c6b0    | Rectifier0 | Rectifier 0  | PFC_01.03-SEC_02.10                                         |
| x1003c6b0    | BMC        | BMC          | cc.1.5-31-shasta-release.arm64.2021-11-03T03:50:18+00:00.b9ced71    |
| x1003c6b0    | FPGA0      | cFPGA        | 3.03                                                        |
| x1003c6b0    | Rectifier2 | Rectifier 2  | PFC_01.03-SEC_02.10                                         |
| x3001c0r11b0 | BMC        | BMC          | sc.1.7.0-45-slingshot-release.arm64.2022-03-05T22:28:42+00:00.9a31838 |
| x3001c0r11b0 | Recovery   | Recovery     | rec.1.4.22-shasta-release.arm64.2021-04-26T23:22:15+00:00.79c40dd   |
| x3001c0r11b0 | FPGA0      | sFPGA-ROS    | 1.08                                                        |
| x3001c0r11b0 | Packages   | Packages     | na                                                          |
| x3001c0r11b0 | Bootloader | Bootloader   | 1.9-sc-ros-tor                                              |
| x3001c0r11b0 | FPGA1      | sFPGA-ROS-TOR | 1.04                                                       |
+--------------+------------+--------------+-------------------------------------------------------------+
```

CHECK SENSORS

- Obtain sensor readings from BMCs (ChassisBMC, NodeBMC, RouterBMC)
  - Limit the telemetry topics queried to the topics listed
  - The default is to query all topics:
    – cray-telemetry-temperature, cray-telemetry-voltage, cray-telemetry-power, 'cray-telemetry-energy, cray-telemetry-fan, cray-telemetry-pressure

```
ncn-m# sat sensors -x x1003c2s6b1 -t NodeBMC -b 2 --timeout 10 --topic cray-telemetry-temperature
Telemetry data being collected for x1003c2s6b1
Please be patient...
Waiting for metrics for all requested xnames from cray-telemetry-temperature.
Receiving metrics from stream: cray-telemetry-temperature...
Telemetry data received from cray-telemetry-temperature for all requested xnames.
+-------------+---------+--------------------------+-------------------------------+-------------+---------+-----------+---------+------+
| xname       | Type    | Topic                    | Timestamp                     | Location | Parental Context | Physical Context | Index |Value |
+-------------+---------+--------------------------+-------------------------------+-------------+---------+-----------+---------+------+
| x1003c2s6b1 | NodeBMC | cray-telemetry-temperature | 2022-04-01T18:17:57.079525696Z | x1003c2s6b1n0 | Chassis | VoltageRegulator | 0 | 55.4 |
| x1003c2s6b1 | NodeBMC | cray-telemetry-temperature | 2022-04-01T18:17:56.585058025Z | x1003c2s6b1n0 | Chassis | VoltageRegulator | 2 | 45.8 |
| x1003c2s6b1 | NodeBMC | cray-telemetry-temperature | 2022-04-01T18:17:57.081500532Z | x1003c2s6b1n1 | Chassis | VoltageRegulator | 0 | 51.2 |
| x1003c2s6b1 | NodeBMC | cray-telemetry-temperature | 2022-04-01T18:17:56.580577726Z | x1003c2s6b1n1 | Chassis | VoltageRegulator | 2 | 45.8 |
| x1003c2s6b1 | NodeBMC | cray-telemetry-temperature | 2022-04-01T18:17:57.072975044Z | x1003c2s6b1n0 | MISSING | CPU         | 0   | 30.875000 |
| x1003c2s6b1 | NodeBMC | cray-telemetry-temperature | 2022-04-01T18:17:57.072913765Z | x1003c2s6b1n0 | MISSING | CPU         | 1   | 26.500000 |
| x1003c2s6b1 | NodeBMC | cray-telemetry-temperature | 2022-04-01T18:17:57.073033042Z | x1003c2s6b1n1 | MISSING | CPU         | 0   | 29.750000 |
| x1003c2s6b1 | NodeBMC | cray-telemetry-temperature | 2022-04-01T18:17:57.073074561Z | x1003c2s6b1n1 | MISSING | CPU         | 1   | 27.500000 |
+-------------+---------+--------------------------+-------------------------------+-------------+---------+-----------+---------+------+
```

TRANSLATE XNAME AND NID

```
ncn-m# sat bash
(1e2360e3e3f0) sat-container:/sat # sat status | head -4
+---------------+-----------+------+------+-------+------+---------+------+----------+---------+---------+--------+
| xname         | Aliases   | Type | NID  | State | Flag | Enabled | Arch | Class    | Role    | Subrole | NetType|
+---------------+-----------+------+------+-------+------+---------+------+----------+---------+---------+--------+
| x1000c0s0b0n0 | nid000004 | Node | 1000 | Ready | OK   | True    | X86  | Mountain | Compute | None    | Sling  |

(1e2360e3e3f0) sat-container:/sat # sat xname2nid x1000c0s0b0n0
nid001000
(1e2360e3e3f0) sat-container:/sat # sat nid2xname 1000
x1000c0s0b0n0
(1e2360e3e3f0) sat-container:/sat # sat xname2nid x1000c0s0b0
nid001000,nid001001
```

This BMC has two nodes which would be affected by hardware work

```
(1e2360e3e3f0) sat-container:/sat # sat xname2nid x3000c0s19,x1000c0s0b0n0
nid[000001-000004,1000]
```

Recursively expand slot, chassis, and cabinet xnames to a range of nids

```
(1e2360e3e3f0) sat-container:/sat # sat xname2nid -f nid x3000c0s19,x1000c0s0b0n0
nid000001,nid000002,nid000003,nid000004,nid001000
```

Recursively expand slot, chassis, and cabinet xnames to a list of nids

TRACK HARDWARE

- Display hardware component history by xname or Field-Replaceable Unit (FRU) ID by querying HSM
  - FRU ID was added to output of `sat hwinv`

```
ncn-m# sat hwhist --help
usage: sat hwhist [-h] [-f PATH] [-x XNAME] [--format {pretty,yaml,json}] [--no-borders] [--no-headings]
       [--reverse] [--sort-by FIELD] [--show-empty] [--show-missing] [--fields FIELDS] [--filter QUERY]
       [--by-fru] [--fruid FRUID]

Report hardware component history.

optional arguments:
  -h, --help            show this help message and exit
  --by-fru              Display hardware component history by FRU.
  --fruid FRUID, --fruids FRUID
                        A comma-separated list of FRUIDs to include in the hardware component history report.
xnames:
  Options for specifying target xnames.

  -f PATH, --xname-file PATH
     Path to a newline-delimited file of xnames. In order to share the path between the host and container
     when sat is run in a container environment, the path should be either an absolute or relative path of
     a file in or below the home or current directory. Overrides value set in config file.
  -x XNAME, --xname XNAME, --xnames XNAME
     Specify an xname on which to operate. Multiple xnames may be specified via comma-separated entries or
     by providing this option multiple times.
```

HARDWARE MANAGEMENT

HARDWARE MANAGEMENT MICROSERVICES

- System Layout Service (SLS)
  - “Single source of truth" for the system design
- Hardware State Manager (HSM)
  - Operational datastore for current state of all components in the system
- Mountain Endpoint Discovery Service (MEDS)
  - Redfish endpoint discovery for liquid-cooled Olympus (Mountain) hardware
- River Endpoint Discover Service (REDS)
  - Redfish endpoint discovery for air-cooled (River) hardware
- Redfish Translation Service (RTS)
  - Provide Redfish appearance for everything that cannot do Redfish, that such as PDUs with JAWS protocol
- Cray Advanced Platform Monitoring and Control (CAPMC)
  - Power Control and Power Capping for all components
- System Configuration Service (SCSD)
  - Set various BMC and controller parameters
- Firmware Action Service (FAS)
  - Manages firmware for all Out-Of-Band components

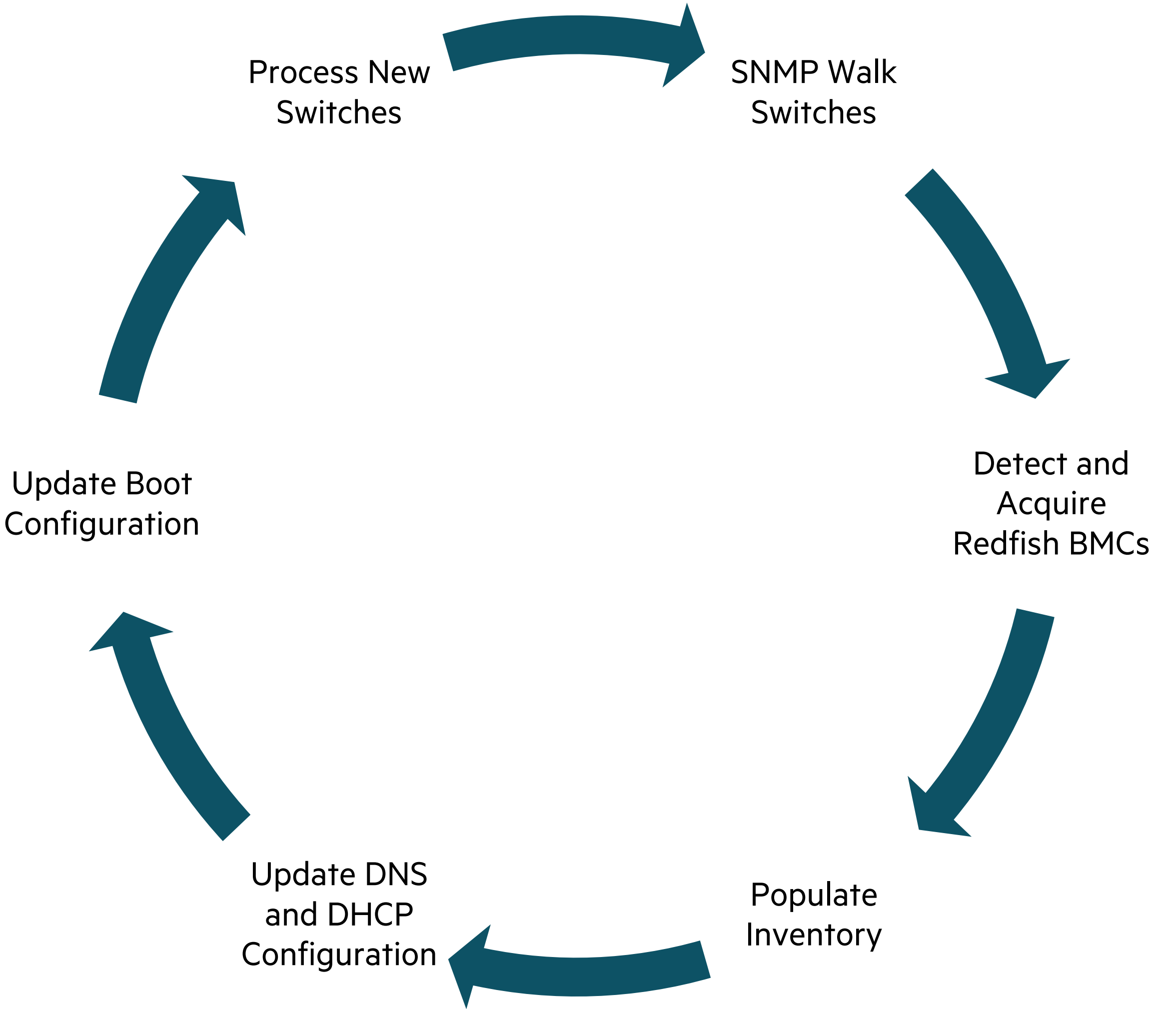
SYSTEM LAYOUT SERVICE (SLS)

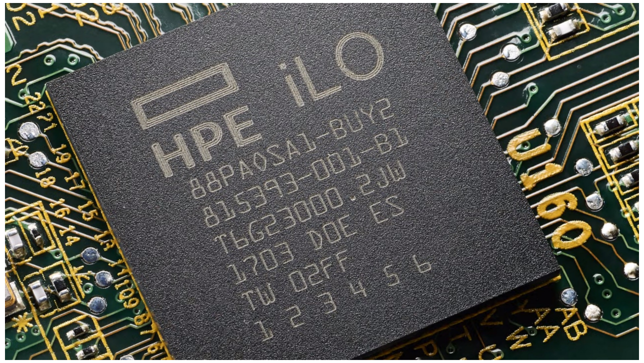
- SLS stores a generalized abstraction of the system that other services can access
  - Populated at install time with system-specific information that describes the hardware and network perimeter
  - Details the physical locations of network hardware, management nodes, application nodes, compute nodes, and cabinets
  - Stores information about the network, such as which port on which switch should be connected to each node
  - Does not need to change as hardware within the system is replaced
- -SLS is responsible for the following:
  - Providing an HTTP API to access site information
  - Storing a list of all hardware
  - Storing a list of all network links
  - Storing a list of all power links
- Changes to system setup which require updating data in SLS
  - Changing system cabling
  - Expanding the system
  - Reducing the system
  - Updating UAN CAN IP addresses
  - Updating UAN hostname aliases

```
ncn# cray sls hardware
ncn# cray sls networks
ncn# cray sls search hardware list
ncn# cray sls search networks list
ncn# cray sls dumpstate
ncn# cray sls loadstate
```

CLOSED LOOP HARDWARE DISCOVERY

- No Static Inventory Required
- Redfish for discovery and acquisition of all devices
- No “test boot” or discovery image required
- Diskless nodes can get completely new images and identity with reboot
- Suitable for any device with Redfish and DHCP/BootP





Redfish

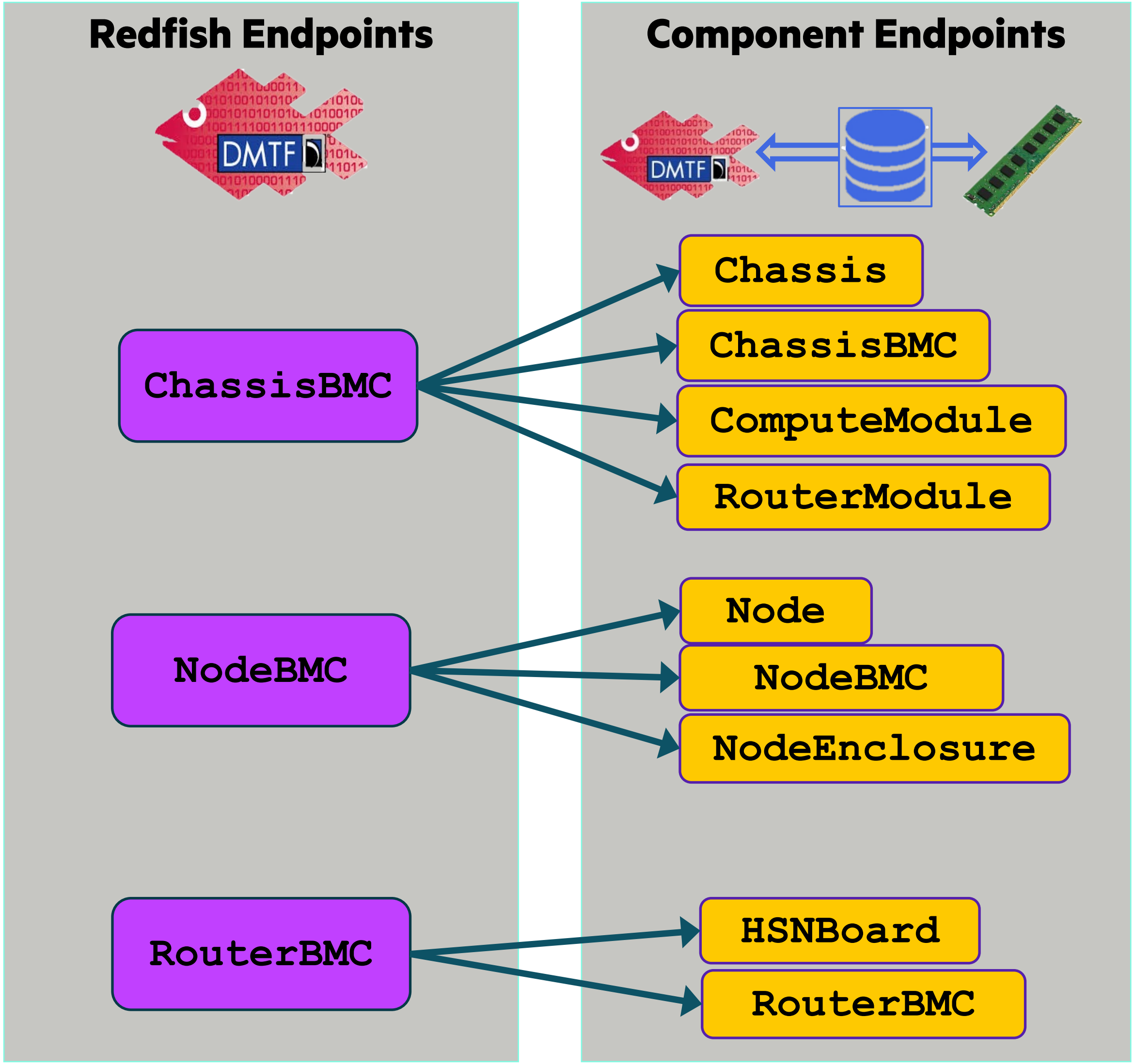
HARDWARE STATE MANAGER (HSM)

- Provides Redfish endpoint management and discovery
  - Discovery of node, switch, and chassis controllers running Redfish top-level entry points
    - Provides Redfish endpoint information to **H**ardware **M**anagement **S**ervices, which allows HMS to interact directly with parent endpoints of system components
    - Provides hardware inventory and component state data to other system services, which limits the number of systems that interact directly with Redfish
- Performs hardware inventory discovery
  - Uses raw data collected during discovery of Redfish endpoints
  - Stores detailed information on hardware present in the system at discovery
    - Includes information that is independent of the component's current location (serial number for FRU tracking)
- Provides component state management
  - Tracks logical component states and other dynamic information (e.g., node roles) needed for most common administrative and operational functions

HARDWARE STATE MANAGER DATA STRUCTURES

- Redfish Endpoints
  - The entity where Redfish runs (nC, sC, cC, BMC)
- Component Endpoints
  - They are the management representation of system components and are linked to the parent RedfishEndpoint
- Components
  - Provide a higher-level HSM representation of the component, including State, NID, Role (i.e. compute/service), Subtype, and so on
- NodeMaps
  - A mapping of one xname ID to a NID and, optionally, other default information
- Role and Subrole
  - A node has exactly one Role and Subrole value
  - Role Management with Subrole (master, worker, storage)
  - Role Application with Subrole (UAN, Gateway, and multiple site-definable Subroles)
  - Role Compute with no Subroles
- Groups
  - A group is a division of the system that groups components using a site-definable group name
  - Components can be members of multilple groups
- Partitions
  - Partitions are used as an access control mechanism
  - Each component may belong to at most one partition
- Memberships
  - a mapping of a component xname to its set of group labels and partition names

HSM DATA HIERARCHY



All components that are tracked by HSM must be associated with a Redfish endpoint!

```
ncn# cray hsm inventory componentEndpoints list --
redfish-ep x1000c0b0 |grep -w Type |sort -u
Type = "Chassis"
Type = "ChassisBMC"
Type = "ComputeModule"
Type = "RouterModule"
```

```
ncn# cray hsm inventory componentEndpoints list --
redfish-ep x1000c0s1b0 |grep -w Type |sort -u
Type = "Node"
Type = "NodeBMC"
Type = "NodeEnclosure"
```

```
ncn# cray hsm inventory componentEndpoints list --
redfish-ep x1000c0r3b0 |grep -w Type |sort -u
Type = "HSNBoard"
Type = "RouterBMC"
```

HSM SAMPLE DATA: REDFISHENDPOINTS

```
ncn# cray hsm inventory redfishEndpoints describe x1000c0s1b0
  "ID": "x1000c0s1b0",
  "Type": "NodeBMC",
  "Hostname": "x1000c0s1b0",
  "Domain": "",
  "FQDN": "x1000c0s1b0",
  "Enabled": true,
  "User": "root",
  "Password": "",
  "MACAddr": "02:03:E8:00:31:00",
  "RediscoverOnUpdate": true,
  "DiscoveryInfo": {
    "LastDiscoveryAttempt": "2021-10-29T21:19:03.935137Z",
    "LastDiscoveryStatus": "DiscoverOK",
    "RedfishVersion": "1.2.0"
```

Retrieving Redfish Endpoint inventory
Liquid-cooled compute node

Note the MAC address on a proprietary liquid-cooled blade, the MACAddr is algorithmically assigned to the node based on its position in the system

```
ncn# cray hsm inventory redfishEndpoints describe x3000c0s20b2
  "ID": "x3000c0s20b2",
  "Type": "NodeBMC",
  "Hostname": "x3000c0s20b2",
  "Domain": "",
  "FQDN": "x3000c0s20b2",
  "Enabled": true,
  "UUID": "b42e9978-5486-be03-0010-debfe042c46d",
  "User": "root",
  "Password": "",
  "MACAddr": "b42e99785486",
  "RediscoverOnUpdate": true,
  "DiscoveryInfo": {
    "LastDiscoveryAttempt": "2021-04-05T18:32:17.643790Z",
    "LastDiscoveryStatus": "DiscoverOK",
    "RedfishVersion": "1.7.0"
```

Retrieving Redfish Endpoint inventory
Air-cooled node

Note the MAC address on a commodity node, the MACAddr is assigned to the device by the manufacturer

HSM SAMPLE DATA: COMPONENT ENDPOINTS

```
ncn# cray hsm inventory componentEndpoints describe x1000c0s1b0n0
{
  "ID": "x1000c0s1b0n0",
  "Type": "Node",
  "RedfishType": "ComputerSystem",
  "RedfishSubtype": "Physical",
  "OdataID": "/redfish/v1/Systems/Node0",
  "RedfishEndpointID": "x1000c0s1b0",
  "Enabled": true,
  "RedfishEndpointFQDN": "x1000c0s1b0",
  "RedfishURL": "x1000c0s1b0/redfish/v1/Systems/Node0",
  "ComponentEndpointType": "ComponentEndpointComputerSystem",
  "RedfishSystemInfo": {
    "Name": "Node0",
    "Actions": {
      "#ComputerSystem.Reset": {
        "ResetType@Redfish.AllowableValues": [

"EthernetNICInfo": [
        "RedfishId": "HPCNet0",
        "@odata.id": "/redfish/v1/Systems/Node0/EthernetInterfaces/HPCNet0",
        "Description": "Shasta Timms NMC REV04 (HSN)",
        "MACAddress": "Not Available",
        "PermanentMACAddress": "00:40:a6:83:57:16"

        "RedfishId": "ManagementEthernet",
        "@odata.id": "/redfish/v1/Systems/Node0/EthernetInterfaces/ManagementEthernet",
        "Description": "Node Maintenance Network",
        "MACAddress": "00:40:a6:83:63:39",
        "PermanentMACAddress": "00:40:a6:83:63:39"
```

You would see:
```
 "ForceOff",
 "Off",
 "On"
```

Retrieving Component Endpoint inventory
liquid-cooled compute node

Output in slide has been trimmed to fit in slide

HSM SAMPLE DATA: COMPONENT ENDPOINTS

Retrieving Component Endpoint inventory **air-cooled node**

```
ncn# cray hsm inventory componentEndpoints describe x3000c0s17b1n0
ID = "x3000c0s17b1n0"
Type = "Node"
RedfishType = "ComputerSystem"
RedfishSubtype = "Physical"
UUID = "32324C58-6E35-3054-3031-505030313635"
OdataID = "/redfish/v1/Systems/1"
RedfishEndpointID = "x3000c0s17b1"
Enabled = true
RedfishEndpointFQDN = "x3000c0s17b1"
RedfishURL = "x3000c0s17b1/redfish/v1/Systems/1"
ComponentEndpointType = "ComponentEndpointComputerSystem"

[RedfishSystemInfo]
Name = "Computer System"
PowerURL = "/redfish/v1/Chassis/1/Power"
[[RedfishSystemInfo.EthernetNICInfo]]
RedfishId = "1"
"@odata.id" = "/redfish/v1/Systems/1/EthernetInterfaces/1"
MACAddress = "94:40:c9:c1:61:d4"
MemberId = "0"
PowerCapacityWatts = 1600
…
[RedfishSystemInfo.Actions."#ComputerSystem.Reset"]
"ResetType@Redfish.AllowableValues" = [ "On", "ForceOff", "GracefulShutdown", "ForceRestart", "Nmi", "PushPowerButton",]
"@Redfish.ActionInfo" = ""
target = "/redfish/v1/Systems/1/Actions/ComputerSystem.Reset"
```

HARDWARE INVENTORY DATA SAMPLE

```
ncn# cray hsm inventory hardware describe x3000c0s17b1 --
format json | jq '.NodeBMCLocationInfo'
{
  "DateTime": "2021-03-19T18:51:13Z",
  "DateTimeLocalOffset": "+08:00",
  "Description": "",
  "FirmwareVersion": "iLO 5 v2.12",
  "Id": "1",
  "Name": "Manager"
}
```

Retrieving ILO5 Information

```
ncn# cray hsm inventory hardware describe x3000c0s17b1n0d1 --
format json | jq '.PopulatedFRU.MemoryFRUInfo'
{
  "BaseModuleType": "RDIMM",
  "BusWidthBits": 72,
  "CapacityMiB": 16384,
  "DataWidthBits": 64,
  "ErrorCorrection": "MultiBitECC",
  "Manufacturer": "Micron",
  "MemoryType": "DRAM",
  "MemoryDeviceType": "DDR4",
  "OperatingSpeedMhz": 3200,
  "PartNumber": "18ASF2G72PDZ-3G2E1",
  "RankCount": 2,
  "SerialNumber": "266463A6"
}
```

Retrieving DIMM Information

```
ncn# cray hsm inventory hardware describe x3000c0s20b2n0p0
{
  "ID": "x3000c0s20b2n0p0",
  "Type": "Processor",
  "Ordinal": 0,
  "Status": "Populated",
  "HWInventoryByLocationType": "HWInvByLocProcessor",
  "ProcessorLocationInfo": {
    "Id": "1",
    "Name": "Processor 1",
    "Description": "Processor Instance 1",
    "Socket": "P0"
  },
  "PopulatedFRU": {
    "FRUID": "Processor.AdvancedMicroDevicesInc.2B48D1E76B14022",
    "Type": "Processor",
    "Subtype": "",
    "HWInventoryByFRUType": "HWInvByFRUProcessor",
    "ProcessorFRUInfo": {
      "InstructionSet": "x86-64",
      "Manufacturer": "Advanced Micro Devices, Inc.",
      "MaxSpeedMHz": 3350,
      "Model": "AMD EPYC 7702 64-Core Processor                ",
      "SerialNumber": "2B48D1E76B14022",
      "PartNumber": "",
      "ProcessorArchitecture": "x86",
      "ProcessorId": {
        "EffectiveFamily": "AMD Zen Processor Family",
        "EffectiveModel": "0x31",
        "IdentificationRegisters": "178bfbff00830f10",
        "MicrocodeInfo": "",
        "Step": "0x0",
        "VendorID": "AuthenticAMD"
      },
      "ProcessorType": "CPU",
      "TotalCores": 64,
      "TotalThreads": 128,
      "Oem": null
    }
```

Retrieving Processor Information

HARDWARE INVENTORY DATA SAMPLE – ethernetInterfaces

```
ncn# cray hsm inventory ethernetInterfaces list --format json | jq 'map(select(.ComponentID =="x3000c0s17b1n0"))'
[
  {
    "ID": "9440c9c161d4",
    "Description": "",
    "MACAddress": "94:40:c9:c1:61:d4",
    "LastUpdate": "2021-03-11T12:18:43.539116Z",
    "ComponentID": "x3000c0s17b1n0",
    "Type": "Node",
    "IPAddresses": []
  },
  {
    "ID": "ecebb88dec20",
    "Description": "",
    "MACAddress": "ec:eb:b8:8d:ec:20",
    "LastUpdate": "2021-03-11T12:42:02.44448Z",
    "ComponentID": "x3000c0s17b1n0",
    "Type": "Node",
    "IPAddresses": [
      {
        "IPAddress": "10.254.1.23"
      }
    ]
.
.
]
```

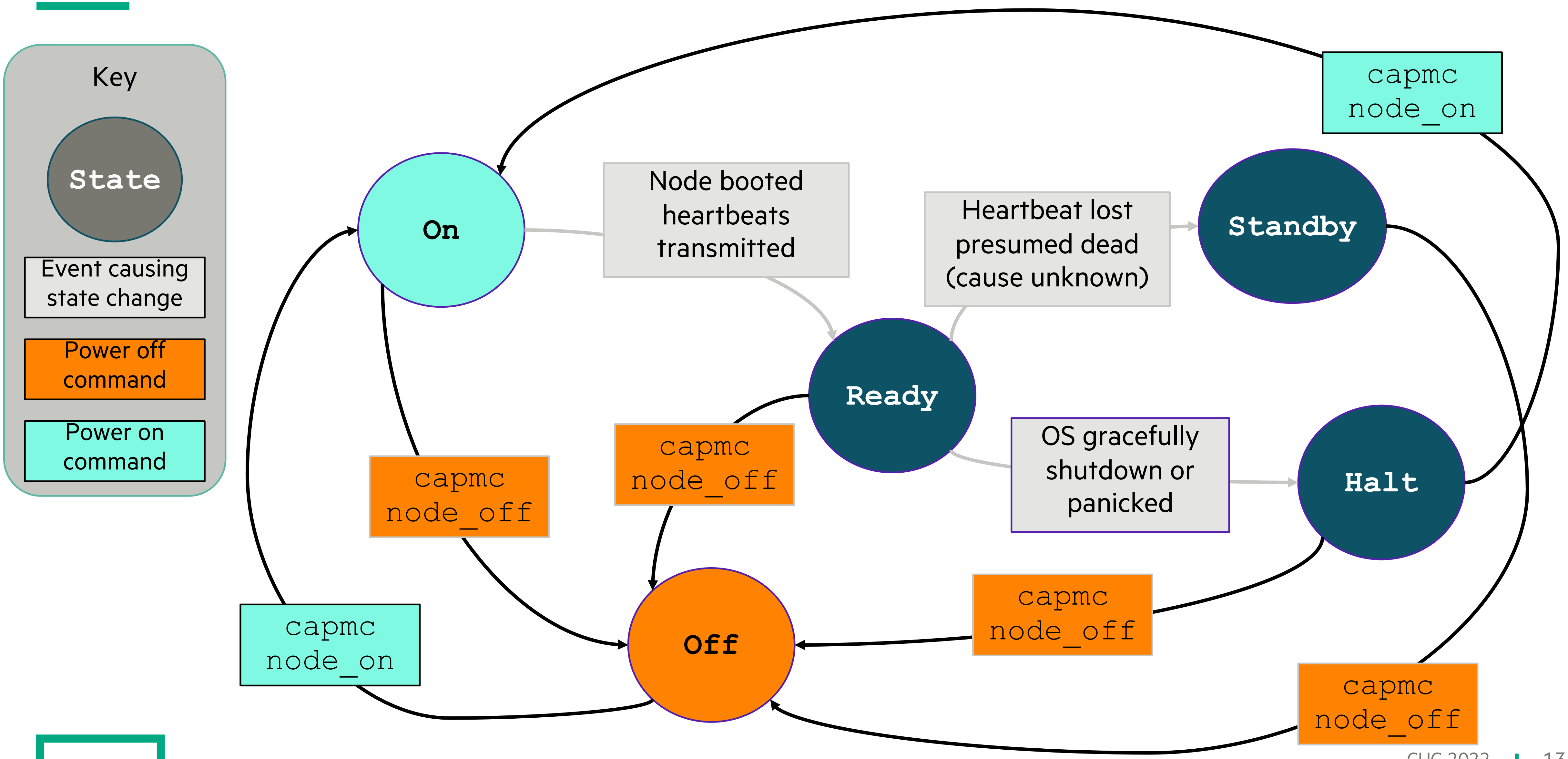
For the bmc interface(s) query for the xname of the bmc

(Some lines omitted for clarity)

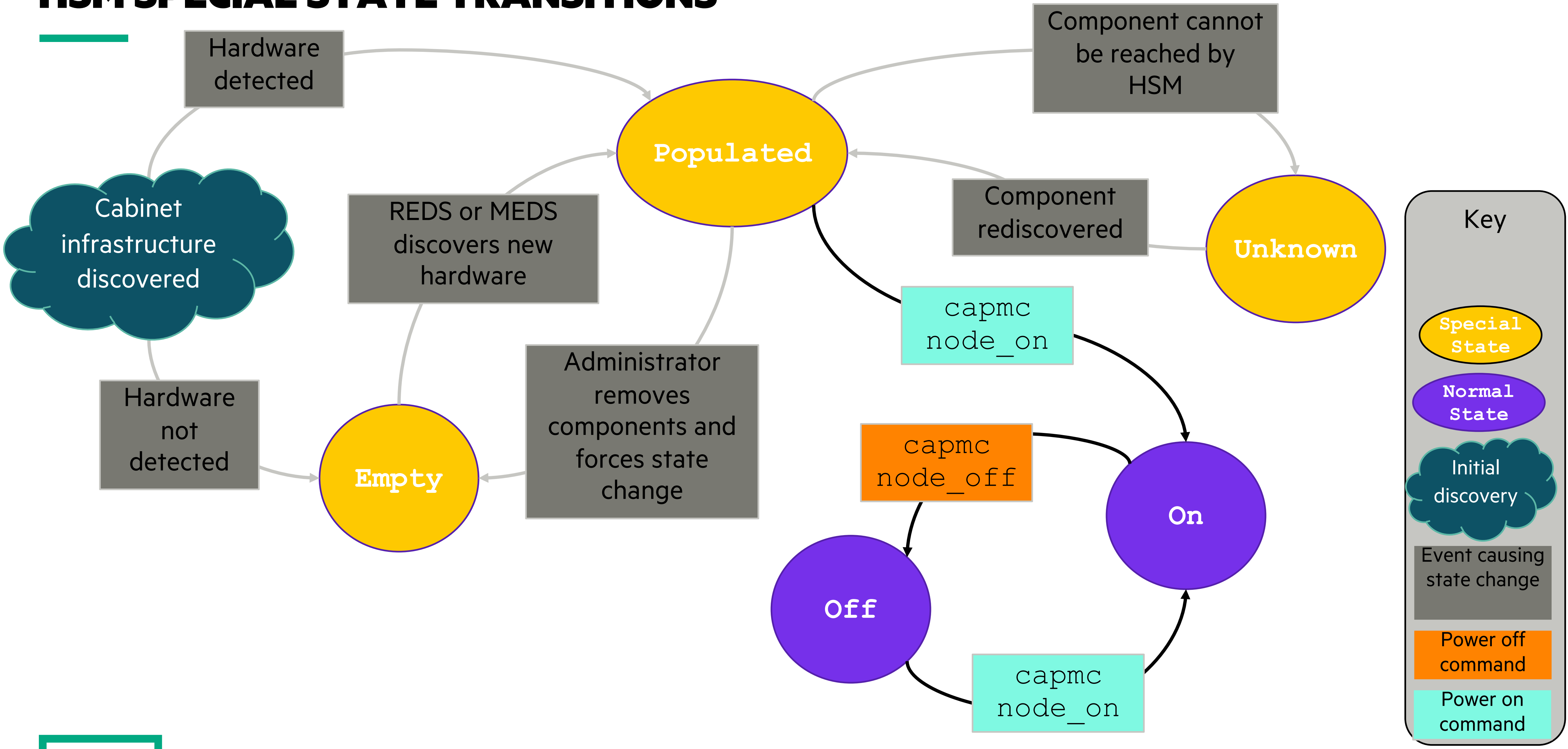
HSM STATE DEFINITIONS

HSM State	Definition
Unknown	The state is unknown. Appears missing but has not been confirmed as empty
Empty	The location is not populated with a component
Populated	Present (not empty), but no further tracking can or is being done
Off	Present but powered off
On	Powered on. If no heartbeat mechanism is available, its software state may be Unknown
Standby	No longer Ready and presumed dead. It typically means heartbeat has been lost (with alert)
Halt	No longer Ready and halted. OS has been gracefully shutdown or panicked (with alert)
Ready	Both On and Ready at Linux multi-user state to provide its expected role in the system

HSM STATE TRANSITION MAP



HSM SPECIAL STATE TRANSITIONS



DISABLING NODES IN HSM

<u>Problem</u> – One unhealthy node fails to boot

```
ncn# kubectl logs -f -n services boa-6729097c-8f06-4169-8008-c40a06087677-mjwqq -c boa
...
2022-04-19 19:43:17,448 - ERROR   - cray.boa.agent - Nodes were not ready: Number of retries: 361 exceeded allowed amount: 360; 1 nodes
were not in the state: Ready
2022-04-19 19:43:17,483 - ERROR   - cray.boa.agent - Traceback (most recent call last):
2022-04-19 19:43:17,483 - ERROR   - cray.boa.agent - These nodes failed to reboot. {'x3000c0s27b0n0'}
2022-04-19 19:43:17,483 - ERROR   - cray.boa.agent - You can attempt to reboot these nodes by issuing the command:
cray bos v1 session create --template-uuid uan-sessiontemplate-2.3.2-cos-2.2.101-gpfs --operation reboot --limit x3000c0s27b0n0
...
ncn# cray hsm state components enabled update x3000c0s19b3n0 --enabled false
ncn# cray hsm state components describe x3000c0s19b3n0
Type = "Node"
Enabled = false
State = "Off"
NID = 3
ID = "x3000c0s19b3n0"
Flag = "OK"
Role = "Compute"
NetType = "Sling"
Arch = "X86"
Class = "River"

ncn# cray hsm state components bulkEnabled update x3000c0s19b3n0,x3000c0s19b3n1,x3000c0s3b1n0,x3000c0s3b1n0 --enabled true
```

<u>Solution</u> – Disable the unhealthy node in HSM and relaunch the boot

<u>Note:</u> Can also affect a list of several components with a single call

CREATING AND UPDATING HSM GROUPS

```
ncn# cray hsm groups list
results = []

ncn# cray hsm groups create --label blue
[[results]]
URI = "/hsm/v1/groups/blue"

ncn# cray hsm groups update --description ”All compute nodes that are blue" blue

ncn# cray hsm groups members create --id  x3000c0s24b4n0 blue
[[results]]
URI = "/hsm/v1/groups/blue/members/x3000c0s24b4n0"

ncn# cray hsm groups describe blue
description = "All compute nodes that are blue"
label = ”blue"

[members]
ids = [ "x3000c0s24b4n0",]

ncn# cray hsm groups members delete  x3000c0s24b4n0 blue
message = "deleted 1 entry"
code = 0
```

ADDING MULTIPLE MEMBERS TO AN HSM GROUP

```
ncn# cray hsm groups members create --id x3000c0s19b[1-4]n0 blue
Usage: cray hsm groups members create [OPTIONS] GROUP_LABEL
Try 'cray hsm groups members create --help' for help.

Error: Bad Request: invalid xname ID
ncn# for XNAME in $(sat status --filter role=Compute --no-borders --no-headings | awk '{print $1}');
do cray hsm groups members create --id $XNAME blue; done
[[results]]
URI = "/hsm/v1/groups/blue/members/x3000c0s19b1n0"

[[results]]
URI = "/hsm/v1/groups/blue/members/x3000c0s19b2n0"
...
[[results]]
URI = "/hsm/v1/groups/blue/members/x3000c0s24b4n0"

ncn# cray hsm groups describe blue
description = "All compute nodes that are blue"
label = ”blue"

[members]
ids = [ "x3000c0s19b1n0", "x3000c0s19b2n0", "x3000c0s19b3n0", "x3000c0s19b4n0", "x3000c0s21b1n0",
"x3000c0s21b2n0", "x3000c0s21b3n0", "x3000c0s21b4n0", "x3000c0s24b1n0", "x3000c0s24b2n0",
"x3000c0s24b3n0", "x3000c0s24b4n0",]
```

Due to a parsing limitation in the API adding multiple nodes in a single command is not supported

For now the work around is to add one node at a time in a shell script loop

CRAY ADVANCED PLATFORM MONITORING AND CONTROL (CAPMC)

- The Cray Advanced Platform Monitoring and Control (CAPMC) API
  - Enables direct hardware control of power on/off, power monitoring, or system-wide power, telemetry and configuration parameters from Redfish
  - Implements a simple interface for powering on/off compute nodes, querying node state information, and querying site-specific service usage rules
    - These controls enable external software to more intelligently manage system-wide power consumption or configuration parameters
- Features
  - Retrieve Redfish power status and power management capabilities of components
  - Control single components via NID or xname
  - Control grouped components
  - Control the entire system
  - Can specify ancestors (--prereq) and descendants (--recursive) of single component
  - Provide a --force option for immediate power off
  - Power sequencing

CAPMC GET STATUS BY NODE OR XNAME

```
ncn# cray capmc get_node_status create --filter show_all
{
  "e": 0,
  "err_msg": "",
  "off": [ 1006, 1038, 1108, 1110 ],
  "on": [ 1076, 49168960 ],
  "ready": [ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14,
    15, 16, 1004, 1005, 1007, 1020, 1021, 1022, 1023, 1028,1029, 1030, 1031, 1036, 1037, 1039, 1077,
    1078, 1079, 1109, 1111, 1180, 1181, 1182, 1183, 1252, 1253, 1254, 1255, 100001, 100002, 100003,
    100004, 100005, 100006, 100007, 100008, 100009, 49168832, 49168896 ]
}
ncn# cray capmc get_xname_status create --filter show_all
{
  "e": -1,
  "err_msg": "Errors encountered with 1/109 Xnames for Status",
  "off": [ "x1000c0s1b1n0", "x1000c1s1b1n0", "x1000c3s3b0n0", "x1000c3s3b1n0" ],
  "on": [ "x1000c0", "x1000c0r3", "x1000c0r3e0", "x1000c0r7", "x1000c0r7e0", "x1000c0s1", "x1000c0s1b0n0",
    "x1000c0s1b0n1", "x1000c0s1b1n1", "x1000c0s5", "x1000c0s5b0n0", "x1000c0s5b0n1", "x1000c0s5b1n0",
    "x1000c0s5b1n1", "x1000c0s7", "x1000c0s7b0n0", "x1000c0s7b0n1", "x1000c0s7b1n0", "x1000c0s7b1n1",
    ... << clipped for space >>
    "x3000c0s37b3n0", "x3000c0s37b4n0" ],
  "undefined": [ "x3000c0s2b0n0" ]
}
```

capmc get_xname_status command to show all ***nodes*** and their status.

capmc get_xname_status command to list all nodes by their ***xnames*** and their power status.

SAMPLE REDFISH API CALL

```
ncn# curl -k -u admin:password -s https://x3000c0s20b1/redfish/v1/Systems/Self/ResetActionInfo |jq
{
  "@odata.context": "/redfish/v1/$metadata#ActionInfo.ActionInfo",
  "@odata.etag": "W/\"1601653292\"",
  "@odata.id": "/redfish/v1/Systems/Self/ResetActionInfo",
  "@odata.type": "#ActionInfo.v1_1_1.ActionInfo",
  "Description": "This action is used to reset the Systems",
  "Id": "ResetAction",
  "Name": "ResetAction",
  "Parameters": [
    {
      "AllowableValues": [
        "ForceRestart",
        "On",
        "GracefulShutdown",
        "ForceOff"
      ],
      "DataType": "String",
      "Name": "ResetType",
      "Required": true
    }
  ]
}
```

Every Redfish endpoint on the system can be interacted with for hardware monitoring and management with the proper URL and credentials

The allowable values can be POSTed to the action URL to change the state of the node.

For example: Sending an HTTP POST of **{"ResetType": "On"}** to the API path **redfish/v1/Systems/Self/Actions/ComputerSystem.Reset** will attempt to power up the node

CAPMC POWER CONTROL

```
ncn# cray capmc node_off create --nids 7
e = 0
err_msg = ""
```

The capmc call shown generated the Redfish calls shown in the logs below.

```
ncn# kubectl logs -n services --since=1m -l app.kubernetes.io/instance=cray-hms-capmc -c cray-capmc
2020/04/13 16:35:40 [DEBUG] GET http://cray-vault.vault:8200/v1/secret/hms-creds/x3000c0s23b3n0
2020/04/13 16:35:40 nodectl.go:134: Info: Node power command: Off, nids: [7], reason:
2020/04/13 16:35:40 capmcd.go:305: Info: --> HTTP POST http://cray-smd/hsm/v1/locks
2020/04/13 16:35:40 capmcd.go:326: Info: <-- HTTP 201 Created POST http://cray-smd/hsm/v1/locks
(6.588427ms)
2020/04/13 16:35:40 bmcapi.go:494: Info: Node: 'x3000c0s23b3n0', NodeBMC: '10.254.2.16', Command: 'Off'
2020/04/13 16:35:40 capmcd.go:305: Info: --> HTTP POST
https://10.254.2.16/redfish/v1/Systems/Self/Actions/ComputerSystem.Reset
2020/04/13 16:35:49 capmcd.go:326: Info: <-- HTTP 204 No Content POST
https://10.254.2.16/redfish/v1/Systems/Self/Actions/ComputerSystem.Reset (8.746677165s)
2020/04/13 16:35:49 capmcd.go:305: Info: --> HTTP DELETE http://cray-smd/hsm/v1/locks/ab2e36fd-0a0a-49e6-
bb11-02cbf2102946
2020/04/13 16:35:49 capmcd.go:326: Info: <-- HTTP 200 OK DELETE http://cray-smd/hsm/v1/locks/ab2e36fd-
0a0a-49e6-bb11-02cbf2102946 (7.933897ms)
2020/04/13 16:35:49 capmcd.go:272: Info: --> 127.0.0.1:40582 HTTP 200 OK POST /capmc/v1/node_off
(8.800847806s)
```

- Note: The System Admin Toolkit command **`sat status`** will give a list of nodes `xnames` and their nid number

BOOTING AND SHUTTING DOWN THE SYSTEM

- The entire system can be booted or shut down in the documented, ordered stages
  https://github.com/Cray-HPE/docs-csm/blob/release/1.0/operations/power_management/System_Power_Off_Procedures.md

```
ncn-m# sat bootsys boot --list-stages
ncn-power
platform-services
k8s-check
cabinet-power
bos-operations

ncn-m# sat bootsys shutdown --list-stages
capture-state
session-checks
bos-operations
cabinet-power
platform-services
ncn-power
```

- Can boot or shutdown multiple BOS session templates at the same time

```
ncn-m# sat bootsys boot --stage bos-operations --bos-templates A,B,C
```

SYSTEM CONFIGURATION SERVICE (SCSD)

```
ncn# cray scsd bmc cfg  describe x3000c0r42b0 --format yaml
Force: false
Params:
  SyslogServerInfo:
    ProtocolEnabled: true
    SyslogServers:
    - 10.94.100.53
    Port: 514
    Transport: udp
  NTPServerInfo:
    NTPServers:
    - 10.254.0.5
    ProtocolEnabled: true
    Port: 123
  SSHKey: "ssh-rsa
AAAAB3NzaC1yc2EAAAADAQABAAACAQC/GrzGi0ff8nhKCP9E09sFf+gNf0ibP53DOr/a25JZxhAlw7QKJcpBhK/JOi/ch8QAM8YxLTae4
0p7jjfI5bJ58Y0HeSgVUmUTWh6QIFMd+CqQRU2jSv6m3gISrzSOmEkBtcl0RuTmOJxvg5tKL9Qm6ymVQwSK0KWFQAwTI0I7DkyinqcRbV
R3HVC4vnCQDWjPumQHzheRzIDfkUtOfX/YcFjH5GtvQzqolh0mLEQm2mpjWoXjAXK6RkI3SttxnMW/IM2RplynvC/ffGAYJwu0xZXb48y
ga0yhd7REL6Kcvahlc2jQqgXgqC0siMSoLSqKooKiZPZrxh1IdWf9Ic1mvqg7wbx3NDBik0vjU+ZChDicZw80vHtl217QegvzONT+XoEJ
o/F2Arur2UISZIT5THzJvMaFqnXQvgF6y+ejg+Il13NnWA5kds/sBnsl3VhxkOUWYHi6v0CPWmkBmQPdoah+K4gY1lfOQtu9wf8EXyLB+
NYjotf2D31URmJEBVGnL1CpTKPMxLF4zZVPu2jc1rilRfv34fbngzGJurne0TifmULFlT3yfzkJiIj+vJLzVsaoHRypzduvQZgTSP8hl5
ERPbfHIuGHDc3hKOdzq//JxF4qrw4voUab5+u/0YH6f4frzKQD7YcxMHvCjCFKynxvdVbouQctIPUrEi3Qvw==\
    \ \n"
```

`scsd` is used to retrieve and set redfish controller (BMC) parameters including:

- SSH keys
- NTP server
- Syslog server
- BMC/Controller passwords

See *System Configuration Service* documentation for details

FIRMWARE ACTION SERVICE (FAS)

- FAS updates firmware of Redfish-enabled hardware

Manufacturer	Redfish Endpoint	Firmware Target
HPE Cray	Liquid-cooled node BMC	BMC, Node0.BIOS, Node1.BIOS, Recovery
HPE Cray	Chassis BMC	BMC, Recovery
HPE Cray	Slingshot switch BMC	BMC, Recovery
Gigabyte	Air-cooled node BMC	BMC, BIOS
HPE	Air-cooled node BMC	iLO 5 (BMC), System ROM (BIOS)

FAS TERMINOLOGY

- Action
  - A collection of operations initiated by user request to update to the firmware images on a set of hardware
- Operation
  - An update (upgrade/downgrade) to a specific device's Firmware Target
- Snapshot
  - Point-in-time record of what firmware images were running on the system
  - Used to 'RESTORE' the system back to specific firmware versions
- Image
  - A JSON object that contains
    - Key data including `deviceType`, `manufacturer`, `model` and other information to identify the firmware
    - Process Guides that tell FAS how to update the firmware in question
    - S3 link (URL) where the firmware binary can be retrieved

UPDATING FIRMWARE WITH FAS

- Complete a dry-run using a JSON file describing which component types, and dry-run flag

```
ncn# vi fas_file.json
ncn# cray fas actions create fas_file.json
```

- Interpret the output of the dry-run
  - Poll the status of the action until the action `state` is `completed

```
mcn# cray fas actions describe actionID --format json
```

  - NoOp: Nothing to do, already at version.
  - NoSol: No viable image is available; this will not be updated.
  - succeeded:
    - IF dryrun: The operation should succeed if performed as a live update, FAS COULD update a component name (xname) + target with the declared strategy
    - IF live update: the operation succeeded, and has updated the component name (xname) + target to the identified version
  - failed:
    - IF dryrun: There is something that FAS could do, but it likely would fail; most likely because the file is missing
    - IF live update: the operation failed, the identified version could not be put on the component name (xname) + target
- If succeeded count > 0 now perform a real update
- Change the flag from dry-run to live in JSON file an do the real update

```
ncn# vi fas_file.json
ncn# cray fas actions create fas_file.json
```

- Interpret the output of the real update
  - Poll the status of the action until the action `state` is `completed

```
ncn# cray fas actions describe newactionID –format json
```

NETWORK MANAGEMENT

- Slingshot fabric manager
- Slingshot topology tool

ACCESSING THE SLINGSHOT FABRIC MANAGER CONTAINER

```
ncn# kubectl get pod -n services | grep slingshot
slingshot-fabric-manager-7bddfccc87-bhx9x                2/2     Running     0          45d
ncn# FMN_POD=$(kubectl get pod -n services | grep slingshot | awk '{print $1}')
ncn# echo $FMN_POD
slingshot-fabric-manager-7bddfccc87-bhx9x

ncn# kubectl exec -it -n services $FMN_POD -c slingshot-fabric-manager -- /bin/bash
slingshot-fabric-manager-7bddfccc87-bhx9x:/opt/slingshot # fmn_version
FMN : 1.0.4-63-20210421180621_89c5fbb
FMN Scripts : 1.0.4
FMN CLI : 1.0.4
Slingshot Fabric Manager : 1.0.4
Slingshot Certificate Manager : 1.0.4
Slingshot Tools : 1.0.4
Slingshot UI : not installed
Slingshot Web Server : not installed
slingshot-fabric-manager-7bddfccc87-bhx9x:/opt/slingshot # exit
exit
ncn#
```

The slingshot fabric manager runs in a container inside a Kubernetes pod

Executing bash in the slingshot fabric manager container causes the prompt to change

exit from within the container will quit the bash session

IMPORTANT FILES WITHIN THE FABRIC MANAGER CONTAINER

```
slingshot-fabric-manager-7bddfccc87-bhx9x:~ # head /opt/cray/fabric_template.json
{
    "links": [
        {
            "endpoint1": "x1000c0r3j1p1",
            "endpoint2": "x3000c0r42j31p1"
        },
        {
            "endpoint1": "x1000c0r3j1p0",
            "endpoint2": "x3000c0r42j31p0"
        },
slingshot-fabric-manager-7bddfccc87-bhx9x:~ # head  /opt/cray/etc/sct/Shasta_system_hsn_pt_pt.csv
cable_id,src_conn_a,src_conn_b,dst_conn_a,dst_conn_b,stage,src_egress_a,src_egress_b,dst_egress_a,dst_egress_b,link
_type,src_group,dst_group,part_number,part_length,calculated_distance,route
1000.1000.00.0080,x1000c0r3j10,none,x1000c5r7j11,none,1,none,none,none,n
1000.1000.00.0086,x1000c0r3j11,none,x1000c7r3j14,none,1,none,none,none,n
1000.1000.00.0090,x1000c0r3j12,none,x1000c7r7j11,none,1,none,none,none,n
1000.1000.00.0027,x1000c0r3j13,none,x1000c6r7j13,none,1,none,none,none,n
1000.1000.00.0025,x1000c0r3j14,none,x1000c6r3j13,none,1,none,none,none,n
1000.1000.00.0022,x1000c0r3j16,none,x1000c4r7j13,none,1,none,none,none,none,local,1,1,102234306,2.13,1.9636838,[]
1000.1000.00.0018,x1000c0r3j18,none,x1000c4r3j14,none,1,none,none,none,none,local,1,1,102234305,1.91,1.7582719,[]
3000.1000.00.0000,x1000c0r3j1,none,x3000c0r42j31,none,1,x3000-LEFT,none,x1000-
CENTRE,none,global,1,0,102253304,19M,5.7431222,['x3000_BACK-x1000_BACK']
1000.1000.00.0013,x1000c0r3j20,none,x1000c2r7j14,none,1,none,none,none,none,local,1,1,102234304,1.64,1.4212273,[]
```

fabric_template.json is the slingshot topology file used to initialize the slingshot fabric. It is a complete json formatted description of the slingshot switches and cables

Shasta_system_hsn_pt_pt.csv is a Slingshot cabling configuration file that is built from the SHCD and used to initially set up a slingshot network

FABRIC_TEMPLATE.JSON

- The template will be located at: `/opt/cray/fabric_template.json`
- Structure:

```
{
   "links": [
      {
         "endpoint1": "x1000c7r3j13p0",
         "endpoint2": "x1000c2r3j11p0"
      },...
   ],
   "maxNumLocalSwitches": 16,
   "numGroups": 2,
   "switches": [
      {
         "IP": "x1000c0r3b0",
         "edgePorts": [ {"id": "x1000c5r7a0l32", "meta": {"conn_port": "x1000c5r7j103p0"} }, ... ]
         "fabricPorts": [ {"id": "x1000c5r7a0l6", "meta": {"conn_port": "x1000c5r7j4p1"} }, ... ]
         "grpId": 1,
         "swcNum": 0
      },...
   ]
}
```

`links` are used for setting up routing between switches

The `fabricPorts` connect to other switches (L1 “Local” or L2 “Global”) and and `edgePorts` connect to hosts (L0)

Switch Group

Switch number within Group

SLINGSHOT FABRIC MANAGER COMMAND OPTIONS

fmctl

- Built on native REST APIs
  - Autogenerated from APIs, so automatically extended when new functionality is added
  - Reflect stable APIs that are not expected to change
- Simple method to interact using CLI
- Can output data in human-readable or machine-readable format, making them useful for DevOps

fmn_*

- “Helper” orchestration scripts
  - Usually, configuration or diagnostic related
- Not yet built into higher-level API services
- Designed for human-readable text/shell output only

FMCTL OVERVIEW

- Provides direct interaction with the Slingshot fabric API
  - Connects to the fabric API based on the OpenAPI specifications
- Allows basic CRUD operations and patch
  - Get, create, update, replace, delete
- Integrated into the Fabric Management container
  - Also available as an RPM for download and install
- Two modes of input:
  - Interactive mode
  - Command-line arguments

```
slingshot-fabric-manager-7bddfccc87-bhx9x:~ # fmctl
Usage:

fmctl { help | version }
fmctl { get | create | update | replace | delete } help
fmctl { get } <resource> [--fmn-endpoint ENDPOINT] [--timing] [--verbose lvl] [--raw] [--select key,key,...] [--response-code] [--
api-runtime-check]
fmctl { delete } <resource> [--fmn-endpoint ENDPOINT] [--timing] [--verbose lvl] [--raw] [--select key,key,...] [--response-code] [--
api-runtime-check]
fmctl { create | update | replace } <resource> [--fmn-endpoint ENDPOINT] [--timing] [--verbose lvl] [--raw] [--select key,key,...] [-
-response-code] [--api-runtime-check] {[--file payload.json] | [key=value ...]}
fmctl { interactive } [--fmn-endpoint ENDPOINT] [--timing]

Flag details:
    --verbose: silent, low, high, debug
    --raw: emit raw JSON
    --select: Comma separated list of fields to print, \n separator

Default config is written to: /root/.fmctlrc
```

FMCTL GET SWITCHES

```
slingshot-fabric-manager-7bddfccc87-bhx9x:~ # fmctl get switches
+---------------+-------------------------------+
|      KEY      |             VALUE             |
+---------------+-------------------------------+
| documentCount |                            17 |
| documentLinks | /fabric/switches/x3000c0r42b0 |
|             1 | /fabric/switches/x1000c6r3b0  |
|             2 | /fabric/switches/x1000c5r7b0  |
|             3 | /fabric/switches/x1000c2r7b0  |
|             4 | /fabric/switches/x1000c0r3b0  |
|             5 | /fabric/switches/x1000c3r7b0  |
|             6 | /fabric/switches/x1000c2r3b0  |
|             7 | /fabric/switches/x1000c7r3b0  |
|             8 | /fabric/switches/x1000c0r7b0  |
|             9 | /fabric/switches/x1000c7r7b0  |
|            10 | /fabric/switches/x1000c1r3b0  |
|            11 | /fabric/switches/x1000c1r7b0  |
|            12 | /fabric/switches/x1000c3r3b0  |
|            13 | /fabric/switches/x1000c5r3b0  |
|            14 | /fabric/switches/x1000c6r7b0  |
|            15 | /fabric/switches/x1000c4r3b0  |
|            16 | /fabric/switches/x1000c4r7b0  |
| totalCount    |                            17 |
+---------------+-------------------------------+
```

FMCTL GET SWITCHES/ANY_SWITCH

```
slingshot-fabric-manager-7bddfccc87-bhx9x:~ # fmctl get switches/x3000c0r42b0 | head -7
+------------------+---------------------------------------+
|       KEY        |                 VALUE                 |
+------------------+---------------------------------------+
| IP               | x3000c0r42b0                          |
| agentLink        | /fabric/agents/x3000c0r42b0           |
| documentSelfLink | /fabric/switches/x3000c0r42b0         |
| edgePortLinks    | /fabric/ports/x3000c0r42j12p0         |


slingshot-fabric-manager-7bddfccc87-bhx9x:~ # fmctl get switches/x3000c0r42b0 | tail -7
|                2 | map[id:x3000c0r42a010                 |
|                  | meta:map[conn_port:x3000c0r42j1p1]]   |
|                3 | map[id:x3000c0r42a011                 |
|                  | meta:map[conn_port:x3000c0r42j1p0]]   |
| grpId            |                                     0 |
| swcNum           |                                     0 |
+------------------+---------------------------------------+
slingshot-fabric-manager-7bddfccc87-bhx9x:~ # fmctl get switches/x3000c0r42b0 --raw \
| jq '.fabricPortLinks'
[
  "/fabric/ports/x3000c0r42j1p1",
  "/fabric/ports/x3000c0r42j1p0",
  "/fabric/ports/x3000c0r42j31p0",
  "/fabric/ports/x3000c0r42j31p1"
]
```

FMCTL PRIMARY COMMANDS

Command	Definition
`fmctl get`	View the current status of a resource or metric. Use with a resource path or OData parameter.
`fmctl create`	Creates a resource with the given parameters
`fmctl update`	Updates the value of a specified field
`fmctl replace`	Replaces the entire value of the resource to the new given parameters
`fmctl delete`	Deletes a resource
`fmctl interactive`	Start interactive mode
`fmctl get help`	View static resource paths. To view dynamic resource paths, use `fmctl get switches`. Example resource paths: • `/fabric/switches/<xname>` • `/fabric/routing-engines/dragonfly`
`fmctl version`	Print the `fmctl` version

COMMON FMN_* COMMANDS

Command	Definition
`fmn_pw`	Set password
`fmn_version`	Display version information
`fmn_switch_reset`	Reset switch (warm boot)
`fmn_update_switch_firmware`	Update firmware or display version
`fmn_fabric_bringup`	Start switch synchronization with FMN
`fmn_status`	Display fabric status
`fmn_cert_provision`	Provision certificates to switches
`fmn_shasta_dns`	Add or delete or print DNS records

FMN_* EXAMPLES

```
slingshot-fabric-manager-7bddfccc87-bhx9x:/opt/slingshot # fmn_status
Edge: 52 / 284
Fabric: 488 / 488
Ports Reported: 772 / 772
Fully Synchronized Switches: 17 / 17
slingshot-fabric-manager-7bddfccc87-bhx9x:/opt/slingshot # fmn_port_state --check x1000c4r7j1
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100   214  100   214    0     0  12588      0 --:--:-- --:--:-- --:--:-- 13375
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100   225  100   225    0     0  75000      0 --:--:-- --:--:-- --:--:--  109k
slingshot-fabric-manager-7bddfccc87-bhx9x:/opt/slingshot # fmn_fabric_snapshot
Tue Jul 13 05:25:45 UTC 2021
tar: Removing leading `/' from member names
/var/tmp/triage-data/
/var/tmp/triage-data/fmn_fabric_snapshot.log
/var/tmp/triage-data/fabric_template.json
/var/tmp/triage-data/routing-engine-data.txt
/var/tmp/triage-data/fabric-agent-x3000c0r42b0.txt
/var/tmp/triage-data/fabric-agent-x1000c7r3b0.txt
/var/tmp/triage-data/fabric-agent-x1000c6r3b0.txt
...
```

SLINGSHOT TOPOLOGY TOOL (STT)

- Runs in the Fabric Management container
- An integration of several Python-based tools generate configuration files for the Fabric Manager to perform diagnostics on the network
- Can take input from the SHCD file (configuration file), and builds the full "topology file" that is a map of the port-to-port connections for a Slingshot fabric: Shasta_system_hsn_pt_pt.csv
- Used via an interactive CLI
- Enables the following:
  - Import and export of different file formats (SHCD files, point-to-point files)
  - Generation of topologies from plugin algorithms
  - Inspection, modification, generation, and validation of topologies
  - Consolidation of diagnostic test scripts from various repositories, test execution, and result reporting
- Use of STT is optional
  - Slingshot fabric does not require STT to function

STT INTERACTIVE COMMAND LINE INTERFACE

```
slingshot-fabric-manager-7bddfccc87-bhx9x:/opt/slingshot # slingshot-topology-tool
Using Fabric Manager URL http://localhost:8000
STT diags log directory -  /opt/slingshot/stt_diags_logs
STT diags log directory -  /opt/slingshot/stt_diags_logs/default
Loading  point2point file /opt/cray/etc/sct/Shasta_system_hsn_pt_pt.csv to default topology
Loading fabric template file /opt/cray/fabric_template.json to default topology
Welcome to the Slingshot Topology Tool v1.0.4-20.
     General Usage is <command> <arguments>
     Type help or ? to list commands.

(STT) help

Documented commands (use 'help -v' for verbose/'help <topic>' for details):
===========================================================================
add                  copy      help     refresh_data  set_active
clear_cache          del       history  remove        set_threadpool_size
combine              exit      list     run           shell
compute_nodes_creds  filter    load     save          show
configure            generate  new      set           snapshot_data

(STT)
```

STT DIAGNOSTICS

```
(STT) help run
        Run diagnostic command
        Usage: run <command> <command options>
        Usage: run <command> <help|summary>

Available list of diagnostic commands:
         dgrperfcheck                   - Rosetta Diagnostics for performance check.
         dgrlinkstat                    - Rosetta Diagnostics for links statistics.
         dgrerrstat                     - Rosetta Diagnostics for error statistics.
         dgrheadshellstat               - Rosetta Diagnostics for Headshell statistics.
         dgrflowdebug                   - Rosetta Diagnostics for flow control debugging.
         dgrcounters                    - Rosetta Diagnostics for capturing counter data.
         check-fabric                   - Validates current state of a fabric with fabric template file.
         check-switches                 - Validates L1/L2 cabling with p2p file.
         fmn_status                     - Provides summary of fabric switch ports status.
         simple_discovery               - Finds connected Switch/Node/Chassis BMCs to the SMS/NCN.
         services_rosetta               - Provides summary of services running on Rosetta.
         services_platform              - Provides summary of services running on Switch platform.
         dgrvalidatesyscfg              - Provides health snapshot of switches
         linkdbg                        - Provides health snapshot of links
         fabric_snapshot                - Collects fabric information from the FMN.
         show-flaps                     - Provides link flapping information.
         compute_snapshot               - Collects Network Config Snapshot from the CNs.
         dgrcsr                         - Dumps Rosetta CSR data of switches
```

STT SHOW CABLES

```
(STT) show cables

Working with 'default' topology and 'default' filter profile.
Collecting data using 'check-switches' script.
Collecting data using 'check-fabric' script.
Warning: login credentials for compute nodes is not set in STT.
Use 'compute_nodes_creds' command to input compute node login credentials.
Trying to access compute nodes without password using SSH.

Collecting data using 'dgrlinkstat' script.
check-fabric   :  Start time: 07/13/2021, 04:47:30 , End time: 07/13/2021, 04:47:33
dgrlinkstat   :  Start time: 07/13/2021, 04:47:30 , End time: 07/13/2021, 04:47:40
check-switches   :  Start time: 07/13/2021, 04:47:30 , End time: 07/13/2021, 04:47:53
+-----------------+-----------------+------------------+------------------+--------+---------------+--------------------------------------------------+
|       srca      |       srcb      |       dsta       |       dstb       |  type  |     status    |                    serial_ids                    |
+-----------------+-----------------+------------------+------------------+--------+---------------+--------------------------------------------------+
|  x1000c0r3j10p1 |  x1000c0r3j10p0 |  x1000c5r7j11p1  |  x1000c5r7j11p0  | fabric |   Connected   |   0619190050, 0619190050, 0619190050, 0619190050  |
|  x1000c0r3j11p1 |  x1000c0r3j11p0 |  x1000c7r3j14p1  |  x1000c7r3j14p0  | fabric |   Connected   |   0828190034, 0828190034, 0828190034, 0828190034  |
|  x1000c0r3j12p1 |  x1000c0r3j12p0 |  x1000c7r7j11p1  |  x1000c7r7j11p0  | fabric |   Connected   |   0828190074, 0828190074, 0828190074, 0828190074  |
|  x1000c0r3j13p1 |  x1000c0r3j13p0 |  x1000c6r7j13p1  |  x1000c6r7j13p0  | fabric |   Connected   |   0828190024, 0828190024, 0828190024, 0828190024  |
|  x1000c0r3j14p1 |  x1000c0r3j14p0 |  x1000c6r3j13p1  |  x1000c6r3j13p0  | fabric |   Connected   |   0619190001, 0619190001, 0619190001, 0619190001  |
|  x1000c0r3j16p1 |  x1000c0r3j16p0 |  x1000c4r7j13p1  |  x1000c4r7j13p0  | fabric |   Connected   |   0619190051, 0619190051, 0619190051, 0619190051  |
|  x1000c0r3j18p1 |  x1000c0r3j18p0 |  x1000c4r3j14p1  |  x1000c4r3j14p0  | fabric |   Connected   |   0828190047, 0828190047, 0828190047, 0828190047  |
|  x1000c0r3j1p1  |  x1000c0r3j1p0  | x3000c0r42j31p1  | x3000c0r42j31p0  | fabric |   Connected   | UH294G00255, UH294G00255, UH294G00255, UH294G00255 |
|  x1000c0r3j20p1 |  x1000c0r3j20p0 |  x1000c2r7j14p1  |  x1000c2r7j14p0  | fabric |   Connected   |   0910190282, 0910190282, 0910190282, 0910190282  |
|  x1000c0r3j22p1 |  x1000c0r3j22p0 |  x1000c2r3j20p1  |  x1000c2r3j20p0  | fabric |   Connected   |   0612190337, 0612190337, 0612190337, 0612190337  |
|  x1000c0r3j24p1 |  x1000c0r3j24p0 |  x1000c0r7j24p1  |  x1000c0r7j24p0  | fabric |   Connected   |   0612190556, 0612190556, 0612190556, 0612190556  |
|  x1000c0r3j2p1  |  x1000c0r3j2p0  |  x1000c1r3j24p1  |  x1000c1r3j24p0  | fabric |   Connected   |   0527190006, 0527190006, 0527190006, 0527190006  |
|  x1000c0r3j4p1  |  x1000c0r3j4p0  |  x1000c1r7j20p1  |  x1000c1r7j20p0  | fabric |   Connected   |   0612190287, 0612190287, 0612190287, 0612190287  |
...
```

IMAGE MANAGEMENT

- Nexus repository manager
- Image Management Service

PACKAGE MANAGEMENT

- Package management is performed with the Sonatype Nexus Repository Manager, or simply Nexus
- Nexus resources are orchestrated by Kubernetes
  - By default, all Nexus resources are in the `nexus` namespace
  - Support for multiple repository formats
  - Support for multiple types of repositories
    - hosted
    - group
    - Proxy
- Primary access to nexus is through a web-based interface
  - https://nexus.EX_SHASTA_DOMAIN/
  - Command line interaction with Nexus is via Kubernetes `kubectl` commands and through api calls

```
ncn# kubectl get pods -A |grep nexus
nexus          cray-precache-images-q6ntx          1/1   Running   1   68d
nexus          cray-precache-images-sshkq          1/1   Running   1   68d
nexus          cray-precache-images-xmcth          1/1   Running   2   68d
nexus          nexus-868d7b8466-t4cnb              2/2   Running   0   57d
```

NEXUS REPOSITORIES INCLUDED BY DEFAULT

```
ncn# curl -s https://packages.local/service/rest/v1/repositories -H "Content-type: application/json" \
|jq 'map(select(.type == "hosted"))' | jq 'map(select(.format == "raw")) | .[].name' |sort
"HFP-firmware-2.0.101916-0"
"HFP-firmware-2.0.111516-0"
"SUSE-21.16.0-SLE-Module-Basesystem-15-SP2-x86_64-Debug"
"SUSE-21.16.0-SLE-Module-Basesystem-15-SP2-x86_64-PTF"
"SUSE-21.16.0-SLE-Module-Basesystem-15-SP2-x86_64-Updates"
"SUSE-21.16.0-SLE-Module-Containers-15-SP2-x86_64-Updates"
```

Nexus supports multiple repository formats including raw, yum, helm, and docker as shown below, note that the predominant format is “raw”

```
ncn# curl -s https://packages.local/service/rest/v1/repositories -H "Content-type: application/json" \
|jq 'map(select(.type == "hosted"))' | jq 'map(select(.format == "yum")) | .[].name'
"aocc-sle-15-cn"
ncn# curl -s https://packages.local/service/rest/v1/repositories -H "Content-type: application/json" \
|jq 'map(select(.type == "hosted"))' | jq 'map(select(.format == "helm")) | .[].name'
"charts"
ncn# curl -s https://packages.local/service/rest/v1/repositories -H "Content-type: application/json" \
|jq 'map(select(.type == "hosted"))' | jq 'map(select(.format == "docker")) | .[].name'
"registry"
```

NCN AND COMPUTE RPM REPOSITORIES

```
ncn# zypper lr -P
#  | Alias                                       | Name                                        | Enabled | GPG Check | Refresh | Priority
---+---------------------------------------------+---------------------------------------------+---------+-----------+---------+---------
 1 | SUSE-SLE-Module-Basesystem-15-SP2-x86_64-Pool    | SUSE-SLE-Module-Basesystem-15-SP2-x86_64-Pool    | Yes     | (  ) No   | Yes     |   99
 2 | SUSE-SLE-Module-Basesystem-15-SP2-x86_64-Updates | SUSE-SLE-Module-Basesystem-15-SP2-x86_64-Updates | Yes     | (  ) No   | Yes     |   99
 3 | SUSE-SLE-Module-Containers-15-SP2-x86_64-Pool    | SUSE-SLE-Module-Containers-15-SP2-x86_64-Pool    | Yes     | (  ) No   | Yes     |   99
 4 | SUSE-SLE-Module-Containers-15-SP2-x86_64-Updates | SUSE-SLE-Module-Containers-15-SP2-x86_64-Updates | Yes     | (  ) No   | Yes     |   99
 5 | SUSE-SLE-Module-HPC-15-SP2-x86_64-Pool           | SUSE-SLE-Module-HPC-15-SP2-x86_64-Pool           | Yes     | (  ) No   | Yes     |   99
 6 | SUSE-SLE-Module-HPC-15-SP2-x86_64-Updates        | SUSE-SLE-Module-HPC-15-SP2-x86_64-Updates        | Yes     | (  ) No   | Yes     |   99
 7 | cray-sdu-rda                                     | cray-sdu-rda                                     | Yes     | ( p) Yes  | Yes     |   99
 8 | csm-sle-15sp2                                    | CSM SLE 15 SP2 Packages (added by Ansible)       | Yes     | ( p) Yes  | Yes     |   99
 9 | sat-sle-15sp2                                    | sat-sle-15sp2                                    | Yes     | (  ) No   | Yes     |   99
10 | sma-sle-15sp2                                    | sma-sle-15sp2                                    | Yes     | (  ) No   | Yes     |   99

nid001004# zypper lr -P
#  | Alias                                                     | Name                                                      | Enabled | GPG Check | Refresh | Priority
---+-----------------------------------------------------------+-----------------------------------------------------------+---------+-----------+---------+---------
 1 | SUSE-Backports-SLE-15-SP2                                 | SUSE-Backports-SLE-15-SP2                                 | Yes     | ( p) Yes  | No      |    3
 2 | SUSE-SLE-Module-Basesystem-15-SP2-x86_64-PTF              | SUSE-SLE-Module-Basesystem-15-SP2-x86_64-PTF              | Yes     | ( p) Yes  | No      |    3
 3 | SUSE-SLE-Module-Basesystem-15-SP2-x86_64-Pool             | SUSE-SLE-Module-Basesystem-15-SP2-x86_64-Pool             | Yes     | (r ) Yes  | No      |    3
 4 | SUSE-SLE-Module-Basesystem-15-SP2-x86_64-Updates          | SUSE-SLE-Module-Basesystem-15-SP2-x86_64-Updates          | Yes     | (r ) Yes  | No      |    3
 5 | SUSE-SLE-Module-Desktop-Applications-15-SP2-x86_64-Pool   | SUSE-SLE-Module-Desktop-Applications-15-SP2-x86_64-Pool   | Yes     | (r ) Yes  | No      |    3
 6 | SUSE-SLE-Module-Desktop-Applications-15-SP2-x86_64-Updates | SUSE-SLE-Module-Desktop-Applications-15-SP2-x86_64-Updates | Yes     | (r ) Yes  | No      |    3
 7 | SUSE-SLE-Module-Development-Tools-15-SP2-x86_64-Pool      | SUSE-SLE-Module-Development-Tools-15-SP2-x86_64-Pool      | Yes     | (r ) Yes  | No      |    3
 8 | SUSE-SLE-Module-Development-Tools-15-SP2-x86_64-Updates   | SUSE-SLE-Module-Development-Tools-15-SP2-x86_64-Updates   | Yes     | (r ) Yes  | No      |    3
 9 | SUSE-SLE-Module-HPC-15-SP2-x86_64-Pool                    | SUSE-SLE-Module-HPC-15-SP2-x86_64-Pool                    | Yes     | (r ) Yes  | No      |    3
10 | SUSE-SLE-Module-HPC-15-SP2-x86_64-Updates                 | SUSE-SLE-Module-HPC-15-SP2-x86_64-Updates                 | Yes     | (r ) Yes  | No      |    3
11 | SUSE-SLE-Module-Legacy-15-SP2-x86_64-Pool                 | SUSE-SLE-Module-Legacy-15-SP2-x86_64-Pool                 | Yes     | (r ) Yes  | No      |    3
12 | SUSE-SLE-Module-Legacy-15-SP2-x86_64-Updates              | SUSE-SLE-Module-Legacy-15-SP2-x86_64-Updates              | Yes     | (r ) Yes  | No      |    3
13 | SUSE-SLE-Module-Public-Cloud-15-SP2                       | SUSE-SLE-Module-Public-Cloud-15-SP2                       | Yes     | (r ) Yes  | No      |    3
14 | SUSE-SLE-Module-Public-Cloud-15-SP2-x86_64-Updates        | SUSE-SLE-Module-Public-Cloud-15-SP2-x86_64-Updates        | Yes     | (r ) Yes  | No      |    3
15 | SUSE-SLE-Module-Python2-15-SP2-x86_64-Pool                | SUSE-SLE-Module-Python2-15-SP2-x86_64-Pool                | Yes     | (r ) Yes  | No      |    3
16 | SUSE-SLE-Module-Python2-15-SP2-x86_64-Updates             | SUSE-SLE-Module-Python2-15-SP2-x86_64-Updates             | Yes     | (r ) Yes  | No      |    3
17 | SUSE-SLE-Module-Server-Applications-15-SP2-x86_64-Pool    | SUSE-SLE-Module-Server-Applications-15-SP2-x86_64-Pool    | Yes     | (r ) Yes  | No      |    3
18 | SUSE-SLE-Module-Server-Applications-15-SP2-x86_64-Updates | SUSE-SLE-Module-Server-Applications-15-SP2-x86_64-Updates | Yes     | (r ) Yes  | No      |    3
…
nid00104# zypper lr -P | wc -l
33
```

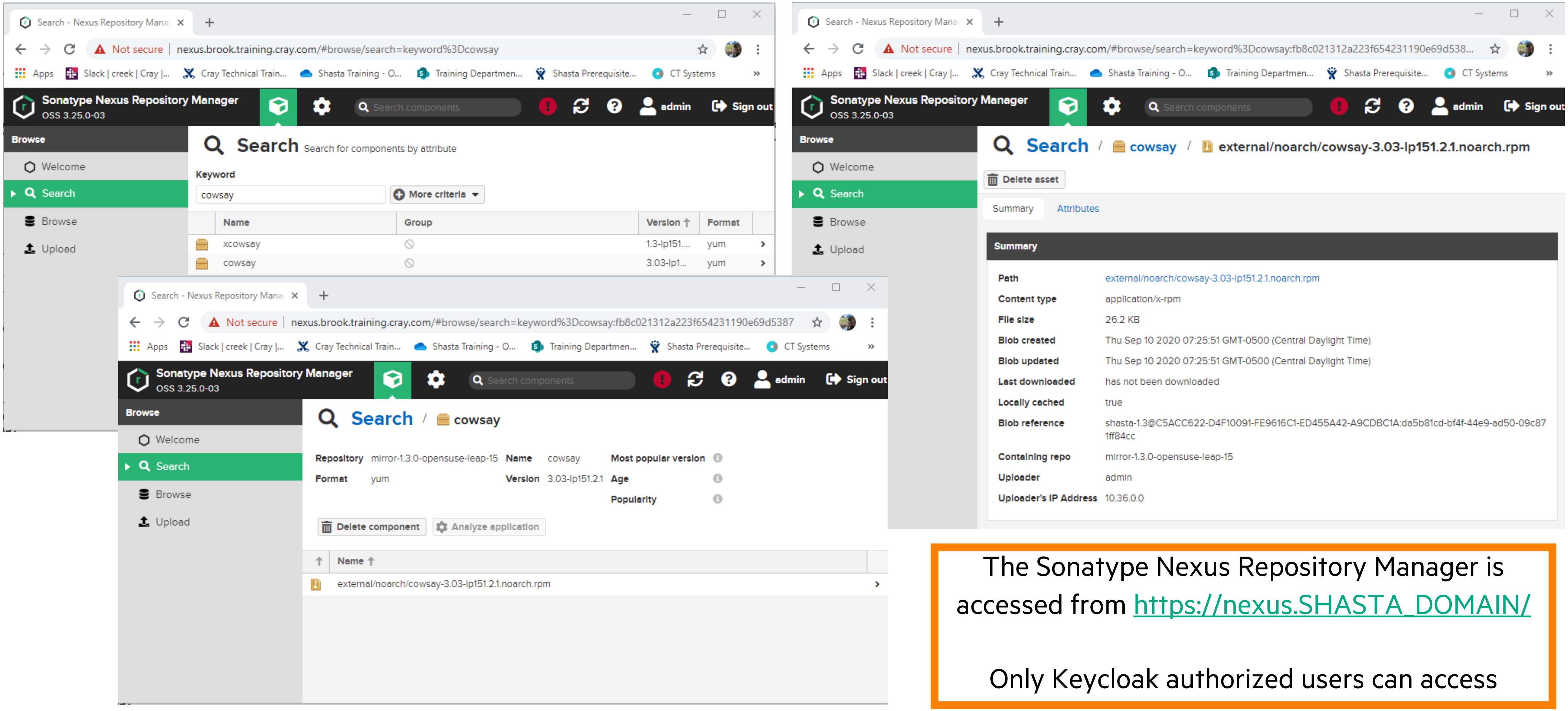
HOSTED VERSUS GROUP REPOSITORIES

```
ncn# curl -s https://packages.local/service/rest/v1/repositories \
-H "Content-type: application/json" |jq 'map(select(.name == "cray-sdu-rda"))'
[
  {
    "name": "cray-sdu-rda",
    "format": "raw",
    "type": "group",
    "url": "https://packages.local/repository/cray-sdu-rda",
    "attributes": {}
  }
]
ncn# curl -s https://packages.local/service/rest/v1/repositories \
-H "Content-type: application/json" |jq 'map(select(.name == "cray-sdu-rda-1.1.7"))'
[
  {
    "name": "cray-sdu-rda-1.1.7",
    "format": "raw",
    "type": "hosted",
    "url": "https://packages.local/repository/cray-sdu-rda-1.1.7",
    "attributes": {}
  }
]
ncn# zypper lr -up |grep -v ^- | awk -F "|" '{print $2 $8}' | grep sdu
 cray-sdu-rda                                 https://packages.local/repository/cray-sdu-rda
```

Group repositories combine hosted repositories into “meta repositories versioned by minor releases and used by `zypper`

Hosted repositories are versioned according the patch release

SONATYPE NEXUS REPOSITORY MANAGER



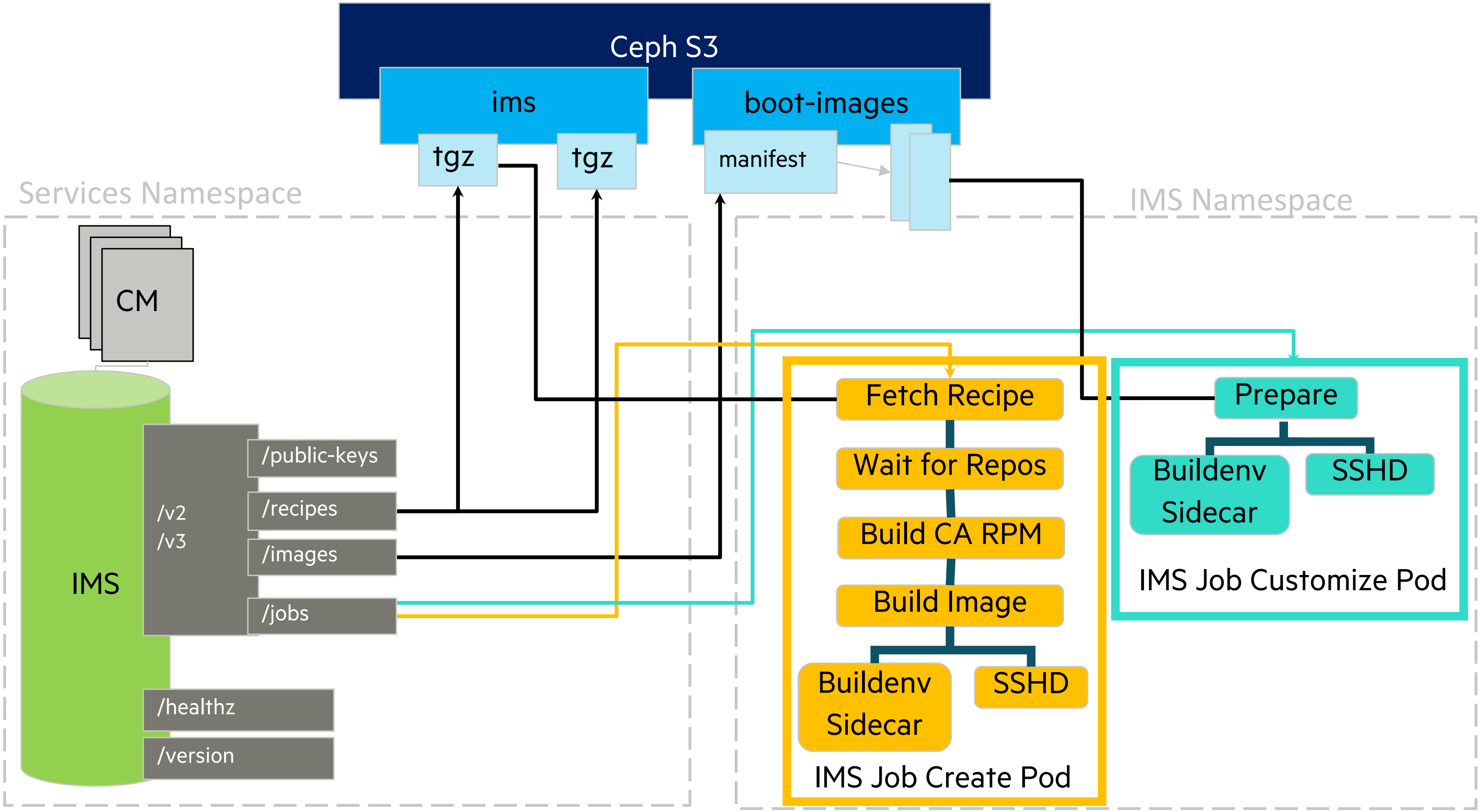
The Sonatype Nexus Repository Manager is accessed from https://nexus.SHASTA_DOMAIN/

Only Keycloak authorized users can access

IMAGE MANAGEMENT SERVICE (IMS)

- Allows administrators and authorized users to build or customize (pre-boot) images.
- IMS supports the following REST endpoints:
  - Public key management
    - Public keys to enable SSH access to debug and customize images
  - Recipe management
    - Recipes that can be used to build an image
  - Image and image artifact management
    - An image can consist of multiple image artifacts including the image root, kernel and `initrd`
  - Job management
    - The workflow to create or customize an image via a Kubernetes job

IMS API, BUCKETS, AND JOBS

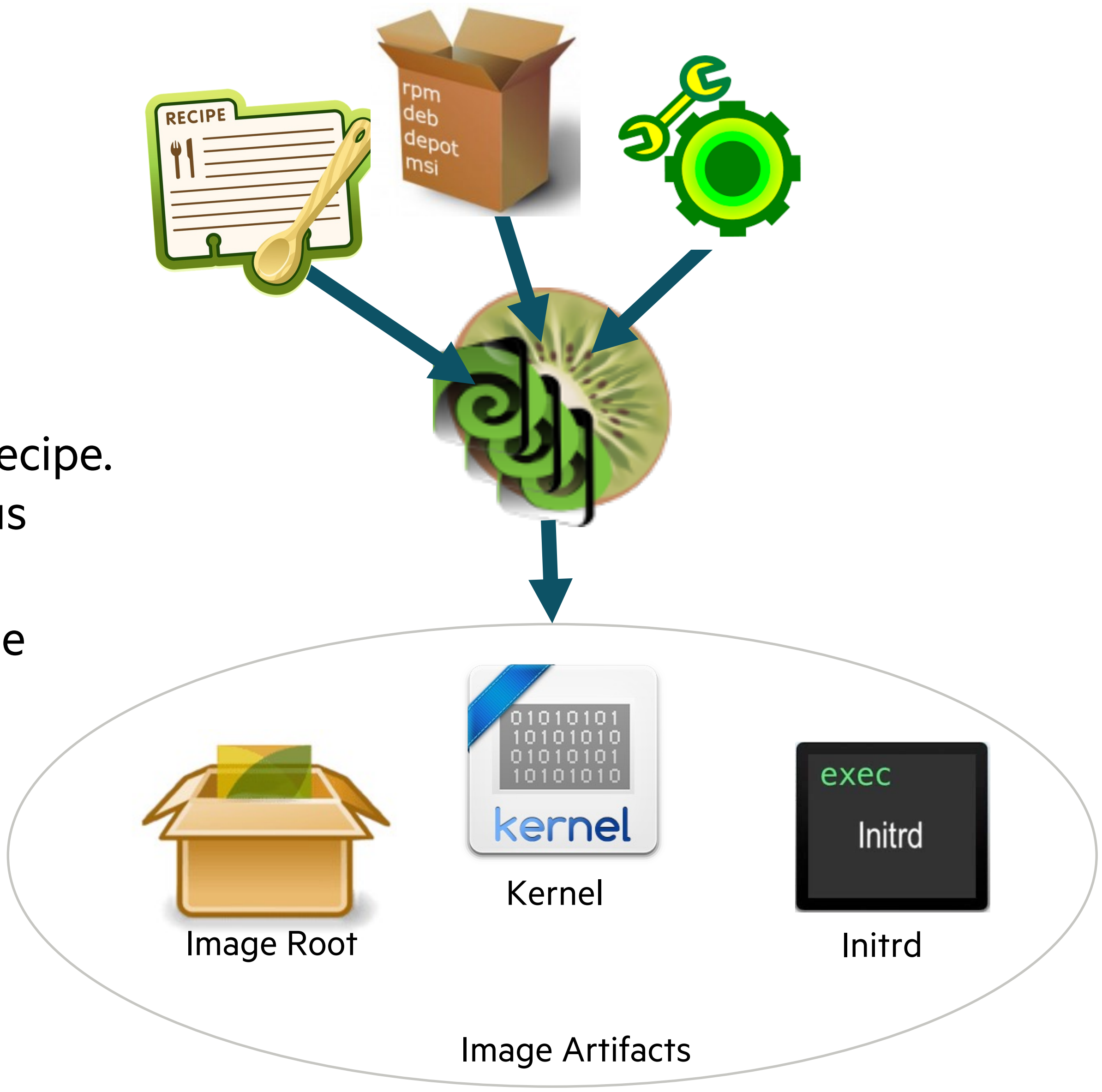


KIWI-NG

- IMS uses the open-source tool KIWI-NG to build images from Kiwi Image Descriptions
- KIWI-NG is the next generation (or updated version) of the Kiwi Appliance Builder
  - An appliance is just a ready to use image for an operating system
  - Kiwi can create images that boot via PXE
- Kiwi supports building images of various Linux distributions
  - Currently supported, SLES 15
- Image Description
  - Specification to define an appliance
  - The image description is a collection of human readable files in a directory
  - The contents of the Image Description (“*recipe*”) are archived and stored in S3
    - – The artifact ID of the recipe is stored in IMS
- Image
  - The result of a KIWI build process
  - Consists of the kernel, `initrd`, image root, and possibly other artifacts
  - Image artifacts are stored in S3 with a link to the artifact in IMS

CREATING AN IMAGE

1. Admin submits a “create job” to IMS
2. IMS establishes a new Kubernetes pod to build the image
3. The recipe is downloaded from S3 and passed to KIWI-NG running in the new pod
4. KIWI-NG installs the RPM packages listed in the recipe. RPMs are retrieved from repos setup by the Nexus Repository Manager
5. KIWI-NG runs configuration scripts specified in the recipe to the image
6. When KIWI-NG completes, the image artifacts are collected and stored in S3



LISTING IMS RECIPES

```
ncn# cray ims recipes list | grep name |sort
    "name": "cpe-barebones-sles15sp2.x86_64-21.11.7",
    "name": "cray-shasta-compute-sles15sp2.x86_64-1.5.63",
    "name": "cray-shasta-compute-sles15sp2.x86_64-1.5.66",
    "name": "cray-shasta-uan-cos-sles15sp2.x86_64-0.2.35",
    "name": "cray-shasta-uan-cos-sles15sp2.x86_64-0.2.37",
    "name": "cray-shasta-uan-cos-sles15sp2.x86_64-0.2.39",
ncn# cray ims recipes list --format yaml | grep -B7 -A 1 cray-shasta-compute-sles15sp2.x86_64-1.5.66
- created: '2021-12-14T22:15:27.662675+00:00'
  id: 667b38e5-2b09-429d-b09a-f848fcb48d84
  link:
    etag: 5afae742dab16ef9f549ab03f1747962
    path: s3://ims/recipes/667b38e5-2b09-429d-b09a-f848fcb48d84/recipe.tar.gz
    type: s3
  linux_distribution: sles15
  name: cray-shasta-compute-sles15sp2.x86_64-1.5.66
  recipe_type: kiwi-ng
```

Latest versions

DOWNLOADING AN IMAGE RECIPE

```
ncn# cray ims recipes describe 667b38e5-2b09-429d-b09a-f848fcb48d84
{
  "created": "2021-12-14T22:15:27.662675+00:00",
  "id": "667b38e5-2b09-429d-b09a-f848fcb48d84",
  "link": {
    "etag": "5afae742dab16ef9f549ab03f1747962",
    "path": "s3://ims/recipes/667b38e5-2b09-429d-b09a-f848fcb48d84/recipe.tar.gz",
    "type": "s3"
  },
  "linux_distribution": "sles15",
  "name": "cray-shasta-compute-sles15sp2.x86_64-1.5.66",
  "recipe_type": "kiwi-ng"
}

ncn# cray artifacts get ims recipes/667b38e5-2b09-429d-b09a-f848fcb48d84/recipe.tar.gz cray-
shasta-compute-sles15sp2.x86_64-1.5.66.tar.gz
ncn# file cray-shasta-compute-sles15sp2.x86_64-1.5.66.tar.gz
cray-shasta-compute-sles15sp2.x86_64-1.5.66.tar.gz: gzip compressed data, from Unix
ncn# ls -l cray-shasta-compute-sles15sp2.x86_64-1.5.66.tar.gz
-rw-r--r-- 1 root root 7413 Jan  5 15:11 cray-shasta-compute-sles15sp2.x86_64-1.5.66.tar.gz
```

EXTRACTING AN IMAGE RECIPE

```
ncn# tar -xvf cray-shasta-compute-sles15sp2.x86_64-1.5.66.tar.gz
./
./config.sh
./images.sh
./root/
./root/root/
./root/root/bin/
./root/root/bin/zypper-addrepo.sh
./config.xml
```

- **config.xml** - Image definition file provides *image name* and *type* and the packages and patterns to make the image
- **images.sh** - Optional script called at the beginning of the image creation process
- **config.sh** - Optional script called at the end of the installation while in chroot, but before package scripts have run
- **root/** - Subdirectory that contains special files, directories, and scripts for adapting the image environment after the installation of all the image packages. The entire directory is copied into the root of the image tree using `cp -a`
- **config/** - Optional subdirectory with Bash scripts called after the installation of all the image packages to remove the parts of a package that are not needed

CONFIG.XML

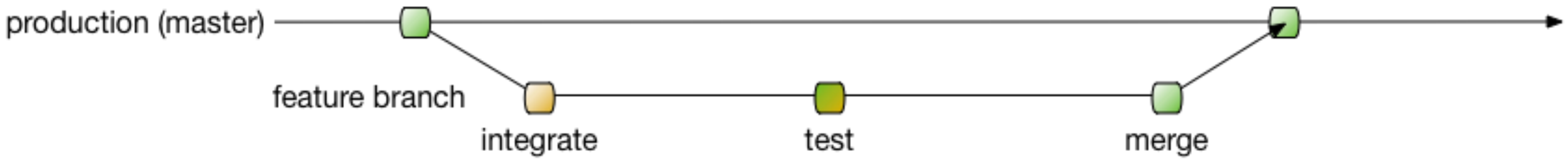
- The config.xml file consist of the following elements or tags
  - **image**
    - The top-level tag for the image description provides the name of the image and the XML schema
  - **description**
    - This tag provides information on the `author` of the image description and some additional information
  - **preferences**
    - This tag contains information about the supported image type(s), the package manager used, the version of this image, and optional attributes
    - Each `preferences` block must define at least one *`type`* element. Multiple *`type`* elements can be specified in any `preferences` block. The image *`type`* to be created is determined by the value of the image attribute
  - **users**
    - This tag allows for the creation of users within the image. Each child tag defines one user
  - **repository**
    - Specifies the location and type of a repository to be used by the package manager
    - Each repository tag can include a `priority` attribute. The Zypper package manager prefers packages from a repository with a lower priority over packages from a repository with higher priority values
  - **packages**
    - This tag specifies the list of packages to be used with the image.
    - The value of the type attribute specifies how the packages and patterns listed are handled
- Tags in purple are mandatory

CONFIGURATION MANAGEMENT

- Version Control Service
- Configuration Framework Service

CONFIGURATION WITH CFS AND VCS

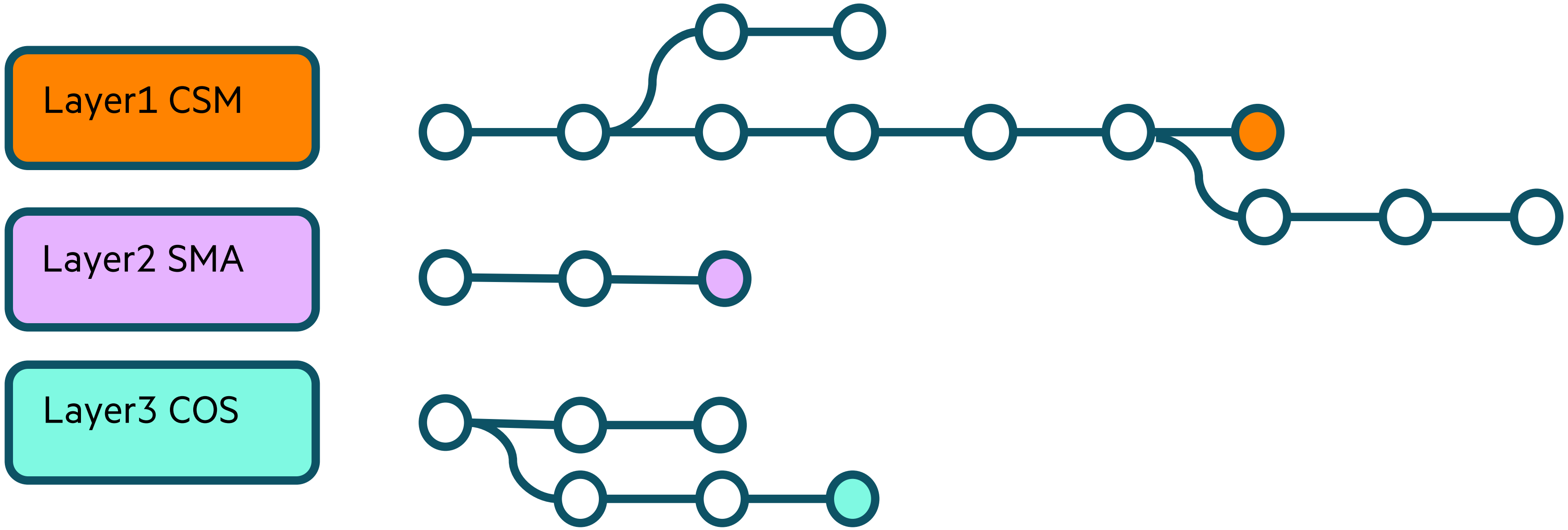
- Version Control Service (VCS)
  - Manages configuration data and content
    - Compute image configuration YAML files
  - *Gitea* server holds configuration content



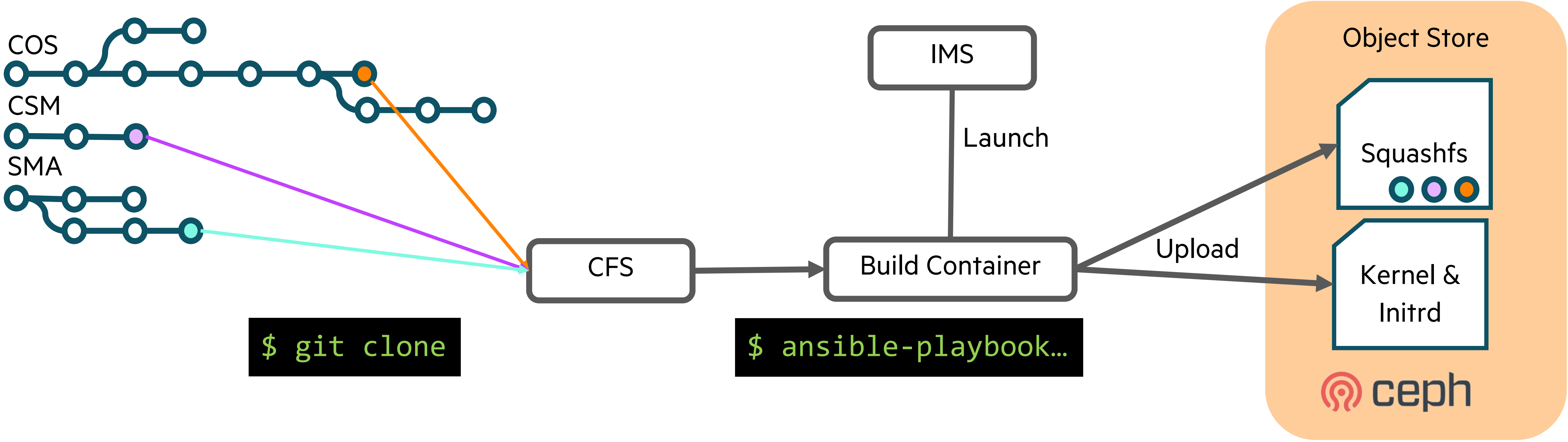
- Configuration Framework Service (CFS)
  - Manages the launch of configuration actions
  - Does `git-clone` of configuration data and content from VCS
  - Launches Ansible Execution Environment (AEE) which runs Ansible playbook for target inventory
    - Either hostnames of nodes for node personalization or reconfiguration
    - Or IMS build environment for image customization
  - Aggregates status to show how many targets passed/failed the Ansible run

USING GIT FOR MANAGING CFS CONFIGURATION

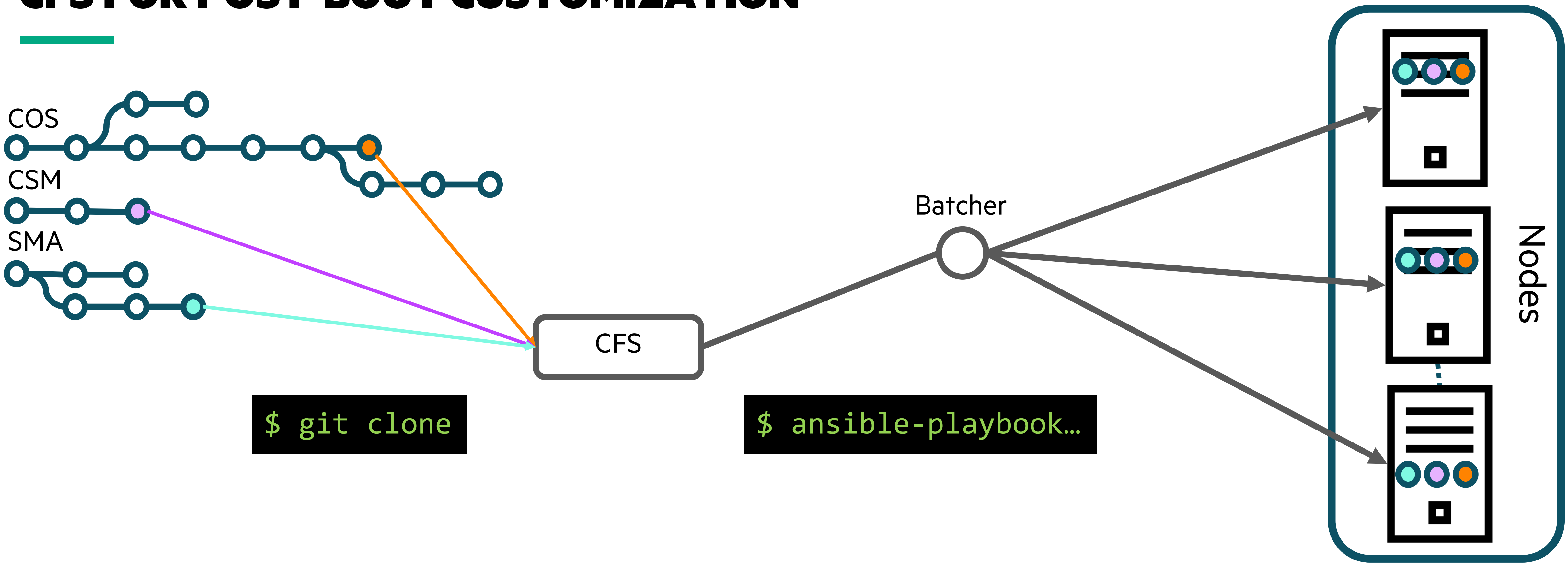
- Stores Ansible to apply to nodes at lifecycle events
- All Ansible in git repositories with branches to allow site customization
- Ordered configuration management across multiple repositories
- CFS sessions as part of pre-boot Image Customization as well as post-boot Node Personalization



CFS FOR IMAGE CUSTOMIZATION



CFS FOR POST-BOOT CUSTOMIZATION



LIST GIT REPOSITORIES

```
ncn# kubectl get cm -n services cray-product-catalog -o json| \
jq -r '.data' | sed 's/\\n/\n/g' | grep clone_url | sort -u
    clone_url: https://vcs.creek.training.hpe.com/vcs/cray/analytics-config-management.git
    clone_url: https://vcs.creek.training.hpe.com/vcs/cray/cos-config-management.git
    clone_url: https://vcs.creek.training.hpe.com/vcs/cray/cpe-config-management.git
    clone_url: https://vcs.creek.training.hpe.com/vcs/cray/csm-config-management.git
    clone_url: https://vcs.creek.training.hpe.com/vcs/cray/sat-config-management.git
    clone_url: https://vcs.creek.training.hpe.com/vcs/cray/slurm-config-management.git
    clone_url: https://vcs.creek.training.hpe.com/vcs/cray/sma-config-management.git
    clone_url: https://vcs.creek.training.hpe.com/vcs/cray/uan-config-management.git
```

Retrieve the clone urls of the git repositories from Kubernetes

```
ncn# VCSPWD=$(kubectl get secret -n services vcs-user-credentials \
    --template={{.data.vcs_password}} | base64 --decode)
```

Retrieve the vcsuser password from kubernetes

```
ncn# # curl -s -u crayvcs:${VCSPWD} -X 'GET' 'https://api-gw-service-
nmn.local/vcs/api/v1/repos/search' | jq -r .data[].name
analytics-config-management
cos-config-management
cpe-config-management
csm-config-management
sat-config-management
slurm-config-management
sma-config-management
uan-config-management
```

This command retrieves the repository names directly from git

PRODUCT RELEASE DETAILS

```
ncn# kubectl get cm -n services cray-product-catalog -o json | jq -r '.data.cos'  | sed 's/\\n/\n/g'
2.2.101:
  configuration:
    clone_url: https://vcs.groot.dev.cray.com/vcs/cray/cos-config-management.git
    commit: a736bc12032330d5236456f1cec207a431620098
    import_branch: cray/cos/2.2.101
    import_date: 2022-03-16 18:34:23.475935
    ssh_url: git@vcs.groot.dev.cray.com:cray/cos-config-management.git
  images:
    cray-shasta-compute-sles15sp3.x86_64-2.2.38:
      id: 30068b70-9244-46e6-90d7-6d6000fe6339
  recipes:
    cray-shasta-compute-sles15sp3.x86_64-2.2.38:
      id: c4cba248-51c4-45d5-af3c-ea1f4856ce67
2.2.76:
  configuration:
    clone_url: https://vcs.groot.dev.cray.com/vcs/cray/cos-config-management.git
    commit: 6e18c471782b2c3d460e4edfc7e349eb71620540
    import_branch: cray/cos/2.2.76
    import_date: 2022-02-11 18:57:45.134106
    ssh_url: git@vcs.groot.dev.cray.com:cray/cos-config-management.git
  images:
    cray-shasta-compute-sles15sp3.x86_64-2.2.29:
      id: 982efd61-fe63-4b14-90dc-93805437f75c
  recipes:
    cray-shasta-compute-sles15sp3.x86_64-2.2.29:
      id: 1292a468-2978-4db5-a157-f5e6c41b6d5f
```

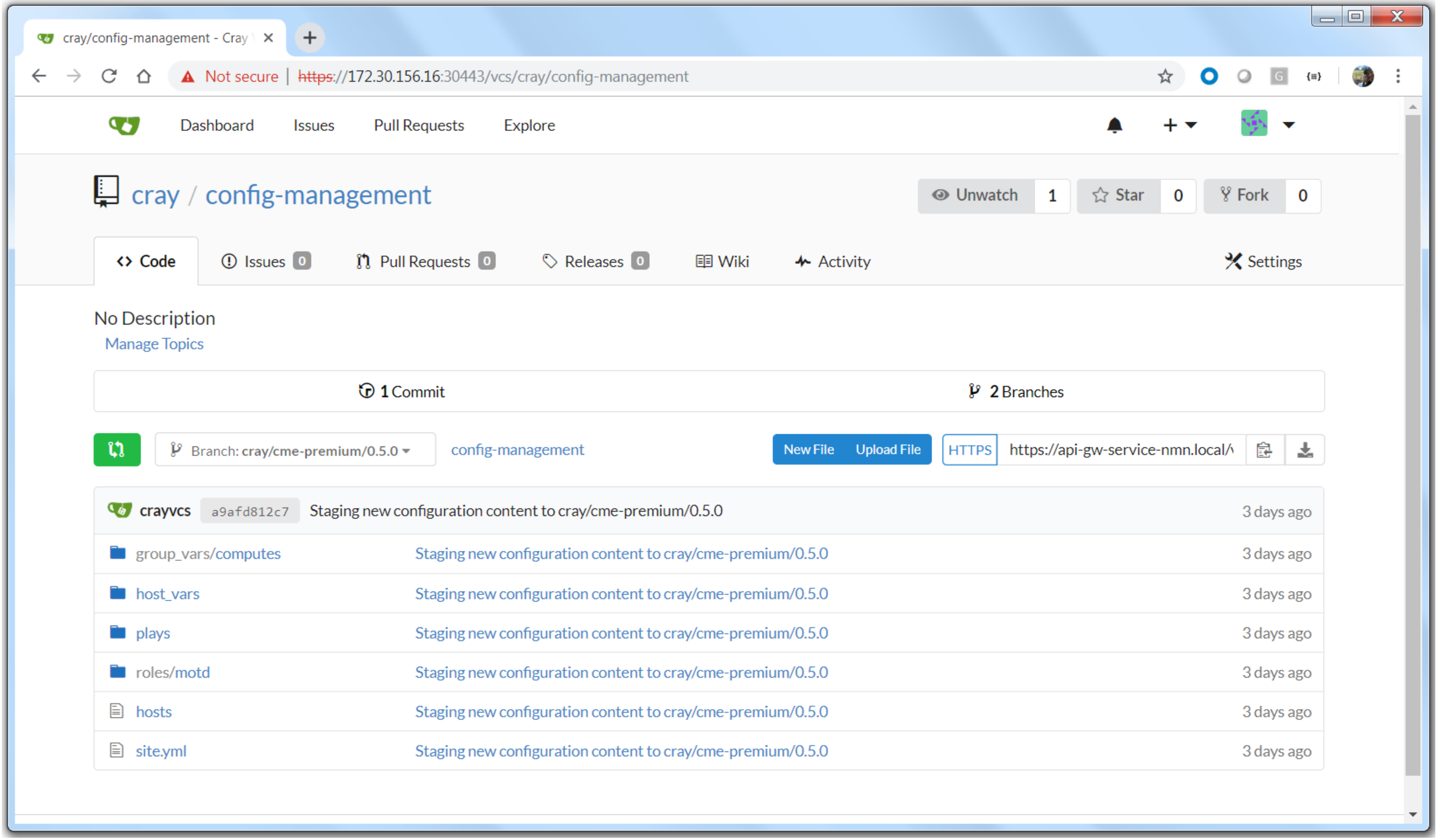
Available releases:
- cos
- cpe
- csm
- hfp
- pbs
- sat
- sdu
- slingshot
- slurm
- sma
- uan

This command provides: git branches, images, and recipes included with each product release

GIT COMMANDS

Area	Git CLI command	Description
Setup & Init	git init <project name>	Create a new project and local repository
	git clone <url>	Download a project
Stage & Snapshot	git status	List all new or modified files to be committed
	git add <file name>	Stages files for version control
	git reset <file name>	Unstages a file
	git diff	Shows file differences that are not yet staged for version control
	git diff <branch 1> <branch 2>	Show the differences between two branches
	git commit –m <commit message>	Records all added files in version history
Branch & Merge	git branch	List all local branches
	git branch <branch name>	Create a new branch
	git checkout <branch name>	Switche to the specified branch and updates the working directory
	git merge <branch name>	Combines the specified branch history into the current branch
	git log	Show all commits in the current branch history
Share & Update	git push	Push local committed changes back to remote git repository
	git pull	Pull content from remote git repository

VCS WEB PORTAL



SAMPLE GIT SEQUENCE

```
ncn# git clone https://api-gw-service-nmn.local/vcs/cray/uan-config-management.git
Username for 'https://api-gw-service-nmn.local': crayvcs
Password for 'https://crayvcs@api-gw-service-nmn.local':
```

Checkout from VCS

```
ncn# cd uan-config-management
ncn# git checkout cray/uan/2.3.2
ncn# git pull
ncn# git checkout -b integration && git merge cray/uan/2.3.2
```

Make local branch

```
ncn# vi <file(s) to be edited>
ncn# git mv <file(s) to be renamed>
ncn# git rm <file(s) to be removed>
```

Change something

```
ncn# git status
ncn# git diff
ncn# git add <file(s) in repo that were added or edited>
ncn# git status
```

Compare and Update

```
ncn# git commit -m "<some message about the change>"
```

Describe change

```
ncn# git push --set-upstream origin integration
```

Push changes to git

```
ncn# git rev-parse --verify HEAD
ecece54b1eb65d484444c4a5ca0b244b329f4667
```

Git commit ID to be used on CFS layer

CONFIGURATION FRAMEWORK SERVICE

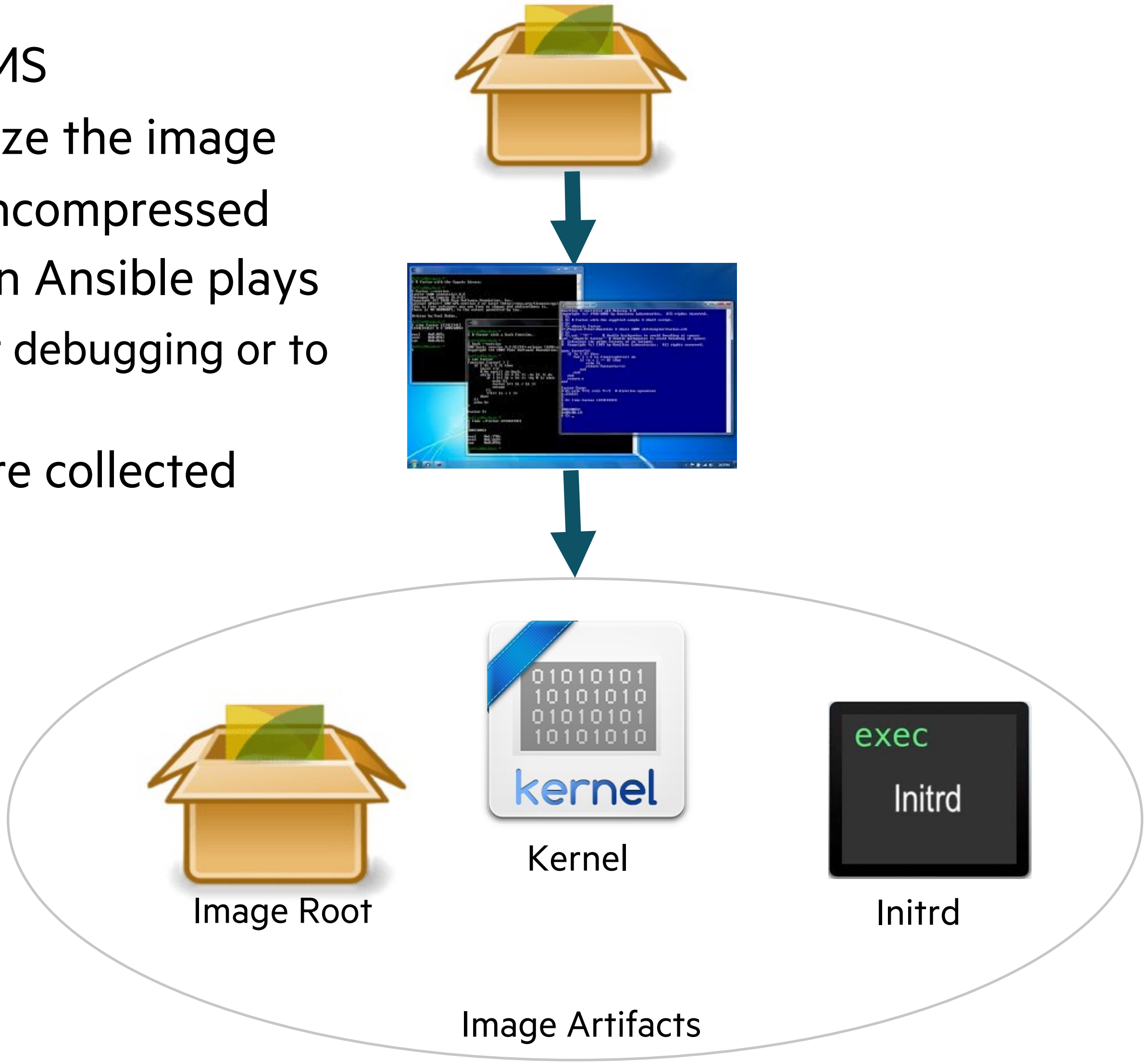
- Provides a configuration framework for HPE and customers which integrates industry-standard configuration management tooling (Ansible) with Cray services
- Flexible workflow
  - Pre-boot image customization
  - Post-boot node personalization
- Provides dynamic inventory plugins to target Cray nodes for configuration
- CFS is integrated with other Cray Management Services:
  - Image Management Service (IMS)
  - Nexus Repository Manager
  - Version Control Service (VCS)
  - Boot Orchestration Service (BOS)
  - Artifact Repository / S3
- Configurations are applied in layers
- Configurations are processed in batches

CONFIGURATION OPTIONS

- Image customization options (pre-boot)
  - IMS via manual SSH configuration environment
  - IMS via automatic Ansible plays in SSH configuration environment
- Node personalization options (post-boot)
  - Node personalization via Ansible plays on booted node
  - Node personalization via manual configuration
  - Live update (post-boot) via Zypper/Yum updates to RPM on booted node
- Reconfiguration of node (without rebooting)
  - Same methods as node personalization
- Any customer-provided methods for image customization, node personalization, or reconfiguration
- With each option a `cfs` configuration must be specified

CUSTOMIZING AN IMAGE

1. The Administrator submits a "*customize job*" to IMS
2. IMS establishes a new Kubernetes pod to customize the image
3. The existing image is downloaded from S3 and uncompressed
4. An SSH environment is established so CFS can run Ansible plays
   1. Or an administrator could request manual access for debugging or to make any required manual changes
5. When configuration is done, the image artifacts are collected and stored in S3 as new artifacts



CFS CONFIGURATIONS

```
ncn# cray cfs configurations describe compute-slurm-cpe-21.6.5 --format json
{
  "lastUpdated": "2021-06-24T18:58:25Z",
  "layers": [
    {
      "cloneUrl": "https://api-gw-service-nmn.local/vcs/cray/cos-config-management.git",
      "commit": "97209cb3e6c128e0b8c1eaae0e683227c57910ee",
      "name": "cos-integration-2.1.70",
      "playbook": "site.yml"
    },
    {
      "cloneUrl": "https://api-gw-service-nmn.local/vcs/cray/slurm-config-management.git",
      "commit": "b302e1b672e27f74c36ceacfd2ed6bd50ed14c0a",
      "name": "slurm-integration-0.1.3",
      "playbook": "site.yml"
    },
    {
      "cloneUrl": "https://api-gw-service-nmn.local/vcs/cray/cpe-config-management.git",
      "commit": "43f3a36bca35d693a583d1643fe1cebb0ccaf7fe",
      "name": "cpe-integration-21.6.5",
      "playbook": "pe_deploy.yml"
    }
  ],
  "name": " compute-slurm-cpe-21.6.5 "
}
```

CFS COMPONENTS

```
ncn# cray cfs components describe x1000c0s5b0n1 --format json
{
  "configurationStatus": "configured",
  "desiredConfig": " compute-slurm-cpe-21.6.5 ",
  "enabled": true,
  "errorCount": 0,
  "id": "x1000c0s5b0n1",
  "retryPolicy": 3,
  "state": [
    {
      "cloneUrl": "https://api-gw-service-nmn.local/vcs/cray/cos-config-management.git",
      "commit": " 97209cb3e6c128e0b8c1eaae0e683227c57910ee",
      "lastUpdated": "2021-11-17T18:44:41Z",
      "playbook": "site.yml",
      "sessionName": "batcher-f80ebbdb-c4ec-4025-8156-68205b22ccdf"
    },
    {
      "cloneUrl": "https://api-gw-service-nmn.local/vcs/cray/slurm-config-management.git",
      "commit": " b302e1b672e27f74c36ceacfd2ed6bd50ed14c0a",
      "lastUpdated": "2021-11-17T19:47:29Z",
      "playbook": "site.yml",
      "sessionName": "batcher-b57c437f-33e9-46d7-9416-8c955f773504"
    },
<< snip >>
    {
      "cloneUrl": "https://api-gw-service-nmn.local/vcs/cray/cpe-config-management.git",
      "commit": " 43f3a36bca35d693a583d1643fe1cebb0ccaf7fe",
      "lastUpdated": "2021-12-06T20:42:02Z",
      "playbook": "pe_deploy.yml",
      "sessionName": "batcher-bdea16db-dae5-4f7a-bffe-40f0a179d328"
    }
  ],
  "tags": {
    "bos_session": "d5f69110-dca6-4ecb-890f-3622957589fe"
```

The configuration for a component and whether it is enabled are set by BOS according to the `sessiontemplate`

If configuration fails it will be automatically retried up to the number specified in the `retryPolicy`

To see configuration (ansible) output check the cfs sessions to find configuration jobs and then check the logs of the ansible-x pods within those jobs.

CFS SESSIONS

```
ncn# cray cfs sessions describe batcher-080ba574-0a99-409b-a639-a45c73c25e63 --format json
{
  "ansible": {
    "config": "cfs-default-ansible-cfg",
    "limit": "x3000c0s26b0n0",
    "verbosity": 0
  },
  "configuration": {
    "limit": "",
    "name": "uan-config-2.0.0"
  },
  "name": "batcher-080ba574-0a99-409b-a639-a45c73c25e63",
  "status": {
    "artifacts": [],
    "session": {
      "completionTime": "2021-10-18T20:34:18",
      "job": "cfs-e78738d3-99a9-4b73-bce1-a720b34a714d",
      "startTime": "2021-10-18T20:31:15",
      "status": "complete",
      "succeeded": "true"
    }
  },
  "tags": {
    "bos_session": "bf88ad75-6a02-470c-85ca-4708a7f9fe0d"
  },
  "target": {
    "definition": "dynamic",
    "groups": null
  }
}
ncn# kubectl logs -n services cfs-e78738d3-99a9-4b73-bce1-a720b34a714d-ps4ls
error: a container name must be specified for pod cfs-e78738d3-99a9-4b73-bce1-a720b34a714d-ps4ls, choose one of: [inventory ansible-0 ansible-1
ansible-2 istio-proxy] or one of the init containers: [git-clone-0 git-clone-1 git-clone-2 istio-init]
```

The limit shows which node(s) are configured by each session

Kubernetes jobs control one or more pods and the job name is typically the start of the pod name

Each layer will be executed by a different container within the cfs job or possibly a different job

The containers names will have the format ansible-N (e.g., ansible-0)

CFS-BATCHER SCHEDULING RULES

- Every 10 seconds the batcher checks for components that need configuration
- Components (nodes) are assigned to a batch if:
  - They need configuration
  - They are not disabled
  - They are currently not assigned to a batch
- Components are grouped according to their desired state information.
- A new batch is created if
  - no partial batches match the desired state
  - all similar batches are full
- Batches are scheduled as CFS sessions when either
  - The batch is full
  - The batch window time has been exceeded

```
ncn# cray cfs options list --format json
{
  "additionalInventoryUrl": "",
  "batchSize": 25,
  "batchWindow": 60,
  "batcherCheckInterval": 10,
  "defaultAnsibleConfig": "cfs-default-ansible-
cfg",
  "defaultBatcherRetryPolicy": 1,
  "defaultPlaybook": "site.yml",
  "hardwareSyncInterval": 10,
  "sessionTTL": "7d"
}
```

WHY ISN'T CFS RUNNING?

```
ncn-m001:~ # kubectl logs -n services cray-cfs-batcher-5d58b8964c-tdsm2 -c cray-cfs-batcher
2021-09-16 09:19:54,225 - INFO    - batcher.batch - Successfully submited 1 batches for configuration
2021-09-16 09:20:54,910 - INFO    - batcher.batch - 1 batches/sessions have completed
2021-09-16 09:21:15,163 - INFO    - batcher.batch - Successfully submited 1 batches for configuration
2021-09-16 09:21:25,250 - INFO    - batcher.batch - 1 batches/sessions have completed
2021-09-16 09:22:15,759 - INFO    - batcher.batch - Successfully submited 1 batches for configuration
2021-09-16 09:23:26,546 - INFO    - batcher.batch - 1 batches/sessions have completed
2021-09-16 09:23:26,547 - WARNING - batcher.batch - The 20 most recent configuration sessions have failed. Halting session creation for 60 seconds
2021-09-16 09:24:27,136 - INFO    - batcher.batch - 1 batches/sessions have completed
2021-09-16 09:24:27,136 - WARNING - batcher.batch - The 20 most recent configuration sessions have failed. Halting session creation for 120 seconds
2021-09-16 09:26:28,170 - INFO    - batcher.batch - Successfully submited 2 batches for configuration
2021-09-16 09:27:49,098 - INFO    - batcher.batch - Successfully submited 1 batches for configuration
2021-09-16 09:28:49,865 - INFO    - batcher.batch - 2 batches/sessions have completed
2021-09-16 09:28:49,866 - WARNING - batcher.batch - The 20 most recent configuration sessions have failed. Halting session creation for 240 seconds
2021-09-16 09:29:50,468 - INFO    - batcher.batch - 1 batches/sessions have completed
2021-09-16 09:32:52,036 - INFO    - batcher.batch - Successfully submited 2 batches for configuration
2021-09-16 09:34:53,393 - INFO    - batcher.batch - 2 batches/sessions have completed
2021-09-16 09:34:53,393 - WARNING - batcher.batch - The 20 most recent configuration sessions have failed. Halting session creation for 480 seconds
2021-09-16 09:42:57,206 - INFO    - batcher.batch - Successfully submited 1 batches for configuration
2021-09-16 09:44:28,008 - INFO    - batcher.batch - 1 batches/sessions have completed
2021-09-16 09:44:28,008 - WARNING - batcher.batch - The 20 most recent configuration sessions have failed. Halting session creation for 960 seconds
2021-09-16 10:00:35,565 - INFO    - batcher.batch - Successfully submited 1 batches for configuration
2021-09-16 10:00:45,689 - INFO    - batcher.batch - Successfully submited 1 batches for configuration
2021-09-16 10:02:26,775 - INFO    - batcher.batch - 2 batches/sessions have completed
2021-09-16 10:02:26,775 - INFO    - batcher.batch - A session has succeeded.  Resuming normal operations
```

CFS has implemented a crash loop back off style behavior to avoid creating an infinite number of failed configuration sessions

If the last 20 CFS session have failed, then it will pause increasing intervals to allow the problems to be corrected

ANSIBLE PRIMER

WRITE ANSIBLE CODE FOR CFS

- CFS uses Ansible for configuration management
  - Create a configuration with one or more layers within a specific VCS git repository, and commit it to be executed by Ansible
  - Target a node, boot image, or group of nodes to apply the configuration
  - Create a configuration session to apply and track the status of Ansible, applying each configuration layer to the targets specified in the session metadata
- VCS is populated during software installation with Ansible code to configure each product
- Customers can write their own Ansible plays and roles to augment CFS configuration or implement new features
  - Ansible playbook best practices
    - https://docs.ansible.com/ansible/latest/user_guide/playbooks_best_practices.html
  - Ansible Examples
    - https://github.com/ansible/ansible-examples

ANSIBLE – TERMS

- Playbook
  - One or more plays
- Play
  - Maps groups of hosts to tasks
- Task
  - Sequence of actions performed against group of hosts that match a pattern in the play
- Modules
  - Large Ansible library of common code
    - Manage basic system resources
    - Send notifications
- Roles
  - Abstraction for naming a group of things that perform same function
- Separate code from data
  - Jinja2 templates (code)
  - Variables (data)
- Jinja2
  - Python-based template engine
  - Templates have placeholders for parameter values which can be replaced with variables
- Data
  - Facts
    - Automatically available
    - Discovered at run time
  - Variables
    - User-defined

ANSIBLE CODE STRUCTURE

- Each repository directory matches Ansible documentation
  - https://docs.ansible.com/ansible/2.9/user_guide/playbooks_best_practices.html#content-organization
  - The default playbook site.yml is found at the top level, if it exists
  - Ansible roles and variables are in their appropriately named directories
  - Inventory directories like `group_vars` and `host_vars` may exist, but they are empty and left for variable overrides and customizations as needed by the customer

```
group_vars/
  group1.yml # here we assign variables to particular groups
  group2.yml
host_vars/
  hostname1.yml # here we assign variables to particular nodes
  hostname2.yml
site.yml # master playbook
roles/
  common/ # this hierarchy represents a "role"
    tasks/ #
      main.yml # <-- tasks file can include smaller files if warranted
    handlers/ #
      main.yml # <-- handlers file
    templates/ # <-- files for use with the template resource
      ntp.conf.j2 # <------- templates end in .j2
    files/ #
      bar.txt # <-- files for use with the copy resource
      foo.sh # <-- script files for use with the script resource
    vars/ #
      main.yml # <-- variables associated with this role
    defaults/ #
      main.yml # <-- default lower priority variables for this role
    meta/ #
      main.yml # <-- role dependencies
    library/ # roles can also include custom modules
    module_utils/ # roles can also include custom module_utils
    lookup_plugins/ # or other types of plugins, like lookup in this case
  fooapp/ # "" same kind of structure as "common" was above but for fooapp
```

ANSIBLE – BEST PRACTICES FOR PLAYBOOKS/ROLES

- Ansible expects that all tasks are idempotent
  - (action performed only once, even if play is run more than once)
  - Care should be taken to ensure that tasks prescribe the desired state of the running system, making changes only when necessary
  - See “Resource Model” at https://docs.ansible.com/ansible/latest/reference_appendices/glossary.html
- When modifying files on a running system
  - Keep in mind that other services may access the file
  - Take the appropriate measures to ensure the modifications do not interfere with other operations
  - Leave a breadcrumb that the file is updated by an automated process
    - The “insertbefore” or “insertafter” options in the Ansible “lineinfile” module are well-suited to help with this
- If you find that you are trying to do something that is difficult to achieve in a few simple steps
  - It is likely that Ansible already has a module that provides the functionality

WRITE PLAYBOOKS FOR MULTIPLE NODE TYPES

- Ansible playbook can designate which node groups the various tasks and roles will run against
  - This is designated using the `hosts` parameter
  - Users can create additional sections that target other node types, or adjust the hosts that the included roles will run against
  - Can target multiple groups within a section of a playbook or specify complex targets, such as nodes that are in one group and not in another group
    - https://docs.ansible.com/ansible/latest/user_guide/intro_patterns.html#common-patterns
  - Hosts can be in more than one group at a time if there are user-defined groups
    - Ansible will run all sections that match the node type against the node

CFS INVENTORY

- Dynamic inventory generates Ansible hosts file with data from HSM
  - Can target an HSM group

```
ncn# cray hsm groups list --format json | jq .[].label
"blue"
"green"
```

  - Can target HSM-reported hardware roles and sub-roles
    - "Compute", "Management", "Application"
    - "Application_UAN", "Management_Worker", other Application subroles for the system
- Static inventory can target specific groups of nodes
  - Good for testing configuration changes on a small scale in a configuration repository

```
ncn# mkdir -p hosts; cd hosts; cat > static <<EOF
[test_nodes]
x3000c0s25b0n0
EOF
ncn# cd ..; git add hosts/static
ncn# git commit -m "Added a single node to static inventory for test_nodes"
ncn# git push
```

- Image Customization by IMS
  - IMS image IDs are used as hosts and grouped according to input to the session creation

```
ncn# cray cfs sessions create --name example --configuration-name configurations-example \
--target-definition image --target-group Compute IMS_IMAGE_ID
```

CFS PERFORMANCE AND SCALING TIPS

- Use image customization to limit how many times a task is run and improve boot times
- Use image customization for configuration that is the same for all nodes of a type
  - Target a task to be run only when customizing image

    ```
    when: "{{ cray_cfs_image | default(false) }}"
    ```
  - Target a task to be run only on booted node during node personalization

    ```
    when: "{{ not cray_cfs_image | default(false) }}"
    ```
- Import roles rather than playbooks
  - Each time a new playbook starts, Ansible automatically gathers facts for all the systems it is running against
  - This is not necessary more than once and can slow down Ansible execution
- Turn off facts that are not needed in a playbook by setting `gather_facts: false`
  - If only a few facts are required, it is also possible to limit fact gathering by setting `gather_subset`
    - For more information on `gather_subset`, see https://docs.ansible.com/ansible/latest/modules/setup_module.html
- Use loops rather than individual tasks where modules are called multiple times
  - Some Ansible modules will optimize the command, such as grouping package installations into a single transaction https://docs.ansible.com/ansible/latest/user_guide/playbooks_loops.html

ANSIBLE DEBUGGING

- Name tasks uniquely and use debug

```
tasks:
  - name: find nid match in external hosts file, capture IP address
    shell: “grep {{nid}} /etc/mysitelocal/hosts-external | head -1 | awk '{ print $4 }’”
    register: external_ipaddr
  - name: add ListenAddress/external options to file
    lineinfile:
      dest: /etc/sshd/sshd_config
      regexp="^SSHD_OPTS="
      line="SSHD_OPTS=‘-u0 -f /etc/ssh/sshd_config.external -o ListenAddress={{external_ipaddr}}’"
      backup: yes
    when:
      external_ipaddr is defined
  - debug: “Did not find external interface to start SSHD on...”
    when: external_ipaddr is not defined
```

- Ansible tasks and playbooks can be profiled to determine execution times and identify poor runtime performance
  - Edit the default CFS Ansible.cfg

    ```
    ncn# kubectl edit cm cfs-default-ansible-cfg -n services
    ```

  - Uncomment this line

    ```
    #callback_whitelist    = cfs_aggregator, timer, profile_tasks, profile_roles
    ```

  - New sessions will be created with profiling information available in the Ansible logs of the CFS session pods

TROUBLESHOOT ANSIBLE PLAY FAILURES IN CFS SESSIONS

- Find the CFS pod that is in an error state

```
ncn# kubectl get pods -n services | grep Error
    NAME                                         READY    STATUS    RESTARTS    AGE
    cfs-e8e48c2a-448f-4e6b-86fa-dae534b1702e-pnxmn   0/3      Error     0           25h
```

- Check to see what containers are in the pod

```
ncn# kubectl logs -n services $CFS_POD_NAME
    Error from server (BadRequest): a container name must be specified for pod cfs-e8e48c2a-448f-4e6b-86fa-dae534b1702e-pnxmn,
choose one of: [inventory ansible-0 istio-proxy] or one of the init containers: [git-clone-0 istio-init]
```

- Check the git-clone-0, inventory, ansible-0 containers in that order

```
ncn# kubectl logs -n services CFS_POD_NAME git-clone-0
ncn# kubectl logs -n services CFS_POD_NAME inventory
        Sidecar available
        2019-12-05 15:00:12,160 - INFO    - cray.cfs.inventory - Starting CFS Inventory version=0.4.3, namespace=services
        2019-12-05 15:00:12,171 - INFO    - cray.cfs.inventory - Inventory target=dynamic for cfsession=boa-2878e4c0-39c2-4df0-989e-
053bb1edee0c
        2019-12-05 15:00:12,227 - INFO    - cray.cfs.inventory.dynamic - Dynamic inventory found a total of 2 groups
        2019-12-05 15:00:12,227 - INFO    - cray.cfs.inventory - Writing out the inventory to /inventory/hosts
ncn# kubectl logs -n services CFS_POD_NAME ansible-0
        Waiting for Inventory
        TASK [ncmp_hsn_cns : SLES Compute Nodes (HSN): Create/Update ifcfg-hsnx File(s)] ***
        fatal: [x3000c0s19b1n0]: FAILED! => {"msg": "'interfaces' is undefined"}
        fatal: [x3000c0s19b2n0]: FAILED! => {"msg": "'interfaces' is undefined"}
        NO MORE HOSTS LEFT *************************************************************
        PLAY RECAP *********************************************************************
        x3000c0s19b1n0             : ok=28   changed=20   unreachable=0    failed=1    skipped=77   rescued=0    ignored=1
        x3000c0s19b2n0             : ok=27   changed=19   unreachable=0    failed=1    skipped=63   rescued=0    ignored=1
```

HPE CRAY EX SYSTEM OVERVIEW
MANAGEMENT SERVICES
WHAT IS HAPPENING ON MY SYSTEM?
MANAGING USER ENVIRONMENTS
RESOURCES

WHAT IS HAPPENING ON MY SYSTEM?

- Booting Processes
- System Health
  - Prometheus, Jaeger, Kiali, Alertmanager, Grafana, Dashboards
- Logs and Dumps
  - ConMan, Elasticsearch, Logstash, SMA-Kibana
- Monitoring
  - LDMS, SMA-Grafana, Alerts and Notifications, Dashboards
- System Testing
- Troubleshooting Tips

BOOTING PROCESS

- Booting overview
- Boot Script Service
- Content Projection Service
- Boot Orchestration Service

BOOT FLOWCHART WITH BOS AND S3

The Boot Orchestration Service (BOS) is responsible for booting, configuring, or shutting down collections of nodes.

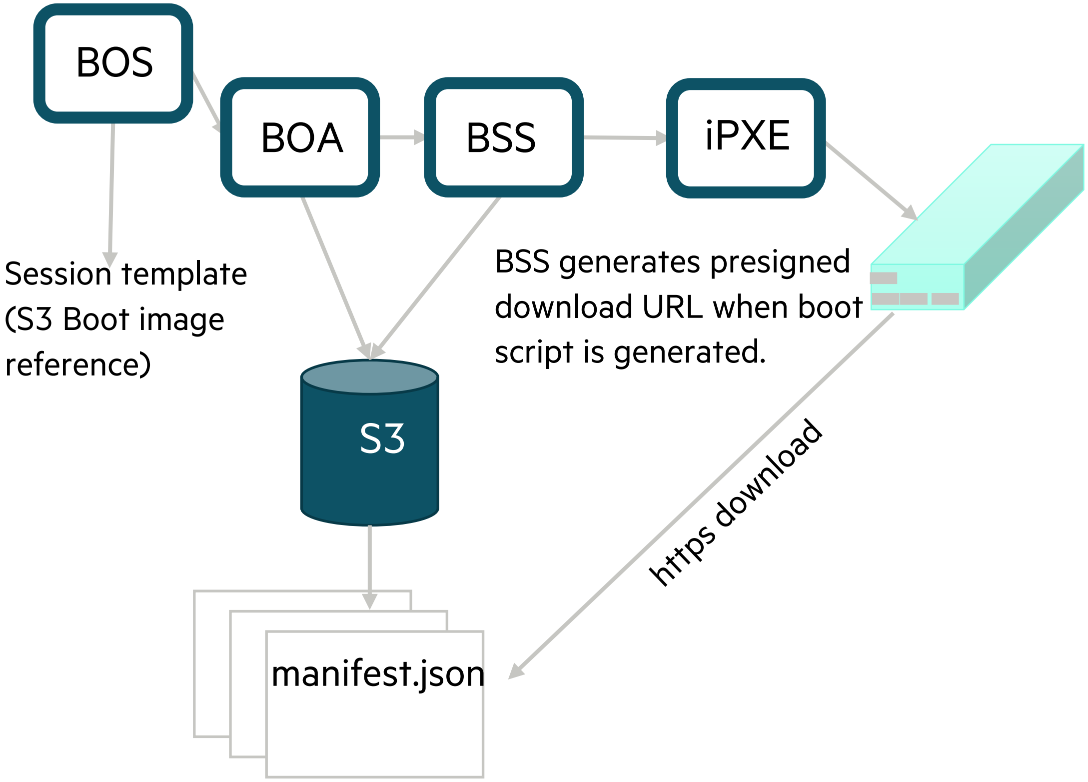
The Boot Orchestration Service has the following components:

- **Boot Orchestration Session Template** – a collection of one or more boot set objects
  - A boot set defines a collection of nodes and the information about the boot artifacts and parameters
- **Boot Orchestration Session** – An instance of a BOS operation that manages Boot Orchestration Agents
- **Boot Orchestration Agent** (BOA) – Executes actions submitted to the BOS API

BOS coordinates with several services to boot compute nodes:

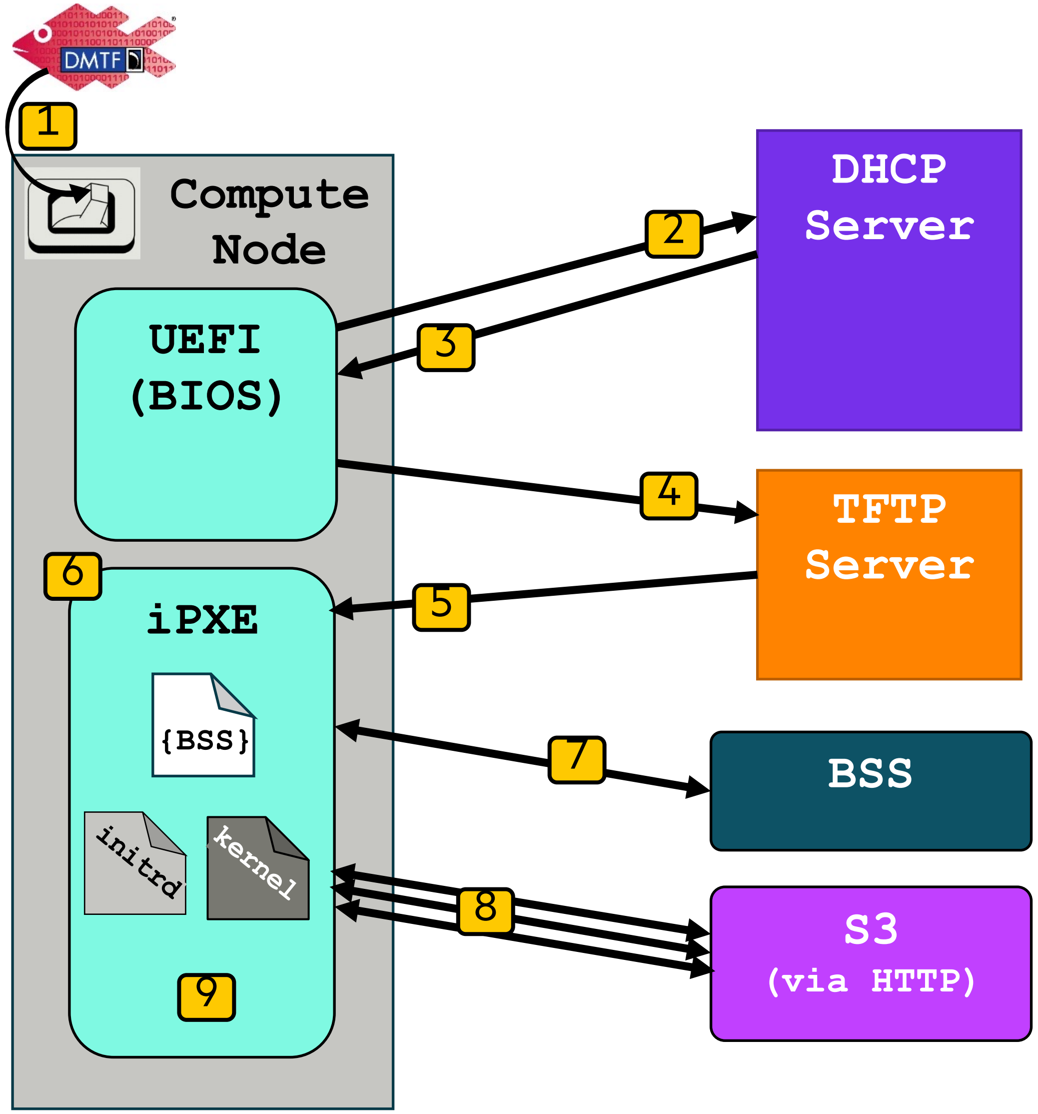
- **Hardware State Manager** (HSM) – Tracks the state of each node and holds their group and role associations
- **Image Management Service** (IMS) – Manages image records (kernel, `initrd`, image root)
- **Simple Storage Service** (S3) – Stores boot artifacts (kernel, `initrd`, image root)
- **Boot Script Service** (BSS) – Stores per-node information about iPXE boot script
- **Cray Advanced Platform and Monitoring Control** (CAPMC) – provides system-level power control for nodes in the system
- **Configuration Framework Servic**e (CFS) – Configures node(s) using configuration framework

During boot, BOS/BOA will get the S3 reference to boot image. BOA will need to access the image to read boot parameters. At the point that BSS generates the iPXE bootscript, BSS will generate the pre-signed S3 Download URL for the kernel and `initrd`. CPS will similarly need to be updated to project the `rootfs`.



COMPUTE NODE BOOT SEQUENCE

1. The compute node is powered on
2. The BIOS issues a DHCP discover request
3. DHCP Server responds with:
   - The IP address of the TFTP server
   - The name of the file to download
4. The node sends a request to the TFTP server
5. The TFTP server sends `ipxe.efi` to the node
6. The node chainloads the iPXE binary
7. iPXE downloads an `ipxe` boot script from BSS
8. Following the boot script, iPXE downloads the kernel, `initrd`, and kernel parameters from S3
9. The node attempts to boot using the boot artifacts pulled from S3



BOOT SCRIPT SERVICE (BSS)

Boot Script Service (BSS)

- REST API to interact with HSM and provide nodes with boot artifacts and cloud-init payloads
- Stores the configuration information that is used to boot each hardware component
- Nodes consult BSS for their boot artifacts and boot parameters when nodes boot or reboot

- The BSS stores the current image and parameters that are assigned to each node
- The boot parameters stored in BSS for a node when a node is powered on will be used for that boot
- The Boot Orchestration Service (BOS) is used to update the boot script for a given node
  - Updating the boot script for a node in the BSS directly is not recommended
  - BSS does not have any information about how a node should be configured after it boots
  - Post-boot configuration (node personalization) is controlled by the Configuration Framework Service (CFS)
    - BOS calls CFS as part of the process of orchestrating the boot process

RETRIEVING A BOOT SCRIPT FROM BSS

- The boot script for a node includes the following boot artifacts (highlighted):

```
ncn# cray bss bootscript list --name x3000c0s23b2n0
#!ipxe
kernel --name kernel http://rgw-vip.nmn/boot-images/1c4f7f49-bfaf-4c25-9110-f5b46440c9a2/kernel?
X-Amz-Algorithm=AWS4-HMAC-SHA256&X-Amz-Credential=L18PWYUE7B8KBQR3X4NB%2F20220105%2Fdefault%2Fs3%2Faws4_request&X-Amz-
Date=20220105T012211Z&X-Amz-Expires=86400&X-Amz-SignedHeaders=host&X-Amz-
Signature=8aa3bdb208d5e216a0331c41c66f4346f6bf75b75b0f5f0addf0caf4bde3fd7e
initrd=initrd console=ttyS0,115200 bad_page=panic crashkernel=360M hugepagelist=2m-2g intel_iommu=off
intel_pstate=disable iommu=pt numa_interleave_omit=headless oops=panic pageblock_order=14 pcie_ports=native
rd.neednet=1 rd.retry=10 rd.shell turbo_boost_limit=999 biosdevname=0 ip=dhcp quiet
spire_join_token=8900a2f6-3bee-4757-bccb-75247893a6d0
root=craycps-s3:s3://boot-images/1c4f7f49-bfaf-4c25-9110-f5b46440c9a2/rootfs:
c91e4b1462822da009f191c206d8c9fa-205:dvs:api-gw-service-nmn.local:300:nmn0 nmd_data=url=s3://boot-images/1c4f7f49-bfaf-
4c25-9110-f5b46440c9a2/rootfs,etag=c91e4b1462822da009f191c206d8c9fa-205 bos_session_id=f8937b77-2c10-4a05-93bd-06cff8ee076b
xname=x3000c0s23b2n0 nid=6 ds=nocloud-net;s=http://10.92.100.81:8888/ || goto boot_retry
initrd --name initrd http://rgw-vip.nmn/boot-images/1c4f7f49-bfaf-4c25-9110-f5b46440c9a2/initrd?
X-Amz-Algorithm=AWS4-HMAC-SHA256&X-Amz-Credential=L18PWYUE7B8KBQR3X4NB%2F20220105%2Fdefault%2Fs3%2Faws4_request&X-Amz-
Date=20220105T012211Z&X-Amz-Expires=86400&X-Amz-SignedHeaders=host&X-Amz-
Signature=0dc66fb06761dd2e8f022446da6a5d31f9320c0bdb0c054cc2e7a10d0af4a972 || goto boot_retry
boot || goto boot_retry
:boot_retry
sleep 30
chain https://api-gw-service-nmn.local/apis/bss/boot/v1/bootscript?mac=b4:2e:99:7f:0d:24&retry=1
```

kernel image

Kernel parameters

root file system squashfs image

initrd image

MAC address of node's NIC

BSS LOGS

- It is useful to monitor the logs of the cray-bss container within the BSS pods.

```
03/08 19:00:33 ncn# kubectl get pods -n services | grep bss
cray-bss-647fb9775f-jmxs7                                  2/2     Running     0          54d
cray-bss-647fb9775f-k4gl5                                  2/2     Running     0          53d
cray-bss-647fb9775f-qzxf5                                  2/2     Running     0          53d
cray-bss-etcd-4kvjphv69p                                   1/1     Running     0          53d
cray-bss-etcd-7lxvcq4drk                                   1/1     Running     0          54d
cray-bss-etcd-brp85brbnd                                   1/1     Running     0          119d
03/08 19:01:05 ncn# for POD in $(kubectl get pods -n services | grep bss |grep -v etcd | awk '{ print$1}');
do kubectl logs -n services --since 10m $POD -c cray-bss; done
03/08 19:01:23 ncn# ssh x1000c1s1b0n1 reboot
Connection to x1000c1s1b0n1 closed by remote host.
03/08 19:01:35 ncn# sleep 480
03/08 19:11:07 ncn# for POD in $(kubectl get pods -n services | grep bss |grep -v etcd | awk '{ print$1}');
do kubectl logs -n services --since 10m $POD -c cray-bss | grep -v DEBUG; done
2022/03/08 19:10:18 Retrieving state info from http://cray-smd/hsm/v1
2022/03/08 19:10:18 GET /meta-data, xname: x1000c1s1b0n1 ip: 10.100.0.114
2022/03/08 19:10:18 http: superfluous response.WriteHeader call from main.metaDataGetAPI
(cloudInitAPI.go:209)
'
', &spireResp): { 0  9a4f8130-7dee-4180-a4cd-63b22138c03c}
2022/03/08 19:07:34 BSS request succeeded for MAC 00:40:a6:83:63:34 (x1000c1s1b0n1)
```

Like other core boot services, BSS runs insides a Kubernetes pod

`reboot` is NOT the recommended way to reboot a node; BOS should be used

WHAT IS THE CONTENT PROJECTION SERVICE (CPS)

- The Content Projection Service (CPS) is a container-based microservice managed by Kubernetes
  - The main components of CPS are:
    - CPS Brokers
    - Content Managers
    - Projection Managers
- At node boot the Boot Script Service (BSS) provides
  - The Linux kernel
  - `initrd`
  - Boot parameter data
- CPS provides
  - Node's root file system image (operating system image)
  - HPE Cray Programming Environment (CPE) images
  - Analytics images

`cray cps contents` provides a list of images being managed by the content manager

`cray cps deployment` provides a list of CPS pods and their statuses

`cray cps transports` provides a list images currently being exported (served) to nodes

CPS COMPONENTS AND THEIR PODS

```
ncn# kubectl get pods -n services -o wide | grep cps
cray-cps-59db74b89f-7v2ps   2/2   Running   0   41d   10.39.0.241   ncn-w001   <none>   <none>
cray-cps-59db74b89f-gv4h9   2/2   Running   0   41d   10.40.0.216   ncn-w002   <none>   <none>
cray-cps-cm-pm-8bmfk        4/4   Running   0   41d   10.37.0.223   ncn-w003   <none>   <none>
cray-cps-cm-pm-lj5ph        4/4   Running   0   41d   10.39.1.55    ncn-w001   <none>   <none>
cray-cps-cm-pm-sr9gr        4/4   Running   0   41d   10.40.0.240   ncn-w002   <none>   <none>
cray-cps-etcd-f98mlv2n4g    1/1   Running   0   41d   10.39.0.232   ncn-w001   <none>   <none>
cray-cps-etcd-f9f9lhcw5g    1/1   Running   0   41d   10.40.1.15    ncn-w002   <none>   <none>
cray-cps-etcd-p7q44q5pdt    1/1   Running   0   41d   10.37.0.242   ncn-w003   <none>   <none>
```

CPS Broker

- Provides the API service
- Runs in the cray-cps pod

CPS Content Manager ("CM")

Retrieves file system images from S3 to make them available to the CPS Projection Manager

CPS Projection Manager ("PM")

Makes artifacts available to other nodes via network file systems "transports" such as DVS

CPS state manager (etcd)

Facilitates communication between CPS components about the current or desired state of the CPS service.

IDENTIFYING THE IMAGE IN USE BY A NODE

```
ncn# cray bss bootparameters  list --name x3000c0s14b0n0 --format json | jq '.[].kernel'
"s3://boot-images/1c329db9-3a32-49b8-be7c-2b09d47a609f/kernel"

ncn# cray bss bootparameters  list --name x3000c0s14b0n0 --format json | jq '.[].params'
"console=ttyS0,115200 bad_page=panic crashkernel=360M hugepagelist=2m-2g intel_iommu=off
intel_pstate=disable iommu=pt ip=nmn0:dhcp numa_interleave_omit=headless numa_zonelist_order=node
oops=panic pageblock_order=14 pcie_ports=native printk.synchronous=y quiet rd.neednet=1 rd.retry=10
rd.shell turbo_boost_limit=999 ifmap=net2:nmn0,lan0:hsn0,lan1:hsn1 spire_join_token=${SPIRE_JOIN_TOKEN}
root=craycps-s3:s3://boot-images/1c329db9-3a32-49b8-be7c-2b09d47a609f/rootfs:
4f862288a668ed8328158a438f276ab3-190:dvs:api-gw-service-nmn.local:300:nmn0 nmd_data=url=s3://boot-
images/1c329db9-3a32-49b8-be7c-2b09d47a609f/rootfs,etag=4f862288a668ed8328158a438f276ab3-190
bos_session_id=43254b57-d787-4797-8b45-ab621ca0b327"

ncn# ssh x3000c0s14b0n0 cat /proc/cmdline
kernel initrd=initrd console=ttyS0,115200 bad_page=panic crashkernel=360M hugepagelist=2m-2g
intel_iommu=off intel_pstate=disable iommu=pt ip=nmn0:dhcp numa_interleave_omit=headless
numa_zonelist_order=node oops=panic pageblock_order=14 pcie_ports=native printk.synchronous=y quiet
rd.neednet=1 rd.retry=10 rd.shell turbo_boost_limit=999 ifmap=net2:nmn0,lan0:hsn0,lan1:hsn1
spire_join_token=d399ee35-c191-46c7-9f40-da63f895d368 root=craycps-s3:s3://boot-images/1c329db9-3a32-49b8-
be7c-2b09d47a609f/rootfs:4f862288a668ed8328158a438f276ab3-190:dvs:api-gw-service-nmn.local:300:nmn0
nmd_data=url=s3://boot-images/1c329db9-3a32-49b8-be7c-2b09d47a609f/rootfs,
etag=4f862288a668ed8328158a438f276ab3-190 bos_session_id=43254b57-d787-4797-8b45-ab621ca0b327
xname=x3000c0s14b0n0 nid=49168832 ds=nocloud-net;s=http://10.92.100.81:8888/
```

TRACKING AN IMAGE FROM NODE TO CPS TO S3

```
ncn# cray cps contents list --format json | grep 1c329db9-3a32-49b8-be7c-2b09d47a609f/rootfs
    "s3path": "s3://boot-images/1c329db9-3a32-49b8-be7c-2b09d47a609f/rootfs",

ncn# cray cps contents list --format json | jq 'map(select(.s3path == "s3://boot-images/1c329db9-3a32-49b8-
be7c-2b09d47a609f/rootfs")) | .[].artifactID'
"e2e335eda4055fd1b293de4f2c9ab6ce"

ncn# cray cps contents list --format json | jq 'map(select(.s3path == "s3://boot-images/1c329db9-3a32-49b8-
be7c-2b09d47a609f/rootfs")) | .[].exportPath'
"/var/lib/cps-local/e2e335eda4055fd1b293de4f2c9ab6ce"
```

When a node requests a new image from CPS the content manager (CM) downloads the squashfs file from S3 to the Kubernetes worker node hosting each cray_cps_cm_pm_ pod. The squashfs files are stored on local disk in the worker nodes until CPS deletes the content

```
ncn# ssh ncn-w001
Last login: Thu Jul 15 04:53:58 2021 from 10.252.1.9

ncn# file /var/lib/cps-local/e2e335eda4055fd1b293de4f2c9ab6ce/rootfs
/var/lib/cps-local/e2e335eda4055fd1b293de4f2c9ab6ce/rootfs: Squashfs filesystem, little endian, version
4.0, 1589565630 bytes, 90812 inodes, blocksize: 131072 bytes, created: Tue Jun 29 17:23:47 2021
```

COMPUTE NODE ROOT FILE SYSTEM MOUNTS

- All files in the compute node root file system (rootfs) are provided from a squashFS image stored in S3 (Ceph)
- Compute node rootfs images are projected by CPS pods and mounted via DVS
- Rootfs images are mounted on compute nodes with `/opt/cray/cps-utils/bin/cpsmount.sh` and are mounted read-only
  - A compute node local overlay file system is configured to enable writes "on top of" the `rootfs` to an ephemeral in-memory file system
- DVS mount content is accessed over the network on demand
  - When a block is first referenced, DVS caches the content in the node-local Linux page cache so future references to that data will not involve the network
    - If available memory gets too low, Linux can evict these pages, and thus the data will be accessed over the network again (and cached again) if/when they are referenced again
    - Overlay Preload can permanently "pin" files in memory on the compute node at boot time so they can never be evicted
- DVS can also project other filesystems unrelated to CPS
  - Projections of user file systems using DVS can be configured as read-write or read-only

TEMPLATE OF BOS SESSION TEMPLATE

- Use the provided empty session template template as a JSON framework and edit all the fields

```
ncn# cray bos sessiontemplatetemplate list --format json
{
  "boot_sets": {
    ”boot_set1": {
      "boot_ordinal": 1,
      "etag": "your_boot_image_etag",
      "kernel_parameters": "your-kernel-parameters",
      "network": "nmn",
      "node_list": ["x3000c0s19b1n0", "x3000c0s19b1n1", "x3000c0s19b2n0"]
      "path": "your-boot-path",
      "rootfs_provider": "your-rootfs-provider",
      "rootfs_provider_passthrough": "your-rootfs-provider-passthrough",
      "type": "your-boot-type"
    },
    "boot_set2": { ... }
  },
  "cfs": {
    "configuration": "desired-cfs-config"
  },
  "enable_cfs": true,
  "name": "name-your-template"
}
```

Multiple boot sets can be defined that will have same CFS configuration to be applied, but different kernel parameters or different path to boot artifacts

Can specify nodes one of these ways:
"node_list": ["x3000c0s19b1n0", "x3000c0s19b1n1", "x3000c0s19b2n0"]
"node_groups": ["green", "white", "pink"]
"node_roles_groups": ["Compute"]

BOS SESSION TEMPLATE DETAIL

```
ncn# cray bos sessiontemplate describe cos-sessiontemplate-2.2.101 --format json
{
  "boot_sets": {
    "compute": {
      "boot_ordinal": 2,
      "etag": "b29bb9e8cd8c64541f4ff025e108f7a6",
      "kernel_parameters": "ip=dhcp quiet spire_join_token=${SPIRE_JOIN_TOKEN}",
      "network": "nmn",
      "node_roles_groups": [
        "Compute"
      ],
      "path": "s3://boot-images/c26034f1-4acf-4a45-b898-c5842d711ef6/manifest.json",
      "rootfs_provider": "cpss3",
      "rootfs_provider_passthrough": "dvs:api-gw-service-nmn.local:300:hsn0,nmn0:0",
      "type": "s3"
    }
  },
  "cfs": {
    "configuration": "cos-config-2.2.99"
  },
  "enable_cfs": true,
  "name": "cos-sessiontemplate-2.2.101"
}
```

boot_sets: A collection of nodes & the images they should boot with. One or more boot_sets may be specified per session template

etag: 'entity tag helps identify the version of the manifest.json file. Currently not used but cannot be left blank

network: The network over which the node will boot

kernel parameters : Kernel parameters passed to the operating system

Path: s3 location of the components of the boot image file ([IMS_Image_ID] `manifest.json`). Processed based on the “type”

rootfs_provider: The root file system provider

rootfs_provider_passthrough: Additional kernel parameters that will be appended to the 'rootfs=' kernel parameter

cfs or **cfs_url:** The repository configuration file or clone URL for the repository providing the configuration

enable_cfs: Whether to enable the Configuration Framework Service (CFS)

name: Name of the Session Template. The length of the name is restricted to 45 characters

CREATE A BOS SESSION TEMPLATE

```
ncn# cat INPUT_FILE.json
{
  "name": "cos-sessiontemplate-2.2.101",
  "boot_sets": {
    "test_compute": {
      "network": "nmn",
      "boot_ordinal": 1,
      "kernel_parameters": "ip=dhcp quiet spire_join_token=${SPIRE_JOIN_TOKEN}",
      "rootfs_provider": "cpss3”,
      "node_list": [ "x3000c0s19b1n0" ],
      "etag": "90b2466ae8081c9a604fd6121f4c08b7",
      "path": "s3://boot-images/06901f40-f2a6-4a64-bc26-772a5cc9d321/manifest.json",
      "rootfs_provider_passthrough": "dvs:api-gw-service-nmn.local:300:eth0",
      "type": "s3" }
    },
  "cfs": {
    "configuration": "cos-config-2.2.101"
  },
  "enable_cfs": true
}
ncn# cray bos sessiontemplate create --file INPUT_FILE.json --name cos-sessiontemplate-2.2.101
ncn# cray bos sessiontemplate list --format json | jq '.[].name'
"cos-sessiontemplate-2.2.101"
"uan-sessiontemplate-2.3.2-cos-2.2.101"
```

Display a list of all session templates in your system, filtering the output with jq for the `.name`

CREATE BOS SESSION

- A BOS Session represents an operation on a Session Template
  - boot – Boot nodes that are off
  - configure – Reconfigure the nodes using the Configuration Framework Service (CFS)
  - reboot – Gracefully power down nodes that are on and then power them back up
  - shutdown – Gracefully power down nodes that are on
- Use `cray bos session create` to create a BOS session

```
ncn# cray bos session create --template-uuid cos-sessiontemplate-2.2.101 --operation reboot
operation = "Reboot"
templateUuid = "cos-sessiontemplate-2.2.101"
[[links]]
href = "/v1/session/158fc371-d279-4494-a60e-fcac5612d605"
jobId = "boa-158fc371-d279-4494-a60e-fcac5612d605"
rel = "session"
type = "GET"

[[links]]
href = "/v1/session/158fc371-d279-4494-a60e-fcac5612d605/status"
rel = "status"
type = "GET‶
```

When a BOS session is created it initiates one or more Boot Orchestration Agent (BOA) jobs. The name of the `session` created will be labeled `href` and included in the BOA `jobid` – which is part of the BOA pod name

- BOS supports an optional --limit parameter when creating a session
  - List of nodes, HSM groups, or HSM roles to limit the nodes that BOS runs against
  - Components are treated as OR operations unless preceded by “&” for AND or “|” for NOT

```
cray bos session create --template-uuid cos-sessiontemplate-2.2.101 --operation reboot --limit
x3000c0s20b2n0
```

VIEW RUNNING BOS SESSION INFORMATION

- Use `cray bos session describe` to view progress of the BOS job
- Use `kubectl get pods` to view the status of the Boot Orchestration Agent (BOA) job associated with the BOS job

```
ncn# cray bos session describe 158fc371-d279-4494-a60e-fcac5612d605

boa_job_name = "boa-158fc371-d279-4494-a60e-fcac5612d605"
complete = false
error_count = 0
in_progress = true
operation = "Reboot"
start_time = "2021-06-28 08:40:14.949422"
status_link = "/v1/session/158fc371-d279-4494-a60e-fcac5612d605/status"
templateUuid = "team1_session_template"
```

When a BOS session is created it initiates one or more Boot Orchestration Agent (BOA) jobs. The name of the session created will be labeled `href` and included in the BOA jobid – which is part of the BOA pod name

`cray bos session describe <JOB ID>` is used to view the status and progress of the job.
`boa_job_name` – Boot Orchestration Agent job name.

Monitoring the BOA JOB with **`kubectl get pods`** command.

```
ncn# kubectl get pods -n services -l job-name=boa-158fc371-d279-4494-a60e-fcac5612d605

NAME                                            READY   STATUS    RESTARTS   AGE
boa-158fc371-d279-4494-a60e-fcac5612d605-xw4xh  2/2     Running   0          2m47s
```

VIEW BOS SESSION STATUS

```
ncn# cray bos session status describe CATEGORY_NAME PHASE_NAME BOOT_SET_NAME SESSION_ID --format json
```

- BOS session status Phases
  - shutdown
  - boot
  - configure
- BOS session status Categories
  - not_started
  - succeeded
  - failed
  - excluded
  - in_progress

```
ncn# cray bos session status describe succeeded shutdown compute fb808925-2dd6-440d-8d6c-834892472036
name = "succeeded"
node_list = [ "x3000c0s19b4n0", "x3000c0s19b2n0", "x3000c0s19b3n0", "x3000c0s19b1n0",]
ncn# cray bos session status describe failed boot compute fb808925-2dd6-440d-8d6c-834892472036
name = ”failed"
node_list = [ "x3000c0s19b4n0",]
ncn# cray bos session status describe in_progress configure compute fb808925-2dd6-440d-8d6c-834892472036
name = "in_progress"
node_list = ["x3000c0s19b2n0", "x3000c0s19b3n0", "x3000c0s19b1n0",]
```

VIEW COMPLETED BOS SESSION INFORMATION

- Use `cray bos session describe` to view progress of the BOS job.
- Use `kubectl get pods` to view the status of the Boot Orchestration Agent (BOA) job associated with the BOS job.

```
ncn# cray bos session describe 158fc371-d279-4494-a60e-fcac5612d605

boa_job_name = "boa-158fc371-d279-4494-a60e-fcac5612d605"
complete = true
error_count = 0
in_progress = false
operation = "Reboot"
start_time = "2021-06-28 08:40:14.949422"
status_link = "/v1/session/158fc371-d279-4494-a60e-fcac5612d605/status"
stop_time = "2021-06-28 08:53:50.711327"
templateUuid = "team1_session_template"
```

Monitoring the BOS job with **`cray bos session describe <JOB ID>`** to completion

Monitoring the BOA job with **`kubectl get pods`** command to completion

```
ncn# kubectl get pods -n services -l job-name=boa-158fc371-d279-4494-a60e-fcac5612d605
NAME                                          READY   STATUS      RESTARTS   AGE
boa-158fc371-d279-4494-a60e-fcac5612d605-xw4xh   0/2     Completed   0          14m
```

SYSTEM HEALTH

- Prometheus
- Alertmanager
- Istio with Kiali and Jaeger
- Grafana
- Dashboards

SYSTEM MANAGEMENT HEALTH SERVICE

Is the system healthy?

- Independent from the System Monitoring Framework (SMF)
- Does not monitor computes!

System Mgmt Health Service 10d

Helm chart includes

- **Prometheus** to federate metrics
- **Alertmanager** for custom notifications
- **Grafana** with dashboards for Kubernetes, Istio, Ceph

Export Prometheus metrics

SMF 30d

Kubernetes 4h

Prometheus-operator chart features Prometheus with support for

- K8s nodes
- Etcd
- K8s internals
- K8s workloads

Istio 4h

Istio chart includes

- Prometheus which collects Istio metrics
- Kiali and Jaeger

Ceph 4h

Prometheus module exposes metrics from ceph-mgr

INDUSTRY STANDARD TOOLS

- Prometheus is the de-facto standard cloud-native metrics and monitoring tool
  - Prometheus operator provides custom resource definitions
    - Scrape metrics from service endpoints
  - Prometheus alerting rules triggers alerts to Alertmanager
  - Alertmanager manages the silencing, inhibition, aggregation, and sending out of notifications
- Grafana supports pulling data from Prometheus
  - Dashboards are readily available
- Istio supports service mesh tracing with Jaeger and observability with Kiali
- Customer integration
  - Customize Alertmanager notifications
    - Email, Slack, custom web hook
  - Consume metrics via SMF Telemetry API
    - Reuse SMF integration strategy
    - Export alert configurations
  - Run components “off system”
    - Integrate with existing Prometheus infrastructure

HEALTH CHECKS

- Prometheus alerts provide coverage across infrastructure and platform
- Coarse-grained and comprehensive, as opposed to fine-grained and exhaustive
- Supports preventive and diagnostic use cases

NON-COMPUTE NODES	UTILITY STORAGE	CONTAINER ORCHESTRATION	SERVICE MESH	WORKLOADS
• CPU and memory utilization • Local storage utilization • Network I/O errors and latency • Clock skew	• Ceph status • Storage utilization • Disk I/O errors and latency	• Kubernetes status • API errors • CPU and memory overcommitments	• Istio status • Service availability • Service request rates • Service response statuses and latency	• Status of pods, deployments, stateful sets, daemon sets, jobs • CPU, memory, network, and storage utilization and errors

RETRIEVING ALERTS FROM PROMETHEUS

```
ncn# kubectl -n sysmgmt-health get svc cray-sysmgmt-health-promet-prometheus
NAME                                    TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)    AGE
cray-sysmgmt-health-promet-prometheus   ClusterIP   10.21.141.187    <none>        9090/TCP   34d
ncn# curl -s http://10.21.141.187:9090/api/v1/alerts |jq -j '.data' | grep alertname | sort -u
        "alertname": "CPUThrottlingHigh",
        "alertname": "CephMgrIsAbsent",
        "alertname": "CephMgrIsMissingReplicas",
        "alertname": "KubeContainerWaiting",
        "alertname": "KubeDeploymentReplicasMismatch",
        "alertname": "KubeJobCompletion",
        "alertname": "KubeJobFailed",
        "alertname": "KubePodNotReady",
        "alertname": "PostgresqlFollowerReplicationLagSMA",
        "alertname": "PostgresqlFollowerReplicationLagServices",
        "alertname": "PostgresqlHighRollbackRate",
        "alertname": "PostgresqlInactiveReplicationSlot",
        "alertname": "PostgresqlNotEnoughConnections",
        "alertname": "TargetDown",
        "alertname": "Watchdog",
```

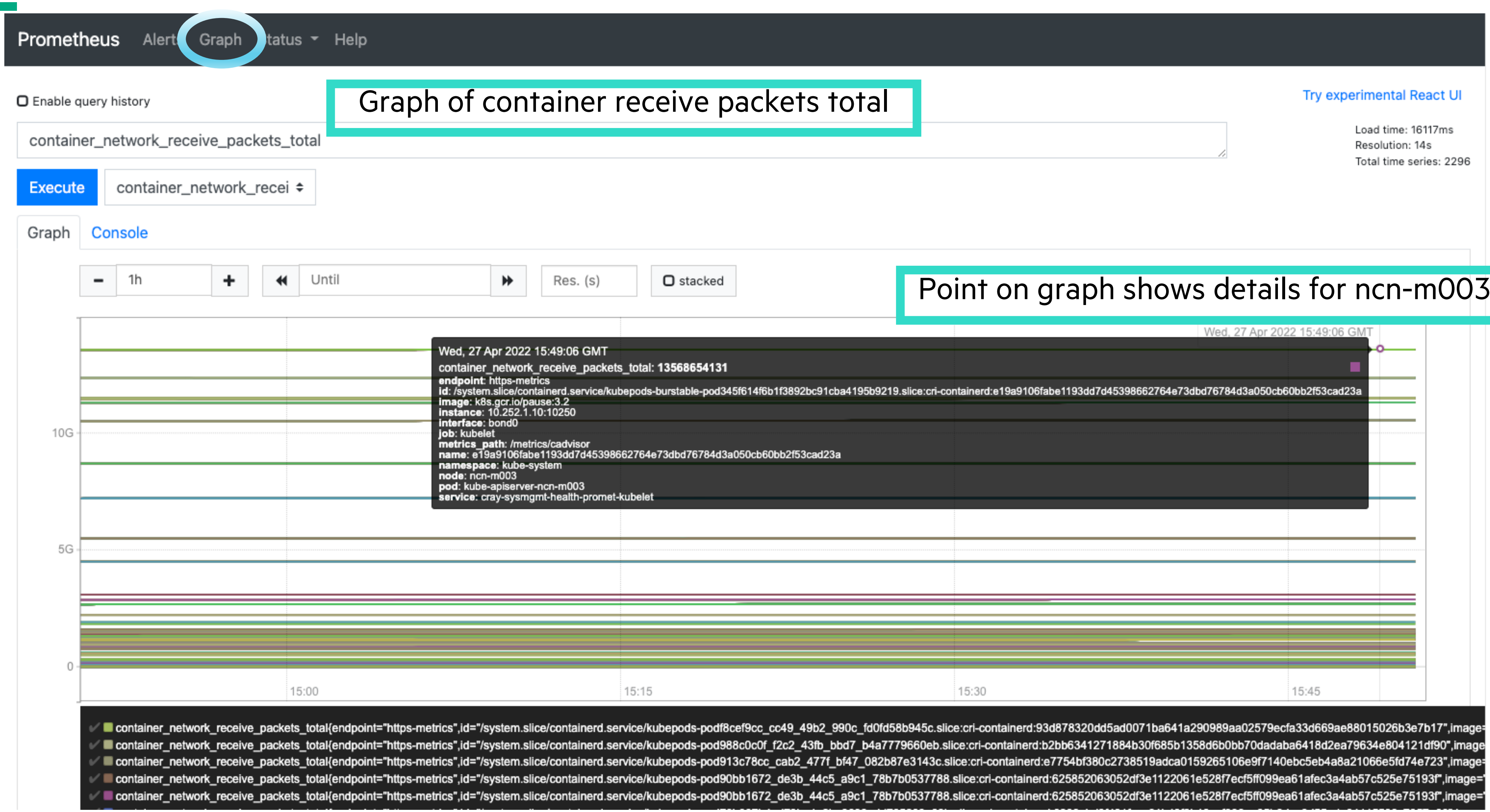
RETRIEVING THE LATEST ALERT FROM PROMETHEUS

```
ncn# curl -s http://10.21.141.187:9090/api/v1/alerts |jq -j '.data.alerts \
| map(select(.labels.alertname == "CPUThrottlingHigh")) | max_by(.activeAt)'
{
  "labels": {
    "alertname": "CPUThrottlingHigh",
    "container": "manager",
    "namespace": "gatekeeper-system",
    "pod": "gatekeeper-controller-manager-588d6476db-d5g8v",
    "severity": "info"
  },
  "annotations": {
    "message": "28.03% throttling of CPU in namespace gatekeeper-system for container manager 
in pod gatekeeper-controller-manager-588d6476db-d5g8v.",
    "runbook_url": "https://github.com/kubernetes-monitoring/kubernetes-
mixin/tree/master/runbook.md#alert-name-cputhrottlinghigh"
  },
  "state": "pending",
  "activeAt": "2022-04-27T16:11:07.129355508Z",
  "value": "2.8030608135320173e-01"
}
```

PROMETHEUS - GRAPH

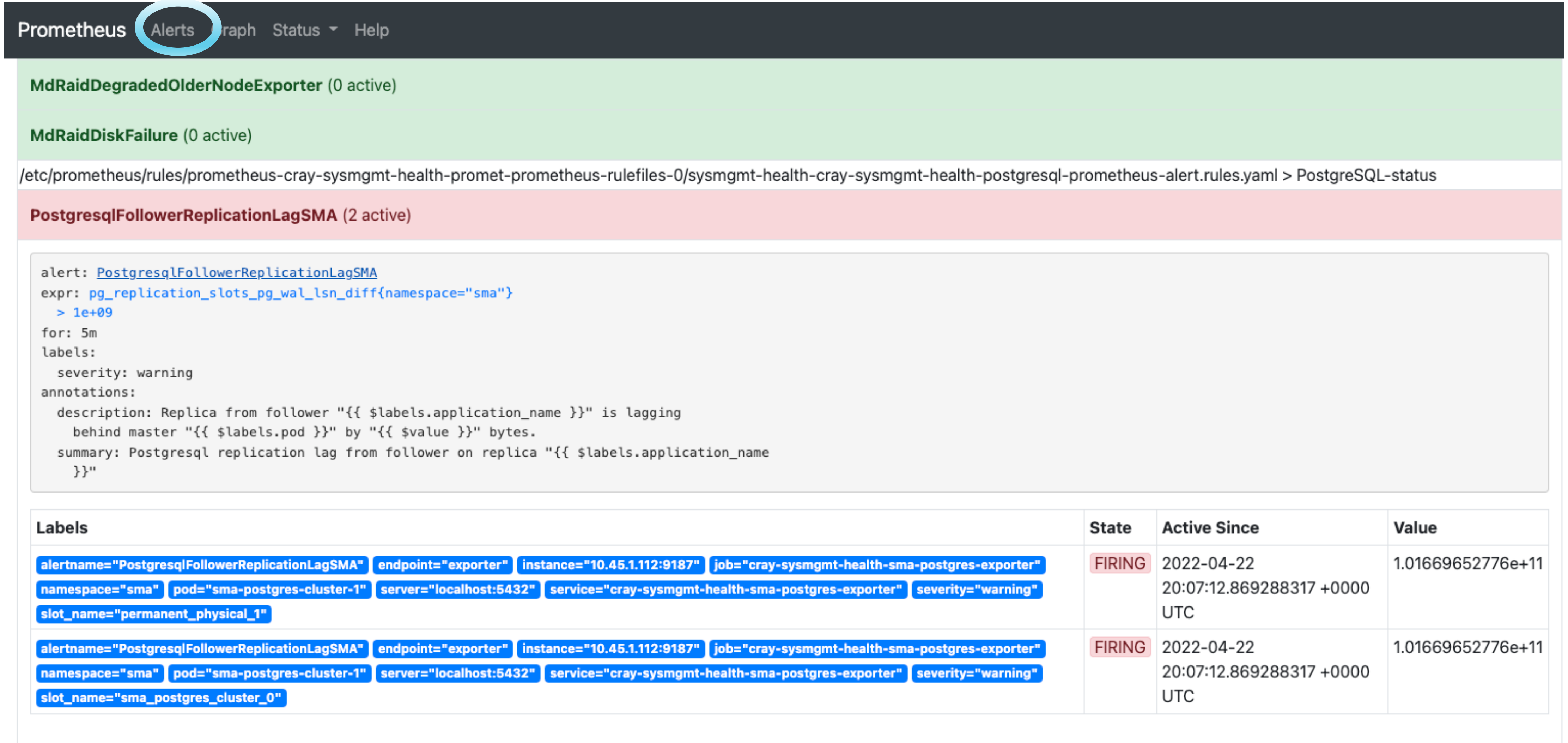
Graph of container receive packets total

Point on graph shows details for ncn-m003



https://prometheus.<systemdomain>

PROMETHEUS - ALERTS


Prometheus Alerts raph Status Help

MdRaidDegradedOlderNodeExporter (0 active)

MdRaidDiskFailure (0 active)

/etc/prometheus/rules/prometheus-cray-sysmgmt-health-promet-prometheus-rulefiles-0/sysmgmt-health-cray-sysmgmt-health-postgresql-prometheus-alert.rules.yaml > PostgreSQL-status

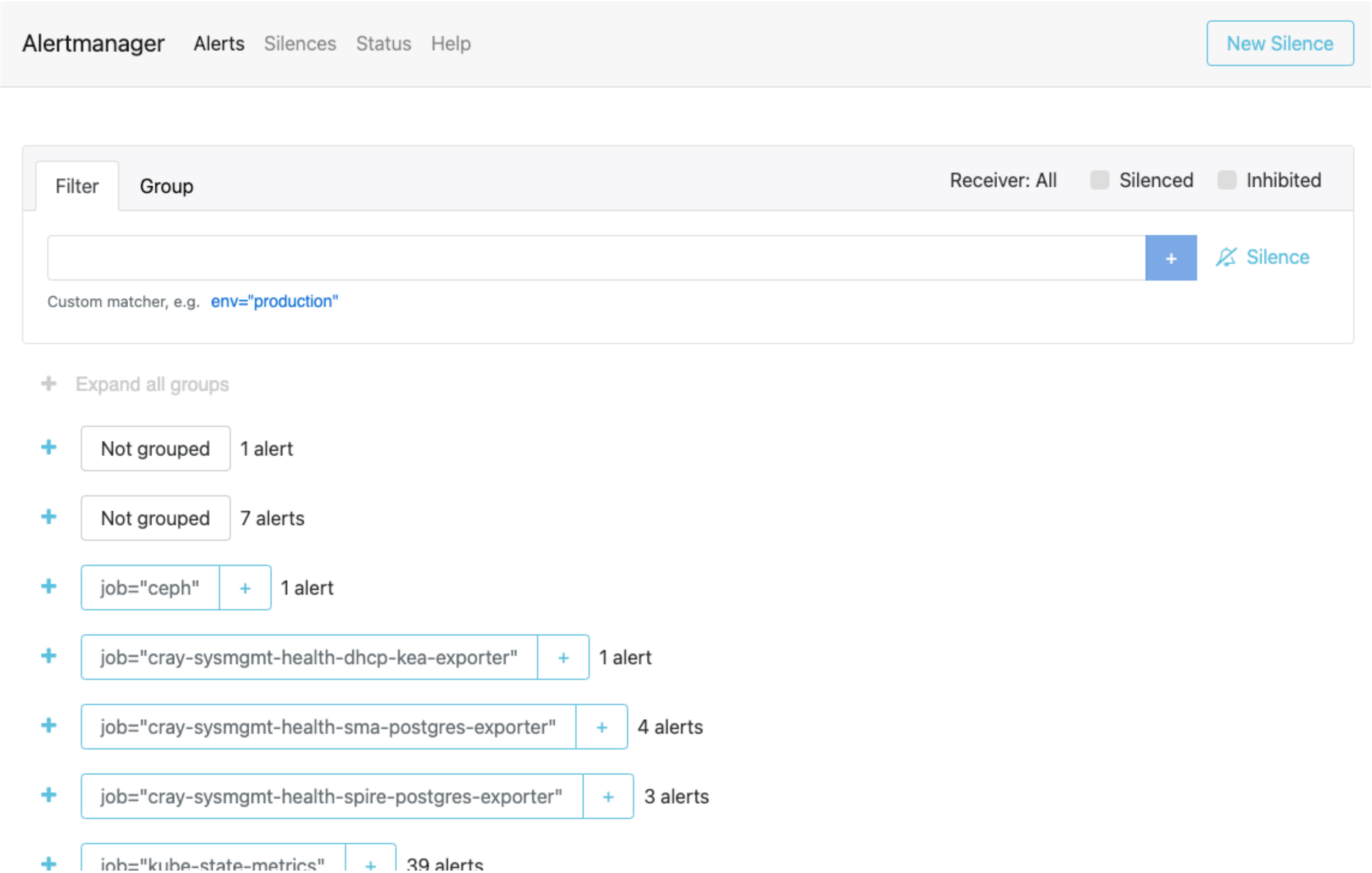
PostgresqlFollowerReplicationLagSMA (2 active)

```
alert: PostgresqlFollowerReplicationLagSMA
expr: pg_replication_slots_pg_wal_lsn_diff{namespace="sma"}
  > 1e+09
for: 5m
labels:
  severity: warning
annotations:
  description: Replica from follower "{{ $labels.application_name }}" is lagging
    behind master "{{ $labels.pod }}" by "{{ $value }}" bytes.
  summary: Postgresql replication lag from follower on replica "{{ $labels.application_name
    }}"
```

Labels	State	Active Since	Value
alertname="PostgresqlFollowerReplicationLagSMA" endpoint="exporter" instance="10.45.1.112:9187" job="cray-sysmgmt-health-sma-postgres-exporter" namespace="sma" pod="sma-postgres-cluster-1" server="localhost:5432" service="cray-sysmgmt-health-sma-postgres-exporter" severity="warning" slot_name="permanent_physical_1"	FIRING	2022-04-22 20:07:12.869288317 +0000 UTC	1.01669652776e+11
alertname="PostgresqlFollowerReplicationLagSMA" endpoint="exporter" instance="10.45.1.112:9187" job="cray-sysmgmt-health-sma-postgres-exporter" namespace="sma" pod="sma-postgres-cluster-1" server="localhost:5432" service="cray-sysmgmt-health-sma-postgres-exporter" severity="warning" slot_name="sma_postgres_cluster_0"	FIRING	2022-04-22 20:07:12.869288317 +0000 UTC	1.01669652776e+11

https://prometheus.<systemdomain>

ALERTMANAGER

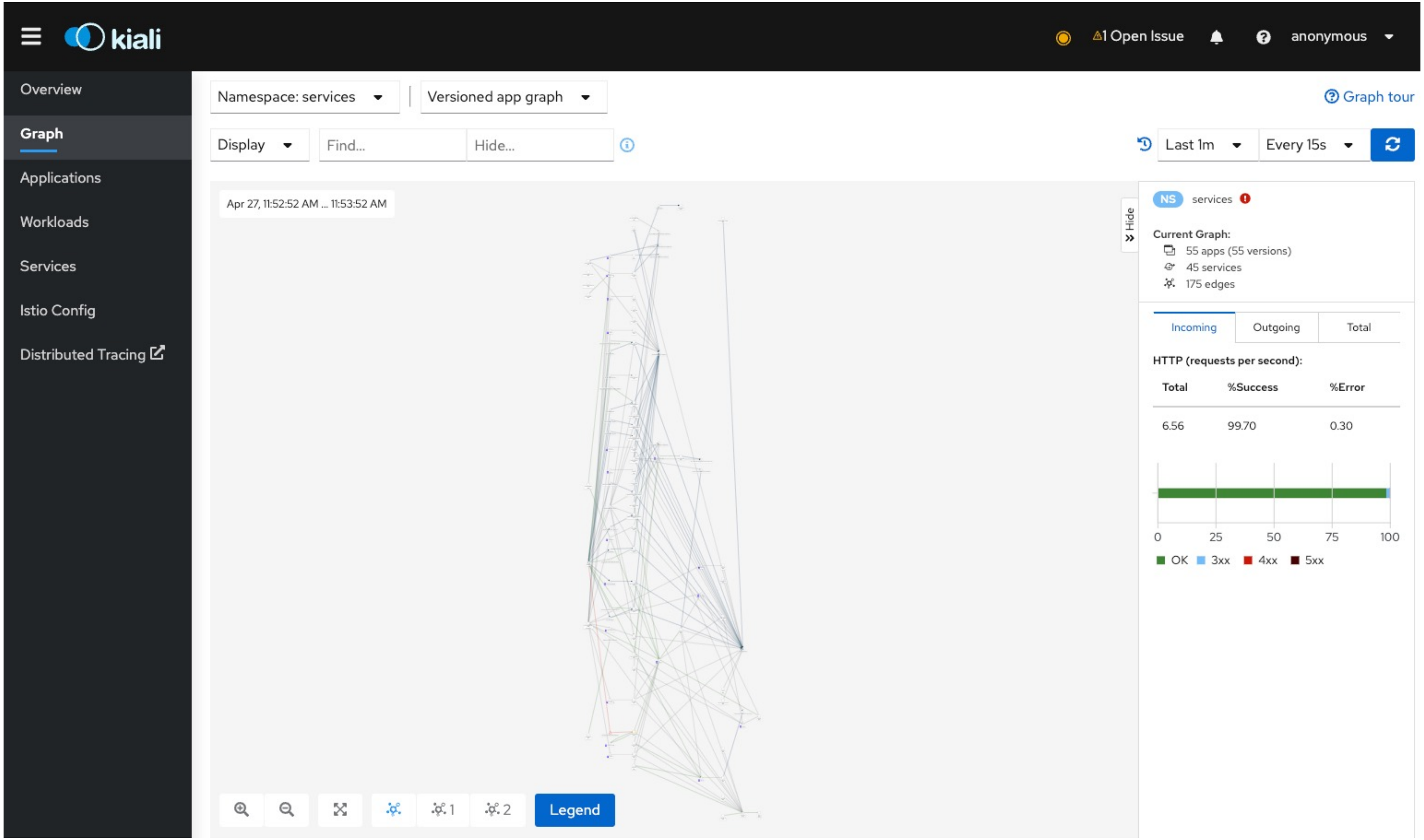


https://alertmanager.<systemdomain>

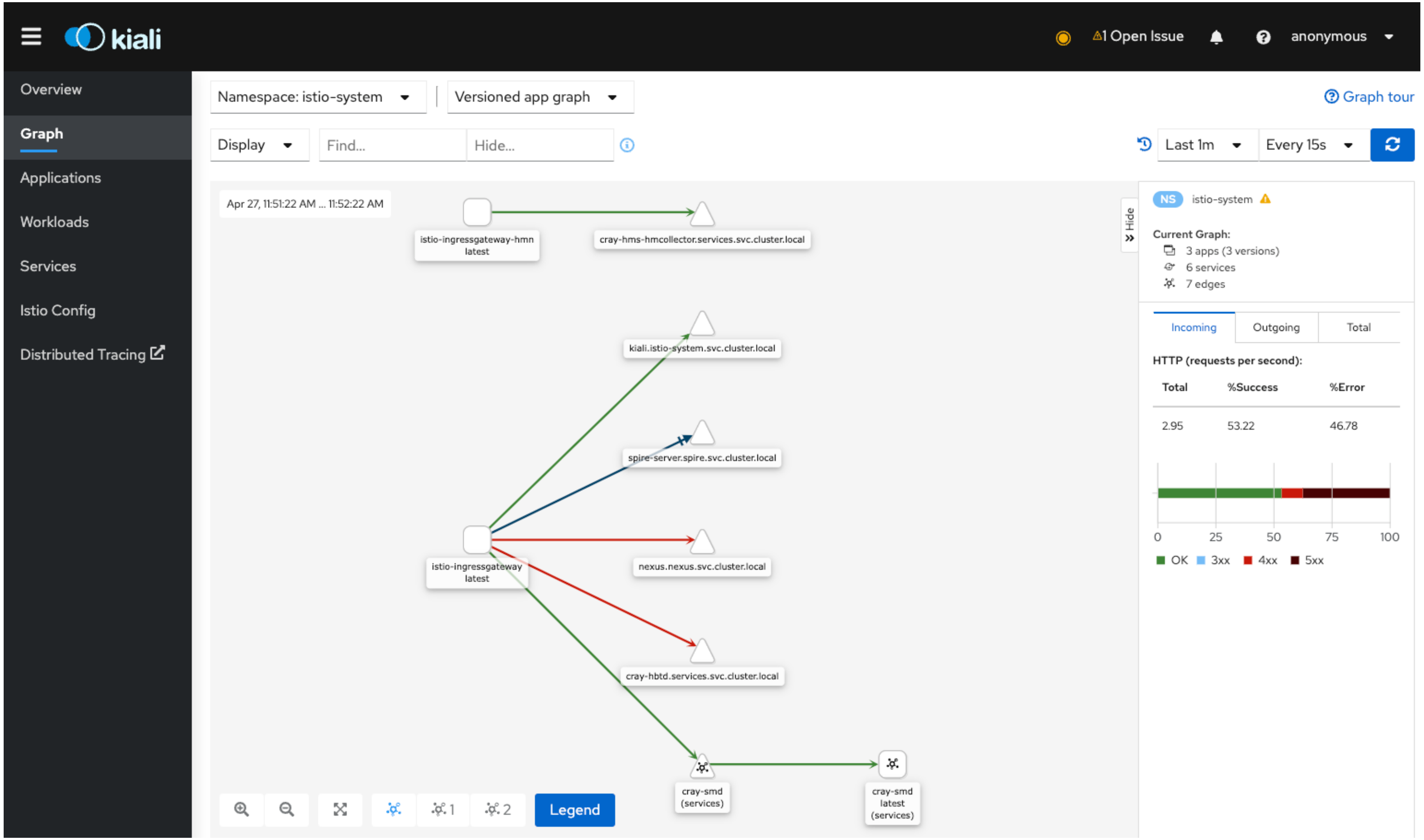
ISTIO WITH KIALI , JAEGER , AND PROMETHEUS

- Kiali
  - Observability console for Istio with service mesh configuration and validation capabilities
  - Helps you understand the structure and health of your service mesh by monitoring traffic flow to infer the topology and report errors
  - Provides detailed metrics and a basic Grafana integration, which can be used for advanced queries
  - Distributed tracing is provided by integration with Jaeger
  - https://kiali-istio.<systemdomain>
  - Documentation https://kiali.io/
- Jaeger
  - Distributed transaction monitoring
  - Performance and latency optimization
  - Root cause analysis
  - Service dependency analysis
  - Distributed context propagation
  - https://jaeger-istio.<systemdomain>
  - Documentation https://www.jaegertracing.io/
- Prometheus
  - Monitoring system and time series database
  - Record metrics that track the health of Istio and of applications within the service mesh
  - https://prometheus-istio.<systemdomain>
  - Documentation https://prometheus.io/

KIALI: SERVICES NAMESPACE

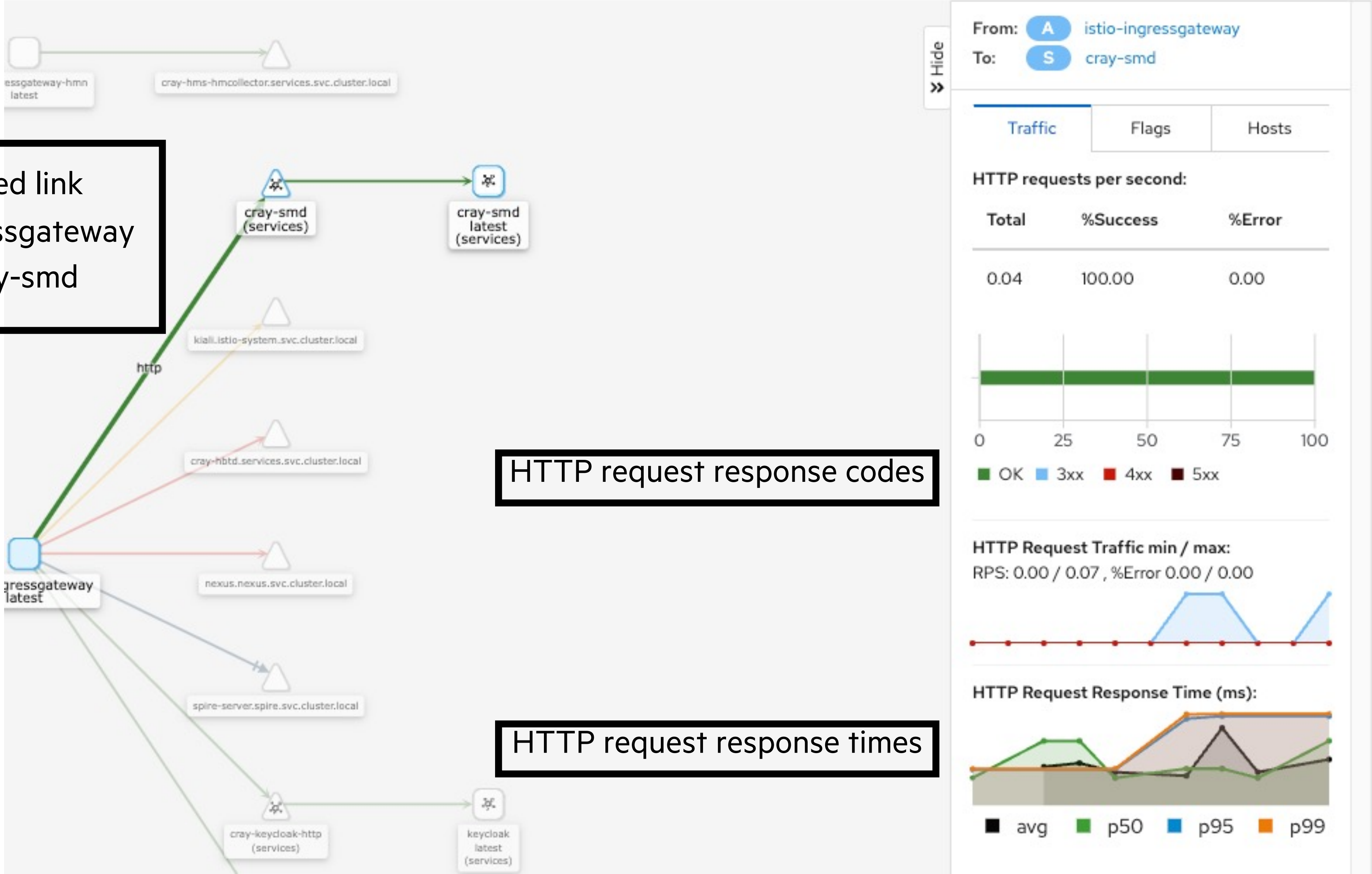


KIALI: ISTIO-SYSTEM NAMESPACE



KIALI: ISTIO-SYSTEM NAMESPACE ONE LINK

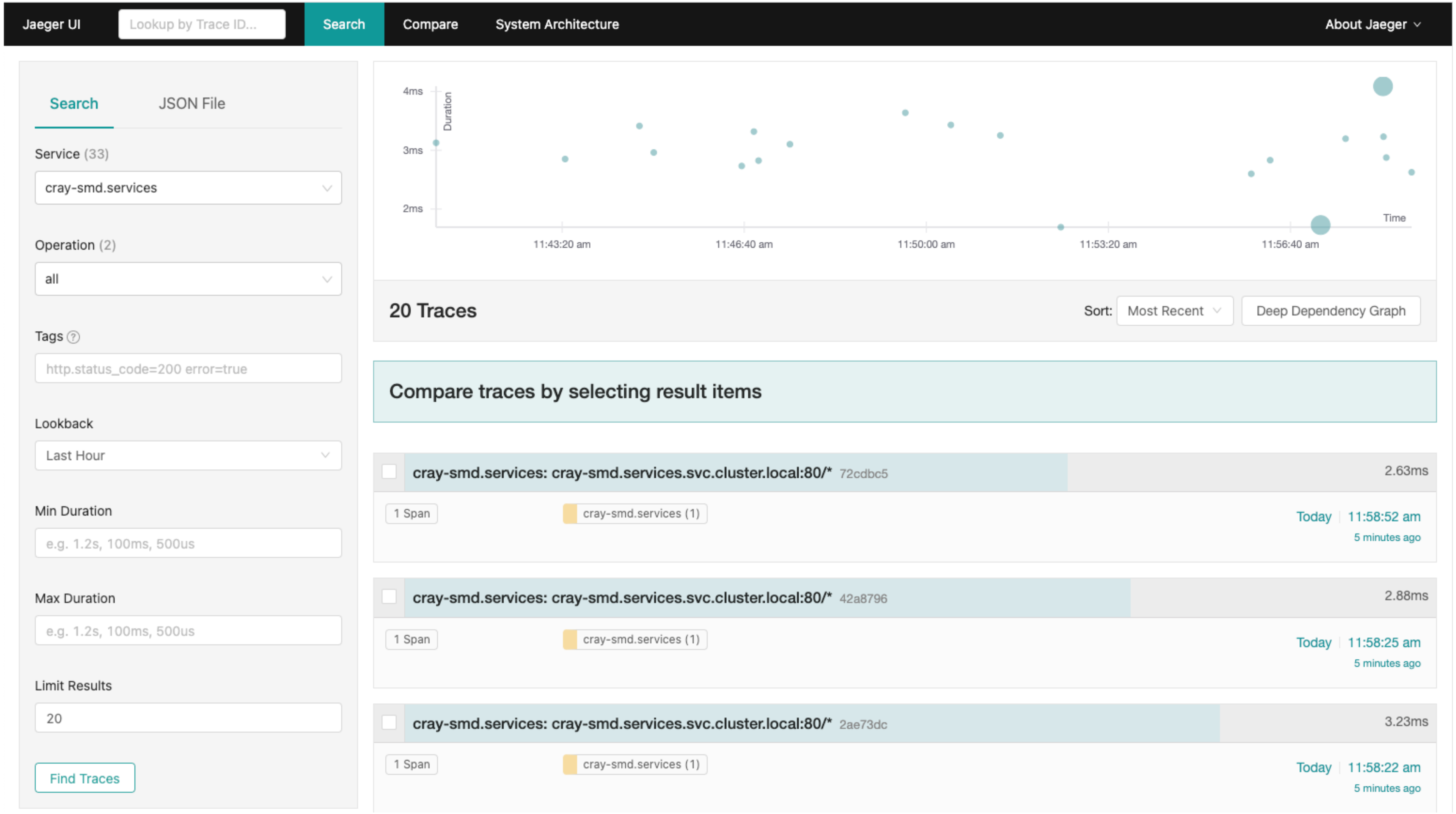
Selected link istio-ingressgateway to cray-smd



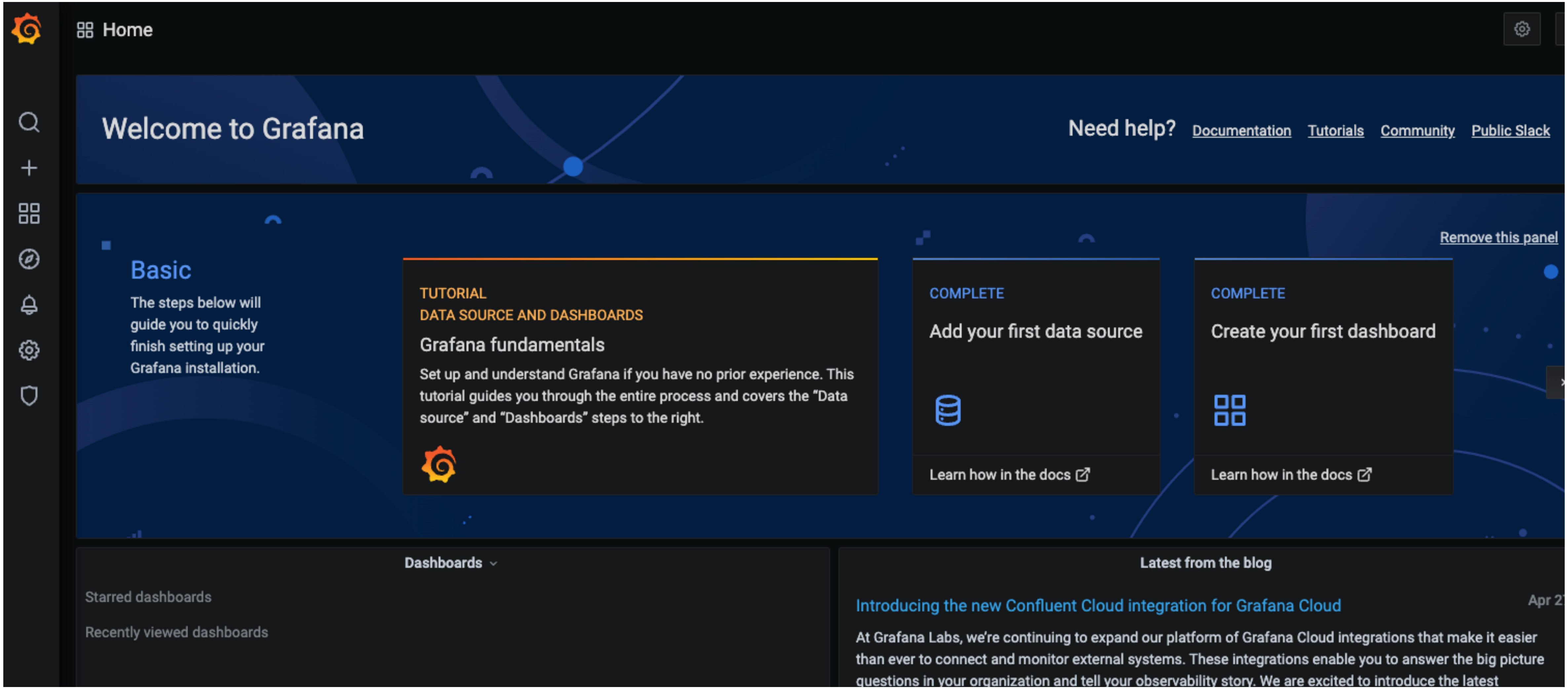
HTTP request response codes

HTTP request response times

JAEGER: CRAY-SMD-SERVICES



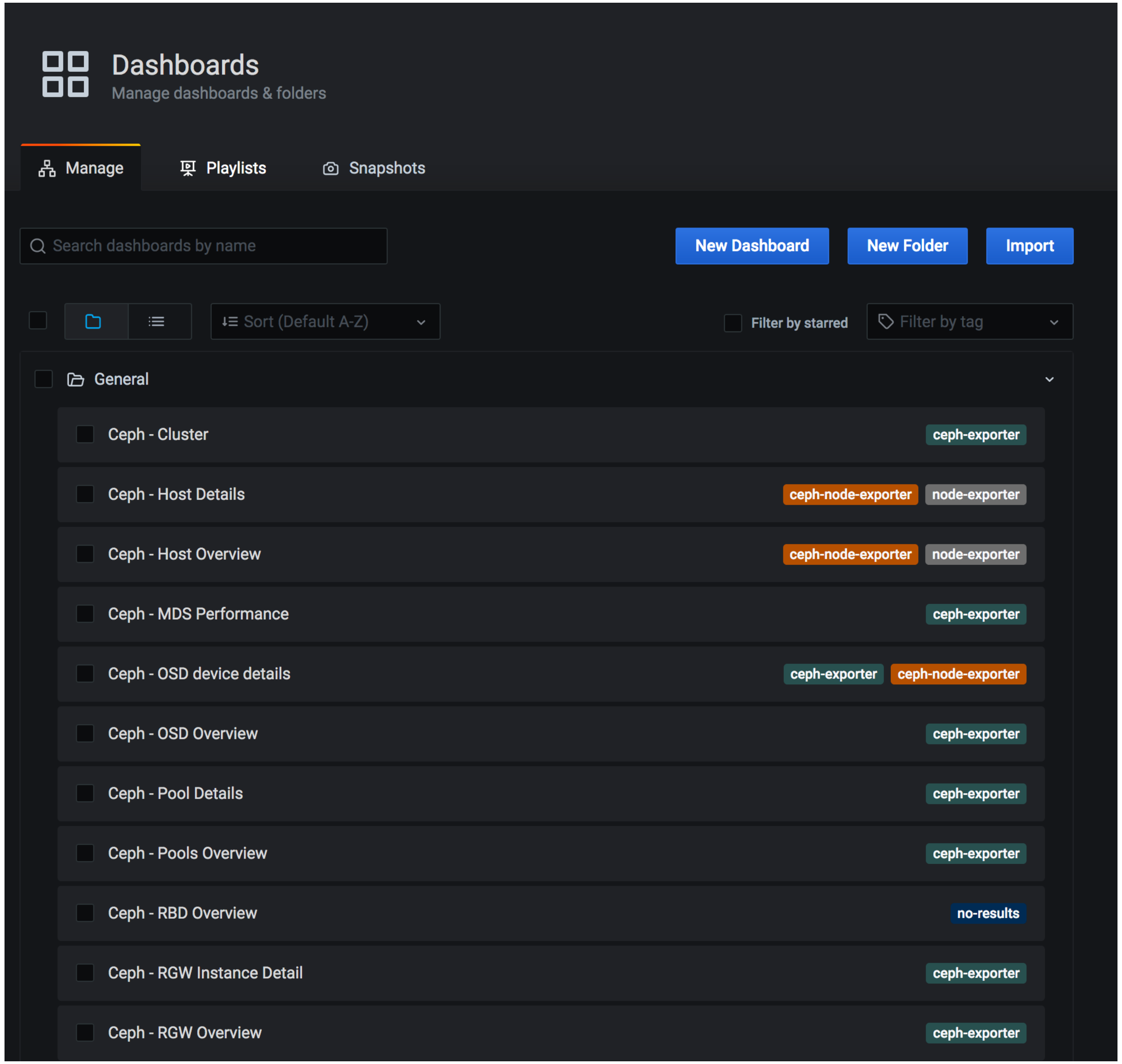
GRAFANA


Home
Welcome to Grafana
Need help? Documentation Tutorials Community Public Slack
Remove this panel
Basic
The steps below will guide you to quickly finish setting up your Grafana installation.
TUTORIAL
DATA SOURCE AND DASHBOARDS
Grafana fundamentals
Set up and understand Grafana if you have no prior experience. This tutorial guides you through the entire process and covers the "Data source" and "Dashboards" steps to the right.
COMPLETE
Add your first data source
Learn how in the docs
COMPLETE
Create your first dashboard
Learn how in the docs
Dashboards
Starred dashboards
Recently viewed dashboards
Latest from the blog
Introducing the new Confluent Cloud integration for Grafana Cloud
At Grafana Labs, we're continuing to expand our platform of Grafana Cloud integrations that make it easier than ever to connect and monitor external systems. These integrations enable you to answer the big picture

GRAFANA DASHBOARDS CATALOG

- Uses Keycloak authentication/authorization
- Secured with TLS sharing cluster certificate bundle
- About 40 included dashboards
  - Ceph
  - CoreDNS
  - Etcd
  - ETCD Clusters
  - Istio
  - Kea-dhcp
  - Kubernetes
  - Node Exporter
  - Nodes
  - PostgreSQL
  - Prometheus

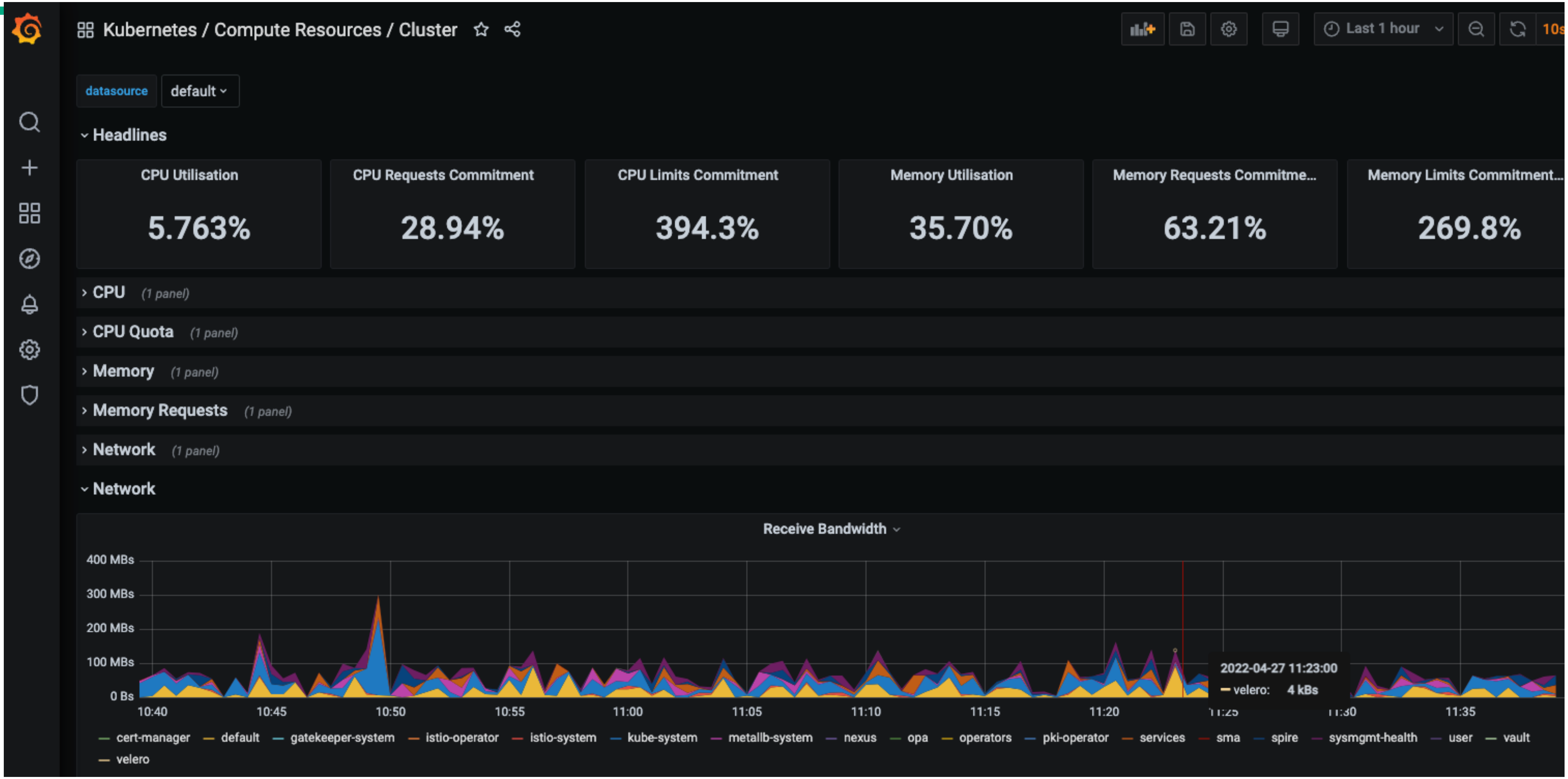
https://grafana.<systemdomain>/dashboards



GRAFANA DASHBOARDS: ETCD

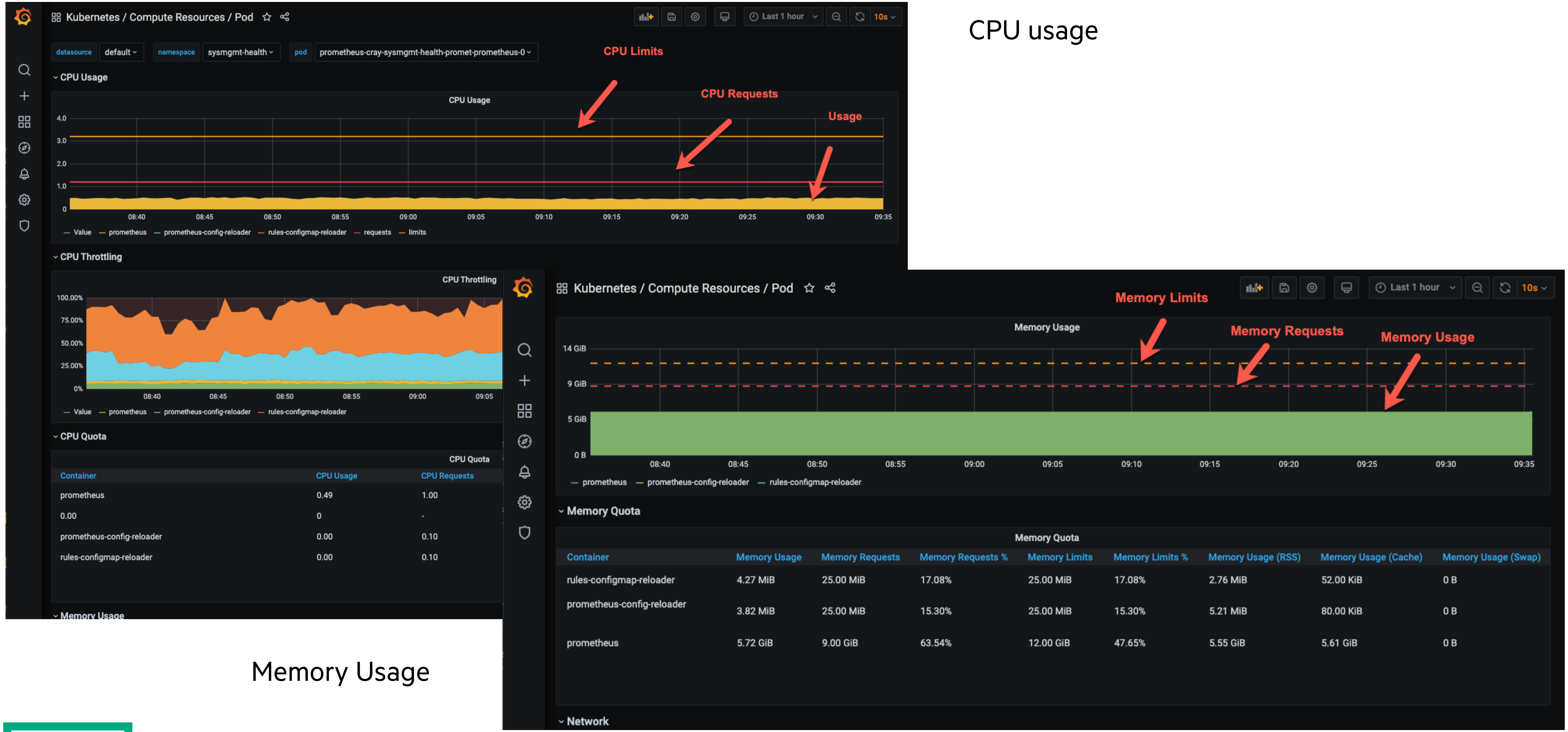
- Nodes up (quorum)
- RPC Rate
- Active Streams
- DB Size
- Disk Sync Duration
- Memory
- Client Traffic in
- Client Traffic Out
- Peer Traffic In
- Peer Traffic Out
- Raft proposals
- Total Leader Elections Per day

GRAFANA DASHBOARDS: KUBERNETES CLUSTER



GRAFANA DASHBOARDS: KUBERNETES POD REQUESTS AND LIMITS

CPU usage



Memory Usage

LOGS AND DUMPS

- Console logs and access
- Log aggregation
- Elasticsearch, Logstash, SMA-Kibana
- Dumps

CONTAINERIZED CONSOLE ACCESS

- ConMan is a serial console management program designed to support a large number of console devices and simultaneous users
- cray-console uses ConMan for interactive remote console access and console log collection
  - Automatically detects nodes which have been added or removed
  - Shared filesystem in Ceph for all cray-console pods to easily view log data
  - Console log data sent to SMA for other log processing
  - Dynamic autoscaling number of cray-console-node pods for size of system
    - Minimally, two pods are started
    - The number of PODs is scaled on
      - 750 Liquid-cooled nodes and/or 2000 "River" nodes
        - The Liquid-cooled nodes each require an ssh connection, so numbers are different.
- Log locations:
  - Logs visible in any `cray-console-node-x` pod
  - Node logs: `/var/log/conman/console.XNAME`
  - ConMan damon logs: `/var/log/conman.log`

```
ncn# kubectl get pods -A |grep cray-console
services                cray-console-data-5cd59677d9-lf4f4
services                cray-console-data-postgres-0
services                cray-console-data-postgres-1
services                cray-console-data-postgres-2
services                cray-console-node-0
services                cray-console-node-1
services                cray-console-operator-7f9894f657-5psn5
```

CONSOLE LOGS WITH CRAY-CONSOLE-NODE

```
ncn# kubectl get pods -A |grep console-node
services              cray-console-node-0        3/3     Running        1      62d
services              cray-console-node-1        3/3     Running        0      68d
ncn# kubectl -it exec -n services cray-console-node-1 -c cray-console-node -- ls /var/log/conman
console.x1000c0s1b0n0  console.x1000c3s3b0n0   console.x3000c0s20b4n0
console.x1000c0s1b0n1  console.x1000c3s3b0n1   console.x3000c0s23b1n0
console.x1000c0s1b1n0  console.x1000c3s3b1n0   console.x3000c0s23b2n0
console.x1000c0s1b1n1  console.x1000c3s3b1n1   console.x3000c0s23b3n0
console.x1000c0s5b0n0  console.x1000c5s5b0n0   console.x3000c0s23b4n0
console.x1000c0s5b0n1  console.x1000c5s5b0n1   console.x3000c0s25b1n0
console.x1000c0s5b1n0  console.x1000c5s5b1n0   console.x3000c0s25b2n0
console.x1000c0s5b1n1  console.x1000c5s5b1n1   console.x3000c0s25b3n0
console.x1000c0s7b0n0  console.x1000c7s7b0n0   console.x3000c0s25b4n0
```

Each pod sees all the console files, only one cray-console-node pod is managing that node and writing its log file

```
ncn# kubectl -it exec -n services cray-console-node-1 -c cray-console-node – \
tail -f /var/log/conman/console.x1000c0s1b0n0
```

Can view log without entering pod

```
ncn# kubectl -it exec -n services cray-console-node-1 -c cray-console-node -- /bin/bash
cray-console-node-1-pod# grep –i error /var/log/conman/console.x1000c0s1b0n0
```

Can view log by entering pod

- Access Console Log Data Via the System Monitoring Framework (SMF)
  https://github.com/Cray-HPE/docs-csm/blob/release/1.0/operations/conman/Access_Console_Log_Data_Via_the_System_Monitoring_Framework_SMF.md

INTERACTIVE CONSOLE EXAMPLE (LONG)

- To join the console, use `conman -j`
  - Retrieve the `cray-console-operator` pod ID

```
ncn# CONPOD=$(kubectl get pods -n services \
    -o wide|grep cray-console-operator|awk '{print $1}')
ncn# echo $CONPOD
cray-console-operator-79bf95964-qpcpp
```

  - Set the `XNAME` variable to the xname of the node whose console you wish to open

```
ncn# XNAME=x1000c0s0b0n0
```

  - Find the `cray-console-node` pod that is managing that node

```
ncn# NODEPOD=$(kubectl -n services exec $CONPOD -c cray-console-operator \
-- sh -c "/app/get-node $XNAME" | jq .podname | sed 's/"//g')
ncn# echo $NODEPOD
cray-console-node-1
```

  - Connect to the node's console using ConMan on the `cray-console-node` pod you found

```
ncn# kubectl exec -it -n services $NODEPOD -- conman -j $XNAME
<ConMan> Connection to console [x1000c0s0b0] opened.
nid000001 login:
```

- To exit console use `&.` command

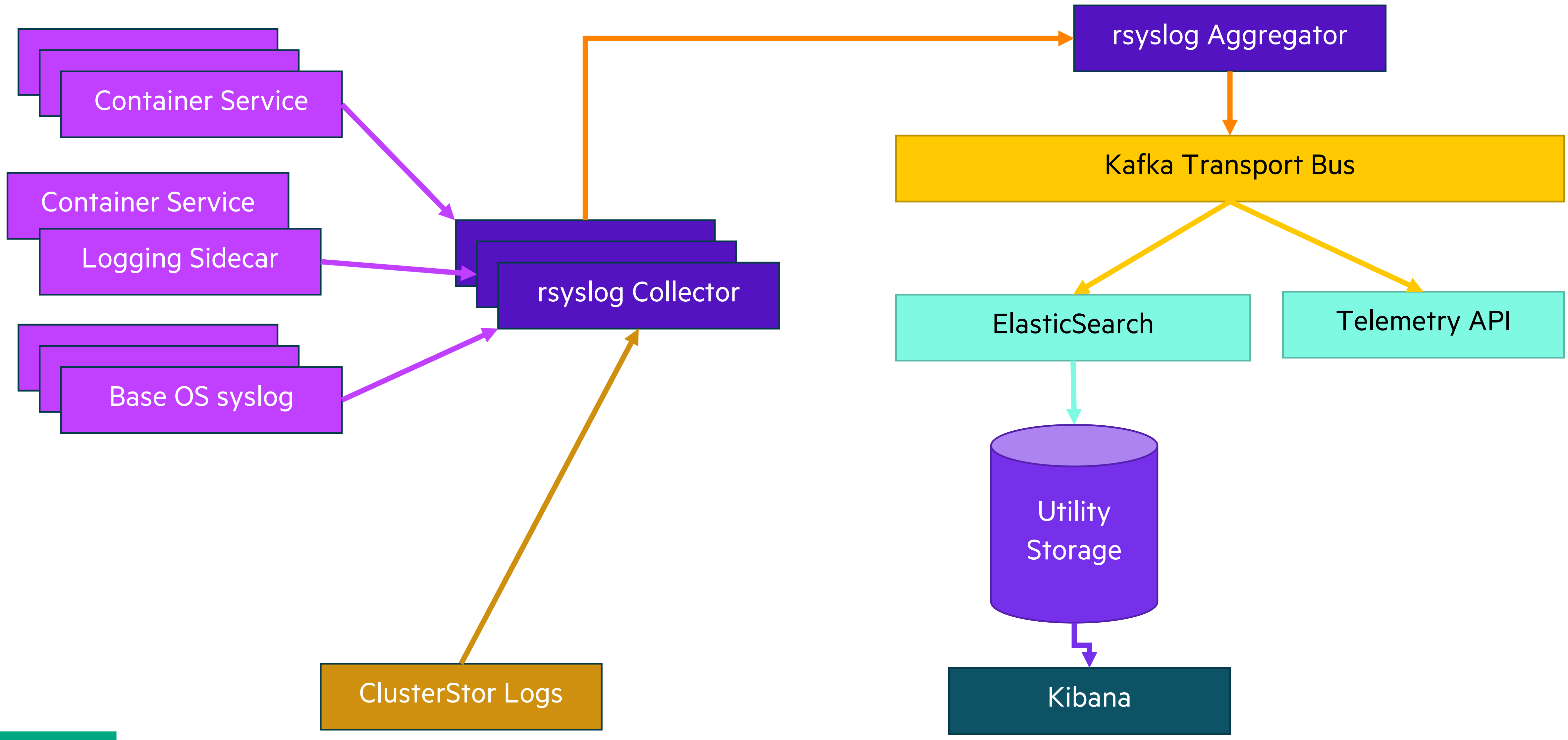
INTERACTIVE CONSOLE EXAMPLE (SHORT)

- Alternate form of previous slide

```
ncn# ConsoleJ () { XNAME=$@; CONPOD=$(kubectl get pods -n services \
| grep cray-console-operator|awk '{print $1}'); \
NODEPOD=$(kubectl exec -n services -c cray-console-operator $CONPOD \
-- sh -c "/app/get-node $XNAME" | jq .podname | tr -d '"'); \
echo conpod = $CONPOD nodepod = $NODEPOD; \
kubectl exec -n services -it $NODEPOD -c cray-console-node \
-- conman -j $XNAME }
ncn# ConsoleJ x1000c0s0b0n0
<ConMan> Connection to console [x1000c0s0b0n0] opened.
nid000001 login:
```

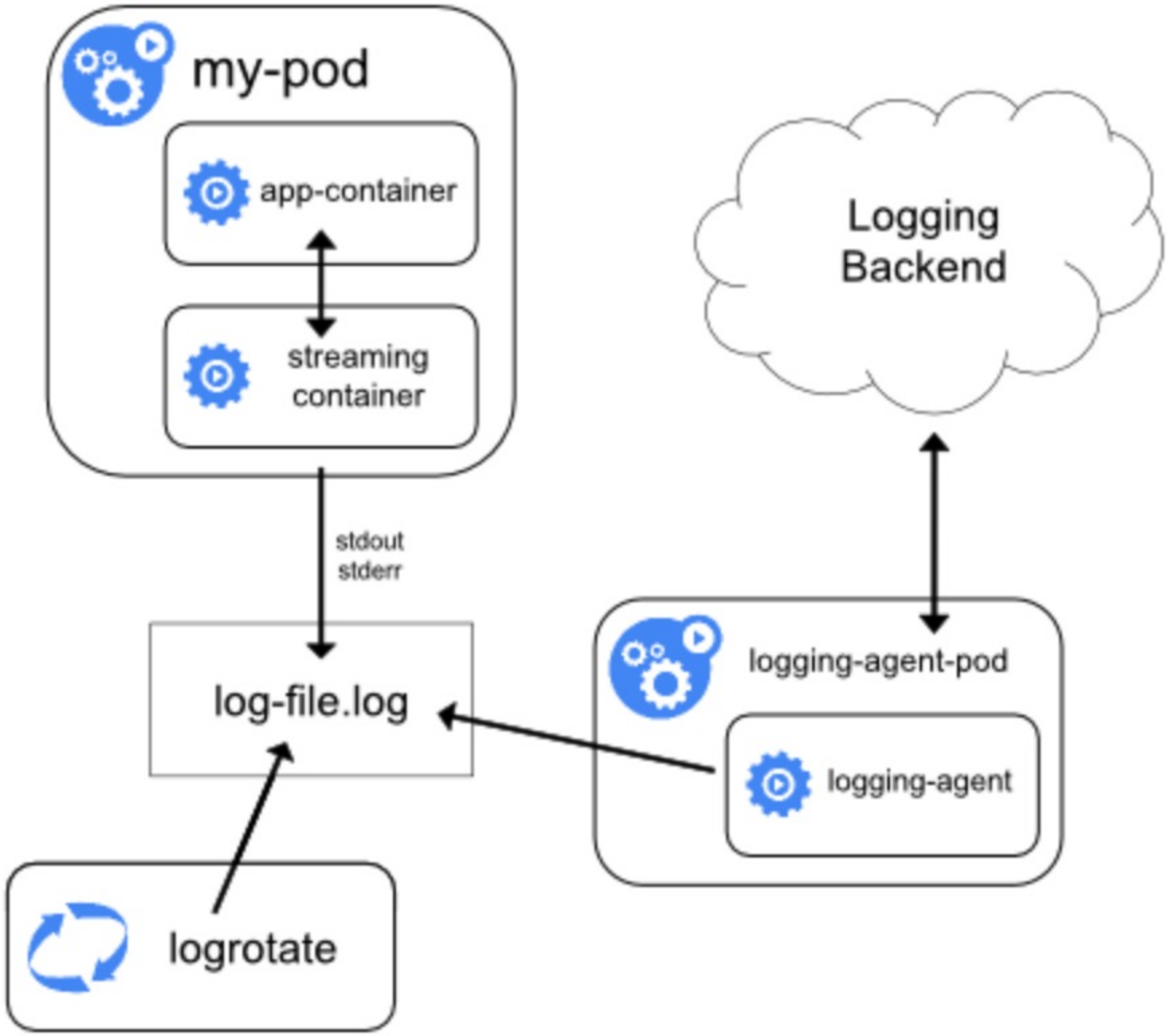
- To exit console use `&.` command
- To view the console read-only instead of joining it read-write, use `conman -m`

LOG AGGREGATION



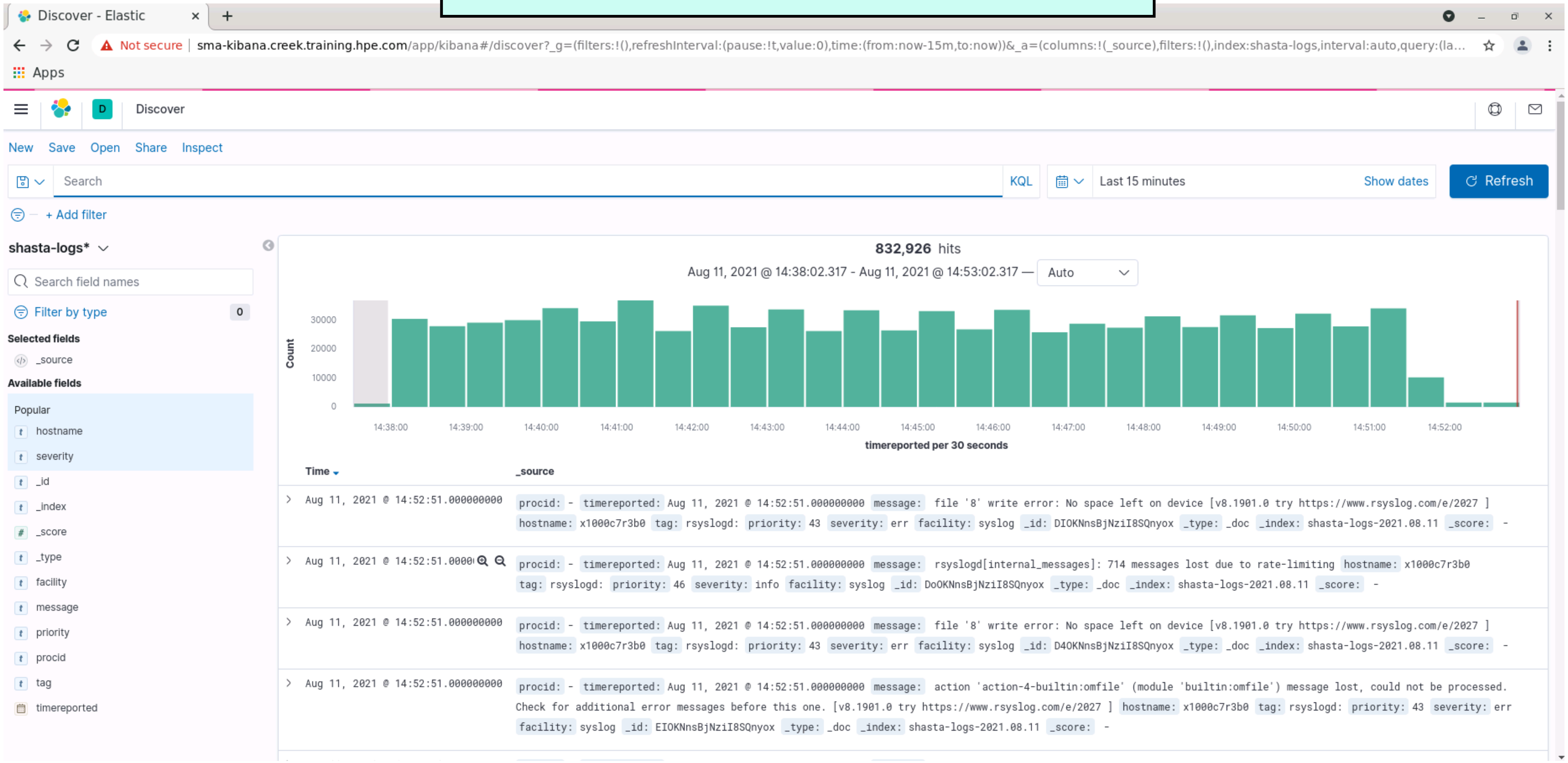
KUBERNETES CONTAINER SIDECAR

- Sidecar runs a logging agent
- Picks up logs from application containers in pod
- Can separate several logs streams from different parts of the application



SMA-KIBANA

https://sma-kibana.creek.training.hpe.com/app/kibana



SAT DASHBOARDS IN SMA-KIBANA

Dashboard	Short Description	Long Description
sat-aer	AER corrected	Corrected Advanced Error Reporting messages from PCI Express devices on each node
sat-aer	AER fatal	Fatal Advanced Error Reporting messages from PCI Express devices on each node
sat-atom	ATOM failures	Application Task Orchestration and Management tests are run on a node when a job finishes. Test failures are logged
sat-atom	ATOM admindown	ATOM test failures can result in nodes being marked admindown. An admindown node is not available for job launch
sat-heartbeat	Heartbeat loss events	Heartbeat loss event messages reported by the hbtd pods that monitor for heartbeats across nodes in the system
sat-kernel	Kernel assertions	The kernel software performs a failed assertion when some condition represents a serious fault. The node goes down
sat-kernel	Kernel panics	The kernel panics when something is seriously wrong. The node goes down
sat-kernel	Lustre bugs (LBUGs)	The Lustre software in the kernel stack performs a failed assertion when some condition related to file system logic represents a serious fault. The node goes down
sat-kernel	CPU stalls	CPU stalls are serous conditions that can reduce node performance, and sometimes cause a node to go down. Technically these are Read-Copy-Update stalls where software in the kernel stack holds onto memory for too long
sat-kernel	Out of memory	An Out Of Memory (OOM) condition has occurred. The kernel must kill a process to continue. The kernel will select an expendable process when possible. If there is no expendable process the node usually goes down in some manner. Even if there are expendable processes the job is likely to be impacted. OOM conditions are best avoided
sat-mce	MCE	Machine Check Exceptions (MCE) are errors detected at the processor level
sat-rasdaemon	rasdaemon errors	Errors from the rasdaemon service on nodes. The rasdaemon service is the Reliability, Availability, and Serviceability Daemon, and it is intended to collect all hardware error events reported by the linux kernel, including PCI and MCE errors
sat-rasdaemon	rasdaemon messages	All messages from the rasdaemon service on nodes

NODE MEMORY DUMP (NMD)

- Standard Linux kdump mechanism
  - Uses `kexec` for booting into the dump-capture kernel (`kdump boot`) immediately after kernel crash
  - Standard `kdump` not scalable to large systems
    - Standard, each node decides on its own to produce a node memory dump
  - Needs a service to initiate dumps of selected nodes
- NMD controls the `kdump` process of the panicked node
  - Initiates and monitors node memory dumps remotely
  - Operates in the management plane
  - Generates the node memory dump only when request is received from the NMD service on the SMS
  - Registers a dump discovery callback with the System Diagnostic Service (SDS)
  - SDS can download existing dumps or create a new dump and download it

```
ncn# cray nmd dumps --help
Usage: cray nmd dumps [OPTIONS] COMMAND [ARGS]...
Options:
  --help  Show this message and exit.
Commands:
  create
  delete
  describe
  list
```

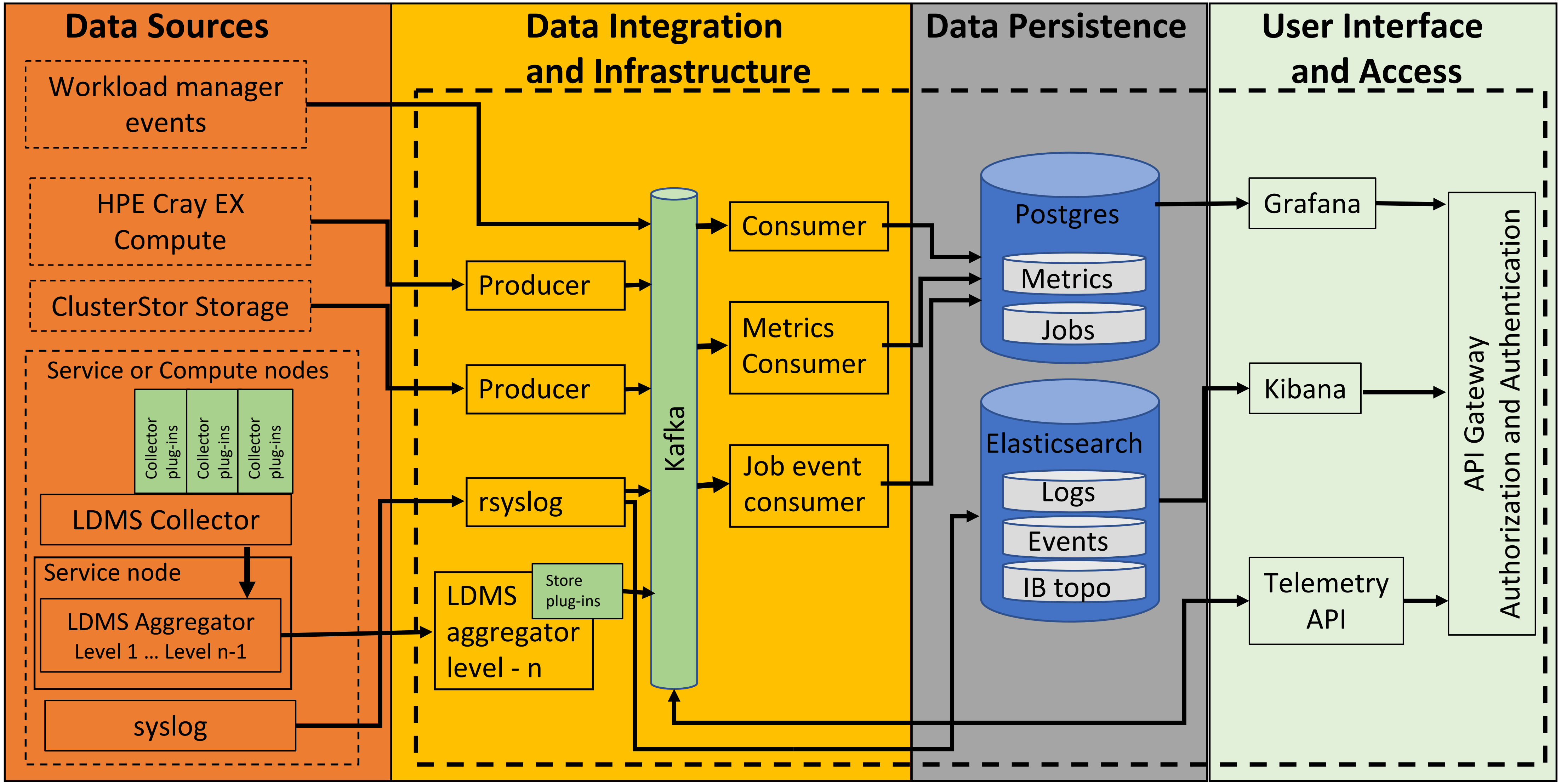
MONITORING

- System Monitoring Framework
- LDMS
- Telemetry API
- SMA-Grafana
- Dashboards

SYSTEM MONITORING FRAMEWORK

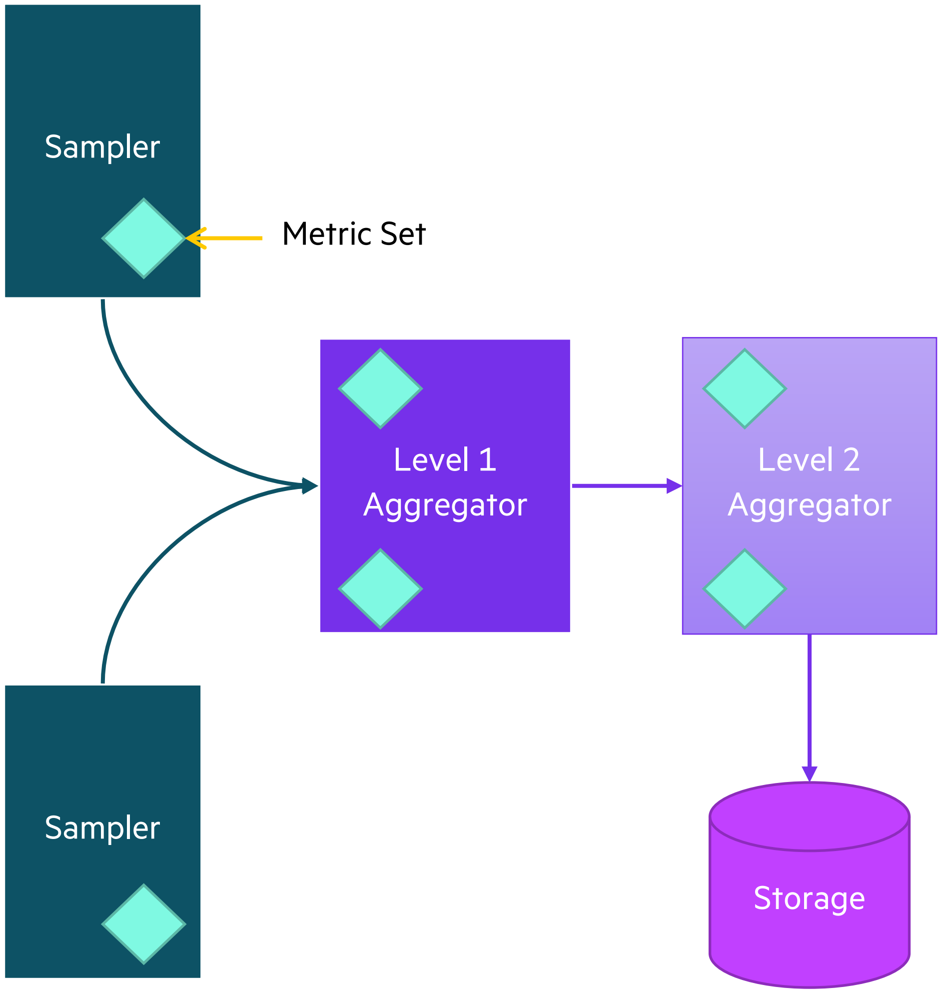
- Tightly-integrated monitoring system
- Provides detailed telemetry information from multiple subsystems:
  - Fabric
  - Environmental
  - Network
  - Storage
  - Operating systems (vmstat and iostat metrics)
- Incorporates the context necessary to understand telemetry data
- Feeds into a common message bus (Kafka), persistence, and minimal UI infrastructure
  - SAT has user interfaces that integrate with the System Monitoring Framework

SYSTEM MONITORING FRAMEWORK DIAGRAM

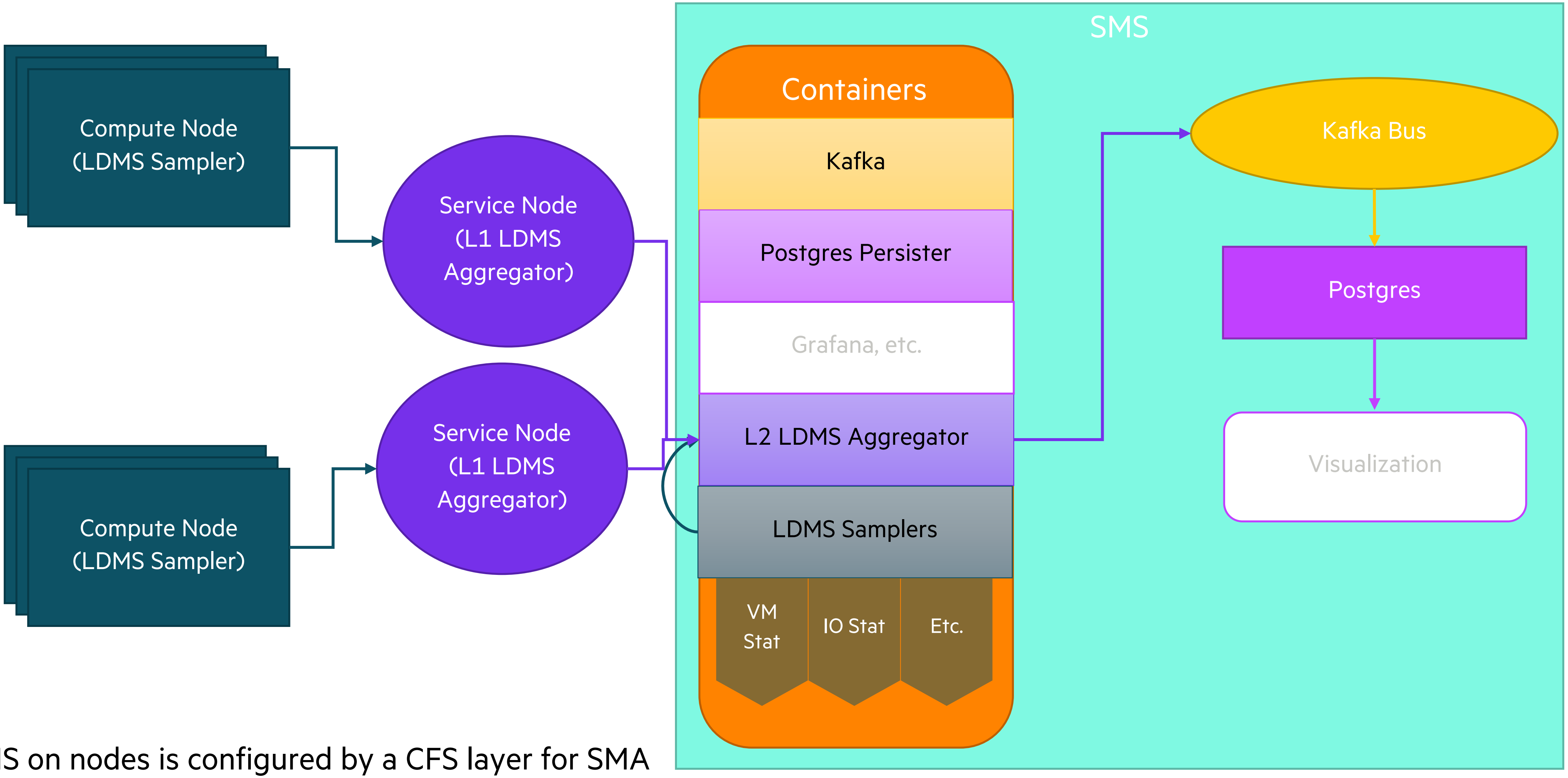


LIGHTWEIGHT DISTRIBUTED METRIC SERVICE (LDMS)

- Developed by Sandia National Lab for Blue Waters Cray XE/XK
- Distributed data collection, transport, and storage tool
- **Samplers** run one or more sampling plugins that periodically sample data on monitored nodes
  - Defines a metric set (a collection of metrics)
  - HA configuration supported
- **Aggregators** periodically collect data in a pull fashion from samplers or other aggregators
- **Storage** plugins periodically write in MySQL or flat file (file per metric name or CSV file per metric set)
  - Incomplete or not updated metric set data is not written to storage



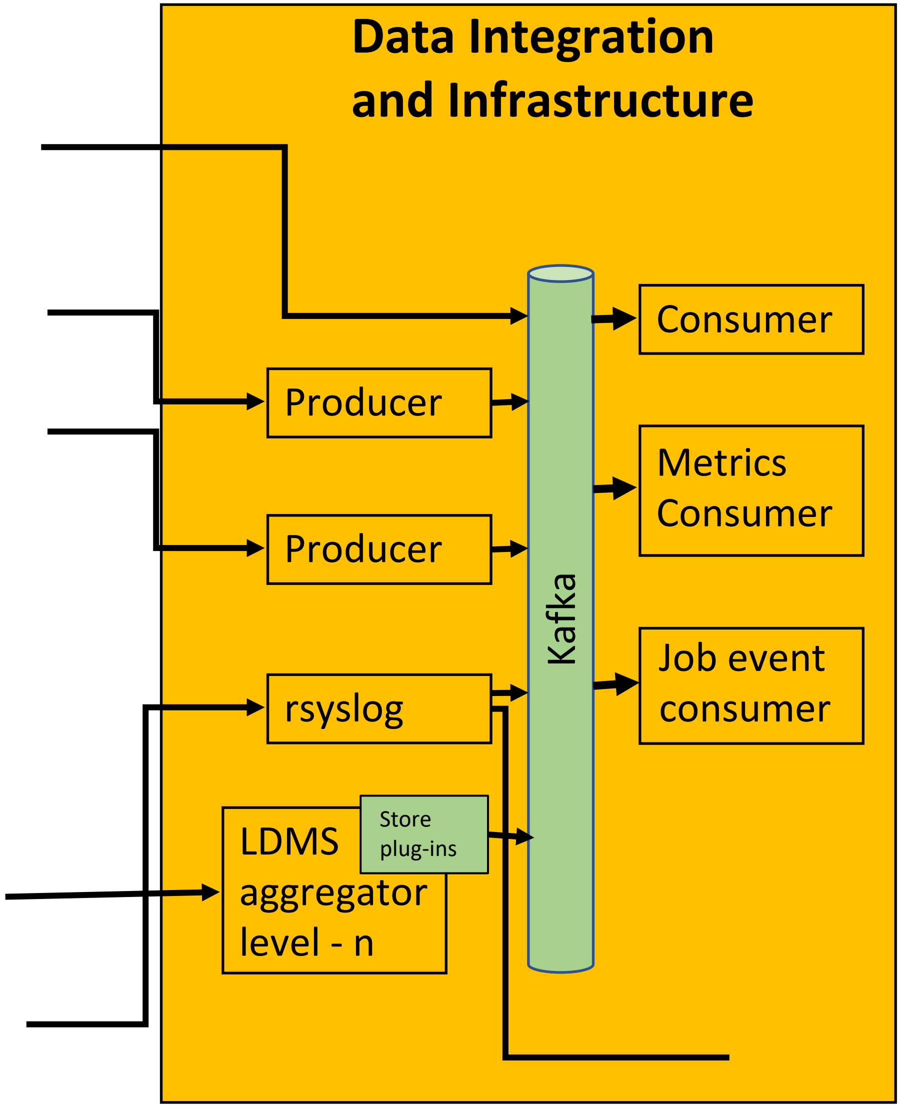
LDMS



- LDMS on nodes is configured by a CFS layer for SMA

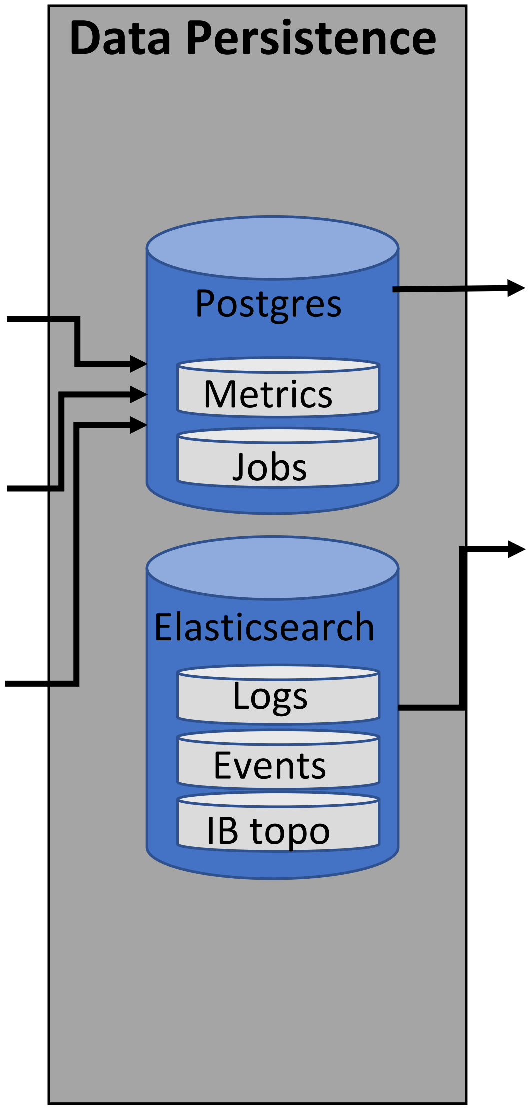
DATA INTEGRATION AND INFRASTRUCTURE LAYER

- Uses a distributed streaming platform to publish and subscribe to streams of records
- Apache Kafka
  - A distributed publish-subscribe messaging system
  - Easy to scale horizontally
  - Supports multiple subscribers and balances consumers during failure
  - Persists messages on disk
  - Supports multiple client-side APIs for consumers and producers
  - Commonly referred to as the “Kafka Bus”



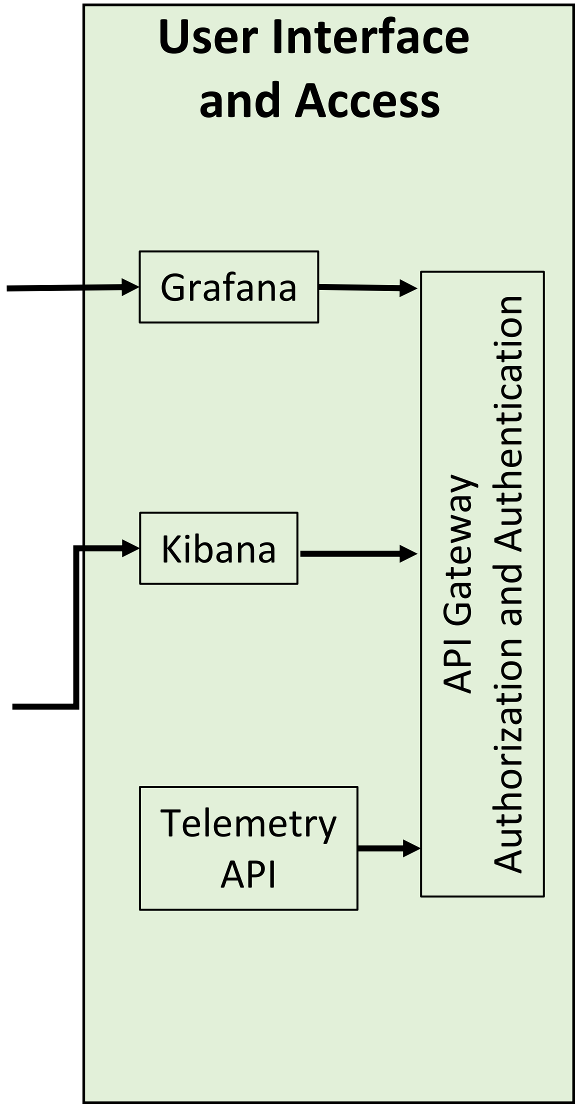
DATA PERSISTENCE LAYER

- Store telemetry data from Cray defined producers, collectors and aggregators
  - It is possible for customers to develop their own data collectors but the data they collect would not be stored in the SMF data persistence databases
    - Data from custom collectors can be streamed via the Telemetry API
- Two main responsibilities:
  - Time scale database (TSDB) optimized for handling time series data
  - Convert raw data into internal documents and store them with full text search
- Two main technologies:
  - Postgres for TSDB
  - Elasticsearch search engine
- Administrators and users should NOT attempt to read or update these databases directly



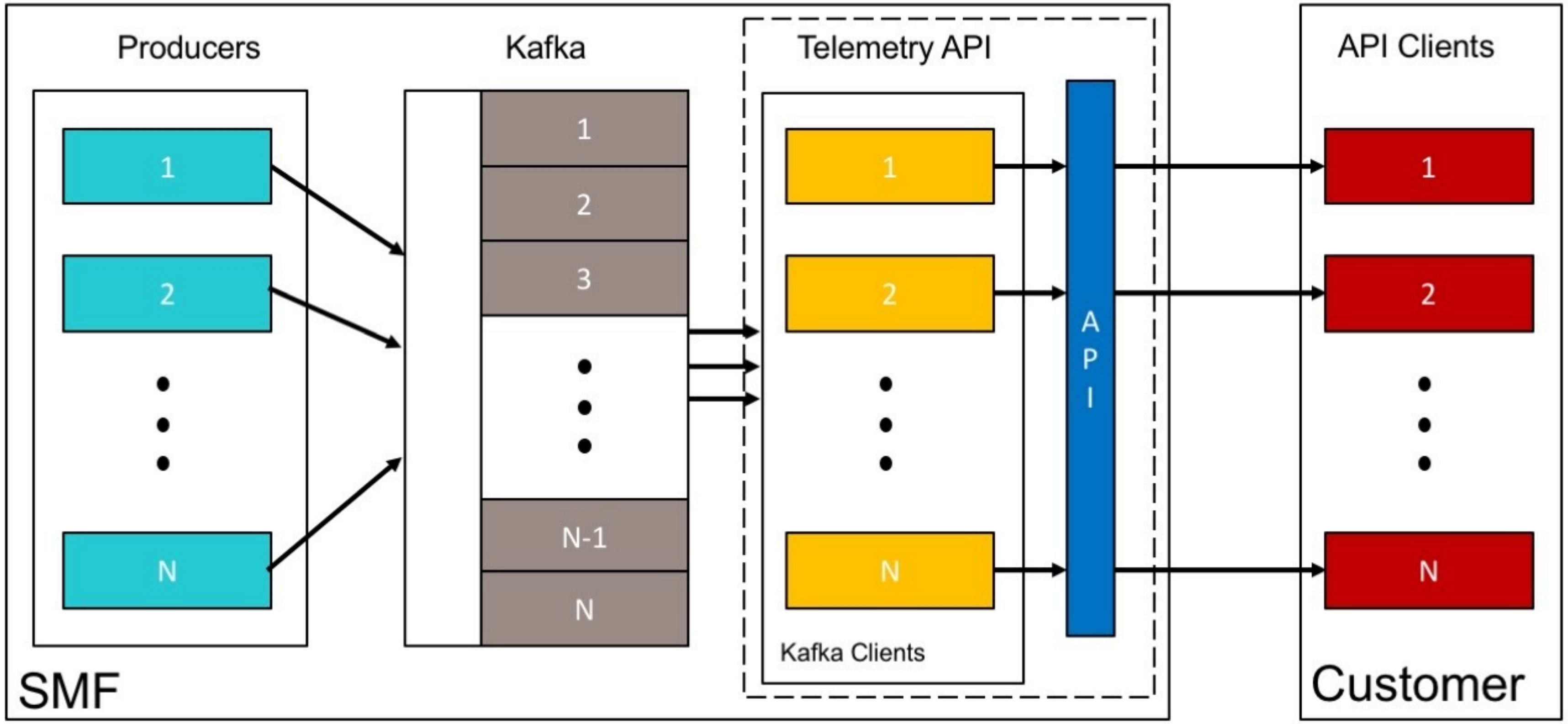
USER INTERFACE AND ACCESS LAYER

- Provides limited end user access to data stored in the SMF
- Allows consumption of streaming data and data that was persisted
  - Creation of custom graphs and panels
  - Generation of custom tables and search dialog boxes
  - Notification generation for metrics in the form of emails, alarms, alerts.
    - Metrics coming in version 1.2
    - Notifications for log data will be later
- LDMS, IO stat, and attached ClusterStor metrics via Grafana
- Log analysis via Kibana
- AuthN and AuthZ provided by API gateway and Keycloak
- Telemetry API used for access to streaming telemetry and data stored in the Data Persistence layer
  - Telemetry API is the only recommended way to pull data from the Kafka bus



TELEMETRY API

Provides access to metrics
Accessible through a RESTful JSON interface
Authenticated using bearer tokens, and token must be included in all HTML requests to the API
Streams telemetry data to clients using Server-Side Events (SSE)



ACCESSING THE TELEMETRY API WITH CURL

```
ncn# CLIENT_SECRET=`kubectl get secrets admin-client-auth -o jsonpath='{.data.client-secret}' | base64 -d`
ncn# TOKEN=$(curl -s -d grant_type=client_credentials -d client_id=admin-client \
-d client_secret=${CLIENT_SECRET} \
https://api-gw-service-nmn.local/keycloak/realms/shasta/protocol/openid-connect/token)
ncn# ACCESS_TOKEN=$(echo ${TOKEN} | jq -r .access_token)
ncn# curl -k -s -H "Authorization: Bearer ${ACCESS_TOKEN}" \
https://api-gw-service-nmn.local/apis/sma-telemetry-api/v1/ping |jq
{
  "api_version": "v1",
  "timestamp": 1591990968
}
ncn# curl -k -s  -H "Authorization: Bearer ${ACCESS_TOKEN}" \
https://api-gw-service-nmn.local/apis/sma-telemetry-api/v1/stream | jq '' |head
{
  "streams": [
    {
      "name": "cray-node",
      "scale_factor": 4
    },
    {
      "name": "cray-logs-clusterstor",
      "scale_factor": 4
    },
```

READING FROM THE TELEMETRY API WITH CURL

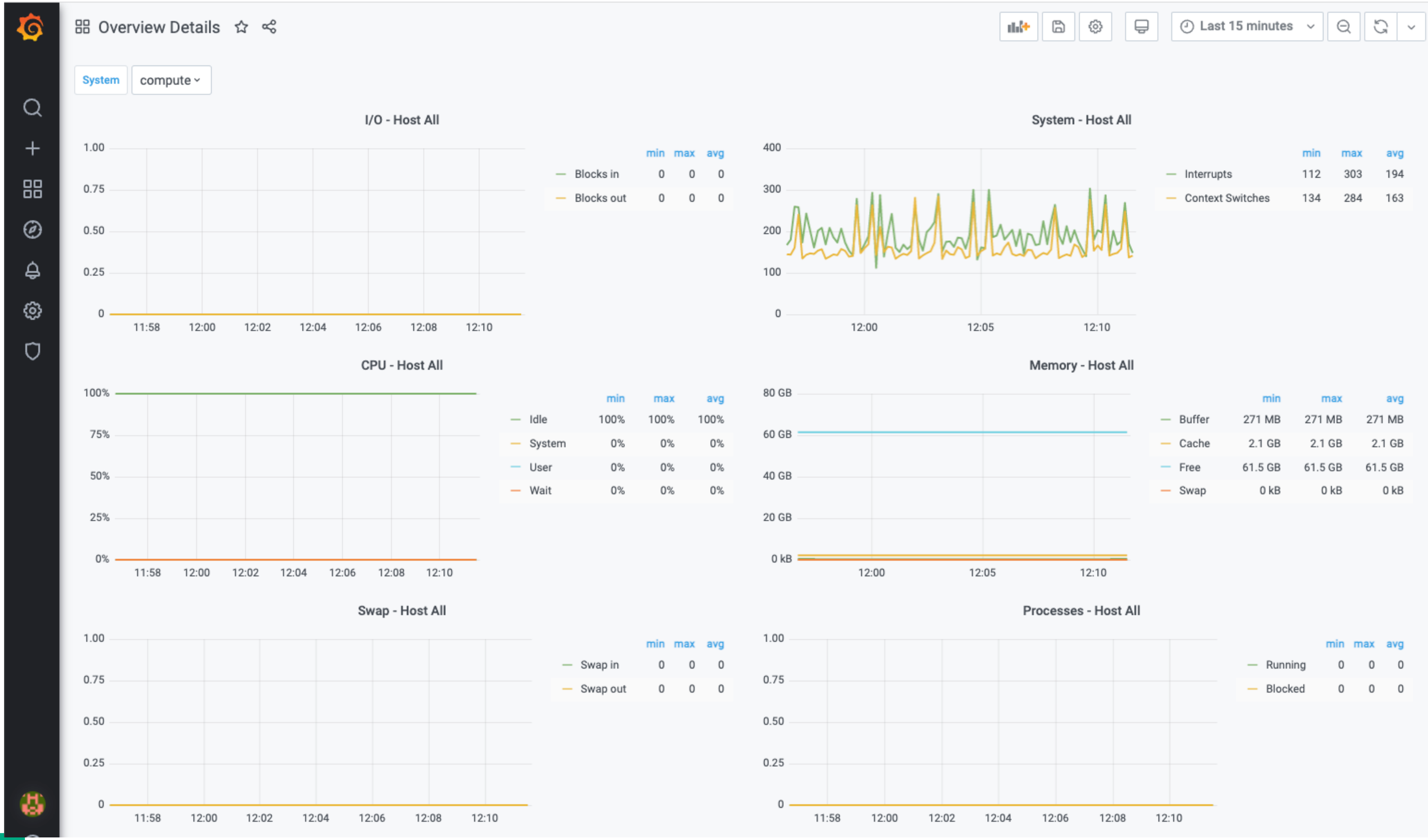
```
ncn# curl -ks --compressed -H "Authorization: Bearer ${ACCESS_TOKEN}" \
https://api-gw-service-nmn.local/apis/sma-telemetry-api/v1/stream/cray-node |head -3 \
| tail -1 | fold -80 | head -5
data: { "metrics": { "messages": [{"metric":{"name":"cray_storage.cray_vmstat.me
m_swpd","dimensions":{"product":"shasta","system":"compute","service":"ldms","co
mponent":"cray_vmstat","hostname":"nid001255","cname":"x1000c7s7b1n1","job_id":"
0"},"timestamp":1599767510102,"value":0},"meta":{"tenantId":"6305a7f186e74d849ad
3f00ade0242a9","region":"RegionOne"},"creation_time":3386706919782612992},{"metr
ncn# curl -ks --compressed -H "Authorization: Bearer ${ACCESS_TOKEN}" \
https://api-gw-service-nmn.local/apis/sma-telemetry-api/v1/stream/cray-node |head -3 \
| tail -1 | cut -c 7- | jq '' | head
{
  "metrics": {
    "messages": [
      {
        "metric": {
          "name": "cray_storage.cray_vmstat.mem_cache",
          "dimensions": {
            "product": "shasta",
            "system": "compute",
            "service": "ldms",
```

To get output from telemetry stream the `--compressed` option is needed

Telemetry stream output is one json object per line of output.

Linux formatting tools are helpful

SMA-GRAFANA

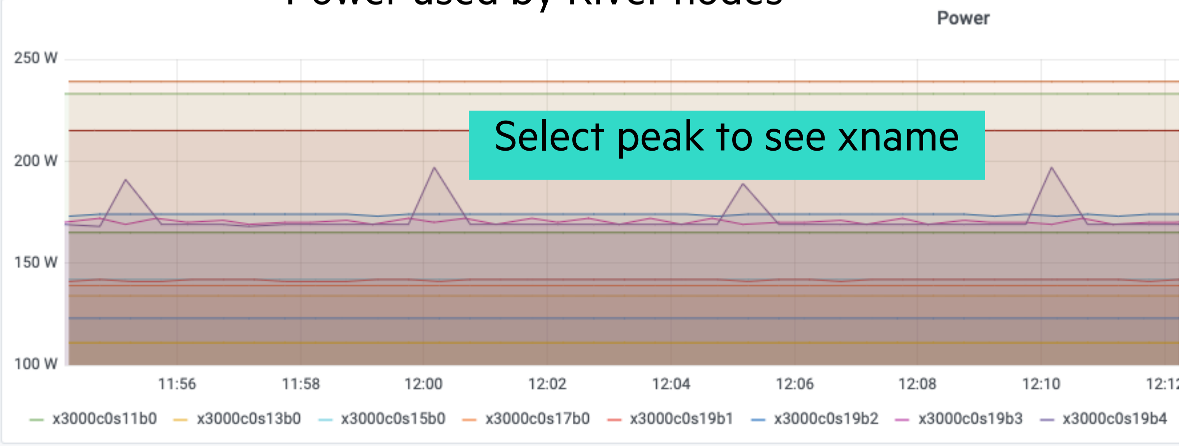

Overview Details
System
compute
Last 15 minutes
I/O - Host All
min max avg
Blocks in 0 0 0
Blocks out 0 0 0
System - Host All
min max avg
Interrupts 112 303 194
Context Switches 134 284 163
CPU - Host All
min max avg
Idle 100% 100% 100%
System 0% 0% 0%
User 0% 0% 0%
Wait 0% 0% 0%
Memory - Host All
min max avg
Buffer 271 MB 271 MB 271 MB
Cache 2.1 GB 2.1 GB 2.1 GB
Free 61.5 GB 61.5 GB 61.5 GB
Swap 0 kB 0 kB 0 kB
Swap - Host All
min max avg
Swap in 0 0 0
Swap out 0 0 0
Processes - Host All
min max avg
Running 0 0 0
Blocked 0 0 0

SMA-GRAFANA DASHBOARDS INCLUDED

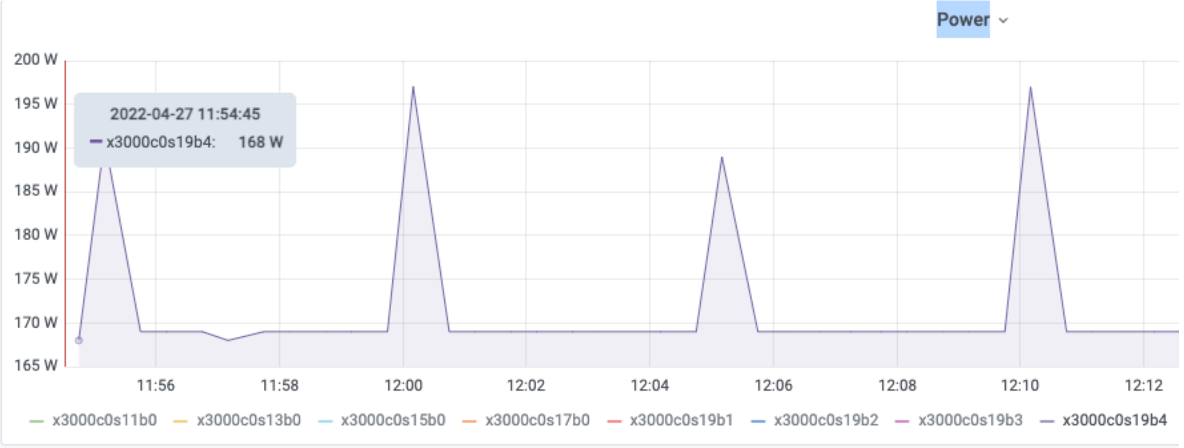
- About 20 included dashboards
  - System CPU, I/O, Kernel, Memory, Processes, Swap
  - Cabinet Controller Sensors
  - CDU Information
  - Fabric Congestion
  - Fabric Errors
  - Fabric Port State
  - Fabric RFC3635
  - Node Controller Sensors
  - Overview Details
  - Overview Device II/O Stats
  - Overview Device I/O Stats Original Overview Mellanox Host Details
  - PDU dashboard
  - Redfish Events
  - River Sensors
  - Switch Controller Sensors
  - System Monitoring Dashboard

https://sma-grafana.<systemdomain>/dashboards

Power used by River nodes



Click on xname to drill into that node



SYSTEM TESTING

- CSM Health Checks
- CSM Diags

CSM HEALTH CHECKS

- CSM documentation describes a series of checks which can be done to validate health for parts of the Shasta system
  - Run before rebooting or rebuilding a management node
  - Run before complete system graceful shutdown
  - Run during complete system graceful startup
  - Run during complete system non-graceful startup
  - Run as part of troubleshooting toolbox
- Platform Health Checks
  - ncnHealthChecks
    ```
    ncn# /opt/cray/platform-utils/ncnHealthChecks.sh
    ```
  - ncnPostgresHealthChecks
    ```
    ncn# /opt/cray/platform-utils/ncnPostgresHealthChecks.sh
    ```
  - BGP Peering Status and Reset
  - Verify KEA has active DHCP leases
  - Verify ability to resolve external DNS
  - Verify Spire Agent is running on management nodes
  - Verify the Vault cluster is health
  - Automated Goss testing
- Hardware Management Services
  - HMS Test execution
  - HSM Discovery Validation
- Software Management Services
  - BOS, TFTP, cray-console, IMS, CFS, VCS, CRUS
- Booting CSM Barebones image
  - Tests whether the booting services infrastructure is functional to boot a compute node
- UAS/UAI tests
  - Validate basic UAS installation
  - Validate UAI creation
  - Troubleshooting UAS/UAI
- See procedures in CSM documentation

  https://github.com/Cray-HPE/docs-csm/tree/release/1.0/operations/validate_csm_health.md

CSM DIAGS

A set of diagnostic tools to perform various node level and system wide tests on compute nodes

- Functional test suites and performance test suites with both MPI and non MPI test suites
- Tests initiated using cray-hms-badger service to submit WLM jobs on compute nodes
  - System Level Diagnostics

    linpack, cwlinpack, nodeperf, stream, olcmt, oldisk, olconf, cwolconf, rank, pandora, cwhpcc
  - GPU Diagnostics

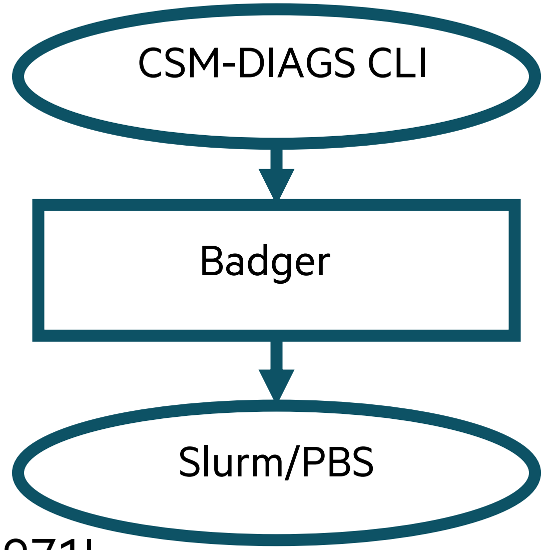
    gpu-burn, xkbandwidth, xkcheck, xkdgemm, xkmemtest, xkmemtest, xkstress, dgnettest
  - OSU Benchmark

    osu_startup, osu_bw_bibw, osu_single_multi_latency, osu_multiplebw_message_rate, osu_multithread_multiprocess_latency, osu_bw_latency_ops, osu_put_bibw, osu_get_acc_latency, osu_collective_blocking_barrier, osu_collective_MPI_blocking_ops, osu_collective_MPI_non_blocking_ibarrier, osu_collective_MPI_non_blocking_ops,
- sdiag_run.py using cray-hms-badger
  - Execute multiple diagnostics (MPI, NON_MPI, GPU, Slingshot) in one shot on multiple compute nodes

CSM DIAGS - CLI

- A CLI (which uses Badger framework) has been provided on the worker nodes to execute multiple diagnostics (MPI, NON_MPI, Slingshot) in a single instance on multiple compute nodes
  - Admin needs to modify the configuration files, with the list of diagnostics that need to be executed
    - sdiag-list.json (List of diagnostics which Admin needs to run)
    - sdiag-arguments.json (Argument Values for each Diagnostic Test)
    - sdiag-gumball.json (Badger Information, Session directory)
    - Nodes (file with the list of node names)
- Can be run by:
  - ->python3 sdiag_run.py
  - out_02:12:02.txt output file has been created in /var/log/cray/shasta-diag
  - Execution completed

gpu-burn: cray badger sessions describe '2912b65c-80dd-417c-93a4-792fb7e5d971'




```
rocket-ncn-w001:~ # cray badger sessions describe "e5d5b58a-f63e-45a5-b3cd-3b383ecdd1df"
notFound = []
loopSuiteUntilTimestamp = ""
analysisStatus = "PASSED"
finishTimestamp = "2020-09-08T04:33:31.977125Z"
underUtilizedNodes = []
cleaned = false
```

```
output_533.0_0_nid001034: TEST has PASSED
GPU 0 - Max Gflops : 16249 , Max Temp : 61 C , Health : OK , Errors: 0
GPU 1 - Max Gflops : 16414 , Max Temp : 52 C , Health : OK , Errors: 0
GPU 2 - Max Gflops : 16172 , Max Temp : 59 C , Health : OK , Errors: 0
GPU 3 - Max Gflops : 17499 , Max Temp : 62 C , Health : OK , Errors: 0
output_533.1_0_nid001033: TEST has PASSED
GPU 0 - Max Gflops : 16271 , Max Temp : 56 C , Health : OK , Errors: 0
GPU 1 - Max Gflops : 16300 , Max Temp : 52 C , Health : OK , Errors: 0
GPU 2 - Max Gflops : 16130 , Max Temp : 58 C , Health : OK , Errors: 0
GPU 3 - Max Gflops : 16492 , Max Temp : 50 C , Health : OK , Errors: 0
```

TROUBLESHOOTING TIPS

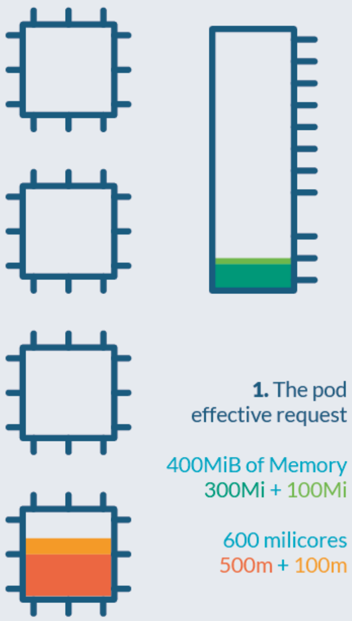
- Kubernetes
- SDU

KUBERNETES LIMITS AND EXCEPTIONS

https://sysdig.com/blog/kubernetes-limits-requests/

A cluster node:

4x vCPUs 16GB RAM



1. The pod effective request

400MiB of Memory
300Mi + 100Mi

600 milicores
500m + 100m

2. Kubernetes assigns 1024 shares per core.

1024 / 0.1 = 102 shares
1024 / 0.5 = 512 shares

The pod - Deployment.yaml

```yaml
kind: Deployment
apiVersion: extensions/v1beta1
metadata:
 name: redis
 labels:
   name: redis-deployment
   app: example-voting-app
spec:
 replicas: 1
 selector:
   matchLabels:
    name: redis
    role: redisdb
    app: example-voting-app
 template:
   spec:
     containers:
       - name: redis
         image: redis:5.0.3-alpine
         resources:
           limits:
             memory: 600Mi
             cpu: 1
           requests:
             memory: 300Mi
             cpu: 500m
       - name: busybox
         image: busybox:1.28
         resources:
           limits:
             memory: 200Mi
             cpu: 300m
           requests:
             memory: 100Mi
             cpu: 100m
```

3. Will be killed if allocates > 600MB.
The whole Pod will fail.

4. Will be throttled if uses more than "1 Core".
1 core = 1000 milicores = 1000m =
100ms of computing time every 100 real ms

Full computing time of the node:
4 vCPUs * 100 real ms =
400ms of computing time = 4000m

5. Killed if allocates > 200MB.

6. Throttled if uses > 30ms of computing time in 100ms

CPU AND MEMORY LIMITS

```
ncn# kubectl get LimitRange --all-namespaces
NAMESPACE        NAME                             CREATED AT
backups          cpu-mem-limit-range              2022-01-19T18:49:07Z
ceph-cephfs      cpu-mem-limit-range              2022-01-19T18:49:06Z
ceph-rbd         cpu-mem-limit-range              2022-01-19T18:49:07Z
default          cpu-mem-limit-range-requests     2022-01-19T18:49:08Z
ims              cpu-mem-limit-range              2022-01-19T18:49:07Z
istio-system     cpu-mem-limit-range              2022-01-19T18:49:07Z
loftsman         cpu-mem-limit-range              2022-01-19T18:49:07Z
metallb-system   cpu-mem-limit-range              2022-01-19T18:49:07Z
operators        cpu-mem-limit-range              2022-01-19T19:29:37Z
pki-operator     cpu-mem-limit-range              2022-01-19T19:29:37Z
services         cpu-mem-limit-range              2022-01-19T19:29:37Z
sma              cpu-mem-limit-range              2022-01-19T18:49:07Z
sysmgmt-health   cpu-mem-limit-range              2022-01-19T18:49:08Z
uas              cpu-mem-limit-range              2022-01-19T19:45:25Z
user             cpu-mem-limit-range              2022-01-19T19:45:25Z
vault            cpu-mem-limit-range              2022-01-19T19:29:37Z
velero           cpu-mem-limit-range              2022-01-19T18:49:08Z
```

POD MEMORY USAGE

```
ncn# kubectl top pod --all-namespaces --sort-by=memory
NAMESPACE           NAME                                                 CPU(cores)   MEMORY(bytes)
sma                 elasticsearch-master-1                               56m          33242Mi
sma                 elasticsearch-master-0                               172m         33163Mi
sma                 elasticsearch-master-2                               166m         33160Mi
sma                 cluster-kafka-0                                      258m         7873Mi
sma                 cluster-kafka-1                                      177m         6813Mi
sma                 cluster-kafka-2                                      173m         6047Mi
sysmgmt-health      prometheus-cray-sysmgmt-health-promet-prometheus-0   383m         5760Mi
istio-system        prometheus-c6f686f44-287qm                           201m         4217Mi
istio-system        prometheus-c6f686f44-jz7xg                           182m         3585Mi
istio-system        prometheus-c6f686f44-8p7p5                           221m         3421Mi
nexus               nexus-7b948976d7-rgzbf                               11m          2408Mi
sma                 sma-monasca-thresh-node-7594fcd77-wrz4d              849m         1633Mi
kube-system         kube-apiserver-ncn-m001                              300m         1563Mi
kube-system         kube-apiserver-ncn-m002                              102m         1408Mi
services            cray-shared-kafka-kafka-2                            52m          1380Mi
services            slingshot-fabric-manager-6d7fbb785f-d7scw            50m          1348Mi
services            cray-shared-kafka-kafka-0                            41m          1283Mi
services            cray-shared-kafka-kafka-1                            40m          1257Mi
sma                 sma-postgres-cluster-1                               14m          1172Mi
sma                 sma-monasca-thresh-dmtf-6c4fcc7c84-2vlzc             845m         1152Mi
sma                 sma-monasca-thresh-metrics-69cf45c768-2kmq9          835m         1144Mi
sma                 cluster-zookeeper-1                                  17m          1031Mi
```

POD CPU USAGE

```
ncn# kubectl top pod --all-namespaces --sort-by=cpu
NAMESPACE           NAME                                                  CPU(cores)   MEMORY(bytes)
sysmgmt-health      prometheus-cray-sysmgmt-health-promet-prometheus-0    1562m        5762Mi
sma                 sma-monasca-thresh-node-7594fcd77-wrz4d               874m         1634Mi
sma                 sma-monasca-thresh-dmtf-6c4fcc7c84-2vlzc              839m         1152Mi
sma                 sma-monasca-thresh-metrics-69cf45c768-2kmq9           832m         1144Mi
kube-system         kube-apiserver-ncn-m001                               312m         1563Mi
sma                 cluster-kafka-0                                       220m         7883Mi
istio-system        prometheus-c6f686f44-8p7p5                            212m         3423Mi
istio-system        prometheus-c6f686f44-jz7xg                            189m         3586Mi
istio-system        prometheus-c6f686f44-287qm                            182m         4217Mi
sma                 elasticsearch-master-2                                167m         33160Mi
sma                 cluster-kafka-2                                       161m         6050Mi
gatekeeper-system   gatekeeper-controller-manager-588d6476db-hrmns        158m         119Mi
sma                 cluster-kafka-1                                       153m         6819Mi
sma                 elasticsearch-master-0                                146m         33164Mi
kube-system         kube-apiserver-ncn-m003                               113m         960Mi
sysmgmt-health      cray-sysmgmt-health-prometheus-node-exporter-5jjgw    110m         212Mi
sysmgmt-health      cray-sysmgmt-health-prometheus-node-exporter-gpb8w    109m         232Mi
kube-system         kube-apiserver-ncn-m002                               102m         1408Mi
```

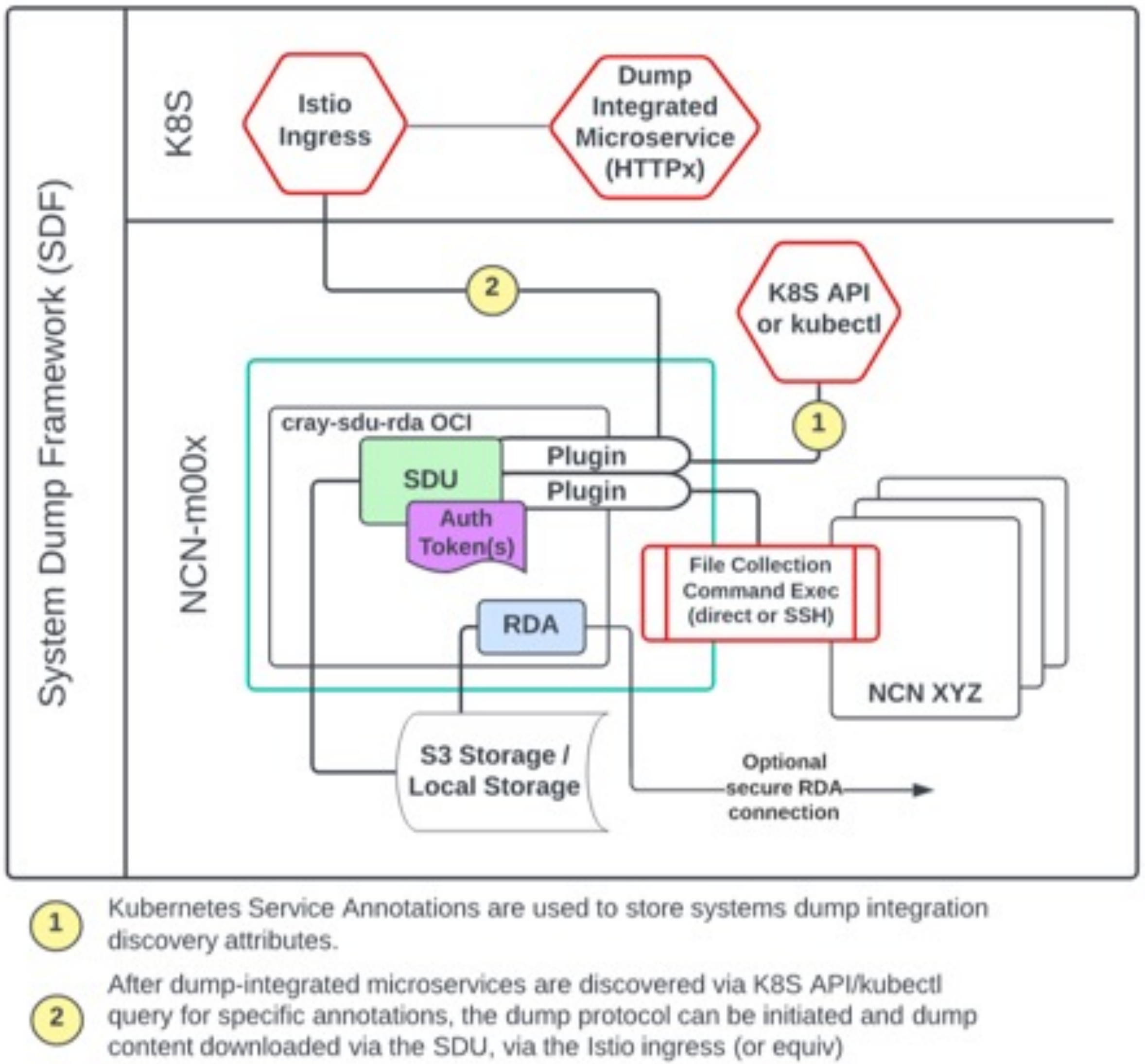
SYSTEM DIAGNOSTIC UTILITY (SDU)

- Pluggable architecture to collect logs, core files, register dumps, and more
  - Can package the output to `tar` to share any useful system triage information
  - Collects data from distributed parts of the system
- Remote Device Access (RDA) is capable of securely transporting this data to HPE
  - AFT (Asynchronous File Transport) is used to securely transport SDU data to HPE
  - IDA (Interactive Device Access) is used to tunnel TCP sessions with HPE
  - Independent from one another and both are opt-in features
- SDU is running in a podman container
  - Container is controlled as a service via systemd on master node
    - `/etc/sysconfig/cray-sdu-rda` – container settings
    - `ncn-m#` **`systemctl start cray-sdu-rda.service`**
      - `/usr/sbin/cray-sdu-rda` – used by systemd to configure, start, and stop container
    - `/usr/sbin/sdu` – passes commands into the cray-sdu-rda podman container
      - allows sdu commands to be run whether on NCN or in the container
- `sdu` commands can be run from the master node or within the container

```
ncn-m001: # sdu bash
ncn-m001-sdu: #    <--- prompt indicates you are inside the SDU/RDA container
```

SYSTEM DUMP FRAMEWORK (SDF)

- Provides a standard system dump feature
  - Onsite triage
  - Onsite to central support
  - Provides a structured data format
- Resiliency model
  - ncn-m001 and ncn-m002 (but SDU can be started on ANY master node, only 1 at a time)
  - Each eligible master node should have a unique RDA configuration



SDU SCENARIOS

- Health
  - Performs a system health collection to gather health information from the system
  - Useful times to run
    - After CSM install.sh completes
    - Before and after NCN reboots
    - After the system is brought back up
    - Any time there is unexpected behavior observed
    - In order to provide relevant information to create support tickets
- Inventory
  - Performs an inventory collection to gather version information for software, firmware, and hardware
  - Useful to run after system upgrades
    - The information collected is used by the HPE Cray Service and R & D organizations to improve customer support
- Triage
  - Performs a triage collection which will gather diagnostic information and logs necessary for HPE Cray Service and R & D to perform problem determination and isolation
  - You are encouraged to provide the --ref 'sfdc:<case number>' command line option to ensure that the snapshot is associated with your service case

SDU TRIAGE SCENARIO

```
ncn-m001# sdu --scenario triage --start_time '-2 days' --reason "Problem with system"
Output similar to the following is expected:
[stdout] INFO Configuration file "/etc/opt/cray/sdu/sdu.conf" and CLI Options Valid.
[stdout] INFO UI master_control status is (enabled) [no control file created]
[stdout] INFO MASTER CONTROLS -> (M:True, U:False)
[stdout] INFO UI CONTROLS -> (C:True, U:True)
[stdout] INFO Exclusive run: Lock file created
@ /var/opt/cray/sdu/lock/sdu.lock_channel-triage_system-devkit
[stdout] INFO COLLECT stage start
[...]
[stdout] INFO dir created in view /var/opt/cray/sdu/collection/triage/view/
2021-02-15T03-10-53_UTC-3c7c6d3040cef5b59b15f15f29c9eda2
[stdout] INFO starting purge
[stdout] INFO work directory removed from '/var/opt/cray/sdu/collection/triage/.work'
[stdout] INFO keeping 10 snapshot(s) max
[stdout] INFO Found 2 snapshot(s) to keep, 0 to purge
[stdout] INFO exiting purge, nothing to do
[stdout] INFO 1813098605.0 raw bytes collected.
[stdout] INFO SDU session stop successfully
[stdout] INFO run took 2431.83 seconds
ncn-m001# cd /var/opt/cray/sdu/collection/triage/view/\
2021-02-15T03-10-53_UTC-3c7c6d3040cef5b59b15f15f29c9eda2
```

All data collected from plugins will be in the view directory

EXPLORE SDU VIEW

- Dump contents are organized first by host or system management component, and then by content type (files and cmds)
  - The following is an example of the directory path:

```
ncn-m001# ls -l
total 3576
drwxr-x--- 4 root root 31 Feb 15 03:51 ceph
drwxr-x--- 3 root root 18 Feb 15 03:51 fmn
drwxr-x--- 3 root root 18 Feb 15 03:51 k8s
drwxr-x--- 3 root root 19 Feb 15 03:51 localhost
drwxr-x--- 3 root root 19 Feb 15 03:51 ncn-m002
drwxr-x--- 4 root root 31 Feb 15 03:51 ncn-m003
drwxr-x--- 4 root root 31 Feb 15 03:51 ncn-m003-sdu
drwxr-x--- 3 root root 19 Feb 15 03:51 ncn-s001
drwxr-x--- 3 root root 19 Feb 15 03:51 ncn-s002
drwxr-x--- 3 root root 19 Feb 15 03:51 ncn-s003
drwxr-x--- 3 root root 19 Feb 15 03:51 ncn-w001
drwxr-x--- 3 root root 19 Feb 15 03:51 ncn-w002
drwxr-x--- 3 root root 19 Feb 15 03:51 ncn-w003
-rw-r--r-- 1 root root 3659206 Feb 15 03:51
session-1613358653-3c7c6d3040cef5b59b15f15f29c9eda2.json
```

- Additional subdirectories exist that contain the logs, core files, register dumps, and more

EXPLORE SDU DATA

- Sample files in subdirectories
  - ceph/cmds/ncn-s001_usr_bin_ceph_status
  - ceph/cmds/ncn-s001_usr_bin_ceph_osd_pool_stats
  - ceph/files/ncn-s001/ncn-s001-ceph-logs.tgz
  - fmn/cmds/usr_bin_fmn_status
  - fmn/cmds/usr_bin_fmctl__get_fabric_switches
  - fmn/cmds/usr_bin_slingshot-topology-tool_--cmd_run_show-flaps
  - fmn/cmds/usr_bin_slingshot-topology-tool_--cmd_show_cables
  - k8s/cmds/usr_bin_kubectl_describe_*
  - k8s/cmds/usr_bin_kubectl_get_*
  - k8s/cmds/usr_bin_kubectl_-n_namespace_describe_pod_*
  - k8s/cmds/usr_bin_kubectl_-n_namespace_logs_*
  - k8s/cmds/usr_bin_kubectl_top_nodes
  - k8s/cmds/usr_bin_kubectl_top_pods
  - localhost/files/report/summary_report
  - ncn-s001/ncn-s001-ceph-logs.tgz
  - ncn-w003/cmds/usr_bin_dmesg
  - ncn-w003/cmds/sbin_lsmod
  - ncn-w003/cmds/sbin_sysctl_-a
  - ncn-w003/cmds/usr_sbin_smartctl_dev_s

Ceph commands and files

Fabric Manager commands and files

Kubernetes commands and files

SDU summary report
- Metadata about the collection
- List of all commands run
- List of files collected
- Exit_code from all plugins

Output from commands run on specific nodes

SDU – KEY DIRECTORIES

- Service (manages SDU container)
  - `/usr/lib/systemd/system/cray-sdu-rda.service`
- Application (inside the container)
  - SDU core: /opt/cray/sdu/default/

```
-ncn-m001-sdu:/ # ls /opt/cray/sdu
    3.3.12-20210624113255_6631f99  default
```

  - SDU Plugins: /opt/cray/sdu/default/plugins
- Configuration
  - `/etc/opt/cray/sdu/sdu.conf`
  - scenario_dir: `/etc/opt/cray/sdu/scenario` (defined in sdu.conf, may have changed from default)
- output (defined in sdu.conf, may have changed from default)
  - log_dir: `/var/opt/cray/sdu/log`
  - lock_dir: `/var/opt/cray/sdu/lock`
  - state_dir: `/var/opt/cray/sdu/run`
  - collection_dir: `/var/opt/cray/sdu/collection`

SDU – MOVING THE COLLECTION (TAR / RDA)

- Tar up collection

```
ncn-m001# cd /var/opt/cray/sdu/collection/<scenario>/view
ncn-m001# tar cvfzh test-system-2020-10-01T00-35-20_UTC-c410d30f1d5656ae006f657aa09d4d27.tgz
2020-10-01T00-35-20_UTC-c410d30f1d5656ae006f657aa09d4d27
```

- RDA Configuration (within the SDU container)
  - `/etc/rda/rda.conf` (if proxy settings are needed)

- RDA Outbox
  - `/var/opt/cray/sdu/outbox`

- Staging files to RDA (to send to HPE) (this will be automated in a future release)

```
ncn-m001-sdu# cd /var/opt/cray/sdu/collection/<scenario>/view
ncn-m001-sdu# sdu-stage-to-rda 2021-02-25T20-09-52_UTC-
f6cade9545082471140 5aa52dade8092
Staging files for RDA transport
Moving files from /var/tmp/RDA_STAGE.7gL3 to RDA outbox /var/tmp/rda/outbox
Done.
```

HPE CRAY EX SYSTEM OVERVIEW

MANAGEMENT SERVICES

WHAT IS HAPPENING ON MY SYSTEM?

MANAGING USER ENVIRONMENTS

RESOURCES

MANAGING USER ENVIRONMENTS

- Site Modifications
- User Access Nodes and Application Nodes
- User Access Instances
- Workload Management
- Cray Programming Environment
- Analytics and AI

SITE MODIFICATIONS

WHAT MAKES MY SYSTEM UNIQUE?

- Site configuration in CFS/VCS
  - Overall versioned configuration name with multiple layers applied to node types
  - Each HPE product may have a layer of configuration which needs site data added to Ansible group_vars or host_vars
    - Lustre filesystem, SpectrumScale (GPFS) filesystem, application node networking, node MOTD, WLM settings, etc.
  - Site may need a layer of configuration from a site-specific config repo in VCS with Ansible plays and related data
    - Use HSM role/subrole and HSM groups to ensure configuration change applies to appropriate type of node
    - Could have a playbook from site repo inserted as a layer before or after any HPE layers
  - Site choice of default versions of CPE and Analytics tools
- Site rpms and package repository in Nexus
- Site changes to image recipe for compute nodes, UANs, or other application nodes with special functions
  - Site rpms or non-default HPE or OS rpms should be placed into the image either via the image recipe or via pre-boot CFS Ansible plays
  - Can adjust scripts used by kiwi-ng to build image recipe into an image root
- Site customizations for UAI classes to provide customized environments for different groups of users
- Site defined names and node membership for HSM groups
  - Used by CFS/Ansible and BOS
- Site BOS session templates specify boot artifacts and configuration to be applied to nodes

USER ACCESS NODES AND APPLICATION NODES

APPLICATION NODES AND UANS

- An application node (AN) is an NCN which is not providing management functions for the HPE Cray EX system
  - The AN is not part of the Kubernetes cluster like management nodes and is not a compute node
  - One special type of AN is the UAN (User Access Node), sometimes called a login node
  - Different systems may have need for other types of ANs
    - Nodes which provide a Lustre routing function (LNet router)
    - Gateways between HSN and Infiniband
    - Data movers between two different network file systems
    - Visualization servers
    - Other special-purpose nodes
- Physical node
  - Mounted in standard air-cooled cabinet
  - More flexibility in node hardware configuration than the management nodes
  - Adjust hardware configuration to meet purpose of node
- Software
  - Default image is similar to the compute node image
    - May be customized, as needed, or entirely different image could be used
  - Configuration with CFS can target special needs of different groups of ANs
  - UAN
    - Shared process space for standard Linux multi-user environment
    - Access via SSH on standard port (22)
    - Workload manager client (natively installed)
    - Cray Programming Environment (mounted, not part of the image)
    - Lustre filesystem (mounted)

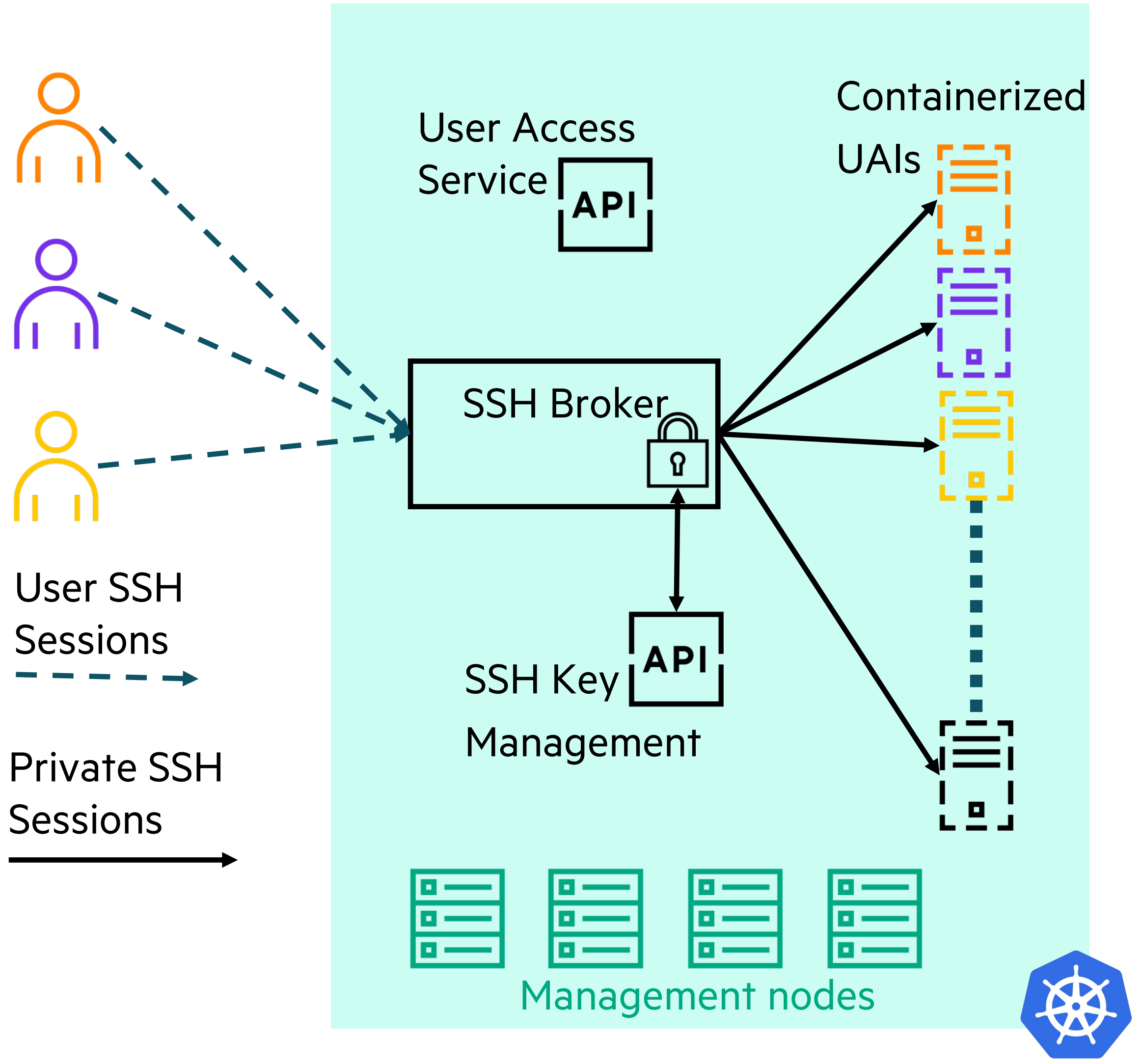
USER ACCESS INSTANCES

ELEMENTS OF A USER ACCESS INSTANCE (UAI)

- Container Image
  - Operating System
  - Preinstalled packages
- Volumes
  - Volumes defined for a UAI provide external access to data provided by the host node
  - Example uses:
    - Workload Manager configuration files
    - Programming Environment libraries and tools
    - Lustre or other external storage for user data
- Resource specifications
  - Sets the minimum amount of memory and CPU to allocate to a UAI by Kubernetes
- Collection of Configuration Items
  - For example:
    - Additional network connections
    - UAI scheduling priority

UAS/UAI RELATIONSHIPS

- User Access Service (UAS) manages UAIs and UAI Configuration
- Broker UAIs
  - Face multiple users on external IP
  - Select or create End-User UAIs on demand
  - Forward SSH sessions to End-User UAIs over private SSH sessions
  - Share private session keys among replicas using key management
- End-User UAIs
  - Each faces a single user on internal Kubernetes IP
- All UAIs are Orchestrated by Kubernetes on worker nodes
- UAI classes are created and managed by System Administrators



LISTING AVAILABLE UAI IMAGES

See documentation for how to create UAI images

```
ncn# cray uas admin config images list
[[results]]
default = false
image_id = "051eb2aa-888c-419c-ba18-d54f148594b6"
imagename = "registry.local/cray/cray-uai-compute:latest"


[[results]]
default = false
image_id = "a7ec4716-8fac-4774-b312-2ec577f173c7"
imagename = "registry.local/cray/cray-uai-cos-2.1.70:latest"


[[results]]
default = true
image_id = "c785276a-aeb5-4a8f-be9d-2a1f6ee77a13"
imagename = "registry.local/cray/cray-uai-sles15sp2:1.0.14"
```

UAI BROKER DEMO – BEFORE UAI CREATION

```
ncn# kubectl get pods -n user -o wide
NAME                              READY   STATUS    RESTARTS   AGE    IP             NODE       NOMINATED NODE   READINESS GATES
slurmctld-59bf5d5597-pfkpn        3/3     Running   0          4d7h   10.42.0.147    ncn-w003   <none>           <none>
slurmdb-74b9cd94f5-2jsg9          1/1     Running   0          4d7h   10.42.0.88     ncn-w003   <none>           <none>
slurmdbd-6bb485cd99-b7lns         3/3     Running   0          4d7h   10.44.0.102    ncn-w001   <none>           <none>
uai-erl-1bdcd856-8bf4c5479-w2nzp  1/1     Running   0          9h     10.44.0.99     ncn-w001   <none>           <none>
ncn# kubectl get pods -n uas -o wide
NAME                                    READY   STATUS    RESTARTS   AGE    IP            NODE       NOMINATED NODE   READINESS GATES
uai-broker-5489b7b4-675f6d849c-9q6hg    2/2     Running   0          3d9h   10.42.0.141   ncn-w003   <none>           <none>
ncn# cray uas admin uais list
[[results]]
uai_age = "3d9h"
uai_connect_string = "ssh broker@10.102.11.145"
uai_host = "ncn-w003"
uai_img = " registry.local/cray/cray-uai-broker:latest"
uai_ip = "10.102.11.145"
uai_msg = ""
uai_name = "uai-broker-5489b7b4"
uai_status = "Running: Ready"
username = "broker"

[[results]]
uai_age = "9h30m"
uai_connect_string = "ssh erl@10.26.5.108"
uai_host = "ncn-w001"
uai_img = "registry.local/cray/cray-uai-sles15sp2:1.0.14"
uai_ip = "10.26.5.108"
uai_msg = ""
uai_name = "uai-erl-1bdcd856"
uai_status = "Running: Ready"
username = "erl"
```

Here we see there is one UAI broker and one end user UAI running on our system

Note the broker and end user UAIs are in different Kubernetes namespaces

UAI BROKER DEMO – REVIEWING THE UAI BROKER

```
ncn# cray uas admin config classes list --format json | jq '.[].class_id, .[].comment'
"1d127b7b-dcb1-4a66-8a85-ce97b00b3300"
"e5c1cba3-035c-4101-a187-92404d7d8256"
"Class for Creating UAI Brokers for PE Supporting UAIs"
"default class for creating legacy mode UAIs with PE support"
"UAI Class for Brokered End-User UAIs -- Supports PE and Slurm"
ncn# cray uas admin config classes list --format json \
| jq 'map(select(.class_id == '1d127b7b-dcb1-4a66-8a85-ce97b00b3300'))' \
| head -12
[
  {
    "class_id": "1d127b7b-dcb1-4a66-8a85-ce97b00b3300",
    "comment": "Class for Creating UAI Brokers for PE Supporting UAIs",
    "default": false,
    "namespace": "uas",
    "opt_ports": [],
    "priority_class_name": "uai-priority",
    "public_ip": true,
    "resource_config": null,
    "uai_compute_network": false,
    "uai_creation_class": "e5c1cba3-035c-4101-a187-92404d7d8256"
ncn# cray uas admin uais list --class-id 1d127b7b-dcb1-4a66-8a85-ce97b00b3300
[[results]]
uai_age = "3d9h"
uai_connect_string = "ssh broker@10.102.11.145"
uai_host = "ncn-w003"
uai_img = "dtr.dev.cray.com/cray/cray-uai-broker:latest"
uai_ip = "10.102.11.145"
uai_msg = ""
uai_name = "uai-broker-5489b7b4"
uai_status = "Running: Ready"
username = "broker"
```

Here we list the class ids and comments to determine which classes exist for UAIs

- First we'll look at the class (cyan) for the UAI broker
- Note it specifies which class (orange) it uses for the UAIs it creates

UAI BROKER DEMO – REVIEWING THE END USER UAI CLASS – 1

```
ncn# cray uas admin config classes list --format json | jq 'map(select(.class_id == \
"e5c1cba3-035c-4101-a187-92404d7d8256"))' | head -15
[
  {
    "class_id": "e5c1cba3-035c-4101-a187-92404d7d8256",
    "comment": "UAI Class for Brokered End-User UAIs -- Supports PE and Slurm",
    "default": false,
    "namespace": "user",
    "opt_ports": [],
    "priority_class_name": "uai-priority",
    "public_ip": false,
    "resource_config": null,
    "uai_compute_network": true,
    "uai_creation_class": null,
    "uai_image": {
      "default": true,
      "image_id": "ab4b789e-8cac-406b-b100-8f94e8c2ba55",
ncn# cray uas admin uais list --class-id e5c1cba3-035c-4101-a187-92404d7d8256
[[results]]
uai_age = "9h47m"
uai_connect_string = "ssh erl@10.26.5.108"
uai_host = "ncn-w001"
uai_img = "registry.local/cray/cray-uai-compute:latest"
uai_ip = "10.26.5.108"
uai_msg = ""
uai_name = "uai-erl-1bdcd856"
uai_status = "Running: Ready"
username = "erl"
```

Here we look at the class for the UAIs created by our broker. Remember that a class is like a template

At this point there is only one UAI running that based on this class

UAI BROKER DEMO – REVIEWING THE END USER UAI CLASS – 2

```
ncn# cray uas admin config classes describe e5c1cba3-035c-4101-a187-92404d7d8256 --format json
{
  "class_id": "e5c1cba3-035c-4101-a187-92404d7d8256",
  "comment": "UAI Class for Brokered End-User UAIs -- Supports PE and Slurm",
  "default": false,
  "namespace": "user",
  "opt_ports": [],
  "priority_class_name": "uai-priority",
  "public_ip": false,
  "resource_config": null,
  "uai_compute_network": true,
  "uai_creation_class": null,
  "uai_image": {
    "default": true,
    "image_id": "ab4b789e-8cac-406b-b100-8f94e8c2ba55",
    "imagename": "registry.local/cray/cray-uai-compute:latest"
  },
  "volume_mounts": [
    {
      "mount_path": "/etc/cray-pe.d",
      "volume_description": {
        "host_path": {
          "path": "/etc/cray-pe.d",
          "type": "DirectoryOrCreate"
        }
      },
      "volume_id": "0d9a6493-d7fc-4650-8a21-a260a07756b5",
      "volumename": "etccrayped"
    },
```

Here we can see the beginning of the class definition.
Items of note are outlined in cyan

UAI BROKER DEMO – REVIEWING THE END USER UAI CLASS – 3

```
ncn# cray uas admin config classes describe e5c1cba3-035c-4101-a187-92404d7d8256 --format json | jq '.volume_mounts
| .[].volume_description.host_path.path'
"/etc/cray-pe.d"
"/opt/modulefiles"
"/opt/forge"
"/opt/totalview_license"
"/opt/R"
"/opt/totalview"
"/opt/lmod"
"/usr/share/lmod"
"/opt/cray/cray-ucx"
"/opt/gcc"
"/usr/local/Modules"
"/etc/ld.so.conf.d/cray-pe.conf"
"/opt/AMD"
"/etc/profile.d"
"/etc/localtime"
"/opt/cray/pe"
"/lus"
"/opt/arm-licenceserver"
"/opt/nvidia/hpc_sdk"
"/opt/intel"
null
"/usr/local/Modules/bin/modulecmd"
"/opt/totalview-support"
"/var/opt/cray/pe/pe_images"
"/opt/toolworks"
null
"/opt/forge_license"
"/opt/cray/modulefiles/cray-ucx"
```

Here we can see all the volumes that are defined in this class. All but two are just mapping directories on the UAI host to the UAI. The other two in orange are a Kubernetes managed secret and a config_map

UAI BROKER DEMO – USING THE BROKER TO CREATE A UAI

```
ncn# cray uas admin uais describe uai-broker-5489b7b4
uai_age = "3d10h"
uai_connect_string = "ssh broker@10.102.11.145"
uai_host = "ncn-w003"
uai_img = "dtr.dev.cray.com/cray/cray-uai-broker:latest"
uai_ip = "10.102.11.145"
uai_msg = ""
uai_name = "uai-broker-5489b7b4"
uai_status = "Running: Ready"
username = "broker"

[uai_portmap]


*cmarsh@linux ~ $ ping 10.102.11.145
PING 10.102.11.145 (10.102.11.145) 56(84) bytes of data.
64 bytes from 10.102.11.145: icmp_seq=1 ttl=54 time=3.41 ms
64 bytes from 10.102.11.145: icmp_seq=2 ttl=54 time=2.43 ms
64 bytes from 10.102.11.145: icmp_seq=3 ttl=54 time=2.39 ms
^C
--- 10.102.11.145 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2002ms
rtt min/avg/max/mdev = 2.396/2.750/3.417/0.475 ms

*cmarsh@linux ~ $ ssh 10.102.11.145
Password:
Creating a new UAI...
The authenticity of host '10.18.134.250 (10.18.134.250)' can't be established.
ECDSA key fingerprint is SHA256:BGN0leKer6z9FL424gBM9Cz79zjwJgloeVYvVbndvZ4.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '10.18.134.250' (ECDSA) to the list of known hosts.
cmarsh@uai-cmarsh-732ed456-98d66f944-6n2mk:~>
```

To create the end user UAI with the broker we just SSH to the broker and authenticate

The second IP address is our UAI's internal IP address – only reachable from the UAI broker

UAI BROKER DEMO – USING THE UAI

```
cmarsh@uai-cmarsh-732ed456-98d66f944-6n2mk:~>
cmarsh@uai-cmarsh-732ed456-98d66f944-6n2mk:~> module list
Currently Loaded Modulefiles:
 1) cpe-cray                5) libfabric/1.11.0.3.66(default)    9) xpmem/2.2.40-
7.0.1.0_1.9__g1d7a24d.shasta(default)
 2) cce/11.0.3(default)     6) craype-network-ofi               10) cray-mpich/8.1.3(default)
 3) craype/2.7.5(default)   7) cray-dsmml/0.1.3(default)        11) cray-libsci/21.03.1.1(default)
 4) craype-x86-rome         8) perftools-base/21.02.0(default)
cmarsh@uai-cmarsh-732ed456-98d66f944-6n2mk:~> sinfo
PARTITION AVAIL  TIMELIMIT  NODES  STATE NODELIST
workq*       up   infinite      2  down* nid[001010,001029]
workq*       up   infinite     16   idle nid[001000-001009,001011-001015,001028]
cmarsh@uai-cmarsh-732ed456-98d66f944-6n2mk:~> whoami
cmarsh
cmarsh@uai-cmarsh-732ed456-98d66f944-6n2mk:~> pwd
/home/users/cmarsh
cmarsh@uai-cmarsh-732ed456-98d66f944-6n2mk:~> srun -n 2 -N 2 hostname
slurmstepd: error: couldn't chdir to `/home/users/cmarsh': No such file or directory: going to /tmp instead
nid001000
slurmstepd: error: couldn't chdir to `/home/users/cmarsh': No such file or directory: going to /tmp instead
nid001001
cmarsh@uai-cmarsh-732ed456-98d66f944-6n2mk:~> exit
logout
Connection to 10.18.134.250 closed.
Connection to 10.102.11.145 closed.
*cmarsh@linux ~ $
```

When we disconnect, we see two connections end

UAI BROKER DEMO – REVIEWING AND DELETING THE NEW UAI

```
ncn# kubectl get pods -n user -o wide
NAME                                 READY   STATUS    RESTARTS   AGE     IP            NODE       NOMINATED NODE   READINESS GATES
slurmctld-59bf5d5597-pfkpn           3/3     Running   0          4d8h    10.42.0.147   ncn-w003   <none>           <none>
slurmdb-74b9cd94f5-2jsg9             1/1     Running   0          4d8h    10.42.0.88    ncn-w003   <none>           <none>
slurmdbd-6bb485cd99-b7lns            3/3     Running   0          4d8h    10.44.0.102   ncn-w001   <none>           <none>
uai-cmarsh-732ed456-98d66f944-6n2mk  1/1     Running   0          6m34s   10.44.0.98    ncn-w001   <none>           <none>
uai-erl-1bdcd856-8bf4c5479-w2nzp     1/1     Running   0          10h     10.44.0.99    ncn-w001   <none>           <none>
ncn# cray uas admin uais list --format json | jq 'map(select(.username == "cmarsh"))'
[
  {
    "uai_age": "8m",
    "uai_connect_string": "ssh cmarsh@10.18.134.250",
    "uai_host": "ncn-w001",
    "uai_img": "registry.local/cray/cray-uai-compute:latest",
    "uai_ip": "10.18.134.250",
    "uai_msg": "",
    "uai_name": "uai-cmarsh-732ed456",
    "uai_portmap": {},
    "uai_status": "Running: Ready",
    "username": "cmarsh"
  }
]
ncn# cray uas admin uais delete --owner cmarsh
results = [ "Successfully deleted uai-cmarsh-732ed456",]

ncn# cray uas admin uais list --format json | jq 'map(select(.username == "cmarsh"))'
[]
ncn# kubectl get pods -n user -o wide
NAME                                 READY   STATUS        RESTARTS   AGE     IP            NODE       NOMINATED NODE   READINESS
GATES
slurmctld-59bf5d5597-pfkpn           3/3     Running       0          4d8h    10.42.0.147   ncn-w003   <none>           <none>
slurmdb-74b9cd94f5-2jsg9             1/1     Running       0          4d8h    10.42.0.88    ncn-w003   <none>           <none>
slurmdbd-6bb485cd99-b7lns            3/3     Running       0          4d8h    10.44.0.102   ncn-w001   <none>           <none>
uai-cmarsh-732ed456-98d66f944-6n2mk  1/1     Terminating   0          9m59s   10.44.0.98    ncn-w001   <none>           <none>
uai-erl-1bdcd856-8bf4c5479-w2nzp     1/1     Running       0          10h     10.44.0.99    ncn-w001   <none>           <none>
```

Currently UAIs persist until they are deleted by an administrator

WORKLOAD MANAGEMENT

WORKLOAD MANAGEMENT

SLURM and PBS PRO

- Actively working with SchedMD and Altair on HPE Cray Ex system check-out and new APIs
- Cray providing integration through a new set of services and APIs
- Both WLMs supported
- Other WLMs can also use the same APIs

CRAY WLM SERVICES

- PALS – Parallel Application Launch Service
  - Used only by PBS Pro
- Application Task Orchestration and Management (ATOM) combines:
  - JACS – Job and application configuration services
  - HATS – Health analysis test service
  - JARS – Job and application reporting service
  - RUR – Resource Utilization Reporting (from Cray XC)

USER ACCESS AND JOB LAUNCH

UAI

- Workload Manager (WLM) clients are installed local to the user access instance (UAI)
  - Commands executed as WLM vendor intended, not proxied
  - No escaping or special handling of the environment
- Access to Lustre mount for job scripts, binaries, and results
  - All UAIs default to the Lustre mount point
- Networking handled by Kubernetes

UAN

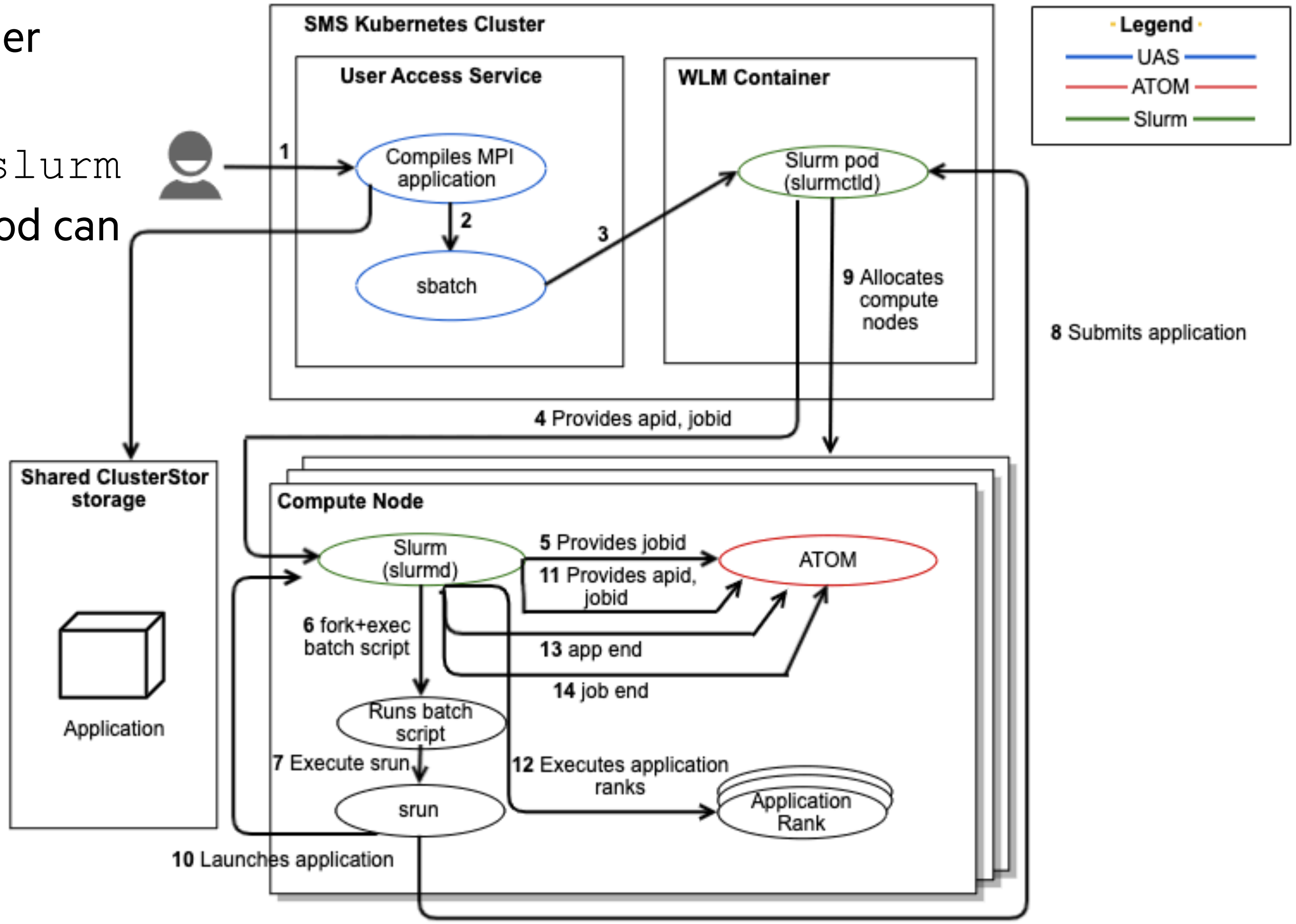
- WLM clients are installed local to the user access node (UAN)
  - Commands executed as WLM vendor intended, not proxied
  - No escaping or special handling of the environment
- Access to Lustre mount for job scripts, binaries, and results
  - All UANs default to the Lustre mount point
- Networking handled by base OS

SLURM OVERVIEW

- Maintained by SchedMD
- Open-source, fault-tolerant, highly scalable cluster management and job scheduling system
- Three key functions:
  - Allocates exclusive and/or non-exclusive access to resources (compute nodes)
  - Provides a framework for starting, executing, and monitoring work
  - Manages a queue of pending work
- Includes an application launcher (`srun`)
- Slurm architecture
  - `slurmctld`
    – Centralized manager to monitor resources and work
    – Runs on management node with an optional fail-over twin
  - `slurmd`
    – Daemon running on each compute node
    – Waits for work, executes that work, returns status, and waits for more work
    – Provides fault-tolerant hierarchical communications
  - `slurmdbd`
    – Optional
    – Used to record accounting information from multiple Slurm-managed clusters in a single database

SLURM WORKFLOW

- Both a launcher and Workload Manager
- Slurm will use a default build
  - For example, the Config file is `/etc/slurm`
- Different options for `slurmctld` pod can be configured



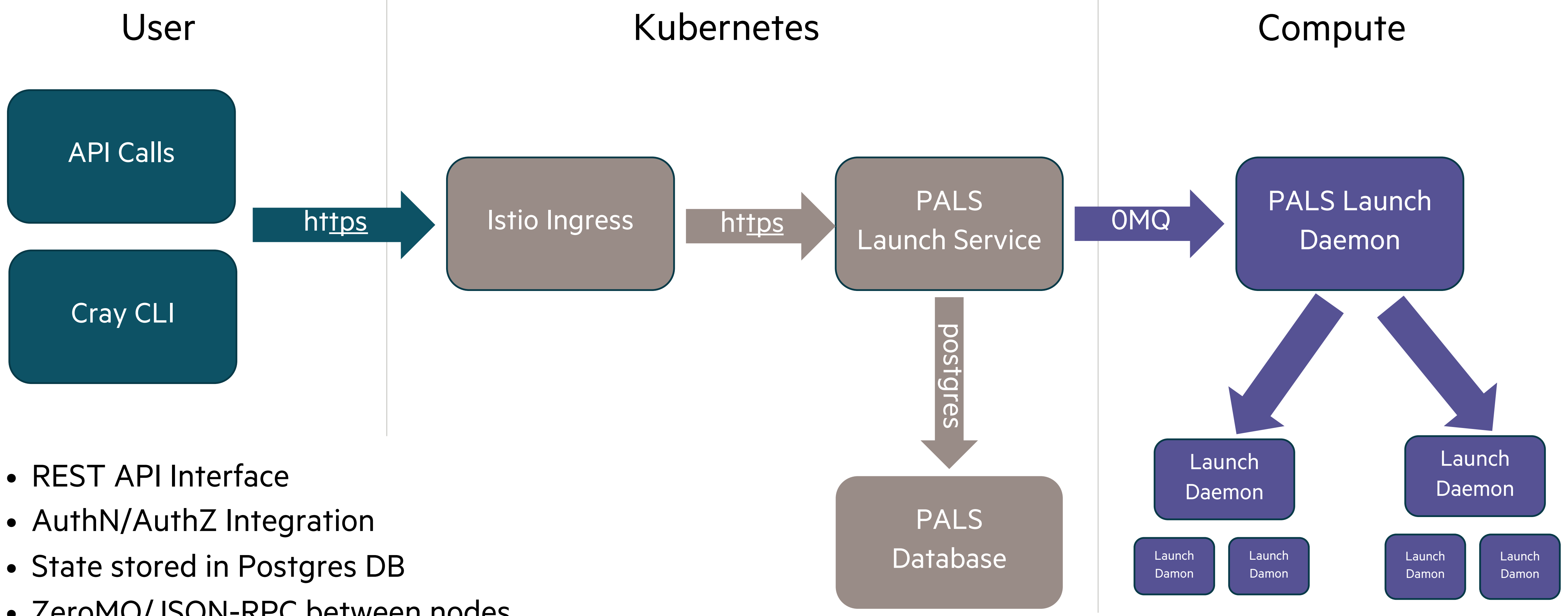
PBS PRO OVERVIEW

- Owned by Altair
- Based on the open source OpenPBS
  - PBS = Portable Batch System
- Automates job scheduling, management, monitoring, and reporting
- Allocates available computing resources to batch requests
- Does not include a launcher
- PBS Pro architecture
  - `pbs_server`
    – Creates batch jobs, modifies jobs, and tracks resources
  - `pbs_sched`
    – Schedules jobs on nodes or vnodes
  - `pbs_mom`
    – MoM (Machine-oriented Miniserver) executes the job on the host
    – Each node must have a MoM to execute a job
  - Works like it currently works in a cluster
  - Uses the HPE-provided *Parallel Application Launch Service* (PALS)
  - Job script executed on a compute node

PARALLEL APPLICATION LAUNCH SERVICE (PALS)

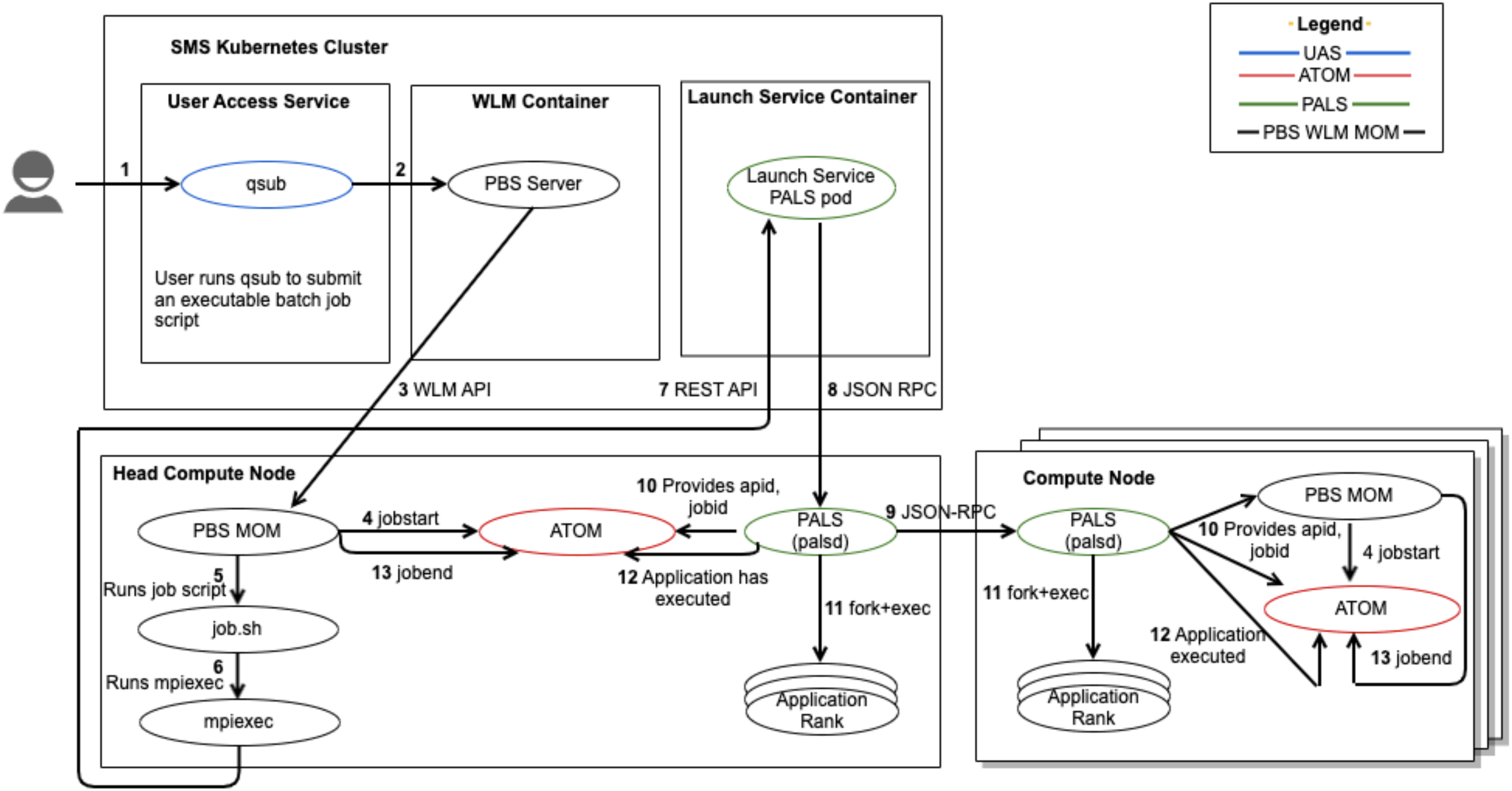
- Application launcher that enables WLMs to function normally
- WLM-specific plugins and configured to access the WLM interfaces
- Launch daemon (`palsd`) integrates with WLMs that have a compute node presence
  - PBS Pro's MoM
- Runs alongside the WLM daemon on the compute node
- Coordinates execution of parallel applications on multiple compute nodes
  - Treats these as a unit rather than separate processes
- Needed for WLMs that do not have a launcher or Cray PMI plugin
  - PBS Pro
- What about Slurm?
  - Already has a launcher (`srun`) and Cray PMI plugin
  - PALS will be disabled

PALS ARCHITECTURE AND COMPONENTS



- REST API Interface
- AuthN/AuthZ Integration
- State stored in Postgres DB
- ZeroMQ/JSON-RPC between nodes
- Cray APIs mainly provide value-add

PALS WORKFLOW WITH PBS PRO



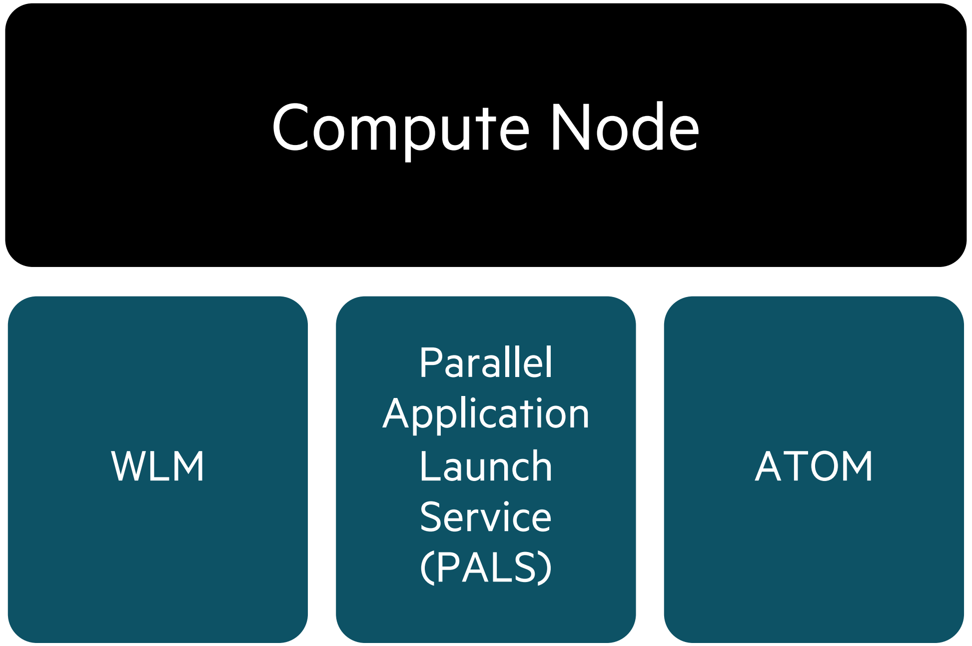
PALS COMMANDS AND ENVIRONMENT VARIABLES

Command	Purpose
`cray mpiexec`	Launch
`cray aprun`	Launch
`cray pals apps signal`	Signal the application
`cray pals apps list`	Gather information about all applications
`cray pals apps describe <apid>`	Gather information about a single application
`cray pals apps files`	Transfer files to compute nodes
`cray pals apps tools`	Run helper processes on compute nodes
`cray pals --help`	View the help page

Environment Variable	Purpose
`PALS_APID`	Unique application identifier
`PALS_RANKID`	Application rank identifier
`PALS_DEPTH`	Number of CPUs per rank for the application
`PALS_NODEID`	Application relative node identifier

APPLICATION TASK ORCHESTRATION AND MANAGEMENT (ATOM)

- Combines functionality of Cray XC system's compute node cleanup, node health, and RUR (Resource Usage and Reporting)
- General purpose job and application prologue and epilogue task runner
  - Configuration
  - Compute node cleanup
  - Node health testing
- ATOM is only called by PALS and WLMs
- ATOM REST API is not exposed on the network
  - Users cannot call ATOM APIs directly



WHY ATOM?

- Allows integration with PALS or the WLM compute node daemon
- Runs a task at a given time
  - ATOM service or daemon start-up
  - Job start or end by WLM Daemon
  - Application start or end by PALS
- Does something if that task fails or succeeds
- Extensible and configurable by the customer
  - New tasks added by dropping in a new task configuration file
  - Runs tasks in lexical order, so sites can choose ordering
- Tasks can be disabled or enabled by site administrator or user
  - Site administrator can force some tasks to run or not permit others to be enabled
- ATOM: compute node daemon runs tasks in the configured order

WHAT IS A TASK?

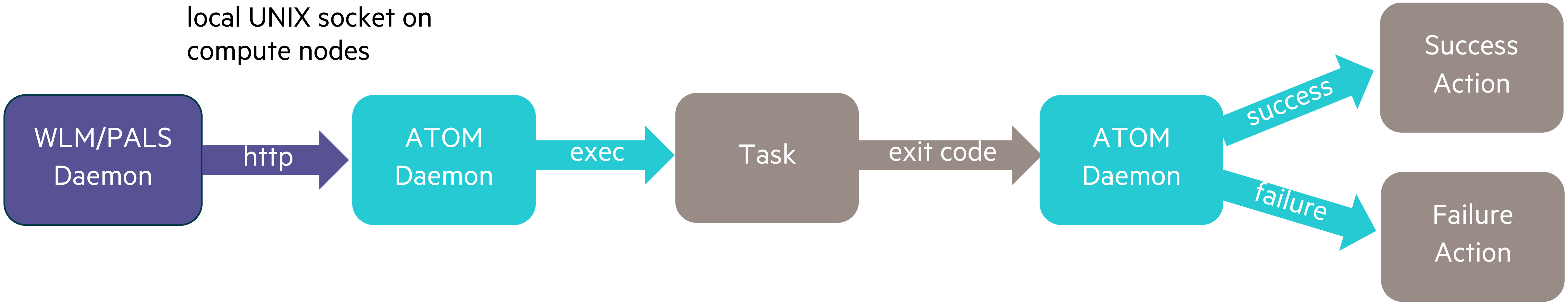
- ATOM daemon startup
  - Initialize Boot FreeMem
- Compute node cleanup
  - Clear VM/Lustre cache
  - Compact memory
- Node health
  - Free memory check
- Reporting
  - Task stats

- Any executable action that is run at a specified time
  - “On this event, run this script and if it fails, do this”
  - “On this event, run this script and if it succeeds, do this”
- Command can be inline commands or executed (Python/shell/binaries)
- Executed in filename lexical order

```
010_bootfreemem_init
{
    "name": “bootfreemem_init",
    "description": "Initialize /proc/boot_freemem",
    "onSuccess": [],
    "onFailure": [],
    "events": ["startup"],
    "timeout": 2,
    "command": ["/bin/sh", "-c", "echo 1 >/proc/boot_freemem"],
    "enabled": true,
    "userControl": false
}
```

ATOM ARCHITECTURE AND COMPONENTS

ATOM only listens on a local UNIX socket on compute nodes



- All tasks and actions run kept in a database only during a job or application's lifespan
  - Task details available through "tasks" endpoint
- All associated tasks and actions are deleted when a job or application is deleted!
- Tasks are considered successful if they exit with 0 exit status before their timeout period has elapsed
- In compute node image, `/etc/sysconfig/atomd` contains configurable variables which control file locations and settings for ATOM daemon

ATOM TASK CONFIGURATION FILE

- File names must begin with three decimal digits
  - Files are executed in numerical order
  - Configuration changes done via customizing the node image or via post-boot node personalization using Ansible JSON object with the following keys:

Key	Type	Required	Description
`name`	String	Yes	Unique task name
`description`	String	No	Human-readable task description
`onSuccess`	Array	No	List of action names to take upon successful completion
`onFailure`	Array	No	List of action names to take upon failure
`events`	Array	Yes	List of times to run this task (startup, jobStart, jobEnd, appStart, appEnd, action)
`timeout`	Number	No	Task timeout in seconds
`command`	Array	Yes	Task argv array
`enabled`	Boolean	No	Enable/disable task by default
`userControl`	Boolean	No	If true, allow users to enable/disable this task

ATOM TASK CONFIGURATION FILES

```
nid001000# ls -1 /etc/atom.d
010_bootfreemem_init.cfg
020_clear_lustre_caches.cfg
020_clear_lustre_caches_job.cfg
025_clean_tmpdirs.cfg
030_clear_vm_cache.cfg
040_compact_memory.cfg
040_compact_memory_job.cfg
090_hugepages_test.cfg
100_freemem_test.cfg
110_zeropage_test.cfg
120_pals_test.cfg
150_filesystem_test.cfg
200_energy_end.cfg
200_energy_start.cfg
800_admindown.cfg
850_reboot.cfg
900_panic.cfg
999_hello_atom.cfg
```

Example task, no actual action

ATOM TASK EXECUTION FILES

- Execution files are in /opt/cray/atom/sbin and are referenced in the “command” field
  - Test for zero page memory corruption at job end

```
nid001000# cat /etc/atom.d/110_zeropage_test.cfg
{
    "name": "zeropage_test",
    "description": "Check for zero page memory
corruption",
    "onSuccess": [],
    "onFailure": ["admindown"],
    "events": ["jobEnd"],
    "timeout": 5,
    "command": ["/opt/cray/atom/sbin/zeropage"],
    "enabled": true,
    "userControl": false,
    "exclusive": false
}
```

- Compact fragmented memory at end of every application and job so hugepage allocations remain efficient

```
nid000001# cat
/etc/atom.d/040_compact_memory.cfg
{
    "name": "compact_memory",
    "description": "Compact fragmented memory to
allow better hugepages allocation",
    "onSuccess": [],
    "onFailure": [],
    "events": ["appEnd", "jobEnd"],
    "timeout": 30,
    "command":
["/opt/cray/atom/sbin/compact_memory.py"],
    "enabled": true,
    "userControl": true
}
```

CFS CONFIGURATION FOR ATOM

- ATOM configuration is done by CFS, so add or change data in VCS (git)
  - Configuration settings can be used to specify directory paths
    - atom_filesystems
      - list of directory paths mounted on all compute nodes to check at application and job end time
    - atom_tmpdirs
      - list of directory paths to be cleaned up at job end time
  - Create the `group_vars/all/atom.yml` file in the pbs-config-management or slurm-config- management git repository
  - Edit and populate it with the desired settings. For example:

```
atom_filesystems:
  - "/scratch"

atom_tmpdirs:
  - "/tmp"
  - "/var/tmp"
  - "/dev/shm"
```

  - Can override or add new ATOM configuration files or tasks

```
roles/atom/files/config/
roles/atom/files/tasks/
```

CRAY PROGRAMMING ENVIRONMENT

HPE CRAY PROGRAMMING ENVIRONMENT

Essential toolset for HPC organizations developing HPC code in-house.

Fully integrated software suite with compilers, tools, and libraries designed to increase programmer productivity, application scalability, and performance.

Complete toolchain

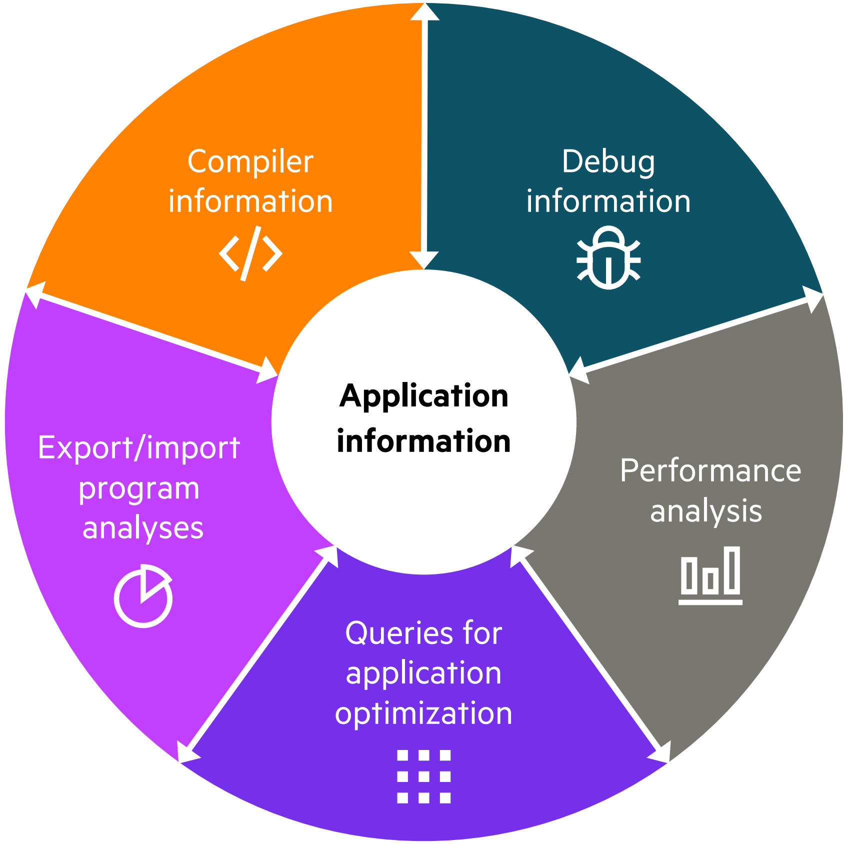
Cross platform

Programmable

Scalable

Holistic support

From HPC experts for HPC experts



COMPREHENSIVE TOOLCHAIN

HPE Cray Programming Environment

Software

Application Development

- C/C++ and Fortran compilers
- I/O, scientific, and math libraries
- HPE Cray MPI
- Machine learning plug-ins

Debugging

- Stack tracing at scale
- Parallelized gbp for scale
- Compare two versions of an application
- Manage core files at scale
- Memory debugging at scale

Performance analysis and Optimization tools

- Whole program profiling and visualization
- HPC optimization tool for scale, parallelization, memory usage, bandwidth

CPE CONFIGURATION

- CPE is installed into images which are projected via CPS/DVS to nodes
- CPE layer in CFS sets which images to use
  - Multiple versions can be installed for cpe-base, aocc (AMD Optimizing Compiler), ARM Forge, Intel oneAPI
  - In VCS, cpe-config-management has pe_deploy.yml Ansible playbook
    - Default version of each tool will be from the first image in the list
- Environment setup with one of these choices
  - Modules and Modulefiles
    - CPE Environment Modules enables users to modify their environment dynamically by using modulefiles
    - The module command provides a user interface to the Modules package
    - The module command system interprets modulefiles, which contain Tool Command Language (Tcl) code, and dynamically modifies shell environment variables such as PATH and MANPATH
    - Can be adjusted in cpe-config-management in roles/cray-.pe_deploy/files
      - cray-pe-configuration.csh and cray-pe-configuration.sh
  - Lmod
    - Lua-based module system that loads and unloads modulefiles, handles path variables, and manages library and header files
    - Hierarchical, manages module dependencies and ensures any module a user has access to is compatible with other loaded modules

ANALYTICS AND AI

ANALYTICS AND AI

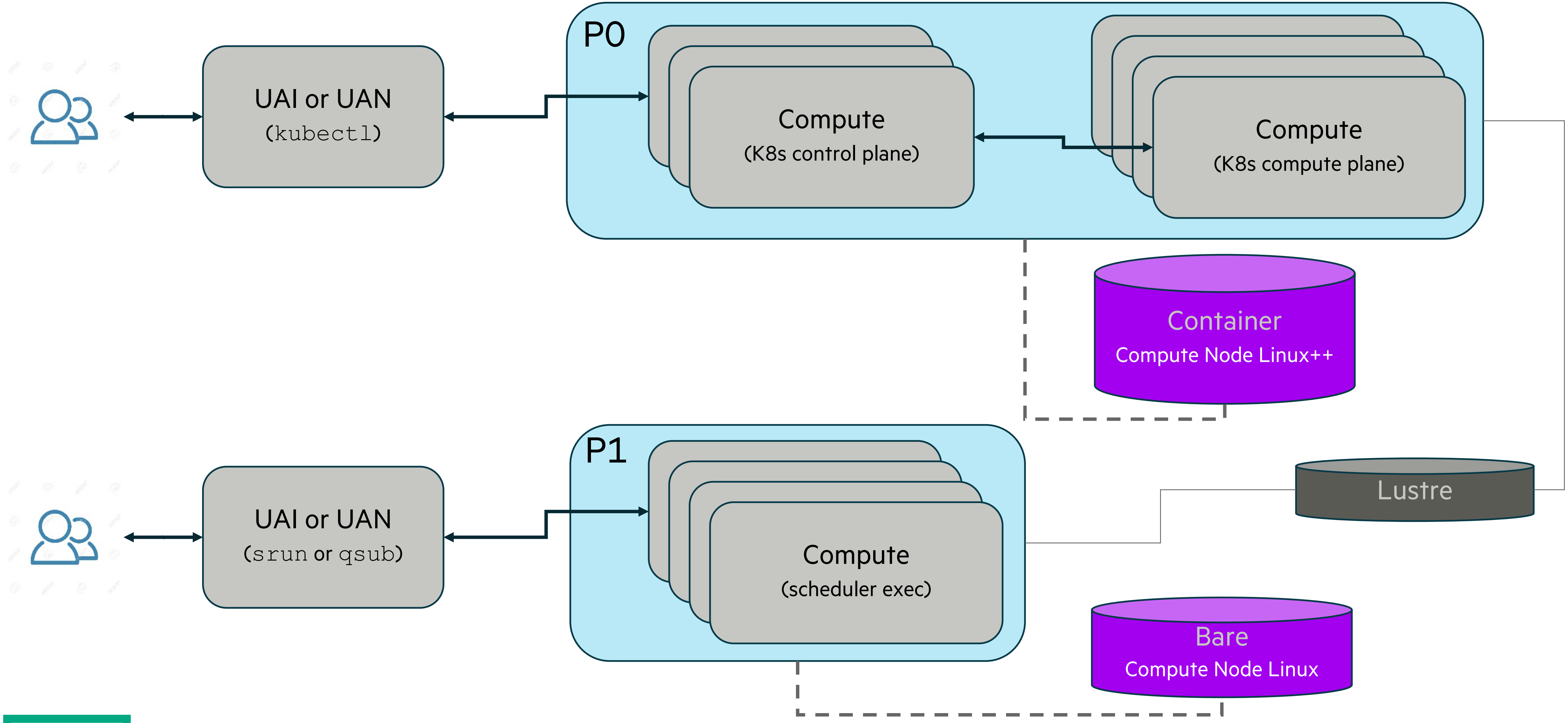
- HPE Cray EX Urika has analytics and AI components for performing big data and deep learning tasks
  - These components run in Docker containers, which are orchestrated via Kubernetes on compute nodes
  - Analytics and AI applications are used through either UAN or UAI
- Supported Analytics and AI Frameworks
  - Apache™ Spark™ - Spark is a general data processing framework that simplifies developing big data applications
    - Provides the means for executing batch, streaming, and interactive analytics jobs
    - Both core Spark components and severalSpark ecosystem components
  - Dask Distributed - Dask Distributed is a centrally managed, distributed, dynamic task scheduler
    - It coordinates several worker processes spread across multiple machines and the concurrent requests of several clients
  - PyTorch™ - PyTorch is an open source optimized tensor library and deep learning framework for Python
    - Designed to be deeply integrated into Python
  - TensorFlow™ - TensorFlow is a software library for dataflow programming across a range of tasks
    - A Math library, which is also used for machine learning applications, such as neural networks

ANALYTICS PROGRAMMING ENVIRONMENT COMPONENTS

- Configuration done via CFS
- Uses modules environment for each user inside their UAI pod
- Requires new compute node image customized to enable Kubernetes on compute nodes
- Installation and configuration in HPE Cray EX Urika Analytics Applications Guide S-8027

Component	Documentation Source
Apache® Maven	https://maven.apache.org
Apache® Spark™	https://spark.apache.org/
Chapel	https://chapel-lang.org/docs/
Dask Distributed	https://docs.dask.org/
Kibana	https://www.elastic.co/products/kibana
ksonnet	https://ksonnet.io
OpenJDK	https://openjdk.java.net
Python	https://python.org
PyTorch	https://pytorch.org
R	https://www.r-project.org
SBT	https://www.scala-sbt.org
TensorFlow	https://www.tensorflow.org/

ORCHESTRATION AND SCHEDULING



HPE CRAY EX SYSTEM OVERVIEW
MANAGEMENT SERVICES
WHAT IS HAPPENING ON MY SYSTEM?
MANAGING USER ENVIRONMENTS
RESOURCES

RESOURCES

- Documentation
- Open Source Software
- Training
- Related Presentations

DOCUMENTATION - INSTALLATION

- HPE Cray EX System Software Getting Started Guide S-8000
- HPE Cray System Management (CSM) Markdown
  - https://github.com/Cray-HPE/docs-csm/tree/release/1.0
- HPE Cray System Management (CSM) HTML
  - https://cray-hpe.github.io/docs-csm/en-10/
- HPE Cray EX System HPC Firmware Pack Installation Guide S-8037
- HPE Cray EX System Admin Toolkit Guide S-8031
- HPE Cray EX System Diagnostic Utility Installation Guide S-8034
- HPE Cray EX System System Monitoring Application Installation Guide S-8030
- HPE SUSE Linux Enterprise Operating System Installation Guide S-8028
- HPE Slingshot Release Notes
- HPE Slingshot Operations Guide
- HPE Cray Operating System Installation Guide CSM on HPE Cray EX Systems S-8025
- HPE Cray User Access Node Software Installation Guide S-8032
- HPE Cray Programming Environment Installation Guide: CSM on HPE Cray EX S-8003
- HPE Cray EX Urika Analytics Applications Guide S-8027

DOCUMENTATION - ADMINISTRATION

- HPE Cray System Management (CSM) Markdown
  - https://github.com/Cray-HPE/docs-csm/tree/release/1.0
  - https://github.com/Cray-HPE/docs-csm/blob/release/1.0/operations/kubernetes/Kubernetes.md
  - https://github.com/Cray-HPE/docs-csm/blob/release/1.0/glossary.md
- HPE Cray System Management (CSM) HTML
  - https://cray-hpe.github.io/docs-csm/en-10/
- HPE Cray EX System Admin toolkit Guide S-8031
- HPE Cray EX System Diagnostic Utility Administration Guide S-8035
- HPE Cray EX System Monitoring Application Administration Guide S-8029
- HPE Cray EX Urika Analytics Applications Guide S-8027
- HPE Cray Operating System Administration Guide CSM on HPE Cray EX Systems S-8024
- HPE Cray User Access Node Software Administration Guide S-8033
- HPE Cray System Management Diagnostics Guide S-8038
- HPE Slingshot Operations Guide
- HPE Slingshot Troubleshooting
- HPE Slingshot Hardware Guide
- HPE Cray Programming Environment User Guide: CSM on HPE Cray EX S-8005

DOCUMENTATION – OPEN SOURCE TOOLS

- https://kubernetes.io/docs/home/
- https://kubernetes.io/docs/reference/kubectl/cheatsheet/
- https://lmgtfy.com/?q=kubernetes+troubleshooting
- https://www.elastic.co/guide/en/kibana/current/index.html
- https://grafana.com/docs/
- https://github.com/aelsabbahy/goss
- http://docs.ansible.com/
- https://kubernetes.io/docs/reference/kubectl/jsonpath/
- https://stedolan.github.io/jq/manual/
- http://www.compciv.org/recipes/cli/jq-for-parsing-json/
- https://osinside.github.io/kiwi/

- CSM
  - MIT License
  - Github Hosted https://github.com/Cray-HPE
  - Community Governance https://github.com/Cray-HPE/community

SUPERCOMPUTING: HPE CRAY EX TRAINING

Where to start?

From HPE Edu
http://www.hpe.com/ww/training

- Select HPE Cray EX Series and ClusterStor Storage

https://education.hpe.com/ww/en/training/portfolio/servers.html#ServersLearningPathsIntro

Course ID	Course Title	Duration	View Schedule
HQ7G6S	HPE Cray EX Series Prerequisite Training Bundle	15 hours	Register →
HQ7D5S	HPE Cray EX System Administration with CSM	5 days	Register →
H9TT2S	HPE Cray EX System Administration with HPE PCM	5 days	Register →
H8PG3S	HPE Cray EX Programming and Optimization	4 days	Register →
HQ6X8AAE	HPE Cray EX Series Overview, Rev. 20.31	8 hours	Register →
HQ6X5AAE	HPE Cray Supercomputer Rack System Hardware Overview, Rev. 20.31	2 hours	Register →
HQ6X6AAE	HPE Cray EX Supercomputer Hardware Overview, Rev. 20.31	3 hours	Register →
HQ6X7AAE	HPE Cray EX Series Test and Development Hardware Overview, Rev. 20.31	2 hours	Register →
HQ7D8S	Cray ClusterStor L300 System Administration	2 days	Register →
HQ7G5S	Cray ClusterStor E1000 Prerequisite Training Bundle	6 hours	Register →
H8PG4S	Cray ClusterStor E1000 System Administration	3 days	Register →
HQ7L0AAE	Cray ClusterStor E1000 System Architecture, Rev. 20.31	2 hours	Register →
HQ7K8AAE	Cray ClusterStor E1000 Overview, Rev. 20.31	2 hours	Register →
HQ7K9AAE	ClusterStor E1000 Install, Rev. 20.31	2 hours	Register →
HQ6Y6AAE	Cray ClusterStor L300 Overview, Rev. 20.31	1 hour	Register →

RELATED PRESENTATIONS AND PAPERS

- CUG 2022
  - HPE Cray EX Shasta 22.03 Cray System Management Overview
  - UAIs Come of Age: Hosting Multiple Custom Interactive Login Experiences Without Dedicated Hardware
  - Dealing with Metrics Data – Where is it, How to get it, What to do with it?
  - Real-Time Machine Learning Analysis of Exascale Integrated Test Failures and Test Coverage
- CUG 2021
  - Managing User Access with UAN and UAI
  - User and Administrative Access Options for CSM-Based Shasta Systems
- CUG 2020
  - Advanced Topics in Configuration Management
  - HPE Cray Supercomputers: System User Access; User Access Node or User Access Instance, Which is Right for Me?
- CUG 2019
  - Shasta Software Technical Workshop
  - Shasta System Management Overview
  - Reimagining Image Management in the New Shasta Environment
  - Hardware Discovery and Maintenance Workflows in Shasta Systems

THANK YOU

harold.longley@hpe.com