Menu

Clusters

Overview

Overview

Creating an On-Premise Cluster

Creating an On-Premise Cluster

etcd Encryption

etcd Encryption

How to

Add External Address for Built-in Registry

Choosing a Container Runtime

Updating Public Repository Credentials

Overview

A cluster is the foundational resource collection for running containerized applications, encompassing nodes, load balancers, storage, and other critical components. It is a prerequisite for successfully running containerized applications on the platform. During initial platform installation, a standard Kubernetes cluster, known as the `global` cluster, is created. Subsequently, multiple clusters can be integrated into the `global` cluster for unified management.

TOC

Cluster Type

On-Premises Cluster

On-Premises cluster is Kubernetes clusters directly created by the platform. Users provide virtual or physical machines, and the platform installs and configures Kubernetes clusters on these machines. This approach is suitable for enterprises with existing hardware resources, allowing full utilization of infrastructure.

Managed Cluster

Managed cluster is Kubernetes clusters provided by cloud service providers, which are integrated into the platform for unified management. Supported integration methods include:

Method	Description	Use Case	Key Characteristics
Import	Integrating existing Kubernetes clusters	Existing clusters with direct network access	• Cluster information submitted to `global` cluster • `global` cluster must have network access to the cluster
Register	Integrating clusters with strict security requirements	Clusters with high security constraints	• Specific plugins installed on the target cluster • Reverse proxy establishes a secure tunnel • Maintains cluster security while enabling management
Proxy Create	Creating clusters through cloud service providers	Leveraging public cloud Kubernetes services	• Cloud service provider credentials required

Method	Description	Use Case	Key Characteristics
			• Platform creates Kubernetes clusters using provided credentials

---

Multi-Cloud and Hybrid Cloud Support

These cluster management approaches meet enterprise needs in multi-cloud and hybrid cloud scenarios, supporting container transformation at different stages:

- Existing Hardware: Create platform-provided clusters
- Existing Clusters: Import or register into the platform
- Elastic Demands: Quickly create public cloud clusters

Implementation Considerations and Limitations

Version Compatibility

- Supported Kubernetes versions: 1.28, 1.29, 1.30, 1.31
- Both On-Premises and Managed clusters must ensure version compatibility
- Version mismatches may result in feature limitations or compatibility issues

Network and Security Requirements

- Ensure network connectivity between `global` and target clusters
- Implement appropriate firewall and network security policies
- Manage access credentials and authentication mechanisms securely

Best Practices for Cluster Management

1. Pre-Implementation Assessment

- Conduct thorough infrastructure and workload analysis
- Identify specific requirements for each cluster
- Develop a comprehensive migration and integration strategy

2. Security and Compliance

- Implement role-based access control (RBAC)
- Use network policies to restrict cluster communication
- Regularly audit and update security configurations
- Ensure compliance with industry standards and regulations

3. Monitoring and Observability

- Set up centralized logging and monitoring
- Implement proactive alerting mechanisms
- Use platform-provided observability tools
- Track cluster performance, resource utilization, and health

4. Backup and Disaster Recovery

- Establish regular backup procedures
- Create and test disaster recovery plans
- Implement multi-cluster backup strategies
- Ensure minimal downtime and data loss

5. Continuous Optimization

- Regularly review cluster configurations

- Optimize resource allocation
- Update to the latest supported Kubernetes versions
- Leverage platform features for automatic updates and scaling

Creating an On-Premise Cluster

TOC

Prerequisites

Node Requirements

1. If you downloaded a single-architecture installation package from Download Installation Package, ensure your node machines have the same architecture as the package. Otherwise, nodes won't start due to missing architecture-specific images.

2. Verify that your node operating system and kernel are supported. See Supported OS and Kernels for details.

3. Perform availability checks on node machines. For specific check items, refer to Node Preprocessing > Node Checks.

4. If node machine IPs cannot be directly accessed via SSH, provide a SOCKS5 proxy for the nodes. The `global` cluster will access nodes through this proxy service.

Load Balancing

For production environments, a load balancer is required for cluster control plane nodes to ensure high availability. You can provide your own hardware load balancer or enable `Self-built VIP`, which provides software load balancing using haproxy + keepalived. We recommend using a hardware load balancer because:

- **Better Performance**: Hardware load balancing performs better than software load balancing.
- **Lower Complexity**: If you're unfamiliar with keepalived, misconfigurations could make the cluster unavailable, leading to lengthy troubleshooting and seriously affecting cluster reliability.

When using your own hardware load balancer, you can use the load balancer's VIP as the `IP Address / Domain` parameter. If you have a domain name that resolves to the load balancer's VIP, you can use that domain as the `IP Address / Domain` parameter. Note:

- The load balancer must correctly forward traffic to ports `6443`, `11780`, and `11781` on all control plane nodes in the cluster.
- If your cluster has only one control plane node and you use that node's IP as the `IP Address / Domain` parameter, the cluster cannot be scaled from a single node to a highly available multi-node setup later. Therefore, we recommend providing a load balancer even for single-node clusters.

When enabling `Self-built VIP`, you need to prepare:

1. An available VRID

2. A host network that supports the VRRP protocol

3. All control plane nodes and the VIP must be on the same subnet, and the VIP must be different from any node IP.

Connecting `global` Cluster and Workload Cluster

The platform requires mutual access between the `global` cluster and workload clusters. If they're not on the same network, you need to:

1. Provide `External Access` for the workload cluster to ensure the `global` cluster can access it. Network requirements must ensure `global` can access ports `6443`, `11780`, and `11781` on all control plane nodes.

2. Add an additional address to `global` that the workload cluster can access. When creating a workload cluster, add this address to the cluster's annotations with the key `cpaas.io/platform-url` and the value set to the public access address of `global`.

Image Registry

Cluster images support Platform Built-in, Private Repository, and Public Repository options.

- **Platform Built-in**: Uses the image registry provided by the `global` cluster. If the cluster cannot access `global`, see Add External Address for Built-in Registry.
- **Private Repository**: Uses your own image registry. For details on pushing required images to your registry, contact technical support.
- **Public Repository**: Uses the platform's public image registry. Before using, complete Updating Public Repository Credentials.

Container Networking

If you plan to use Kube-OVN's Underlay for your cluster, refer to Preparing Kube-OVN Underlay Physical Network.

Creation Procedure

1. Enter the **Administrator** view, and click **Clusters/Clusters** in the left navigation bar.
2. Click **Create Cluster**.

3. Configure the following sections according to the instructions below: Basic Info, Container Network, Node Settings, and Extended Parameters.

Basic Info

Parameter	Description
Kubernetes Version	All optional versions are rigorously tested for stability and compatibility. **Recommendation:** Choose the latest version for optimal features and support.
Container Runtime	Containerd is provided as the default container runtime. If you prefer using Docker as the container runtime, please refer to Choosing a Container Runtime.
Cluster Network Protocol	Supports three modes: IPv4 single stack, IPv6 single stack, IPv4/IPv6 dual stack. **Note:** If you select dual stack mode, ensure all nodes have correctly configured IPv6 addresses; the network protocol cannot be changed after setting.
Cluster Endpoint	`IP Address / Domain`: Enter the pre-prepared domain name or VIP if no domain name is available.

	`Self-Built VIP`: Disabled by default. Only enable if you haven't provided a LoadBalancer. When enabled, the installer will automatically deploy `keepalived` for software load balancing support. `External Access`: Enter the externally accessible address prepared for the cluster when it's not in the same network environment as the `global` cluster.

Container Network

Kube-OVN

An enterprise-grade Cloud Native Kubernetes container network orchestration system developed by Alauda. It brings mature networking capabilities from the OpenStack domain to Kubernetes, supporting cross-cloud network management, traditional network architecture and infrastructure interconnection, and edge cluster deployment scenarios, while greatly enhancing Kubernetes container network security, management efficiency, and performance.

Parameter	Description
Subnet	Also known as Cluster CIDR, represents the **default subnet** segment. After cluster creation, additional subnets can be added.
Transmit Mode	**Overlay**: A virtual network abstracted over the infrastructure that doesn't consume physical network resources. When creating an Overlay default subnet, all Overlay subnets in the cluster use the same cluster NIC and node NIC configuration. **Underlay**: This transmission method relies on physical network devices. It can directly allocate physical network addresses to Pods, ensuring better performance and connectivity with the physical network. Nodes in an Underlay subnet must have multiple NICs, and the NIC used for bridge networking must be exclusively used by

	Underlay and not carry other traffic like SSH. When creating an Underlay default subnet, the cluster NIC is actually a default NIC for bridge networking, and the node NIC is the node NIC configuration in the bridge network. • **Default Gateway**: The physical network gateway address, which is the gateway address for the Cluster CIDR segment (must be within the Cluster CIDR address range). • **VLAN ID**: Virtual LAN identifier (VLAN number), e.g., `0`. • **Reserved IPs**: Set reserved IPs that won't be automatically allocated, such as IPs in the subnet that are already used by other devices.
Service CIDR	IP address range used by Kubernetes Services of type ClusterIP. Cannot overlap with the default subnet range.
Join CIDR	In Overlay transmission mode, this is the IP address range used for communication between nodes and pods. Cannot overlap with the default subnet or Service CIDR.

Calico

Calico is a layer 3 networking solution that provides secure network connections for containers.

Parameter	Description
Default Subnet	Also known as Cluster CIDR, represents the **default subnet** segment. After cluster creation, additional subnets can be added.
Service CIDR	IP address range used by Kubernetes Services of type ClusterIP. Cannot overlap with the default subnet range.

Flannel

Flannel provides a flat network environment for all containers in the cluster, giving containers created on different node hosts a unique virtual IP address across the entire cluster. The pod subnet is divided evenly among the cluster nodes according to the mask, and pods on each node are assigned IP addresses from the segment allocated to that node. This improves communication efficiency between containers without having to consider IP translation issues.

Parameter	Description
Cluster CIDR	IP address range used by pods created when the cluster starts. Supports setting the maximum number of IP addresses that can be allocated to pods on each node under the current container network. **Note**: The platform will automatically calculate the maximum number of nodes the cluster can accommodate based on the above configuration and display it in the hint below the input field. **Important**: After cluster creation, the cluster network cannot be changed, so please plan the network carefully.
Service CIDR	IP address range used by Kubernetes Services of type ClusterIP. Cannot overlap with the container subnet range.

Custom

If you need to install other network plugins, select **Custom** mode. You can manually install network plugins after the cluster is successfully created.

Parameter	Description

Cluster CIDR	IP address range used by pods created when the cluster starts.
Service CIDR	IP address range used by Kubernetes Services of type ClusterIP. Cannot overlap with the container subnet range.

Node Settings

Parameter	Description
Network Interface Card	The name of the host network interface device used by the cluster network plugin. **Note**: • When selecting Underlay transmission mode for the Kube-OVN default subnet, you must specify the network interface name, which will be the default NIC for bridge networking. - The platform's network interface traffic monitoring by default recognizes traffic on interfaces named like `eth.|en.|wl./ww.`. If you use interfaces with different naming conventions, please refer to Collect Network Data from Custom-Named Network Interfaces after cluster onboarding to modify the relevant resources and ensure the platform can properly monitor network interface traffic.
Node Name	You can choose to use either the node IP or hostname as the node name on the platform. **Note**: When choosing to use hostname as the node name, ensure that the hostnames of nodes added to the cluster are unique.

Nodes	**Add nodes** to the cluster, or **Recovery from draft** temporarily saved node information. See the detailed parameter descriptions for adding nodes below.
Monitoring Type	Supports **Prometheus** and **VictoriaMetrics**. When selecting **VictoriaMetrics** as the monitoring component, you must configure the **Deploy Type**: **- Deploy VictoriaMetrics**: Deploys all related components, including **VMStorage**, **VMAlert**, **VMAgent**, etc. **- Deploy VictoriaMetrics Agent**: Only deploys the log collection component, **VMAgent**. When using this deployment method, you need to associate with a VictoriaMetrics instance already deployed on another cluster in the platform to provide monitoring services for the cluster.
Monitoring Nodes	Select nodes for deploying cluster monitoring components. Supports selecting compute nodes and control plane nodes that allow application deployment. To avoid affecting cluster performance, it's recommended to prioritize compute nodes. After the cluster is successfully created, monitoring components with storage type **Local Volume** will be deployed on the selected nodes.

Node Addition Parameters

Parameter	Description
Type	**Control Plane Node**: Responsible for running components such as kube-apiserver, kube-scheduler, kube-controller-manager, etcd,

	container network, and some platform management components in the cluster. When **Application Deployable** is enabled, control plane nodes can also be used as compute nodes. **Worker Node**: Responsible for hosting business pods running on the cluster.
IPv4 Address	The IPv4 address of the node. For clusters created in internal network mode, enter the node's **private IP**.
IPv6 Address	Valid when the cluster has IPv4/IPv6 dual stack enabled. The IPv6 address of the node.
Application Deployable	Valid when **Node Type** is **Control Plane Node**. Whether to allow business applications to be deployed on this control plane node, scheduling business-related pods to this node.
Display Name	The display name of the node.
SSH Connection IP	The IP address that can connect to the node when accessing it via SSH service. If you can log in to the node using `ssh <username>@<node's IPv4 address>`, this parameter is not required; otherwise, enter the node's public IP or NAT external IP to ensure the `global` cluster and proxy can connect to the node via this IP.
Network Interface Card	Enter the name of the network interface used by the node. The priority of network interface configuration effectiveness is as follows (from left to right, in descending order):

	Kube-OVN Underlay: Node NIC > Cluster NIC **Kube-OVN Overlay**: Node NIC > Cluster NIC > NIC corresponding to the node's default route **Calico**: Cluster NIC > NIC corresponding to the node's default route **Flannel**: Cluster NIC > NIC corresponding to the node's default route
Associated Bridge Network	**Note**: When creating a cluster, bridge network configuration is not supported; this option is only available when **adding nodes** to a cluster that already has Underlay subnets created. Select an existing Add Bridge Network. If you don't want to use the bridge network's default NIC, you can configure the node NIC separately.
SSH Port	SSH service port number, e.g., `22`.
SSH Username	SSH username, needs to be a user with root privileges, e.g., `root`.
Proxy	Whether to access the node's SSH port through a proxy. When the `global` cluster cannot directly access the node to be added via SSH (e.g., the `global` cluster and workload cluster are not in the same subnet; the node IP is an internal IP that the `global` cluster

	cannot directly access), this switch needs to be turned on and proxy-related parameters configured. After configuring the proxy, node access and deployment can be achieved through the proxy. **Note**: Currently, only SOCKS5 proxy is supported. **Access URL**: Proxy server address, e.g., `192.168.1.1:1080`. **Username**: Username for accessing the proxy server. **Password**: Password for accessing the proxy server.
SSH Authentication	Authentication method and corresponding authentication information for logging into the added node. Options include: **Password**: Requires a username with root privileges and the corresponding **SSH password**. **Key**: Requires a **private key** with root privileges and the **private key password** .
Save Draft	Saves the currently configured data in the dialog as a draft and closes the **Add Node** dialog. Without leaving the **Create Cluster** page, you can select **Restore from draft** to open the **Add Node** dialog and restore the configuration data saved as a draft. **Note**: The data restored from the draft is the most recently saved draft data.

Extended Parameters

Note:

- Apart from required configurations, it's not recommended to set extended parameters, as incorrect settings may make the cluster unavailable and cannot be modified after cluster creation.
- If a entered **Key** duplicates a default parameter **Key**, it will override the default configuration.

Procedure

1. Click **Extended Parameters** to expand the extended parameter configuration area. You can optionally set the following extended parameters for the cluster:

Parameter	Description
Docker Parameters	`dockerExtraArgs`, additional configuration parameters for Docker, which will be written to `/etc/sysconfig/docker`. Modification is not recommended. To configure Docker through the `daemon.json` file, it must be configured as key-value pairs.
Kubelet Parameters	`kubeletExtraArgs`, additional configuration parameters for Kubelet. **Note**: When the **Container Network**'s **Node IP Count** parameter is entered, a default **Kubelet Parameter** configuration with the key `max-pods` and a value of **Node IP Count** is automatically generated. This sets the maximum number of pods that can run on any node in the cluster. This configuration is not displayed in the interface. Adding a new `max-pods: maximum number of runnable pods` key-value pair in the **Kubelet Parameters** area will override the default value. Any positive integer is allowed, but it's recommended to use the default value (Node IP Count) or enter a value not exceeding `256`.

Controller Manager Parameters	`controllerManagerExtraArgs`, additional configuration parameters for the Controller Manager.
Scheduler Parameters	`schedulerExtraArgs`, additional configuration parameters for the Scheduler.
APIServer Parameters	`apiServerExtraArgs`, additional configuration parameters for the APIServer.
APIServer URL	`publicAlternativeNames`, APIServer access addresses issued in the certificate. Only IPs or domain names can be entered, with a maximum of 253 characters.
Cluster Annotations	Cluster annotation information, marking cluster characteristics in metadata in the form of key-value pairs for platform components or business components to obtain relevant information.

1. Click **Create**. You'll return to the cluster list page where the cluster will be in the **Creating** state.

Post-Creation Steps

Viewing Creation Progress

On the cluster list page, you can view the list of created clusters. For clusters in the **Creating** state, you can check the execution progress.

Procedure

1. Click the small icon **View Execution Progress** to the right of the cluster status.

2. In the execution progress dialog that appears, you can view the cluster's execution progress (status.conditions).

   Tip: When a certain type is in progress or in a failed state with a reason, hover your cursor over the corresponding reason (shown in blue text) to view detailed information about the reason (status.conditions.reason).

Associating with Projects

After the cluster is created, you can add it to projects in the project management view.

etcd Encryption

This guide helps you install, understand, and operate the etcd Encryption Manager in ACP to automate etcd data encryption key rotation within your clusters.

It ensures that sensitive data stored in etcd, such as secrets and configmaps, is encrypted using a secure algorithm, enhancing your cluster's security.

TOC

Installation

See Cluster Plugin for installation instructions.

> **Note:**
>
> - Currently supported:
>   - On-Premises clusters
>   - DCS clusters
> - Not supported:

> - `global cluster`

How it Works

Upon installation, an `etcd-encryption-manager` controller is deployed in the `kube-system` namespace, which:

- Periodically rotates etcd data encryption keys.
- Retains the 8 most recent keys for rollback compatibility.
- Updates encryption configurations on all control nodes.
- Triggers `kube-apiserver` to hot reload new keys.
- Automatically migrates resources to re-encrypt data with new keys.

Cluster stability is maintained throughout these operations.

Default Configuration

Parameter	Value
Encrypted resources	secrets, configmaps
Encryption algorithm	256-bit AES-GCM
Rotation interval	168 hours (7 days)

Operations Guide

Configuration Files

Path	Content
`/etc/kubernetes/encryption-provider.conf`	Current encryption configuration
`/etc/kubernetes/encryption-provider-history.bak`	Historical key records (for recovery)
`/etc/kubernetes/encryption-provider-bak/`	Expired encryption configuration versions

Checking Status

Run the following command to check the current rotation status:

```
kubectl get EtcdEncryptionConfig default -o yaml
```

Example output:

```yaml
apiVersion: cluster.alauda.io/v1alpha1
kind: EtcdEncryptionConfig
metadata:
  name: default
spec:
  resources:
    - secrets
    - configmaps
  rotationInterval: 168h0m0s
  type: aesgcm
status:
  deployStatus:
    192.168.100.1:
      revision: 3
      state: Success
    192.168.100.2:
      revision: 3
      state: Success
    192.168.100.3:
      revision: 3
      state: Success
  migration:
    completeTimestamp: "2025-05-27T05:47:01Z"
    resources:
      - secrets
      - configmaps
    revision: 3
    state: Success
  revision: 3
```

Menu

How to

Add External Address for Built-in Registry

TOC

Overview

When the `global` cluster uses the `Platform Built-in` registry, workload clusters typically also use this registry to pull images. The registry not only serves components within the `global` cluster but must also be accessible to workload cluster nodes.

In certain scenarios, workload cluster nodes cannot directly access the `global` cluster's registry address - for example, when the `global` cluster is in a private data center while workload clusters are in public clouds or edge environments.

This guide explains how to configure an externally accessible address for the platform's default registry to allow workload clusters to pull images.

Prerequisites

Before you begin, prepare the following:

- A domain name accessible by workload cluster nodes

- The IP address that the domain name points to
- A valid SSL certificate for the domain name

WARNING

- The domain name must be different from the platform access address
- Ensure the domain's IP address can forward traffic to all control plane nodes of the `global` cluster

---

Procedure

Configure Certificate and Routing Rules for the Platform Registry

1. Copy the domain's valid certificate to any control plane node of the `global` cluster
2. Create a TLS secret containing the domain certificate:

   ```
   kubectl create secret tls registry-address.tls --cert=<certificate-filename> --key=
   <key-filename> -n kube-system
   ```

   Example:

   ```
   kubectl create secret tls registry-address.tls --cert=custom.crt --key=custom.key -n
   kube-system
   ```

   Note: After creating the certificate, monitor the expiration date of the **registry-address.tls** secret in the **kube-system** namespace of the `global` cluster. Replace the certificate before it expires.

3. Create ingress rules on any control plane node of the `global` cluster:

```
REGISTRY_DOMAIN_NAME=<www.registry.com> # Replace with your accessible domain name
cat << EOF | kubectl create -f -
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  annotations:
    nginx.ingress.kubernetes.io/backend-protocol: HTTPS
  name: registry-address
  namespace: kube-system
  labels:
    service_name: registry
spec:
  rules:
    - host: $REGISTRY_DOMAIN_NAME
      http:
        paths:
          - backend:
              service:
                name: registry
                port:
                  number: 443
            path: /v2/
            pathType: ImplementationSpecific
          - backend:
              service:
                name: registry
                port:
                  number: 443
            path: /v2/_catalog
            pathType: ImplementationSpecific
          - backend:
              service:
                name: registry
                port:
                  number: 443
            path: /v2/.+/tags/list
            pathType: ImplementationSpecific
          - backend:
              service:
                name: registry
                port:
                  number: 443
            path: /v2/.+/manifests/[A-Za-z0-9_+.-:]+
```

```
            pathType: ImplementationSpecific
          - backend:
              service:
                name: registry
                port:
                  number: 443
            path: /v2/.+/blobls/[A-Za-z0-9-:]+
            pathType: ImplementationSpecific
          - backend:
              service:
                name: registry
                port:
                  number: 443
            path: /v2/.+/blobls/uploads/[A-Za-z0-9-:]+
            pathType: ImplementationSpecific
          - backend:
              service:
                name: registry
                port:
                  number: 443
            path: /auth/token
            pathType: ImplementationSpecific
  tls:
    - secretName: registry-address.tls
      hosts:
        - $REGISTRY_DOMAIN_NAME
EOF
```

A response similar to `... created` indicates successful ingress creation.

4. Check if a Registry Service resource exists:

```
kubectl -n kube-system get svc | grep registry
```

If the Service doesn't exist, create it with:

```
cat << EOF | kubectl create -f -
apiVersion: v1
kind: Service
metadata:
  labels:
    name: registry
    service_name: registry
  name: registry
  namespace: kube-system
spec:
  ports:
    - protocol: TCP
      port: 443
      targetPort: 60080
  selector:
    component: registry
  type: ClusterIP
EOF
```

5. Test the configuration by pulling an image from the registry using the domain name:

```
crictl pull <registry-domain-name>/automation/qaimages:helloworld
```

Or

```
docker pull <registry-domain-name>/automation/qaimages:helloworld
```

Choosing a Container Runtime

TOC

Overview

Container Runtime is a core component of Kubernetes, responsible for managing the lifecycle of images and containers.

When creating clusters through the platform, you can choose either Containerd or Docker as your runtime component.

Note: Kubernetes version 1.24 and above no longer officially supports Docker runtime. The officially recommended runtime is Containerd. If you still need to use Docker runtime, you must first enable `cri-docker` in the feature gate before you can select Docker as the runtime component when creating a cluster. For details on using feature gates, see Feature Gate Configuration.

Quick Selection Guide

Choose Containerd	Choose Docker
• Shorter call chain • Fewer components • More stable • Consumes fewer node resources	• Supports docker-in-docker • Allows use of `docker build/push/save/load` commands on nodes • Can call Docker API • Supports docker compose or docker swarm

Differences Between Docker and Containerd

Common Commands

Containerd	Docker	Description
crictl ps	docker ps	View running containers
crictl inspect	docker inspect	View container details
crictl logs	docker logs	View container logs
crictl exec	docker exec	Execute commands in container
crictl attach	docker attach	Attach to container
crictl stats	docker stats	Display container resource usage
crictl create	docker create	Create container
crictl start	docker start	Start container
crictl stop	docker stop	Stop container

Containerd	Docker	Description
crictl rm	docker rm	Remove container
crictl images	docker images	View image list
crictl pull	docker pull	Pull image
None	docker push	Push image
crictl rmi	docker rmi	Delete image
crictl pods	None	View pod list
crictl inspectp	None	View pod details
crictl runp	None	Start pod
crictl stopp	docker images	View images
ctr images ls	None	Stop pod
crictl stopp	docker load/save	Import/export images
ctr images import/export	None	Stop pod
ctr images pull/push	docker pull/push	Pull/push images
ctr images tag	docker tag	Tag images

Call Chain Differences

- Docker as Kubernetes container runtime has the following call relationship:

  kubelet > cri-dockerd > dockerd > containerd > runC

- Containerd as Kubernetes container runtime has the following call relationship:

  kubelet > cri plugin (in containerd process) > containerd > runC

Summary: Although dockerd adds features like swarm cluster, docker build, and Docker API, it can introduce bugs and adds an extra layer in the call chain. Containerd has a shorter call chain, fewer components, greater stability, and consumes fewer node resources.

Log and Parameter Comparison

Comparison	Docker	Containerd
Storage Path	When Docker serves as the Kubernetes container runtime, container logs are stored by Docker in directories like `/var/lib/docker/containers/$CONTAINERID`. Kubelet creates symbolic links in `/var/log/pods` and `/var/log/containers` pointing to the container log files in this directory.	When Containerd serves a the Kubernetes container runtime, container logs are stored by Kubelet in the `/var/log/pods/$CONTAINER_N` directory, with symbolic link created in the `/var/log/containers` direct pointing to the log files.
Configuration Parameters	Specified in the Docker configuration file: `"log-driver": "json-file", "log-opts": {"max-size": "100m","max-file": "5"}`	*Method 1*: Specified in kub parameters: `--container-log-max-files=` `--container-log-max-size="100Mi"` *Method 2*: Specified in KubeletConfiguration: `"containerLogMaxSize": "100Mi",` `"containerLogMaxFiles": 5,`
Saving Container Logs to Data Disk	Mount the data disk to "data-root" (default is `/var/lib/docker`).	Create a symbolic link `/var/log/pods` pointing to a directory under the data dis mount point.

CNI Network Comparison

Comparison	Docker	Containerd
Who Calls CNI	cri-dockerd	cri-plugin built into Containerd (after containerd 1.1)

Comparison	Docker	Containerd
How to Configure CNI	cri-dockerd parameters `--cni-conf-dir` `--cni-bin-dir` `--cni-cache-dir`	Containerd configuration file (toml): `[plugins.cri.cni]` `bin_dir = "/opt/cni/bin"` `conf_dir = "/etc/cni/net.d"`

Updating Public Repository Credentials

TOC

Overview

The `Public Repository` is a platform-provided image registry service available on the public internet. When you want your clusters to use the `Public Repository` as their image registry, you need to update the built-in `public-registry-credential` Cloud Credentials. This ensures your platform has permission to pull images from the public registry.

Procedure

1. Log in to the **Customer Portal** and download your organization's authentication file from the **Enterprise Management** section located in the **User Information** dropdown in the upper right corner.
2. Navigate to **Clusters** > **Cloud Credential** in the left navigation bar of the **Administrator** console.
3. Locate the cloud credential named `public-registry-credential` and click **Update** from the dropdown menu on the right.

4. In the **Upload Public Repository Address** section, upload the authentication file you downloaded from the **Customer Portal**.

5. Click **Update** to apply the changes.