Understanding Docker.io and Containerd.io in Kubernetes

Overview of Docker.io and Containerd.io

This document will be helpful to prepare for interview questions that check deeper understanding on container runtimes and how Kubernetes discontinued docker but embraced containerd.io.

Docker.io:

- Docker is a platform designed to make it easier to create, deploy, and run applications by using containers.
- Containers allow a developer to package up an application with all parts it needs, such as libraries and other dependencies, and ship it all out as one package.
- Docker includes the Docker Engine, a runtime and tooling to manage these containers, and Docker Hub, a service for finding and sharing container images.

Containerd.io:

- Containerd is an industry-standard core container runtime. It is an open-source project designed to handle the core container functionalities, including image transfer and storage, container execution and supervision, and low-level storage and network attachments.
- Containerd is a component of the Docker Engine but can be used independently.

When to Use Docker.io vs. Containerd.io in Kubernetes

Using Docker.io:

- Development and Testing: Docker is particularly useful during development and testing phases because it provides a comprehensive set of tools and an easy-to-use interface for building and managing containers.
- Legacy Systems: If your current system is heavily invested in Docker, it might be practical to continue using Docker to avoid the overhead of migrating to a different runtime.

- Complex Workflows: Docker's ecosystem, including Docker Compose and Docker Swarm, can be beneficial for orchestrating complex multi-container applications during the development phase.

Using Containerd.io:

- Production Environments: Containerd is optimized for use in production environments where you need a lightweight, reliable, and robust container runtime. It has a smaller footprint and is designed to be a minimalistic and efficient core for managing containers.
- Kubernetes Integration: Kubernetes 1.20 and later have deprecated Docker as a container runtime. Kubernetes now directly supports containerd as the underlying runtime through its Container Runtime Interface (CRI). This change is driven by the need for a more streamlined and efficient container runtime that is purpose-built for orchestration.
- Performance and Scalability: If your priority is performance and scalability, containerd is often a better choice due to its streamlined design and reduced overhead compared to Docker.

Docker Support in Kubernetes

As of Kubernetes 1.20, Docker is deprecated as a container runtime. This does not mean that Docker images cannot be used; rather, Kubernetes no longer requires the Docker Engine to manage containers. Kubernetes uses the containerd runtime directly or other CRI-compatible runtimes. Here's what this means for Docker support in Kubernetes:

- CRI Compatibility: Docker is not a CRI-compatible runtime. Kubernetes now uses containerd or other CRI-compliant runtimes directly.
- Docker Shim: Previously, Kubernetes used a component called the "dockershim" to interface with Docker. This shim is being deprecated and removed, necessitating a move to CRI-compatible runtimes like containerd.
- Continued Use of Docker Images: Even with the deprecation, Kubernetes clusters will still pull and run Docker images because containerd and other CRI runtimes support the Docker image format.

Why and How to Use Containerd.io

Why Use Containerd.io:

- Lightweight and Efficient: Containerd is a lightweight container runtime that consumes fewer resources and is highly efficient, making it suitable for large-scale and high-performance applications.
- Direct Kubernetes Integration: Since Kubernetes now integrates directly with containerd via CRI, it eliminates the need for additional shims, reducing complexity and potential points of failure.
- Open-Source and Community Support: As an open-source project with strong backing from the Cloud Native Computing Foundation (CNCF), containerd benefits from community contributions and ongoing development.

How to Use Containerd.io:

1. Installing Containerd: Ensure that containerd is installed on your Kubernetes nodes. This can typically be done via package managers like `apt` or `yum`, or by downloading and installing the binaries directly from the containerd GitHub repository.

   Example for Ubuntu:

   ```
   sudo apt-get update
   sudo apt-get install -y containerd
   ```

2. Configuring Containerd: After installation, configure containerd to ensure it meets your specific needs. This involves setting up the configuration file (`/etc/containerd/config.toml`) and enabling containerd as a service.

   Example to enable and start containerd:

   ```
   sudo systemctl enable containerd
   sudo systemctl start containerd
   ```

3. Kubernetes Configuration: Modify the kubelet configuration to use containerd as the container runtime. This involves updating the kubelet service file or the kubeadm configuration file to specify the CRI socket.

Example for kubelet service file:

```
[Service] ExecStart=/usr/bin/kubelet --container-runtime=remote --
container-runtime-endpoint=unix:///run/containerd/containerd.sock
```

4. Restart Kubelet: Restart the kubelet service to apply the new configuration.

```
sudo systemctl daemon-reload sudo systemctl restart kubelet
```

By adopting containerd in your Kubernetes environment, you can benefit from improved performance, simplified architecture, and seamless integration, aligning with the evolving Kubernetes ecosystem.