Spegel

Stateless cluster local OCI registry mirror

$ whoami

Philip Laine

- Swedish + Finnish living in Berlin
- Spegel creator
- Former FluxCD maintainer

Open to work!

01

Introduction

Where are we today

Distributing Images

Availability

Fault tolerance depends on image availability.

Speed

Startup speed is dependant on the size of the image and registry performance.

Cost

External traffic adds to the cloud bill.

Out in the Wild

Rate Limits

Docker Hub and k8s.registry.io have limits

Outages

GHCR, Docker Hub, Quay, ACR, ECR,

Bandwidth

Public registries limit the bandwidth

Image Removals

Quay removed images from public registry

Two Options

Copying all images

Copy all required images to a private registry and change all image references.

Pull through cache

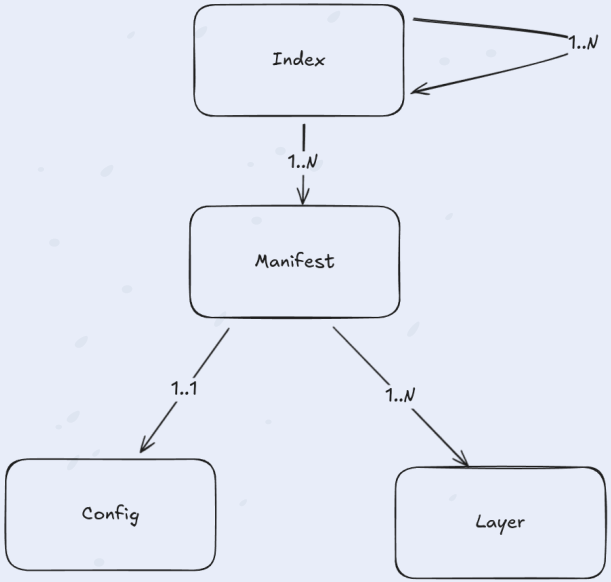
Cache images in private registry as they are being pulled.

02

Background

Let's learn something

Image Spec



Describes the structure of an OCI artifact.

- An image is composed by multiple components
- Index of manifest is top level
- Reference are done through digests

https://oci.dag.dev/?image=ghcr.io%2Fspegel-org%2Fspegel%3Av0.0.30

Distribution Spec

Describes how a client should pull images from a registry.

GET /v2/

Indicate support for the distribution spec.

GET /v2/:name/manifests/:reference

Get manifest content for tag or digest.

GET /v2/:name/blobs/:digest

Get layer content for layers.

03

Solution

An alternative way of looking at things

Planting the Seed

- Presentation about saving a cluster at 1 AM
- Docker Hub is down
- No ability to scale up
- Docker export the images from old node
- SCP image to new node

Open Questions

Images

How are images stored on disk by Containerd?

Registry

Is it possible to implement a registry?

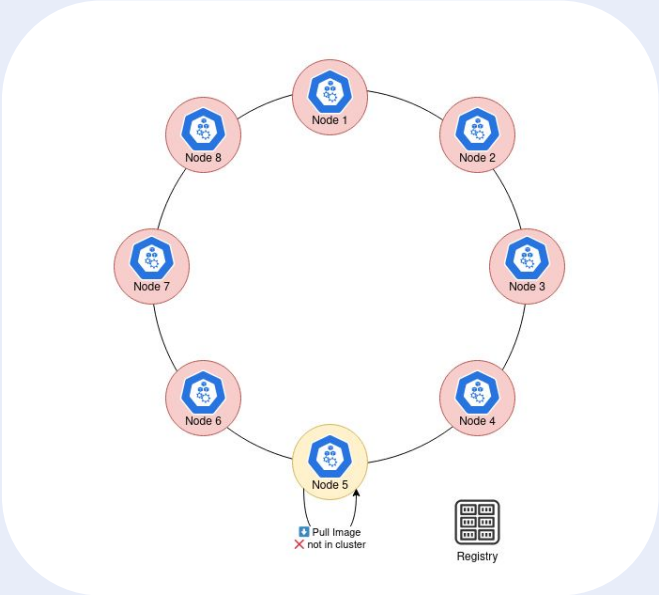
Discovery

How do I discover images on other nodes?

Mirroring

Can mirrors be configured locally?

Spegel



Enables Kubernetes nodes to pull images from each other.

Up to 80%

Increase in image pull performance

Layer Storage

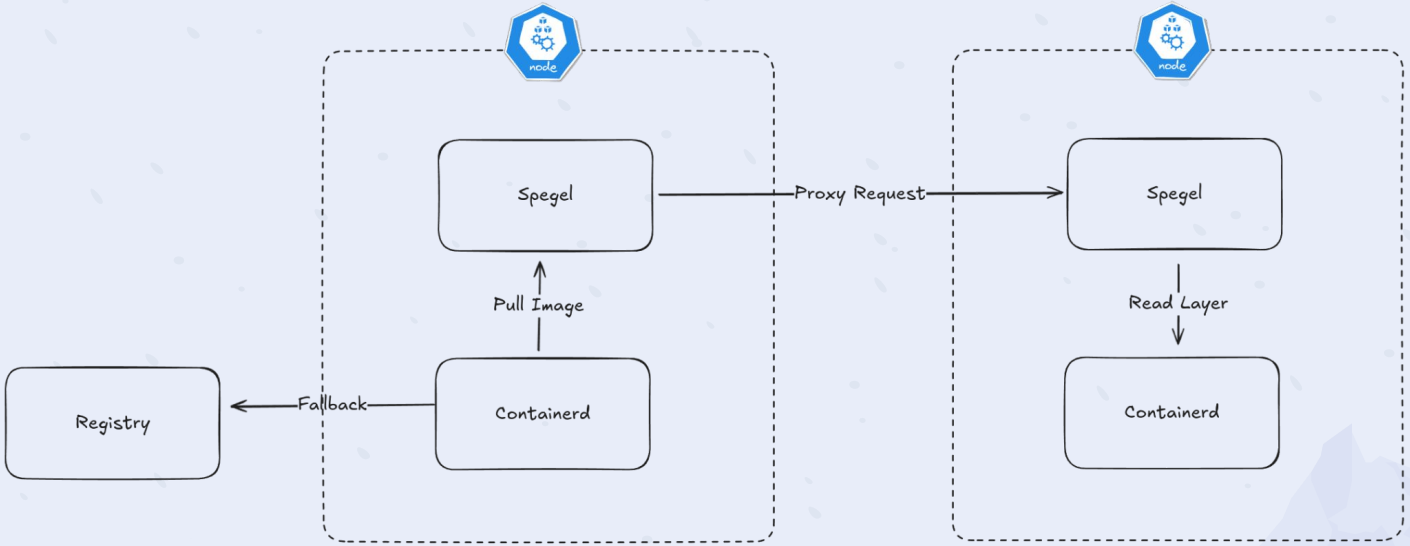
Containerd stores all pulled compressed layers on disk by default which are easily accessible to read from the file system.

/var/lib/containerd/io.containerd.content.v1.content/blobs

Spegel will piggyback on the work the Containerd is doing which is how it is able to be stateless.

In Kubernetes garbage collection is done when disk pressure becomes too high.

Architecture



Content Discovery

A critical part of Spegel is to know which node in the cluster to route the request to.

- Critical that solution is stateless and fault tolerant
- DHT is used to advertise and lookup layers
- All nodes will advertise all digests stored locally on disk
- When request is received lookup is done using digest

https://github.com/libp2p/go-libp2p-kad-dht

Compatibility

Status	Distribution
	AKS
	K3S and RKE2
	EKS
	GKE

https://spegel.dev/docs/getting-started/#compatibility

Use Cases

While the initial intent with Spegel was to build a best effort cache the use cases have expanded past my expectations.

- Best effort cache before primary cache
- Homelabbers avoiding Docker Hub rate limiting
- Air gapped environments
- Optimize distribution of machine learning models

04

Future

Somethings to look forward to

Challenges

Topology Aware Routing

Prioritize peers within the same availability zone.

Fair Load Balancing

Spread traffic to avoid overloading a single instance.

Proxy Performance

Improve Go HTTP copy performance.

OCI Volumes

- Alpha feature since Kubernetes v1.31
- Uses same underlying logic as image pulling
- Useful for AI workloads to load models separately
- Upcoming feature in Containerd

https://github.com/containerd/containerd/issues/10496

Multipart layer fetch

- Splits large layer fetching into multiple requests
- Requests will be run in parallel
- Spegel will enable distributing the load across multiple nodes

https://github.com/containerd/containerd/pull/10177

Thanks!

Do you have any questions?

philip.laine@gmail.com
@phillebaba

https://spegel.dev/
https://github.com/spegel-org/spegel