zotregistry.dev

None

None

Table of contents

1. What's New 4
1.1 v2.1.7 4
1.2 v2.1.6 4
1.3 v2.1.5 4
1.4 v2.1.4 4
1.5 v2.1.3 4
1.6 v2.1.2 5
1.7 v2.1.0 5
1.8 v2.0.4 6
1.9 v2.0.3 6
1.10 v2.0.2 6
1.11 v2.0.1 6
1.12 v2.0.0 6
1.13 v1.4.3 8
2. General 9
2.1 Concepts 9
2.2 Summary of Key Features 12
2.3 Architecture 13
2.4 Extensions 16
2.5 Released Images for zot 18
2.6 Glossary 20
2.7 About the zot Project 24
3. User Guides 25
3.1 Push and Pull Image Content 25
3.2 Use the Web Interface to Find Images 31
3.3 Using the command line interface (zli) 37
4. Installation Guides 44
4.1 Installing zot on Bare Metal Linux 44
4.2 Installing zot with Kubernetes and Helm 48
5. Administrator Guides 53
5.1 Getting Started with zot Administration 53
5.2 Configuring zot 58
6. Developer Guide 67
6.1 Onboarding zot for Development 67
6.2 Developing New Extensions 71

6.3 Using the zot API 73
6.4 zot API Command Reference 79
6.5 Contributing to zot Development 96
7. Articles 98
7.1 Building an OCI-native Container Image CI/CD Pipeline 98
7.2 User Authentication and Authorization with zot 102
7.3 Verifying image signatures 110
7.4 Immutable Image Tags 114
7.5 Software Provenance Workflow Using OCI Artifacts 115
7.6 zot Security Posture 120
7.7 Storage Planning with zot 122
7.8 Configuring zot Tag Retention Policies 132
7.9 OCI Registry Mirroring With zot 136
7.10 zot Clustering 145
7.11 Scale-out clustering 149
7.12 Deploying a Highly Available zot Registry 152
7.13 Monitoring the registry 153
7.14 Using GraphQL for Enhanced Searches 155
7.15 Benchmarking zot with zb 167
7.16 Performance Profiling in zot 170
7.17 Using `kind` for Deployment Testing 173
7.18 *containerd* Mirroring From *zot* 176

1. What's New

1.1

1.1.1 zot OCI Container Images for FreeBSD

FreeBSD community has been releasing [OCI images]{https://download.freebsd.org/releases/OCI-IMAGES/} to enable their container ecosystem. zot is now releasing OCI images to deploy and run as native FreeBSD containers. For example, `ghcr.io/project-zot/zot-freebsd-amd64:latest`.

1.1.2 Bug fixes

Some minor bug fixes.

1.2

1.2.1 Liveness and Readiness Endpoints for Kubernetes Deployments

New HTTP endpoints are added for improved Kubernetes deployments - `/livez`, `/readyz` and `/startupz`. Helm chart has been updated.

1.2.2 OpenID Credentials Stored in Files

Previously OpenID credentials were specified inline in zot configuration. Now, they can be loaded from a separate file and the file can be constructed as a Kubernetes `Secret`.

1.2.3 Bug fixes

Some bug fixes around FreeBSD builds.

1.3

1.3.1 Gitlab Social Login

Gitlab login is now supported in zot UI.

1.3.2 Events Extension Token and Custom HTTP Header Support

Earlier, events extension for HTTP sink supported only basic authentication. Now token authentication is supported. Furthermore, you can also include customer HTTP headers to enable sink side routing, etc.

1.4

1.4.1 Bug fixes

Some bug fixes around authentication and authorization.

1.5

1.5.1 Event generation

zot can now generate registry-significant events that can be published to http or nats endpoints.

1.5.2 Improved `docker` Support in UI

Native docker images are now correctly displayed in the Web UI.

1.5.3 Redis driver Support

Previous releases supported a local BoltDB or a remote DynamoDB database in order to store image and blob metadata. This release now includes support for *Redis*.

1.5.4 AWS ECR Sync Support With Temporary Token Authentication

AWS ECR can now be used as the upstream registry to mirror from and the configuration allows for an authentication helper.

1.5.5 Sync Exclude Regex

While mirroring from an upstream registry, it is now possible to exclude images with a regex pattern.

1.5.6 Windows Binaries

Native Windows binaries (both amd64 and arm64) are now distributed with the release.

1.5.7 Bug fixes

The following security issue has been fixed in this release.

`GHSA-c37v-3c8w-crq8/CVE-2025-48374`

1.6

1.6.1 Compatibility with other image schema types

A new configuration setting allows you to store images using the schema Docker Manifest v2 Schema v2. With this setting enabled, zot makes no modifications to the image's manifest or digest. Such modifications would otherwise break the image's signature and attestations.

1.6.2 Bug fixes

The following security issue has been fixed in this release.

`GHSA-c9p4-xwr9-rfhxi/CVE-2025-23208`

1.7

1.7.1 Scale-out cluster

You can build a scale-out cluster (proxy/shard based on repository name). Scale-out cluster is compatible with "sync" feature.

1.7.2 Bug fixes

The following security issue has been fixed in this release.

`GHSA-55r9-5mx9-qq7r/CVE-2024-39897`

1.8

This is a maintenance release with minor bug fixes.

1.9

This is a maintenance release with minor bug fixes.

1.10

1.10.1 CVE Query Enhancements

It is now possible to bisect CVEs (`zli cve diff`) between two image tags/versions in the same repository. Furthermore, a CVE query for a particular image tag can return a detailed description of CVEs.

1.10.2 Documentation for "Immutable Image Tags"

A new article has been added to document how image tags can be made immutable.

1.10.3 Cross-repo tag search in UI

You can now search for a tag across all repos by starting your query as ':' in the UI, which will return all images that have that tag.

1.10.4 Support for ORAS Artifacts removed

OCI distribution spec 1.1.0 has added support "artifacts" which is likely to gain wider adoption. ORAS artifacts are not widely used or supported.

```
:warning:  Support is removed starting from this version.
```

1.11

1.11.1 Support for hot reloading of LDAP credentials file

Since v2.0.0, LDAP credentials have been specified in a separate file. Starting with this version, the file is watched and changes applied without restarting zot.

1.11.2 Bugfixes and performance improvements

Under some configurations, zot consumes significant CPU and memory resources. This has been fixed in this release.

1.12

1.12.1 Updated OCI support

- Support is added for OCI Distribution Spec v1.1.0-rc3 and OCI Image Spec v1.1.0-rc4. The OCI changes are summarized here. These specifications allow arbitrary artifact types and references so that software supply chain use cases can be supported (SBOMs, signatures, etc). Currently, `oras` and `regclient` support this spec.
- For a demonstration of an end-to-end OCI artifacts workflow, see Software Provenance Workflow Using OCI Artifacts.

  ⚠️ Support is deprecated for earlier OCI release candidates.

1.12.2 Built-in UI support

- Using the new zot GUI, you can browse a zot registry for container images and artifacts. The web interface provides the shell commands for downloading an image using popular third-party tools such as docker, podman, and skopeo.

1.12.3 Support for social logins

- Support is added for OpenID authentication with GitHub, Google, and dex.

1.12.4 Group policies for authorization

- When creating authorization policies, you can assign multiple users to a named group. A group-specific authorization policy can then be defined, specifying allowed access and actions for the group.

  ✏️ **Configuration syntax change**: In the previous release, authorization policies were defined directly under the `accessControl` key in the zot configuration file. With the new ability to create authorization groups, it becomes necessary to add a new `repositories` key below `accessControl`. Beginning with zot v2.0.0, the set of authorization policies are now defined under the `repositories` key.

1.12.5 Signature verification

- The validity of an image's signature can be verified by zot. Users can upload public keys or certificates to zot.

1.12.6 LDAP credentials stored separately from configuration

- The LDAP credentials are removed from zot's LDAP configuration and stored in a separate file. See zot's LDAP documentation.

  ⚠️ This LDAP configuration change is incompatible with previous zot releases. When upgrading, you must reconfigure your LDAP credentials if you use LDAP.

1.12.7 Storage deduplication on startup

- Deduplication, a storage space saving feature, now runs or reverts at startup depending on whether the feature is enabled or disabled. You can trigger deduplication by enabling it and then restarting zot.

1.12.8 Retention policies

- To optimize image storage, you can configure tag retention policies to remove images that are no longer needed.

1.12.9 CVE scanning support for image indexes

- The `trivy` backend now supports vulnerability scanning of image indexes. Previously, only images were scanned.

1.12.10 Bookmarks

- In the zot GUI, you can bookmark an image so that it can be easily found later. Bookmarked images appear in search queries when the bookmarked option is enabled.

1.12.11 Ability to delete tags from the UI

1.12.12 Command line search

- The `zli search` command allows smart searching for a repository by its name or for an image by its repo:tag.

1.12.13 Search by digest

- You can perform a global search for a digest (SHA hash) using either the UI or the CLI. This function is useful when an issue is found in a layer that is used by multiple images. In the UI Search box, for example, begin typing `sha256:` followed by a partial or complete digest value to see a dropdown list of images that contain the layer with the digest value.

1.12.14 GraphQL support for search

- A GraphQL backend server within zot's registry search engine provides efficient and enhanced search capabilities. In addition to supporting direct GraphQL queries through the API, zot hosts the GraphQL Playground, which provides an interactive graphical environment for GraphQL queries.

1.12.15 Scheduling of background tasks

- You can adjust the background scheduler based on your deployment requirements for tasks that are handled in the background, such as garbage collection. See Configuring zot.

1.12.16 Performance profiling for troubleshooting

- You can use zot's built-in profiling tools to collect and analyze runtime performance data.

1.12.17 Binaries for FreeBSD

- zot binary images are available for the FreeBSD operating system. Supported architectures are amd64 and arm64.

  ✏️ zot container images for FreeBSD will be available in a future release.

1.13

1.13.1 Remote-only Storage Support

- The two types of state (images and image metadata) can both now be on remote storage so that zot process lifecycle and its storage can be managed and scaled independently.

1.13.2 Digest Collision Detection During Image Deletion

- When two or more image tags point to the same image digest and the image is deleted by digest causes data loss and dangling references. A new behavior-based policy called *detectManifestCollision* was added to prevent this.

August 3, 2025

2. General

2.1 Concepts

2.1.1 What is zot?

👉 **zot** is a production-ready, open-source, vendor-neutral container image registry server based purely on OCI standards.

Two broad trends are changing how we build, distribute, and consume software. The first trend is the increasing adoption of container technologies. The second trend is that software solutions are being composed by combining elements from various sources rather than being built entirely from scratch. The latter trend raises the importance of software provenance and supply chain security. In both trends, zot intends to play an important role by providing a production-ready, open-source, vendor-neutral container image registry server based purely on OCI standards.

What is an OCI image registry?

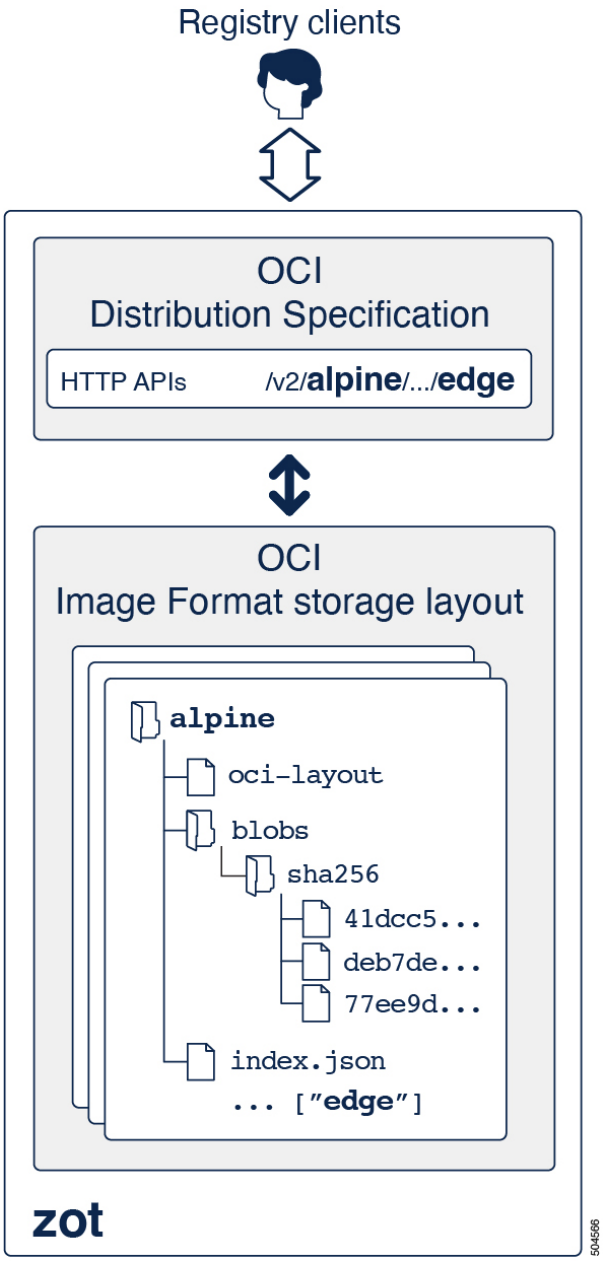
An OCI image registry is a server-based application that allows you to store, manage, and share container images. A developer uploads (pushes) an image to the registry for distribution. Users can then download (pull) the image to run on their systems. The OCI Distribution Specification, published by the Open Container Initiative (OCI), defines a standard API protocol for these and other image registry operations.

An image registry can be a part of your continuous integration and continuous deployment (CI/CD) pipeline when you host zot on your public or private server. In its minimal form, you can also embed a zot registry in a product. In either case, zot provides a secure software supply chain for container images.

2.1.2 Why zot?

👉 zot = OCI Distribution Specification + OCI Image Format

At its heart, zot is a production-ready, vendor-neutral OCI image registry with images stored in the OCI image format and with the OCI distribution specification on-the-wire. zot is built for developers by developers, offering features such as minimal deployment using a single binary image, built-in authentication and authorization, and inline garbage collection and storage deduplication.



Some of the principal advantages of zot are:

- Open source
- OCI standards-only both on-the-wire and on-disk
- Clear separation between core distribution spec and zot-specific extensions
- Software supply chain security, including support for cosign and notation
- Security hardening
- Single binary with many features built-in
- Suitable for deployments in cloud, bare-metal, and embedded devices

zot fully conforms to the OCI Distribution Specification.

The following table lists additional advantages of zot:

Distribution Spec conformance	yes
CNCF project	accepted as a Sandbox Project
License	Apache 2.0
On-premises deployment	yes
OCI conformance*	yes
Single binary image*	yes
Minimal build*	yes
Storage Layout	OCI v1 Image Layout
Authentication	built-in
Authorization	built-in
Garbage collection	inline
Storage deduplication	inline
Cloud storage support	yes
Delete by tag	yes
Vulnerability scanning	built-in
Command line interface (cli)	yes
UI	yes
External contributions	beta available
Image signatures	built-in

✏️ * The **minimal build** feature is the ability to build a minimal Distribution Spec compliant registry in order to reduce library dependencies and the possible attack surface.

May 18, 2023

2.2 Summary of Key Features

- Conforms to OCI distribution spec APIs
- Uses OCI image layout for image storage
- Can serve any OCI image layout as a registry
- Single binary for **all** the features
- Doesn't require *root* privileges
- Clear separation between core dist-spec and zot-specific extensions
- Supports container image signatures - cosign and notation
- Supports helm charts
- Behavior controlled via configuration
- Binaries released for multiple operating systems and architectures
- Supports advanced image queries using *search* extension
- Supports image deletion by tag
- Currently suitable for on-premises deployments (e.g. colocated with Kubernetes)
- Compatible with ecosystem tools such as skopeo and cri-o
- Vulnerability scanning of images
- TLS support Authentication via:
- TLS mutual authentication
- HTTP *Basic* (local *htpasswd* and LDAP)
- HTTP *Bearer* token
- Supports Identity-Based Access Control
- Supports live modifications on the configuration file while zot is running (Authorization configuration only)
- Inline storage optimizations:
- Automatic garbage collection of orphaned blobs
- Layer deduplication using hard links when content is identical
- Data scrubbing
- Serve multiple storage paths (and backends) using a single zot server
- Pull and synchronize from other dist-spec conformant registries
- Supports rate limiting including per HTTP method
- Metrics with Prometheus
- Using a node exporter in case of minimal zot
- Swagger based documentation
- zli: command-line client support
- zb: a benchmarking tool for dist-spec conformant registries
- Released under Apache 2.0 License

September 13, 2023

2.3 Architecture

👉 **zot** is an OCI-native container image registry. This document discusses the design goals, the overall architecture, and the design choices made in the implementation of the design goals.

2.3.1 Design Goals

OCI-first

- HTTP APIs strictly conforms to the OCI Distribution Specification

  zot intends to be a **production** reference implementation for the OCI Distribution Specification. In fact, zot does not support any other vendor protocol or specification.

- Storage layout follows the OCI Image Specification

  The default and only supported storage layout is the OCI Image Layout. The implications of this choice are that any OCI image layout can be served by zot and conversely, zot converts data on-the-wire into an OCI image layout.

Single binary model

zot is a single binary image with all features included so that deployment is extremely simple in various environments, including bare-metal, cloud, and embedded devices. Behavior is controlled by a single configuration file.

Enable Only What You Need

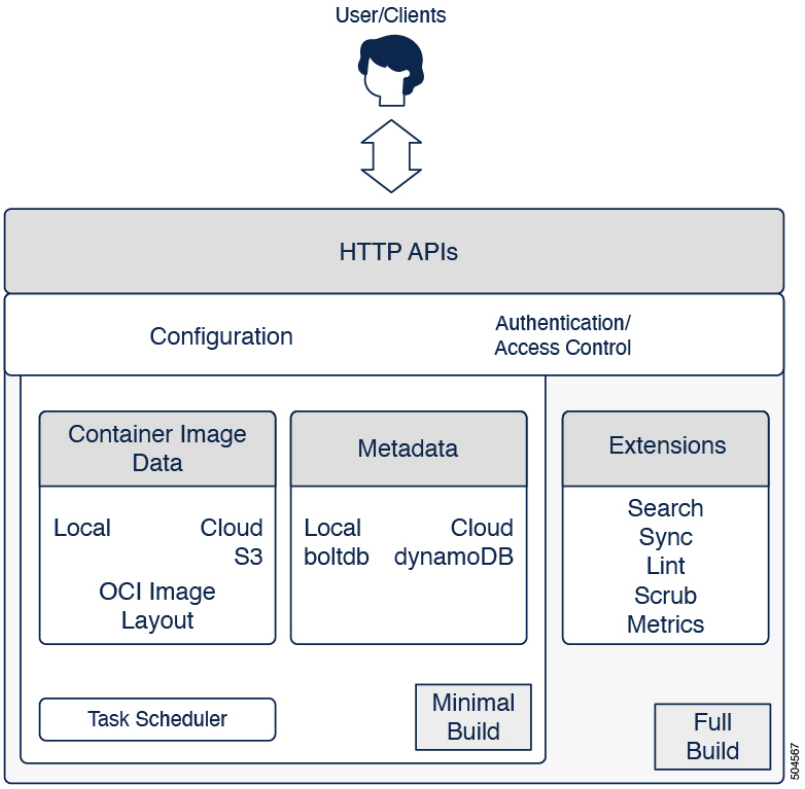
A clear separation exists between (1) the core OCI-compliant HTTP APIs and storage functionality, and (2) other add-on features modeled as **extensions**. The extension features can be selectively enabled both at build-time and run-time.

For more information, see "Conditional Builds" in zot's security posture document.

2.3.2 Overall Architecture

As shown in the following figure, the architecture of zot is organized as:

👉 zot-full = zot-minimal + extensions



The minimal build is the core OCI-compliant registry functionality as described by the OCI Distribution Specification.

The full build adds features that are not a part of the Distribution Specification, but are allowed to be added as Extensions.

External Interaction

External interaction with zot consists of the following two types:

- Client-initiated data or meta-data queries
- Admin-initiated configuration

All client-side interaction occurs over HTTP APIs. The core data path queries are governed by the OCI Distribution Specification. All additional meta-data queries are handled based on the setting of the `search` extension:

- If the `search` extension is enabled, enhanced registry searching and filtering is supported, using graphQL. A database is maintained by zot to efficiently answer complex queries on data stored in the database.
- If the `search` extension is not enabled, basic queries are supported using the core APIs. These queries are less efficient and search actual storage, which is limited in content.

Configuration

A single configuration file governs zot instance behavior. An exception can be made for security concerns, wherein configuration items containing sensitive credentials can be stored in separate files referenced by the main configuration file. Using separate files allows stricter permissions to be enforced on those files if stored locally. Also, modeling as external files allows for storing Kubernetes Secrets.

The configuration file is divided into sections for `http`, `storage`, `log`, and `extension`, governing the behavior of the respective components.

Authentication and Authorization

A robust set of authentication and authorization options are supported natively in zot. These controls are enforced before access is allowed into the storage layer.

For more information, see User Authentication and Authorization with zot.

Storage Driver Support

zot supports any modern local filesystem. Remote filesystems, such as AWS S3 or any AWS S3-compatible storage system, are supported. Additional driver support is planned in the roadmap.

✏️ Deduplication is supported for both local and remote filesystems, but deduplication requires a filesystem with hard-link support.

For more information, see Storage Planning with zot.

Security Scanning

zot integrates with the `trivy` security scanner to scan container images for vulnerabilities. The database is kept current by periodically downloading any vulnerability database updates at a configurable interval. The user remains agnostic of the actual scanner implementation, which may change over time.

Extensions

Additional registry features that are not a part of the Distribution Specification are added as Extensions.

✏️ Extension features of zot are available only with a full zot image. They are not supported in a minimal zot image.

For more information about zot's extensions, see *Extensions*.

Background Tasks

Several periodic tasks occur in the registry, such as garbage collection, sync mirroring, and scrubbing. A task scheduler handles these tasks in the background, taking care not to degrade or interrupt foreground tasks running in the context of HTTP APIs.

September 13, 2023

2.4 Extensions

👉 Extensions provide additional registry features that are not a part of the Distribution Specification.

The following extensions are currently available with zot:

- **Search (enhanced)**
- **Sync**
- **Lint**
- **Scrub**
- **Trust**
- **Metrics**
- **Graphical user interface**
- **User preferences**

💡 For detailed information about configuring zot extensions, see *Configuring zot*.

2.4.1 About extensions

The OCI Distribution Specification supports extending the functionality of an OCI-compliant registry implementation by adding extensions. Extensions are new APIs developed outside of the core OCI specs. Developers may propose their extensions to the OCI for possible future addition to the Distribution Specification.

Wherever applicable, extensions can be dynamically discovered using the extensions support of the OCI Distribution Specification.

⚠️ Extension features of zot are available only with a full zot image. They are excluded from the minimal zot image.

2.4.2 Extensions implemented in zot

The extensions implemented in zot include administrator-configured functionality and end-user features.

✏️ Currently, *search*, *trust*, and *userprefs* are the only zot extensions operable by end users. Only these extensions are accessible through HTTP APIs and are discoverable using the OCI extensions mechanism.

The following extensions are currently supported by zot:

Search

One of the key functions of a container image registry (which is essentially a graph of blobs) is the ability to perform interesting image and graph traversal queries. The user interacts with the **search** extension via a graphQL endpoint. The schema is published with every release.

Examples of queries are:

- "Does an image exist?"
- "What is its size?"
- "Does an image depend on this image via its layers?"
- "What vulnerabilities exist in an image or its dependent images?"

Sync

You can deploy a local mirror pointing to an upstream zot instance with various container image download policies, including on-demand and periodic downloads. The **sync** function is useful to avoid overwhelming the upstream instance, or if the upstream instance has rate-limited access.

💡 `docker.io` is supported as an upstream mirror.

Lint

The **lint** extension helps to avoid image compliance issues by enforcing certain policies about the image or the image metadata. Currently, **lint** can check an uploaded image to enforce the presence of required annotations such as the author or the license.

Scrub

Although container images are content-addressable with their SHA256 checksums, and validations are performed during storage and retrieval, it is possible that bit-rot sets in when not in use. The **scrub** extension actively scans container images in the background to proactively detect errors.

Trust

Images stored in zot can be signed with a digital signature to verify the source and integrity of the image. The digital signature can be verified by zot using public keys or certificates uploaded by the user through the zot API. The **trust** extension enables and configures this function.

Metrics

The **metrics** extension adds a node exporter, which is not present in the minimal build.

Graphical user interface

Using the zot graphical user interface (GUI), you can browse a zot registry for container images and artifacts. From the web interface, you can copy the shell commands for downloading an image using popular third-party tools such as docker, podman, and skopeo.

User preferences

The **userprefs** extension provides an API endpoint for adding configurable user preferences for a repository. This custom extension, not a part of the OCI distribution, is accessible only by authenticated users of the registry. Unauthenticated users are denied access.

The functions currently implemented by this extension include:

- Toggling the star (favorites) icon for a repository.
- Toggling the bookmark icon for a repository.

💡 For information about configuring zot extensions, see *Configuring zot*.

September 13, 2023

2.5 Released Images for zot

👉 This document describes the available zot images for the various supported hardware and software platforms, along with information about image variations, image locations, and image naming formats.

2.5.1 Supported platforms

zot is supported on Linux and Apple MacOS platforms with Intel or ARM processors.

Table: Supported platforms and architectures

OS	ARCH	Platform
linux	amd64	Intel-based Linux servers
linux	arm64	ARM-based servers and Raspberry Pi4
darwin	amd64	Intel-based MacOS
darwin	arm64	ARM-based MacOS (Apple M1)
freebsd	amd64	Intel-based FreeBSD*
freebsd	arm64	ARM-based FreeBSD*

**NOTE:* While binary images are available for FreeBSD, building container images is not supported at this time.

2.5.2 Full and minimal binary images

In addition to variations for specific platforms and architectures, binary images are also available in full and minimal flavors:

- A full zot binary image is compiled with all extensions. Extensions include functions such as metrics, registry synchronization, search, and scrub.
- A minimal distribution-spec conformant zot binary image is compiled with only a minimal set of code and libraries, reducing the attack surface. This option might be optimal for a registry embedded in a shipping product.

2.5.3 Binary image file naming

An executable binary image for zot is named using the target platform and architecture from the Supported platforms and architectures table. The general format of a binary image file name is one of these two:

`zot-<os>-<architecture>`

- A full zot binary image with all extensions has a filename of the form `zot-<os>-<architecture>.` For example, the full binary image for an Intel-based linux server is `zot-linux-amd64`.

`zot-<os>-<architecture>-minimal`

- A minimal distribution-spec conformant zot binary image has a filename of the form `zot-<os>-<architecture>-minimal`. For example, the minimal binary image for an Intel-based linux server is `zot-linux-amd64-minimal`.

💡 For convenience, you can rename the binary image file to simply `zot` after downloading.

2.5.4 Where to get zot

You can download native executable binary images or container (Docker) images.

Getting binary images

The zot project is hosted on GitHub at project-zot.

To download a binary image, go to the zot releases and select a release. Go to the **Assets** section of the release page and download the binary for your platform and architecture.

✏️ You may need to use the `chmod` command to make the image executable.

✏️ When downloading a binary image for MacOS, download the darwin image.

Getting container images

You can download a container image from `ghcr.io` by forming a URL with the desired image name, such as:

```
https://ghcr.io/project-zot/zot-<os>-<architecture>[-<build>]
```

If `<build>` is not specified, the default is `full`. For example, to download the minimal binary image for an Intel-based linux server. The URL is:

```
https://ghcr.io/project-zot/zot-linux-amd64-minimal
```

✏️ When downloading a container image for MacOS, download the linux image, not the darwin image.

2.5.5 Licensing

zot is released under the Apache License 2.0.

July 31, 2023

2.6 Glossary

2.6.1 Documentation Icons

Icon	Description
✏️	Note — A point of emphasis or caution.
💡	Tip — A helpful suggestion or a reference to additional material not covered in this document.
⚠️	Warning — A suggestion or advisory intended to avoid a loss of service or data.

2.6.2 Definitions

Term	Description
artifact	A file of any kind produced during a container build process or associated with the operation of a container. For example, a Helm chart is an artifact that might be stored along with a container.
CNCF	As part of the Linux Foundation, the Cloud Native Computing Foundation provides support, oversight, and direction for open-source, cloud native projects.
cosign	`cosign` is a tool that performs container signing, verification, and storage in an OCI registry.
cosigned	`cosigned` is an image admission controller that validates container images before deploying them.
cri-o	`cri-o` is an implementation of the Kubernetes Container Runtime Interface (CRI) to enable using OCI compatible runtimes. It is a lightweight alternative to using Docker as the runtime for Kubernetes.
deduplication	A storage space saving feature wherein only a single copy of specific content is maintained on disk while many different image manifests may hold references to that same content.
digest	A hashed checksum, such as SHA-256, for verifying the integrity of the downloaded image.
Distribution Specification	The OCI Distribution Specification project defines an API protocol to facilitate and standardize the distribution of content.
extensions	Additional registry features (APIs) that are not a part of the Distribution Specification can be added as Extensions.
helm chart	A helm chart is a package of files that orchestrate the deployment of Kubernetes resources into a Kubernetes cluster.
manifest	An image manifest provides a configuration and set of layers for a single container image for a specific architecture and operating system.
node exporter	A software component that collects hardware and operating system level metrics exposed by the kernel.
OCI	The Open Container Initiative (OCI) is an open governance structure for the express purpose of creating open industry standards around container formats and runtimes.
ORAS	OCI Registry as Storage (ORAS) is a tool for distributing OCI artifacts across OCI registries.
prometheus	Prometheus is a node exporter that exposes a wide variety of hardware- and kernel-related metrics.
referrer	An image containing a non-nil subject field with a descriptor to the referred image.
registry	A service that stores and distributes container images and artifacts.
repository	A collection of images with the same name, differentiated by tags.
skopeo	`skopeo` is a command line utility that performs various operations on container images and image repositories.
stacker	`stacker` is a standalone tool for building OCI images via a declarative `yaml` format. The output of the build process is a container image in an OCI layout.
tag	A label applied to an image that distinguishes the image from other images in the same repository. A common example is a version tag.
zb	A benchmarking tool, available as a zot companion binary, for benchmarking a zot registry or any other container image registry that conforms to the OCI Distribution Specification.
zli	A zot companion binary that implements a set of command line commands for interacting with the zot registry server.
zui	A zot companion binary that implements a graphical user interface (GUI) for interacting with the zot registry server.
zxp	

Term	Description
	A node exporter, available as a zot companion binary, that can be deployed with a minimal zot image in order to scrape metrics from the zot server.

August 4, 2023

2.7 About the zot Project

2.7.1 Project Repository

The zot project is hosted on GitHub:

project-zot/zot

2.7.2 Sponsors

Cisco Systems, Inc.

2.7.3 Adopters

Cisco Systems, Inc.

2.7.4 Presentations

OCI Weekly Discussion - Oct 2, 2019

January 8, 2023

3. User Guides

3.1 Push and Pull Image Content

👉 **zot** is an OCI image registry that allows you to store, manage, and share container images.

A zot registry can store and serve a variety of content, but the type of content may dictate your choice of a client tool.

For various content types, this document shows examples of using a third-party client tool that supports the content. The following table shows which content and client tools are demonstrated.

Content Type	Client
OCI images	skopeo
OCI images	regclient (`regctl`)
OCI images	crane
OCI artifacts	oras
Helm charts	helm

✏️ zot is compatible with kubernetes/cri-o using `docker://` transport, which is the default.

✏️ In the following examples, the zot registry is located at `localhost` , using port number 5000.

3.1.1 Common tasks using skopeo for OCI images

`skopeo` is a command line client that performs various operations on OCI container images and image repositories.

✏️ For detailed information about using skopeo, see the skopeo man page.

Push an OCI image

This example pushes the latest container image for the `busybox` application to a zot registry.

```
$ skopeo --insecure-policy copy --dest-tls-verify=false --multi-arch=all \
   --format=oci docker://busybox:latest \
   docker://localhost:5000/busybox:latest
```

Pull an OCI image

This example pulls the latest container image for the `busybox` application and stores the image to a local OCI-layout directory (`/oci/images`).

```
$ skopeo --insecure-policy copy --src-tls-verify=false --multi-arch=all \
   docker://localhost:5000/busybox:latest \
   oci:/oci/images:busybox:latest
```

Pull an OCI image to a private docker registry

This example pulls the latest container image for the `busybox` application and stores the image to a local private docker registry.

```
$ skopeo --insecure-policy copy --src-tls-verify=false --multi-arch=all \
   docker://localhost:5000/busybox:latest \
   docker://localhost:5000/busybox:latest
```

ck here to view an example of pushing and pulling an image using skopeo.

Authentication

In these examples, authentication is disabled for the source and destination. You can enable authentication by changing the command line options as follows:

```
--src-tls-verify=true
--dest-tls-verify=true
```

You can also add credentials for authenticating with a source or destination repository:

```
--src-creds username:password
--dest-creds username:password
```

3.1.2 Common tasks using regclient for OCI images

regclient is a client interface that performs various operations on OCI container images and image repositories. The command line interface for regclient is `regctl.`

✏️ For detailed information about `regctl` commands, see the regctl Documentation.

Push an OCI image

This example pushes version 1.20 of `golang` to a `tools` repository within the registry.

```
$ regctl registry set --tls=disabled localhost:5000
$ regctl image copy ocidir://path/to/golang:1.20 localhost:5000/tools
```

Pull an OCI image

This example pulls version 1.20 of `golang` to a local OCI-layout directory.

```
$ regctl image copy localhost:5000/tools ocidir://path/to/golang:1.20
```

List all repositories in registry

This example list all repositories in the registry.

```
$ regctl repo ls localhost:5000
```

List tags

This example lists all tags in the `tools` repository within the registry.

```
$ regctl tag ls localhost:5000/tools
```

Pull and push manifest

This example pulls and pushes the manifest in the `tools` repository within the registry.

```
$ regctl manifest get localhost:5000/tools --format=raw-body
$ regctl manifest put localhost:5000/tools:1.0.0 \
--format oci --content-type application/vnd.oci.image.manifest.v1+json \
--format oci
```

Authentication

In the preceding examples, TLS authentication with the zot registry was disabled by the following command:

```
$ regctl registry set --tls=disabled localhost:5000
```

This command allows `regctl` to accept an HTTP response from the zot server. If TLS authentication is enabled on the zot registry server, you can omit this command from your `regctl` session.

3.1.3 Common tasks using oras for OCI artifacts

ORAS (OCI Registry As Storage) is a command line client for storing OCI artifacts on OCI repositories.

✏️ For detailed information about the `oras` commands in these examples, see the ORAS CLI documentation.

Push an artifact

This example pushes version 2 of an artifact file named `hello-artifact` to a zot registry.

```
$ oras push --plain-http localhost:5000/hello-artifact:v2 \
        --config config.json:application/vnd.acme.rocket.config.v1+json \
        artifact.txt:text/plain -d -v
```

Pull an artifact

This example pulls version 2 of an artifact file named `hello-artifact` from a zot registry.

```
$ oras pull --plain-http localhost:5000/hello-artifact:v2 -d -v
```

ck here to view an example of pushing and pulling an artifact using oras.

Attach a reference

```
$ echo '{"artifact": "localhost:5000/hello-artifact:v2", "signature": "pat hancock"}' > signature.json

$ oras attach \
  --artifact-type 'signature/example' \
  localhost:5000/hello-artifact:v2 \
  ./signature.json:application/json

$ oras discover -o tree localhost:5000/hello-artifact:v2
```

Authentication

To authenticate with the zot server, log in at the start of your session using the following command:

```
$ oras login -u myUsername -p myPassword localhost:5000
```

You can also add credentials in the push or pull commands as in this example:

```
$ oras pull -u myUsername -p myPassword localhost:5000/hello-artifact:v2 -d -v
```

✏️ For additional authentication options, including interactive credential entry and disabling TLS, see the ORAS authentication documentation.

3.1.4 Common tasks using helm for helm charts

Helm is a package manager for Kubernetes. Among many other capabilities, helm can store and retrieve helm charts on OCI image repositories.

✏️ For detailed information about the `helm` commands in these examples, see Commands for working with registries in the helm documentation.

Push a helm chart

This example pushes version 1.2.3 of a zot helm chart to a `zot-chart` repository within the registry.

```
$ helm package path/to/helm-charts/charts/zot
$ helm push zot-1.2.3.tgz oci://localhost:5000/zot-chart
```

Pull a helm chart

This example pulls version 1.2.3 of a zot helm chart from a `zot-chart` repository within the registry.

```
$ helm pull oci://localhost:5000/zot-chart/zot --version 1.2.3
```

Authentication

To authenticate with the zot server, log in at the start of your session using the following command:

```
$ helm registry login -u myUsername localhost:5000
```

You will be prompted to manually enter a password.

3.1.5 Common tasks using crane for OCI images

crane is an open-source project that provides a command-line interface (CLI) for interacting with container registries, such as Docker Hub and Google Container Registry.

✏️ For detailed information about `crane` commands, see the crane Documentation.

Push an OCI image

This example pushes the latest container image for the `alpine` application to a registry.

```
$ crane --insecure push \
   oci/images/alpine:latest \
   localhost:5000/alpine:latest
```

Pull an OCI image

This example pulls the latest container image for the `alpine` application and stores the image to a local OCI-layout directory (`/oci/images`).

```
$ crane --insecure pull \
   --format oci \
   localhost:5000/alpine:latest \
   oci/images/alpine:latest
```

Copy an OCI image to a private docker registry

This example pulls the latest container image for the `alpine` application and stores the image to a local private docker registry.

```
$ crane --insecure copy \
   alpine:latest \
   localhost:5000/alpine:latest
```

List tags

This example lists all tags in the `alpine` image within the registry.

```
$ crane ls localhost:5000/alpine
```

Find the digest of an image

This example gets the digest of the `alpine` image within the registry.

```
$ crane digest localhost:5000/alpine:latest
```

Authentication

To authenticate with the registry server, log in at the start of your session using the following command:

```
$ crane auth login -u myUsername localhost:5000
```

July 23, 2024

3.2 Use the Web Interface to Find Images

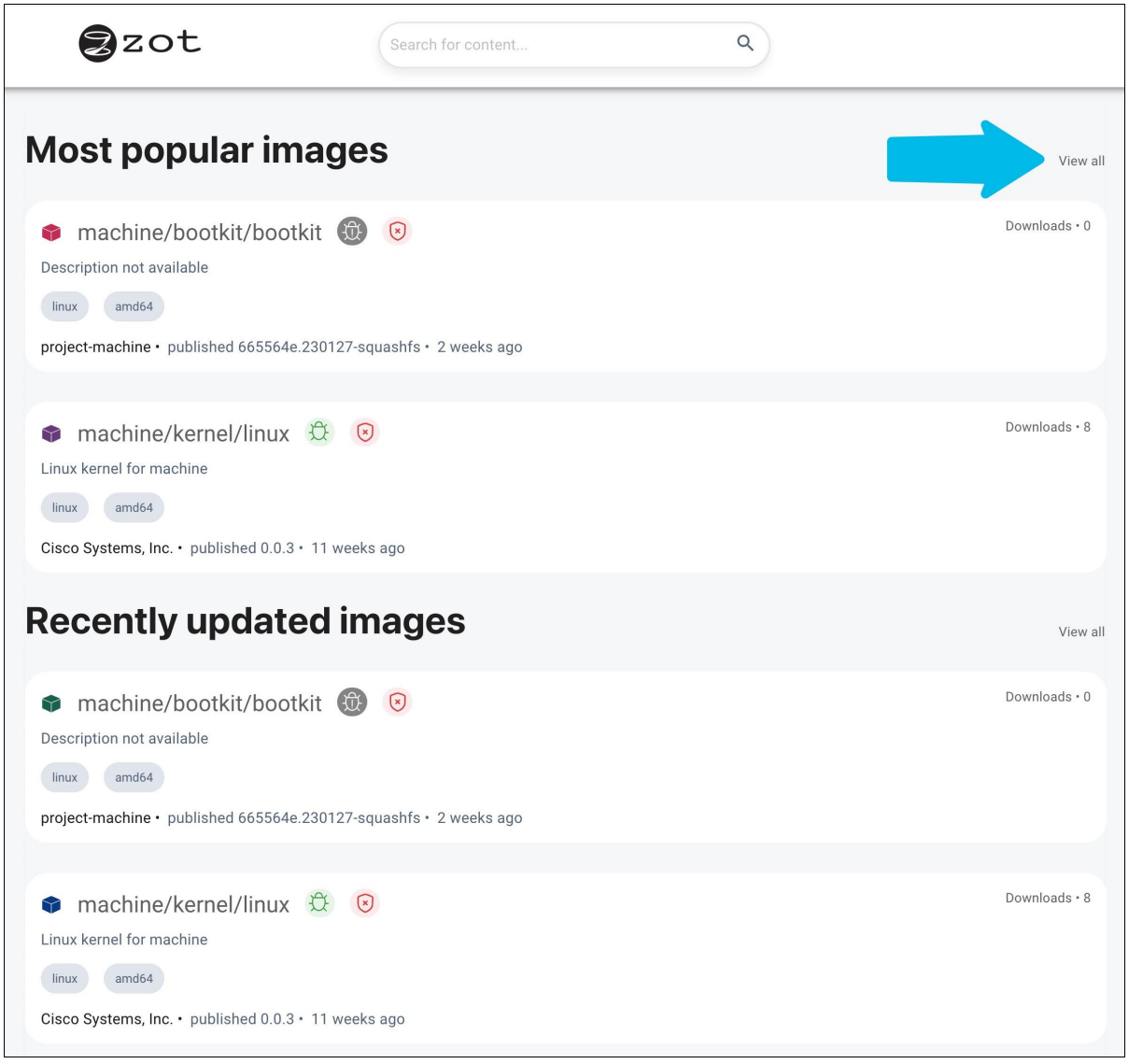
👉 Using a browser, you can browse a **zot** repository for container images and artifacts. The web interface also provides the commands for downloading an image using popular third-party tools.

If a zot registry is built with the optional `zui` package, the registry has a web interface.

3.2.1 Access the registry

✏️ Depending on the security configuration of the zot registry, you might need to authenticate before being given access.

The initial page of the registry displays a sampling of the most popular images and recently updated images. To view all available images in the registry, click the `View all` link in the upper right of the page to go to the `/explore` page.

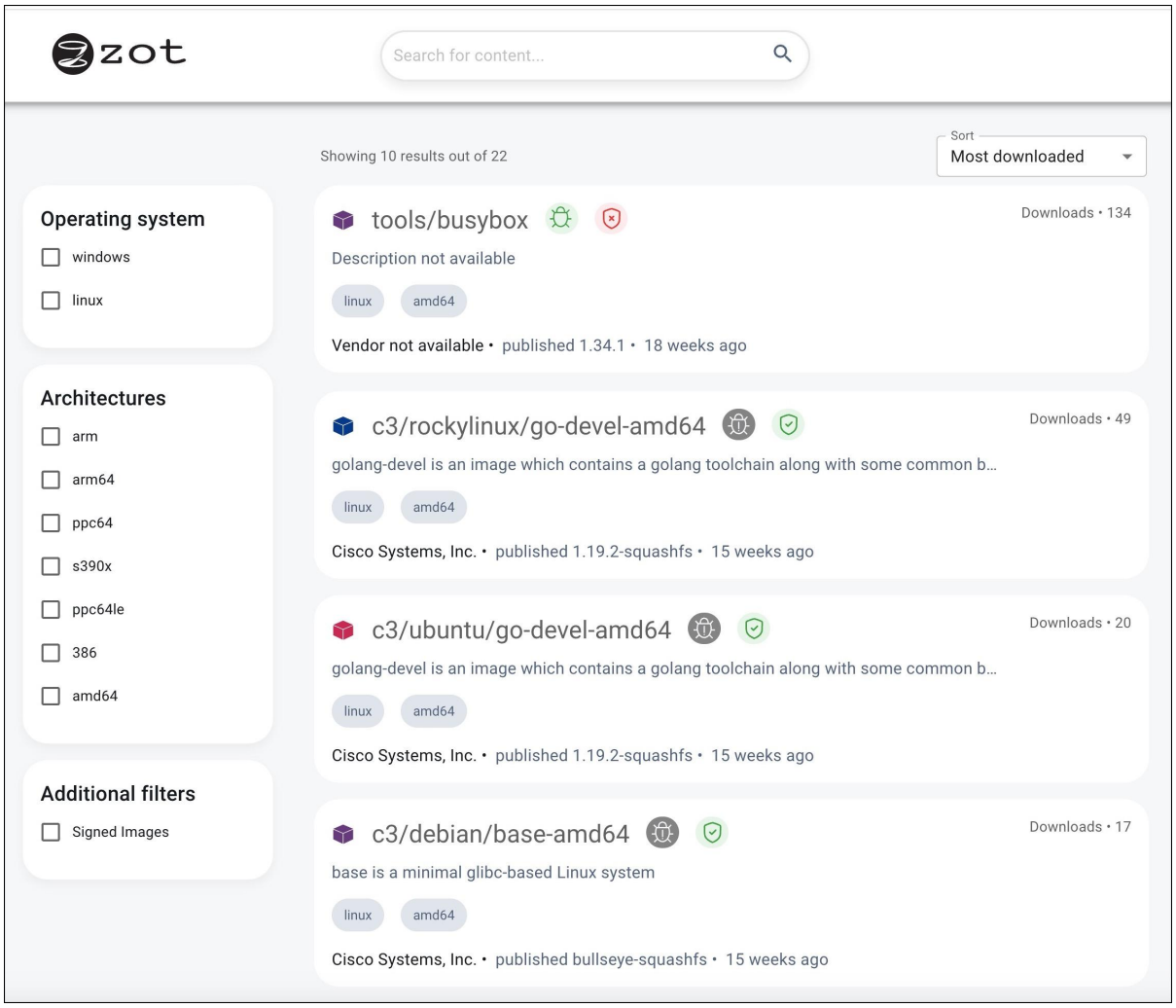


💡 Wherever an image name appears, two icons follow the name, indicating the results of the vulnerability scan and the signature status. These icons are described in Icons and their meanings.

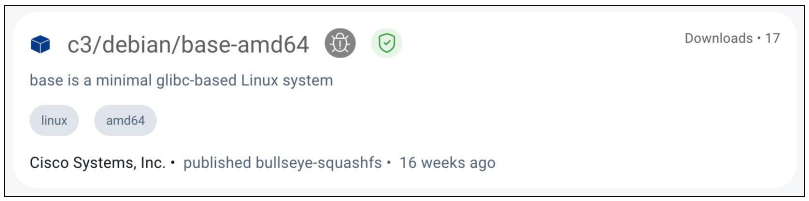
3.2.2 Find an image

To assist you in finding images specific to your requirements, the `/explore` page provides several sorting options along with faceted navigation in addition to a general search box.

- Sorting criteria include relevance, most recently updated, alphabetical, and most downloaded.
- Navigation facets include operating system, CPU architecture, and signature status.



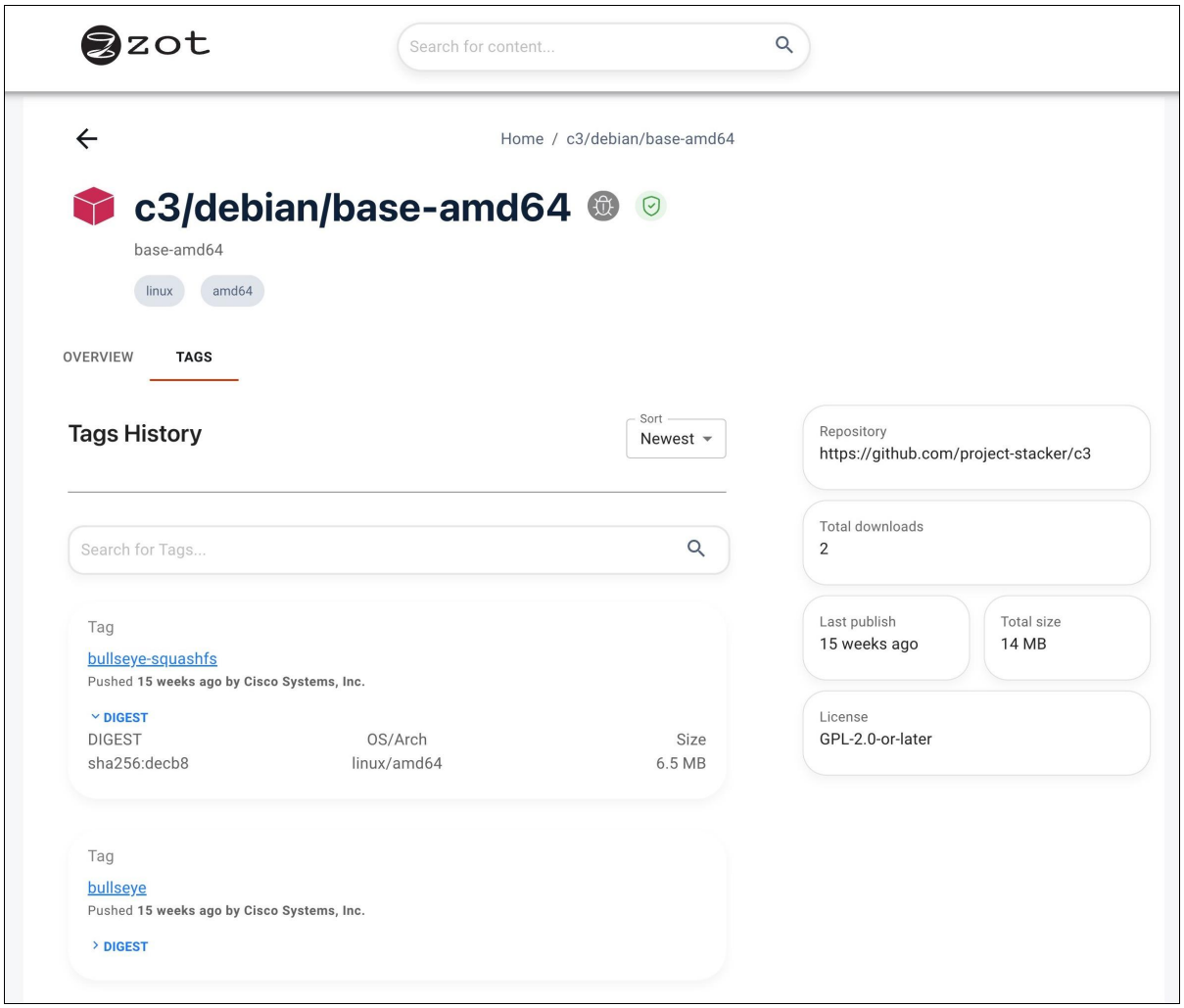
When you locate the desired image, click its tile.



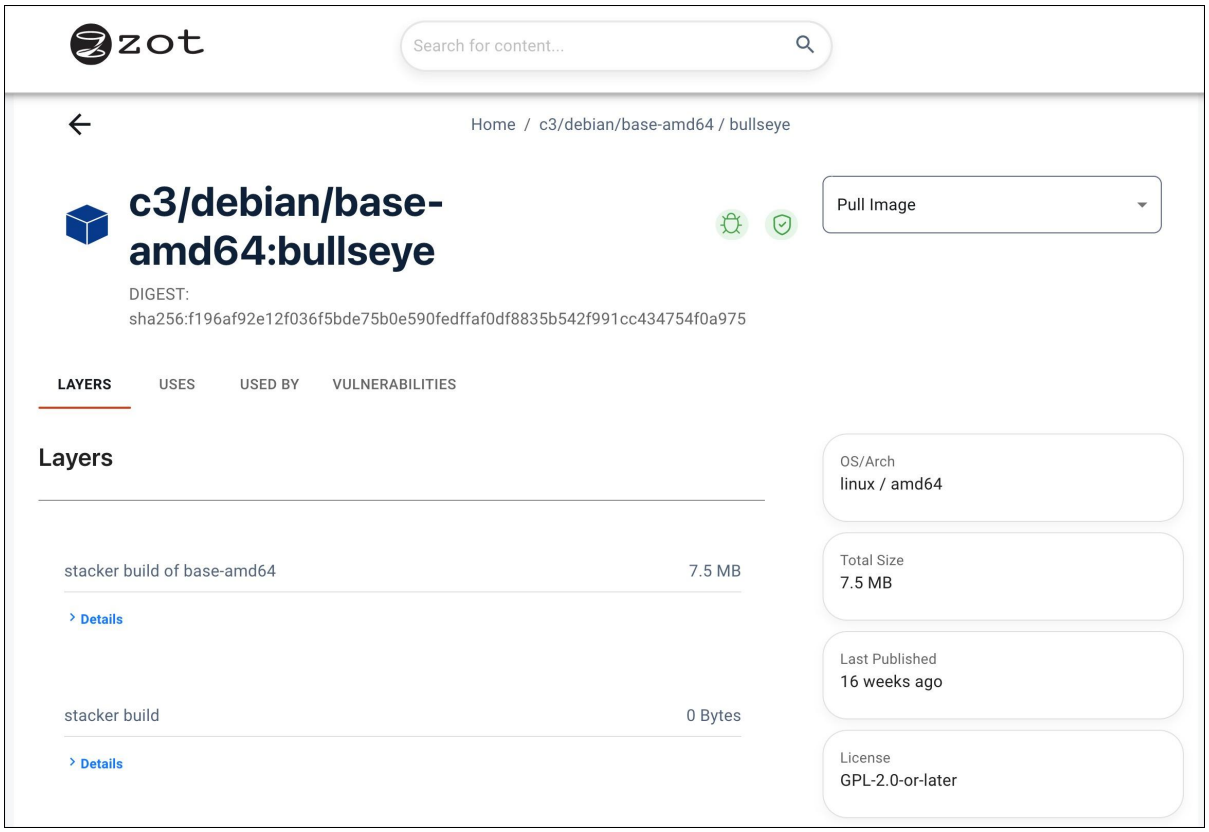
3.2.3 Inspect the image properties

The `OVERVIEW` tab of the initial image description page contains a brief description of the image along with several details, such as the repository URL, number of downloads, last published, size, and license type.

Click the `TAGS` tab to view the available versions of the image.



Click the tag link of the desired version of the image. In the resulting page, you can view detailed information about the image, including its digest hash, and you can obtain a command to pull the image from the repository.



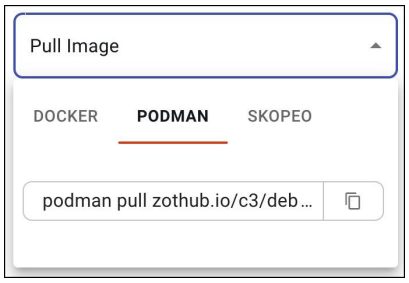
Tabs on this page provide the following information:

Tab	Description
LAYERS	The build command and the digest hash
USES	A list of images used by this image
USED BY	A list of images that use this image
VULNERABILITIES	A list of known vulnerabilities

From the `LAYERS` tab, click `Details` to view an example of the image build command and the layer's digest hash.

3.2.4 Pull an image

After opening the tag link of the desired image, click the `Pull Image` drop-down list to display the available pull commands. Docker, podman, and skopeo image management tools are supported.



Select a tool and click the pages icon next to the command to copy the full command to your clipboard. An example of the pull command using podman is:

```
podman pull zothub.io/c3/debian/base-amd64:bullseye
```

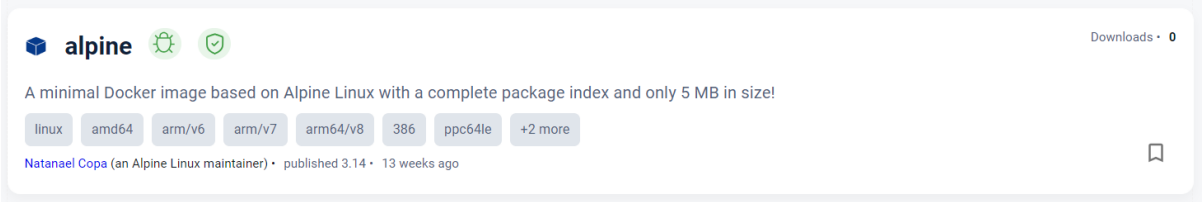
Paste the pull command into your terminal to pull the image.

💡 For information about using image management tools to pull images, see Push and Pull Image Content.

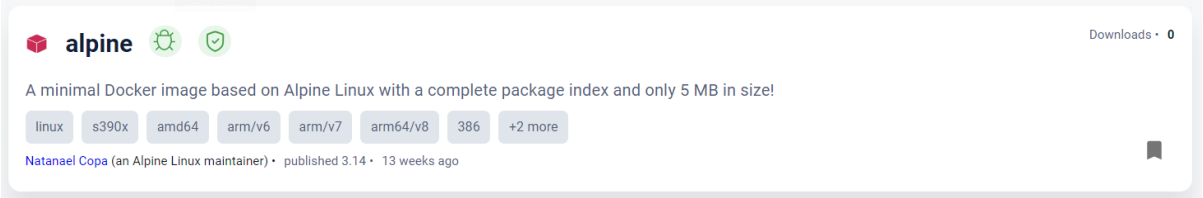
3.2.5 Adding Bookmarks

To mark an image so that it can be easily found later, click the bookmark icon on the image page. Bookmarked images appear in search queries when the Bookmarked option is enabled.

The bookmark icon before selection:



The bookmark icon after selection:



3.2.6 Icons and their meanings

These icons appear next to the image name, indicating the results of the vulnerability scan and the signature status.

Icon	Description
	A vulnerability scan of the image found no vulnerabilities
	A vulnerability scan of the image failed (See severity icon in VULNERABILITIES tab)
	Critical severity
	High severity
	Medium severity
	Low severity
	The image signature is verified
	The image signature is not verified
	The image is not bookmarked by the user
	The image is bookmarked by the user

August 4, 2023

3.3 Using the command line interface (zli)

👉 **zli**: The command line tool for zot servers

3.3.1 What is zli?

zli is a binary that implements a set of command line commands for interacting with the zot registry server.

💡 We recommend installing zli when you install zot.

3.3.2 How to get zli

zli is hosted with zot on GitHub at project-zot. From GitHub, you can download the zli binary or you can build zli from the source.

Supported platforms

zli is supported for the following operating systems and platform architectures:

OS	ARCH	Platform
linux	amd64	Intel-based Linux servers
linux	arm64	ARM-based servers and Raspberry Pi4
darwin	amd64	Intel-based MacOS
darwin	arm64	ARM-based MacOS

Downloading zli binaries

You can download the executable binary for your server platform and architecture under "Assets" on the GitHub zot releases page.

The binary image is named using the OS and architecture from the Supported platforms table. For example, the binary for an Intel-based MacOS server is `zli-darwin-amd64.`

Building zli from source

To build the zli binary, copy or clone the zot project from GitHub and execute the `make cli` command in the `zot` directory. Use the same command options that you used to build zot, as shown:

```
make OS=os ARCH=architecture cli
```

For example, the following command builds zli for an Intel-based MacOS server:

```
make OS=darwin ARCH=amd64 cli
```

In this example, the resulting executable file is `zli-darwin-amd64` in the `zot/bin` directory.

3.3.3 Common tasks using zli

This section includes examples of common zot server tasks using the zli command line interface. For a detailed listing of zli commands, see the zli Command Reference in this guide.

💡 The original filename of the zli executable file will reflect the build options, such as `zli-linux-amd64`. For convenience, you can rename the executable to simply `zli`. The instructions and examples in this guide use `zli` as the name of the executable file.

Adding a zot server URL

You can modify the zot server configuration using the `zli config add` command. This example adds a zot server URL with an alias of `remote-zot`:

```
$ bin/zli config add remote-zot https://server-example:8080
```

Use the `zli config` command to list all configured URLs with their aliases:

```
$ bin/zli config -l

remote-zot   https://server-example:8080
local        http://localhost:8080
```

Listing images

You can list all images hosted on a zot server using the `zli image list` command with the server's alias:

```
$ bin/zli image list --config local

REPOSITORY        TAG       OS/ARCH         DIGEST      SIGNED      SIZE
alpine            latest    linux/amd64     3fc10231    false       84MB
busybox           latest    linux/amd64     9172c5f6    false       2.2MB
```

You can also filter the image list to view a specific image by specifying the image name:

```
$ bin/zli image name busybox:latest --config local

REPOSITORY        TAG       OS/ARCH         DIGEST      SIGNED      SIZE
busybox           latest    linux/amd64     9172c5f6    false       2.2MB
```

Scanning images for known vulnerabilities

Using the `zli cve list` command, you can fetch the CVE (Common Vulnerabilities and Exposures) information for images hosted on the zot server. This example shows how to learn which images are affected by a specific CVE:

```
$ bin/zli cve affected CVE-2017-9935 --config remote-zot

IMAGE NAME        TAG               DIGEST    SIZE
c3/openjdk-dev    commit-5be4d92    ac3762e2  335MB
```

This example displays a list of all CVEs affecting a specific image:

```
$ bin/zli cve list c3/openjdk-dev:0.3.19 --config remote-zot

ID                SEVERITY  TITLE
CVE-2015-8540     LOW       libpng: underflow read in png_check_keyword()
CVE-2017-16826    LOW       binutils: Invalid memory access in the coff_s...
```

This example (--verbose) displays a list of all CVEs affecting a specific image with details:

```
$ bin/zli cve list c3/openjdk-dev:0.3.19 --config remote-zot --verbose
LOW 2, UNKNOWN 1, TOTAL 3

CVE-2015-8540
...
```

Note that the details may display the package path in the image when the information is available.

This example displays the detailed CVEs in JSON format:

```
$ bin/zli cve list c3/openjdk-dev:0.3.19 --config remote-zot -f json
{
  "Tag": "0.3.19",
  "CVEList": [
    {
      "Id": "CVE-2019-17006",
      "Severity": "MEDIUM",
      "Title": "nss: Check length of inputs for cryptographic primitives",
      "Description": "A vulnerability was discovered in nss where input text length was not checked when using certain cryptographic primitives. This could
lead to a heap-buffer overflow resulting in a crash and data leak. The highest threat is to confidentiality and integrity of data as well as system
availability.",
      "PackageList": [
```

```
        {
          "Name": "nss",
          "InstalledVersion": "3.44.0-7.el7_7",
          "FixedVersion": "Not Specified"
        },
        {
          "Name": "nss-sysinit",
          "InstalledVersion": "3.44.0-7.el7_7",
          "FixedVersion": "Not Specified"
        },
        {
          "Name": "nss-tools",
          "InstalledVersion": "3.44.0-7.el7_7",
          "FixedVersion": "Not Specified"
        }]
    }]
}
```

This example lists all images on a specific zot server that are affected by a specific CVE:

```
$ bin/zli cve affected --config remote-zot CVE-2017-9935 --repo c3/openjdk-dev

IMAGE NAME        TAG               DIGEST    SIZE
c3/openjdk-dev    commit-2674e8a    71046748  338MB
c3/openjdk-dev    commit-bd5cc94    0ab7fc76
```

This example lists all images on a specific zot server where the CVE has been fixed:

```
$ bin/zli cve fixed c3/openjdk-dev CVE-2017-9935 --config remote-zot

IMAGE NAME        TAG                       DIGEST    SIZE
c3/openjdk-dev    commit-2674e8a-squashfs   b545b8ba  321MB
c3/openjdk-dev    commit-d5024ec-squashfs   cd45f8cf  321MB
```

This example lists all CVEs that have been found in one image and not the other:

```
$ bin/zli cve diff c3/openjdk-dev:1.0.0 c3/openjdk-dev:2.0.0 --config remote-zot

ID                SEVERITY  TITLE
CVE-2015-8540     LOW       libpng: underflow read in png_check_keyword()
CVE-2017-16826    LOW       binutils: Invalid memory access in the coff_s...
```

For example, the above query lists all CVEs that have been found in c3/openjdk-dev:1.0.0 but not in c3/openjdk-dev:2.0.0

Listing repositories

You can list all repositories hosted on a zot server using the `zli repo` command with the server's alias:

```
Searching... 🌍

REPOSITORY NAME
alpine
busybox
```

Searching for repositories and images

You can locate repositories and images hosted on a zot server using the `zli search` command.

- To search for a repository, specify the full name with a colon or a partial name with no colon.
- To search for an image, specify the full repository name followed by the tag or a prefix of the tag.

This example searches the zot registry named 'local' for a repository whose name contains the substring 'ng':

```
$ bin/zli search query ng --config local

NAME            SIZE        LAST UPDATED                            DOWNLOADS   STARS
nginx           794MB       2023-03-01 18:44:17.707690369 +0000 UTC  0           0
mongo           232MB       2022-10-18 15:03:40.7646203 +0300 +0300  0           0
golang          1.1GB       2023-06-22 00:32:38.613354854 +0000 UTC  0           0
```

This example searches the zot registry named 'local' for a repository named 'nginx'. Because the repository name is followed by a colon, the search results must match the name exactly.

```
$ bin/zli search query nginx: --config local
```

```
REPOSITORY  TAG          OS/ARCH         DIGEST     SIGNED     SIZE
nginx       1.23.1       linux/amd64     d2ad9089   true       57MB
nginx       latest       *               c724afdf   true       448MB
                         linux/amd64     009c6fda   false      57MB
                         linux/arm/v5    1d5d4f53   false      54MB
                         linux/arm/v7    f809744c   false      50MB
                         linux/arm64/v8  ebb807a9   false      56MB
                         linux/386       19cf4b3c   false      59MB
                         linux/mips64le  45ab60e6   false      55MB
                         linux/ppc64le   89511bee   false      63MB
                         linux/s390x     713b9329   false      55MB
nginx       stable-perl  *               4383a0b8   true       534MB
                         linux/amd64     308a37a0   false      68MB
                         linux/arm/v5    0fb8fb71   false      64MB
                         linux/arm/v7    6868f552   false      60MB
                         linux/arm64/v8  aed72c86   false      66MB
                         linux/386       5c7ed456   false      69MB
                         linux/mips64le  546d2bae   false      65MB
                         linux/ppc64le   7db02f5a   false      74MB
                         linux/s390x     800fd86f   false      66MB
```

3.3.4 Sorting the output of a zli command

For a zli command that can result in a lengthy output list, you can use the command flag `--sort-by <option>` to cause the output to be sorted by a specified property of the output data. The available sorting criteria vary for different commands, but examples of sorting criteria options are described in the following table:

flag option	criteria
`alpha-asc`	alphabetical, ascending
`alpha-dsc`	alphabetical, descending
`relevance`	quality of match
`severity`	severity of condition
`update-time`	timestamp

For a given command that results in an output list, you can see the available sorting criteria in the usage information returned by the `--help` flag. For example, `bin/zli image name --help` returns usage information containing the following line under "Flags":

```
--sort-by string Options for sorting the output: [update-time, alpha-asc, alpha-dsc] (default "alpha-asc")
```

According to this information, the list of image names returned by the `bin/zli image name` command can be sorted in order of alphabetical ascending, alphabetical descending, or the timestamp of the latest update of the image. The default sorting method for this command, if no `--sort-by` flag is present, is alphabetical ascending.

3.3.5 Command reference

This section provides detailed usage information for basic first-level zli commands. Many zli commands also support subcommands, which are listed as "Available Commands" in each command description. For example, `zli search` can be extended with either the `query` or `subject` subcommand. To see the detailed usage for each subcommand, type the command with the subcommand and append `--help`, such as `zli search query --help`. The `zli search` description below includes the subcommand help as an example.

zli

```
$ bin/zli --help

Usage:
  zli [flags]
  zli [command]

Available Commands:
  completion  Generate the autocompletion script for the specified shell
  config      Configure zot registry parameters for CLI
  cve         Lookup CVEs in images hosted on the zot registry
  help        Help about any command
  image       List images hosted on the zot registry
  repo        List all repositories
  search      Search images and their tags
```

```
Flags:
  -h, --help      help for zli
  -v, --version   show the version and exit

Use "zli [command] --help" for more information about a command.
```

zli completion

This command generates the autocompletion script for `zli` for the specified shell. See each subcommand's help for details on how to use the generated script.

```
$ bin/zli completion --help

Usage:
  zli completion [command]

Available Commands:
  bash        Generate the autocompletion script for bash
  fish        Generate the autocompletion script for fish
  powershell  Generate the autocompletion script for powershell
  zsh         Generate the autocompletion script for zsh

Flags:
  -h, --help   help for completion

Use "zli completion [command] --help" for more information about a command.
```

zli config

This command configures zot registry parameters for CLI.

```
$ bin/zli config --help

Usage:
  zli config <config-name> [variable] [value] [flags]
  zli config [command]

Examples:
  zli config add main https://zot-foo.com:8080
  zli config --list
  zli config main url
  zli config main --list
  zli config remove main

Available Commands:
  add         Add configuration for a zot registry
  remove      Remove configuration for a zot registry

Flags:
  -h, --help    help for config
  -l, --list    List configurations
      --reset   Reset a variable value

Use "zli config [command] --help" for more information about a command.

Useful variables:
  url       zot server URL
  showspinner   show spinner while loading data [true/false]
  verify-tls    enable TLS certificate verification of the server [default: true]
```

zli cve

This command lists CVEs (Common Vulnerabilities and Exposures) of images hosted on the zot registry

```
$ ./zli cve --help

Usage:
  zli cve [command]

Available Commands:
  affected    List images affected by a CVE
  fixed       List tags where a CVE is fixedRetryWithContext
  list        List CVEs by REPO:TAG or REPO@DIGEST

Flags:
      --config string   Specify the registry configuration to use for connection
      --debug           Show debug output
  -f, --format string   Specify output format [text/json/yaml]
  -h, --help            help for cve
      --url string      Specify zot server URL if config-name is not mentioned
  -u, --user string     User Credentials of zot server in "username:password" format
```

```
      --verbose         Show verbose output

Use "zli cve [command] --help" for more information about a command.

Run 'zli config -h' for details on [config-name] argument
```

zli image

This command lists images hosted on the zot registry.

```
$ ./zli image --help

Usage:
  zli image [command]

Available Commands:
  base        List images that are base for the given image
  cve         List all CVE's of the image
  derived     List images that are derived from given image
  digest      List images that contain a blob(manifest, config or layer) with the given digest
  list        List all images
  name        List image details by name

Flags:
      --config string   Specify the registry configuration to use for connection
      --debug           Show debug output
  -f, --format string   Specify output format [text/json/yaml]
  -h, --help            help for image
      --url string      Specify zot server URL if config-name is not mentioned
  -u, --user string     User Credentials of zot server in "username:password" format
      --verbose         Show verbose output

Use "zli image [command] --help" for more information about a command.

Run 'zli config -h' for details on [config-name] argument
```

zli repo

This command lists all repositories in the zot registry.

```
$ ./zli repo --help

Usage:
  zli repo [command]

Available Commands:
  list        List all repositories

Flags:
      --config string   Specify the registry configuration to use for connection
      --debug           Show debug output
  -h, --help            help for repo
      --url string      Specify zot server URL if config-name is not mentioned
  -u, --user string     User Credentials of zot server in "username:password" format

Use "zli repo [command] --help" for more information about a command.

Run 'zli config -h' for details on [config-name] argument
```

zli search

The `search` command allows smart searching for a repository by its name or for an image by its repo:tag.

```
$ ./zli search --help

Search repos or images

Usage:
  zli search [command]

Available Commands:
  query       Fuzzy search for repos and their tags.
  subject     List all referrers for this subject.

Flags:
      --config string   Specify the registry configuration to use for connection
      --debug           Show debug output
  -f, --format string   Specify output format [text/json/yaml]
  -h, --help            help for search
      --url string      Specify zot server URL if config-name is not mentioned
  -u, --user string     User Credentials of zot server in "username:password" format
      --verbose         Show verbose output
```

```
Use "zli search [command] --help" for more information about a command.

Run 'zli config -h' for details on [config-name] argument
```

ZLI SEARCH QUERY

```
$ ./zli search query --help

Usage:
  zli search query [repo]|[repo:tag] [flags]

Examples:
# For repo search specify a substring of the repo name without the tag
  zli search query "test/repo"

# For image search specify the full repo name followed by the tag or a prefix of the tag.
  zli search query "test/repo:2.1."

Flags:
  -h, --help             help for query
      --sort-by string   Options for sorting the output: [relevance, update-time, alpha-asc, alpha-dsc] (default "relevance")

Global Flags:
      --config string   Specify the registry configuration to use for connection
      --debug           Show debug output
  -f, --format string   Specify output format [text/json/yaml]
      --url string      Specify zot server URL if config-name is not mentioned
  -u, --user string     User Credentials of zot server in "username:password" format
      --verbose         Show verbose output

Run 'zli config -h' for details on [config-name] argument
```

ZLI SEARCH SUBJECT

```
$ ./zli search subject --help

List all referrers for this subject. The subject can be specified by tag(repo:tag) or by digest" or (repo@digest)

Usage:
  zli search subject [repo:tag]|[repo@digest] [flags]

Examples:
# For referrers search specify the referred subject using it's full digest or tag:
  zli search subject "repo@sha256:f9a0981..."
  zli search subject "repo:tag"

Flags:
  -h, --help             help for subject
      --sort-by string   Options for sorting the output: [update-time, alpha-asc, alpha-dsc] (default "alpha-asc")

Global Flags:
      --config string   Specify the registry configuration to use for connection
      --debug           Show debug output
  -f, --format string   Specify output format [text/json/yaml]
      --url string      Specify zot server URL if config-name is not mentioned
  -u, --user string     User Credentials of zot server in "username:password" format
      --verbose         Show verbose output

Run 'zli config -h' for details on [config-name] argument
```

March 13, 2024

4. Installation Guides

4.1 Installing zot on Bare Metal Linux

👉 Using an available executable zot image, you can easily deploy zot on a Linux server.

4.1.1 Before you begin

About binary images

Executable binary zot images are available for multiple platforms and architectures and with full or minimal implementations.

Refer to *Released Images for zot* for information about available zot images along with information about image variations, image locations, and image naming formats.

4.1.2 Installation

Step 1: Get zot

Using `wget,` download the appropriate zot binary image for your platform from the zot GitHub project. Download the image to the`/usr/bin/` directory and rename it to `zot,` as in this example:

```
sudo wget -O /usr/bin/zot https://github.com/project-zot/zot/releases/download/v2.1.7/zot-linux-amd64
```

Then fix permissions to it:

```
sudo chmod +x /usr/bin/zot
sudo chown root:root /usr/bin/zot
```

Step 2: Create a zot configuration file

Create a zot configuration file as `/etc/zot/config.json`.

See Configuration file options for an example file with options and recommendations. You can find other configuration file examples in the zot GitHub project and in *Configuring zot*.

Step 3: Configure a local authentication account

If you want to use local authentication with zot, create a `/etc/zot/htpasswd` file with an initial account entry using the `htpasswd` command as in this example:

```
htpasswd -bnB myUserName myPassword > /etc/zot/htpasswd
```

To add additional local users, use the `>>` redirect as in this example:

```
htpasswd -bnB myUserName2 myPassword2 >> /etc/zot/htpasswd
```

Step 4: Define the zot service

Create a `/etc/systemd/system/zot.service` file to define the zot service in systemd. The following is an example service file for zot:

```
[Unit]
Description=OCI Distribution Registry
Documentation=https://zotregistry.dev/
After=network.target auditd.service local-fs.target

[Service]
Type=simple
ExecStart=/usr/bin/zot serve /etc/zot/config.json
Restart=on-failure
User=zot
```

```
Group=zot
LimitNOFILE=500000
MemoryHigh=30G
MemoryMax=32G

[Install]
WantedBy=multi-user.target
```

✏️ Be sure to configure a dedicated non-root user ID as the User and Group in the zot service definition. The 'zot' user ID in this example is created in the next step.

Step 5: Create a user ID to own the zot service

Create a non-root user ID to be the owner of the zot service and its resources.

In this example, the user ID 'zot' is created with the `adduser` command, and resource ownership is assigned.

```
sudo adduser --no-create-home --disabled-password --gecos --disabled-login zot

sudo mkdir -p /data/zot
sudo chown -R zot:zot /data/zot

sudo mkdir -p /var/log/zot
sudo chown -R zot:zot /var/log/zot

sudo chown -R root:root /etc/zot/
```

With the `adduser` options shown, the 'zot' user ID has no local directory. There is no ability to log into the zot user account, and the account has no finger information.

Step 6: Start zot

Reload systemd config:

```
sudo systemctl daemon-reload
```

Enable and start the zot service with these commands:

```
sudo systemctl enable zot
sudo systemctl start zot
```

Check if zot config is valid:

```
sudo -u zot zot verify /etc/zot/config.json
```

When the zot service has started, you can check its status with this command:

```
sudo systemctl status zot
```

4.1.3 After the installation

If your zot registry server is public facing, we recommend that you test your TLS configuration using a service such as the Qualys SSL Server Test.

Refer to *Configuring zot* for further information about maintaining your zot registry server.

4.1.4 Configuration file options and recommendations

The following zot configuration file (`config.json`) can be used as a template for your own installation. You can modify this file to suit your own environment.

ck here to view the sample configuration file.

```
{
  "distSpecVersion":"1.0.1",
  "storage":{
    "dedupe": true,
    "gc": true,
    "gcDelay": "1h",
    "gcInterval": "6h",
    "rootDirectory":"/data/zot/"
  },
  "http": {
    "address":"0.0.0.0",
    "port":"443",
    "realm":"zot",
    "tls": {
      "cert": "/etc/letsencrypt/live/zothub.io/fullchain.pem",
      "key": "/etc/letsencrypt/live/zothub.io/privkey.pem"
    },
    "auth": {
      "htpasswd": {
        "path": "/etc/zot/htpasswd"
      },
      "failDelay": 5
    }
  },
  "log":{
    "level":"debug",
    "output":"/var/log/zot/zot.log",
    "audit":"/var/log/zot/zot-audit.log"
  },
  "extensions": {
    "search": {
      "enable": true,
      "cve": {
        "updateInterval": "24h"
      }
    },
    "sync": {
      "enable": false,
      "registries": [
        {
          "urls": ["https://mirror.gcr.io/library"],
          "onDemand": true,
          "maxRetries": 3,
          "retryDelay": "5m",
          "pollInterval": "6h"
        },
        {
          "urls": ["https://docker.io/library"],
          "onDemand": true
        }
      ]
    },
    "scrub": {
      "interval": "24h"
    }
  }
}
```

Refer to *Configuring zot* for more details about configuration file options.

TLS encryption

We recommend using a certificate authority such as Let's Encrypt that offers TLS encryption, as shown in this configuration example:

```
"tls": {
  "cert": "/etc/letsencrypt/live/zothub.io/fullchain.pem",
  "key": "/etc/letsencrypt/live/zothub.io/privkey.pem"
}
```

Registry synchronization

The example file enables registry synchronization with two other container registries. In the example, the zot server synchronizes with the Google and Docker container registries, as shown here:

```
"sync": {
  "enable": false,
  "registries": [
    {
      "urls": ["https://mirror.gcr.io/library"],
      "onDemand": true,
      "maxRetries": 3,
```

```
      "retryDelay": "5m",
      "pollInterval": "6h"
    },
    {
      "urls": ["https://docker.io/library"],
      "onDemand": true
    }
  ]
}
```

May 22, 2025

4.2 Installing zot with Kubernetes and Helm

👉 Using Kubernetes with Helm charts for zot, you can easily deploy zot as an application in a Kubernetes cluster.

4.2.1 Before you begin

Prerequisites

- kubectl must be installed and a Kubernetes cluster must be ready.
- Helm must be installed.

Supported platforms

You can install zot on standard Linux platforms with Intel or ARM processors and with systemd installed.

OS	ARCH	Platform
linux	amd64	Intel-based Linux servers
linux	arm64	ARM-based servers and Raspberry Pi4

Supported platforms and architectures

About binary images

Refer to *Released Images for zot* for information about available zot images along with information about image variations, image locations, and image naming formats.

4.2.2 Installing zot

Step 1: Locate the Helm charts in a remote repository

1. Specify a remote repository that contains the Helm charts for zot. Give the repo a local name, such as **project-zot**, as in this example:

```
helm repo add project-zot http://zotregistry.dev/helm-charts
```

```
"project-zot" has been added to your repositories
```

✏️

The Helm charts for zot are currently hosted in these publicly-accessible repositories:
- zotregistry.dev
- artifacthub.io

2. Search the repository to see the Helm charts for zot installation. Search using the keyword 'project-zot' or 'zot', as in this example:

```
helm search repo project-zot
```

```
NAME            CHART VERSION  APP VERSION  DESCRIPTION
project-zot/zot  <chart-version>          v2.1.7       A Helm chart for Kubernetes
```

✏️

The APP VERSION is the version/tag of the zot image used for the deployment.

3. Update to the latest information of available charts from the chart repository, as shown in this example:

```
helm repo update project-zot
```

```
Hang tight while we grab the latest from your chart repositories...
...Successfully got an update from the "project-zot" chart repository
Update Complete. *Happy Helming!*
```

4. Display the default information of the Helm chart, as shown in this example:

`helm show all project-zot/zot`

```
    apiVersion: v2
    appVersion: v2.1.7
    description: A Helm chart for Kubernetes
    name: zot
    type: application
    version: <chart-version>

    # Default values for zot.
    # This is a YAML-formatted file.
    # Declare variables to be passed into your templates.
    replicaCount: 1
    image:
      repository: ghcr.io/project-zot/zot-linux-amd64
      pullPolicy: IfNotPresent
      tag: "v2.1.7"
    serviceAccount:
      create: true
      annotations: {}
      name: ""
    service:
      type: NodePort
      port: 5000
```

Step 2: Determine any needed changes from the Helm chart's defaults

Inspect the default information of the Helm chart, as shown in the previous step. In many cases, the default chart values may be acceptable. If your installation requires any non-default settings, you may be able to specify them during the installation. Not all chart values are configurable, but you can display those that are configurable using the command in the following example:

`helm show values project-zot/zot`

```
  # Default values for zot.
  # This is a YAML-formatted file.
  # Declare variables to be passed into your templates.
  replicaCount: 1
  image:
    repository: ghcr.io/project-zot/zot-linux-amd64
    pullPolicy: IfNotPresent
    tag: "v2.1.7"
  serviceAccount:
    create: true
    annotations: {}
    name: ""
  service:
    type: NodePort
    port: 5000
```

The configurable settings in the chart are listed in the following table:

parameter	description
replicaCount	Desired number of replicas of the application
image.repository	Repository and image name for the application
image.pullPolicy	Whether to pull the image from the repository. If not specified, the policy depends on `image.tag`: • If tag is `:latest` or no tag: `Always` • If tag is other than `:latest`: `IfNotPresent`
image.tag	Identifies different versions the image. default is the chart `appVersion`. Examples: `:latest` (the default) or `:v2.1.7`
serviceAccount.create	Specifies whether a service account should be created
serviceAccount.annotations	Annotations to add to the service account
serviceAccount.name	Name of the service account to use. If `name` is not set and `create` is true, a name is generated using the fullname template.
service.type	ClusterIP (default), NodePort, LoadBalancer, ExternalName, or Headless
service.port	Port number for calling the service
strategy.type	Kubernetes deployment strategy type. [More Info](https://kubernetes.io/docs/concepts/workloads/controllers/deployment/#strategy)

CUSTOMIZING THE HELM CHART USING 'SET'

To override the default values in the chart, you can pass your custom values by adding the `--set` flag in the `helm install` command.

For example, if your servers use an ARM processor instead of Intel, you must change the `image.repository` name from **zot-linux-amd64** to **zot-linux-arm64**:

```
--set image.repository=ghcr.io/project-zot/zot-linux-arm64
```

You can change multiple settings with one `--set` statement. For example, you might want your installation to have more replicas or a different port number:

```
--set replicaCount=2,service.port=5050
```

CUSTOMIZING THE HELM CHART USING A FILE

You can also create a YAML file with your overrides and then add the new file by adding the `-f` flag to the `helm install` command. For example, to override the replica count and port number, the contents of your YAML file (for example, "myfile.yaml") would be:

```yaml
replicaCount: 2
service:
  port: 5050
```

and the following flag would be added to the `helm install` command:

```
-f myfile.yaml
```

ADDITIONAL INFORMATION

See the Helm documentation for further information about modifying the Helm chart.

Step 3: Install zot

Install zot using the `helm install` command. The first example shows how to perform a default installation. The additional examples show different ways to modify the `helm install` command to override default settings in the Helm chart:

Example 1: use default chart parameters

```
helm install zot project-zot/zot
```

```
NAME: zot
LAST DEPLOYED: Thu Aug 11 19:13:02 2022
NAMESPACE: default
STATUS: deployed
REVISION: 1
NOTES:
 Get the application URL by running these commands:
 export NODE_PORT=$(kubectl get --namespace default -o jsonpath="{.spec.ports[0].nodePort}" services zot)
 export NODE_IP=$(kubectl get nodes --namespace default -o jsonpath="{.items[0].status.addresses[0].address}")
 echo http://$NODE_IP:$NODE_PORT
```

Example 2: modify specific chart parameters with 'set'

```
helm install --set replicaCount=2,service.port=5050 zot project-zot/zot
```

Example 3: modify specific chart parameters with a file

```
helm install -f myfile.yaml zot project-zot/zot
```

Example 4: use a specific version of the Helm chart

```
helm install zot project-zot/zot --version 0.1.0
```

Example 5: link to a kubeconfig file

```
helm install zot project-zot/zot --kubeconfig $HOME/.kube/config
```

4.2.3 After the installation

Verify the installation

1. List all releases that are either deployed or failed.

   ```
   helm list
   ```

   ```
   NAME  NAMESPACE  REVISION  UPDATED    STATUS    CHART      APP VERSION
   zot   default    1         <datetime> deployed  <chart-version>  v2.1.7
   ```

   This response indicates that zot is deployed.

2. After making sure that your pods are up and running, execute the following commands:

   ```
   $ export NODE_PORT=$(kubectl get --namespace default -o jsonpath="{.spec.ports[0].nodePort}" services zot)
   $ export NODE_IP=$(kubectl get nodes --namespace default -o jsonpath="{.items[0].status.addresses[0].address}")
   $ echo http://$NODE_IP:$NODE_PORT
   $ curl http://$NODE_IP:$NODE_PORT/v2/_catalog
   ```

   The response should display the current contents of your zot repository, which should be empty immediately after installation:

   ```
   {"repositories":[]}
   ```

Edit the zot configuration file

The zot configuration file is a JSON or YAML file that contains all configuration settings for zot functions such as:

- network
- storage
- authentication
- authorization
- logging
- metrics
- synchronization with other registries
- clustering

The Helm chart installs a minimal JSON configuration file as shown below:

```
{
    "storage":
    {
        "rootDirectory": "/var/lib/registry"
    },
    "http":
    {
        "address": "0.0.0.0",
        "port": "5000"
    },
    "log":
    {
        "level": "debug"
    }
}
```

The zot configuration file is located at `/etc/zot/config.json`.

Refer to *Configuring zot* for complete information on configuring the zot server with the zot configuration file.

Uninstalling zot

Should you need to uninstall zot, use the `helm uninstall` command, as in this example:

```
helm uninstall zot
```

February 22, 2024

5. Administrator Guides

5.1 Getting Started with zot Administration

👉 This document helps you to deploy an appropriate zot image or to build zot if desired.

After deploying zot, proceed to Configuring zot to choose and configure the features you need.

5.1.1 Installing zot

How to get zot

The zot project is hosted on GitHub at project-zot. From GitHub, you can download zot executable binary images or full source code.

SUPPORTED PLATFORMS

zot is officially supported on Linux and Apple MacOS platforms, using Intel or ARM processors. However, development should be possible on any platform that supports the `golang` toolchain.

OS	ARCH	Platform
linux	amd64	Intel-based Linux servers
linux	arm64	ARM-based servers and Raspberry Pi4
darwin	amd64	Intel-based MacOS
darwin	arm64	ARM-based MacOS
freebsd	amd64	Intel-based FreeBSD*
freebsd	arm64	ARM-based FreeBSD*

* NOTE: While binary images are available for FreeBSD, building container images is not supported at this time.

ABOUT BINARY IMAGES

Executable binary zot images are available for multiple platforms and architectures and with full or minimal implementations.

Refer to *Released Images for zot* for information about available zot images along with information about image variations, image locations, and image naming formats.

DEPLOYMENT METHODS

Several options exist for deploying zot:

- You can launch a zot binary as a container service using a container management tool such as Podman, Docker, or Helm.
- You can launch zot as a host-level service by downloading a binary image and running it as a systemd service.
- You can copy or clone the full zot source code and build an image with custom build flags.

Deploying a zot binary image

Executable binary images for supported server platforms and architectures are available from the zot package repository in GitHub.

You can download the appropriate binary image and run it directly on your server, or you can use a container management tool such as Podman, runc, Helm, or Docker to fetch and deploy the image in a container on your server.

💡 For convenience, you can rename the binary image file to simply `zot`.

EXAMPLE: DEPLOYING WITH A CONTAINER MANAGER

Using a container manager such as Podman, runc, Helm, or Docker, you can install a zot binary image, as in the following examples.

Using podman

```
podman run -p 5000:5000 ghcr.io/project-zot/zot-linux-amd64:latest

podman run -p 5000:5000 ghcr.io/project-zot/zot-minimal-linux-amd64:latest
```

ck here to view an example of deploying using podman.

Using docker

```
docker run -p 5000:5000 ghcr.io/project-zot/zot-linux-amd64:latest
```

Each of these example commands pulls a zot binary image from the GitHub Container Registry (ghcr.io) and launches a zot image registry at `http://localhost:5000`.

ck here to view an example of deploying using docker.

Building zot from source

PREREQUISITES

Install golang

Follow the golang instructions to install the `golang` toolchain. After installation, make sure that the `path` environment variable or your IDE can find the toolchain.

✏️ You must use a golang version of at least the minimum specified in go.mod or the build will fail.

BUILDING AN EXECUTABLE BINARY FROM SOURCE

Download or clone the full zot project from GitHub at project-zot. To clone the zot project from GitHub, use this command:

```
git clone https://github.com/project-zot/zot.git
```

To build zot, execute the `make` command in the `zot` directory using the following general syntax:

```
make OS=os ARCH=architecture {binary | binary-minimal}
```

- The operating system and architecture options are listed in the Supported platforms and architectures table. If an option is not specified, the defaults are `linux` and `amd64`.
- The `binary` option builds the full zot binary image with all extensions.
- The `binary-minimal` option builds the minimal distribution-spec conformant zot binary image without extensions, reducing the attack surface.

For example, to build a zot image with extensions for an Intel-based linux server, use the following command:

```
make OS=linux ARCH=amd64 binary
```

The `make` command builds an executable image in the `zot/bin` directory. The original filename of the zot executable image will indicate the build options. For example, the filename of an Intel-based linux minimal image is `zot-linux-amd64-minimal`.

💡 For convenience, you can rename the binary image file to simply `zot`.

BUILDING A ZOT CONTAINER IMAGE FROM SOURCE

with Stacker

Using the settings in stacker.yaml, you can build a container image that runs the latest zot by running the following command:

```
make binary-stacker
```

with Docker

A sample Dockerfile is provided on the zot project page in GitHub. You can edit the sample file with your specific values, such as the desired operating system, hardware architecture, and full or minimal build, as in this example:

```
ARG OS=linux
ARG ARCH=amd64

RUN make COMMIT=$COMMIT OS=$OS ARCH=$ARCH clean binary-minimal
```

Using your edited Dockerfile, you can build a container image that runs the latest zot by running the following command:

```
make image
```

DEPLOYING THE CONTAINER IMAGE

Deploy the image using your container manager, such as Podman, runc, Helm, or Docker, as in these examples:

with Podman

```
podman run --rm -it -p 5000:5000 -v $(pwd)/registry:/var/lib/registry zot:latest
```

with Docker

```
docker run --rm -it -p 5000:5000 -v $(pwd)/registry:/var/lib/registry zot:latest
```

A container image built with the sample Dockerfile and deployed with the example command results in a running registry at `http://localhost:5000`. Registry content is stored at `.registry`, which is bind mounted to `/var/lib/registry` in the container. By default, auth is disabled. As part of the build, a YAML configuration file is created at `/etc/zot/config.yml` in the container.

You can override the configuration file with custom configuration settings in the deployment command and in a local configuration file as shown in this example:

```
podman run --rm -p 8080:8080 \
  -v $(pwd)/custom-config.yml:/etc/zot/config.yml \
  -v $(pwd)/registry:/tmp/zot \
  zot:latest
```

This command causes the registry to listen on port 8080 and to use `/tmp/zot` for content storage.

Additional recommended steps

We recommend that, when deploying zot, you also install the command line (zli) and benchmarking (zb) packages.

Launching zot

The zot service is initiated with the `zot serve` command followed by the name of a configuration file, as in this example:

```
zot serve config.yml
```

💡 For convenience, you can rename the binary image file to simply `zot` .The instructions and examples in this guide use `zot` as the name of the zot executable file and do not include the path to the executable file.

5.1.2 Next Steps

Configuring zot

You configure zot primarily through adding and modifying settings in the zot configuration file. The configuration file is a JSON or YAML file that contains all configuration settings for zot functions.

When you first build zot or deploy an image or container from the distribution, a basic configuration file `config.json` is created. You can modify the initial file or you can create a new file.

Follow the instructions in Configuring zot, to compose a configuration file with the settings and features you require for your zot registry server.

September 13, 2023

5.2 Configuring zot

👉 The registry administrator configures zot primarily through settings in the configuration file.

Using the information in this guide, you can compose a configuration file with the settings and features you require for your zot registry server.

💡 Before launching zot with a new configuration, we recommend that you verify the syntax of your configuration as described in Verifying the configuration file.

5.2.1 Configuration file

The configuration file is a JSON or YAML file that contains all configuration settings for zot functions such as:

- network
- storage
- authentication
- authorization
- logging
- metrics
- synchronization with other registries
-
- clustering

The zot service is initiated with the `zot serve` command followed by the name of a configuration file, as in this example:

```
zot serve config.json
```

✏️ The instructions and examples in this guide use `zot` as the name of the zot executable file. The examples do not include the path to the executable file.

When you first build zot or deploy an image or container from the distribution, a basic configuration file `config.json` is created. This initial file is similar to the following example:

```
{
    "distSpecVersion": "1.0.1",
    "storage": {
        "rootDirectory": "/tmp/zot"
    },
    "http": {
        "address": "127.0.0.1",
        "port": "8080"
    },
    "log": {
        "level": "debug"
    }
}
```

The configuration file contains the Distribution Specification version (`distSpecVersion`). The structure and content of other attributes are described in the later sections of this guide.

Extensions

Additional registry features that are not a part of the Distribution Specification are allowed to be added as Extensions.

With a full (not minimal) zot image, the following extension features can be enabled and configured under an `extensions` attribute in the configuration file as shown in the following example.

```
{
  ...
  "extensions": {
    "metrics": {},
```

```
    "sync": {},
    "search": {},
    "scrub": {},
    "lint": {},
    "trust": {},
    "ui": {}
  }
}
```

⚠️ The extension features are available only with a full zot image. With a minimal zot image, the `extensions` section is ignored if present.

The following features are configured under the `extensions` attribute.

- Metrics
- Sync
- Search
- Scrub
- Lint
- ImageTrust
- UI

An extension feature is usually enabled by the presence of the feature's attribute under `extensions`. An extension feature can then be disabled by either omitting the feature attribute or by including an `enable` attribute with a value of `false`.

✏️ Two API-only extensions, User Preferences and Mgmt, are not enabled or configured under the `extensions` section of the configuration file. These API-only extensions are enabled as follows:

- Mgmt is enabled when the `Search` extension is enabled.
- User Preferences is enabled when both the `Search` and `UI` extensions are enabled.

ENABLING AND DISABLING EXTENSIONS

Following is an example of enabling or disabling a feature in the `extensions` section. The scrub feature is enabled in these two configurations:

```
"extensions": {
  "scrub": {}
}
```

```
"extensions": {
  "scrub": {
    "enable": true
  }
}
```

The scrub feature is disabled in these two configurations:

```
"extensions": {
}
```

```
"extensions": {
  "scrub": {
    "enable": false
  }
}
```

DEVELOPING CUSTOM EXTENSIONS

New functionality can be added to the zot registry by developing custom extensions for integration into zot. For information about developing extensions, see *Developing New Extensions*.

5.2.2 Network configuration

Use the `http` attribute in the configuration file to configure the zot network settings, as shown in the following example.

```
"http": {
  "address":"127.0.0.1",
  "port":"8080",
  "realm":"zot",
  "tls": {
    "cert":"test/data/server.cert",
    "key":"test/data/server.key"
  }
}
```

The following table lists the configurable attributes.

Attribute	Description
`address`	The IP address of the zot server.
`port`	The port number of the zot server.
`realm`	The security policy domain defined for the server.
`tls`	The included attributes in this section specify the Transport Layer Security (TLS) settings for the server.
`cert`	The path and filename of the server's SSL/TLS certificate.
`key`	The path and filename of the server's registry key.

5.2.3 Storage

Exposing flexibility in storage capabilities is a key tenet for catering to the requirements of varied environments ranging from cloud to on-premises to IoT.

Filesystem storage is configured with the `storage` attribute in the zot configuration file, as shown in the following simple example.

```
"storage":{
  "rootDirectory":"/tmp/zot",
  "commit": true,
  "dedupe": true,
  "gc": true,
  "gcDelay": "1h",
  "gcInterval": "24h"
}
```

With zot, you have the option to store your registry image files either in local filesystem storage or in cloud storage, such as an Amazon Simple Storage Service (S3) bucket.

Local storage

zot can store and serve files from one or more local directories (folders). A minimum of one root directory is required for local hosting, but additional hosted directories can be added. When accessed by HTTP APIs, all directories can appear as a single data store.

✏️ Remote filesystems that are mounted and accessible locally such as `NFS` or `fuse` are treated as local filesystems.

Remote storage

zot can also store data remotely in the cloud, using the storage APIs of the cloud service. Currently, zot supports only the AWS S3 storage service.

For detailed information about configuring S3 storage, see the AWS S3 documentation and Storage Planning with zot.

Storage features

COMMIT

Most modern filesystems buffer and flush RAM data to disk after a delay. The purpose of this function is to improve performance at the cost of higher disk memory usage. In embedded devices such as Raspberry Pi, for example, where RAM may be very limited and at a premium, it is desirable to flush data to disk more frequently. The zot storage configuration exposes an option called `commit` which, when enabled, causes data writes to be committed to disk immediately. This option is disabled by default.

DEDUPLICATION

Deduplication is a storage space saving feature wherein only a single copy of specific content is maintained on disk while many different image manifests may hold references to that same content. The deduplication option (`dedupe`) is also available for supported cloud storage backends.

Upon startup, zot enforces the `dedupe` status on the existing storage. If the `dedupe` status upon startup is `true` , zot deduplicates all blobs found in storage, both local and remote. If the status upon startup is `false` , zot restores cloud storage blobs to their original state. There is no need for zot to restore local filesystem storage if hard links are used.

GARBAGE COLLECTION

After an image is deleted by deleting an image manifest, the corresponding blobs can be purged to free up space. However, since Distribution Specification APIs are not transactional between blob and manifest lifecycle, care must be taken so as not to put the storage in an inconsistent state. Garbage collection in zot is an inline feature meaning that it is **not** necessary to take the registry offline. The zot configuration model allows for enabling and disabling garbage collection (`gc`). The model also allows the configuration of a tunable delay (`gcDelay`), which can be set depending on client network speeds and the size of blobs.

SCRUB

The `scrub` function, available as an *extension*, makes it possible to ascertain data validity by computing hashes on blobs periodically and continuously so that any bit rot is caught and reported early.

Configuring storage

For detailed information about configuring local or remote storage and storage features for your zot registry, see Storage Planning with zot.

5.2.4 Security and hardening

Authentication

zot supports authentication by the following methods:

- TLS mutual authentication
- Basic local authentication using an htpasswd file
- LDAP authentication
- Bearer (OAuth2) authentication using an HTTP Bearer token

For detailed information about configuring authentication for your zot registry, see User Authentication and Authorization with zot.

Identity-based authorization

User identity can be used as an authorization criterion for allowing actions on one or more repository paths. For specific users, you can choose to allow any combination of read, create, update, or delete actions on specific repository paths.

For detailed information about configuring access control policies for your zot registry, see User Authentication and Authorization with zot.

Preventing automated attacks with failure delay

Use the `auth` and `failDelay` attributes under `http` in the configuration file to delay the response to an authentication failure. A delayed response helps to prevent automated attacks. The configuration is shown in the following example.

```
"http": {
  "auth": {
    "failDelay": 5
  }
}
```

The `failDelay` attribute specifies a waiting time, in seconds, before zot sends a failure notification to an authenticating user who has been denied access.

Rate limiting

You can limit the rate of API calls from users by configuring the `ratelimit` attribute in the configuration file, as shown in the following example:

```
"http": {
    "address": "127.0.0.1",
    "port": "8080",
    "ratelimit": {
        "rate": 10,
        "methods": [
            {
                "method": "GET",
                "rate": 5
            }
        ]
    }
}
```

In this example, the `rate` attribute directly under `ratelimit` sets a global rate limit of ten API calls per second. You can optionally override the global limit for specific API `Methods`. In this example, API `GET` calls are limited to five per second.

Additional security features

For detailed information about configuring additional security features for your zot registry, see Security Posture.

5.2.5 Monitoring

zot supports a range of monitoring tools including the following:

- Logging

  Logging for zot operations is configured with the `log` attribute in the configuration file.

- Metrics

  Metrics data is available in a Prometheus format. A full zot image with extensions includes a node exporter. A minimal zot image can use an external node exporter such as `zxp`.

- Benchmarking

  The zot project includes the `zb` tool, which allows you to benchmark a zot registry or any other container image registry that conforms to the OCI Distribution Specification.

- Performance profiling

  Performance profiling capabilities within zot allow a zot administrator to collect and export a range of diagnostic performance data such as CPU intensive function calls, memory allocations, and execution traces. The collected data can then be analyzed using Go tools and a variety of available visualization tools.

When zot is deployed in a Kubernetes setup, a site reliability engineering (SRE) operator can monitor service level indicators (SLI) such as metrics and logs. Metrics will appear in Prometheus using the zot `metrics` extension, while logs will appear in the Elasticsearch stack (ELK stack) using Filebeat.

For detailed information about the monitoring tools, see Monitoring the registry.

For detailed information about benchmarking, see Benchmarking zot with zb.

For detailed information about performance profiling, see Performance Profiling in zot.

5.2.6 Clustering zot

To ensure high-availability of the registry, zot supports a clustering scheme with stateless zot instances/replicas fronted by a loadbalancer and a shared remote backend storage. This scheme allows the registry service to remain available even if a few replicas fail or become unavailable. Loadbalancing across many zot replicas can also increase aggregate network throughput.

For detailed information about clustering with zot, see zot Clustering.

5.2.7 Syncing and mirroring registries

A zot registry can mirror one or more upstream OCI registries, including popular cloud registries such as Docker Hub and Google Container Registry. If an upstream registry is OCI distribution-spec conformant for pulling images, you can use zot's `sync` extension feature to implement a downstream mirror, synchronizing OCI images and corresponding artifacts. Synchronization between registries can be implemented by periodic polling of the upstream registry or synchronization can occur on demand, when a user pulls an image from the downstream registry.

As with git, wherein every clone is a full repository, you can configure a local zot instance to be a full OCI mirror registry. This allows for a fully distributed disconnected container image build pipeline.

For detailed information about syncing and mirroring, see OCI Registry Mirroring With zot.

5.2.8 Linting uploaded images

The lint extension can check an uploaded image to enforce the presence of required annotations such as the author or the license.

To configure linting, add the `lint` attribute under `extensions` in the configuration file, as shown in the following example:

```
"extensions": {
    "lint": {
      "enable": true,
      "mandatoryAnnotations": ["annot1", "annot2", "annot3"]
      }
  }
```

The following table lists the configurable attributes of the `lint` extension.

Attribute	Description
`enable`	If this attribute is missing, incoming image linting is disabled by default. Linting can be enabled by including this attribute and setting it to `true`.
`mandatoryAnnotations`	A list of annotations that are required to be present in the image being pushed to the repository.

If the mandatory annotations option is configured when you push an image, linter will verify that the mandatory annotations list present in the configuration is also found in the manifest's annotations list. If any annotations are missing, the push is denied.

5.2.9 Compatibility with other image schema types

As an option, zot can be configured to store images using the schema Docker Manifest v2 Schema v2. In this case, a Docker image can be copied to zot without modifications to the image's manifest or digest. Such modifications would otherwise break the image's signature and attestations.

To enable this compatibility, configure the `compat` attribute under `http` in the zot configuration file. Set the `compat` value to `docker2s2` as shown in the following example:

```
"http": {
    "address": "127.0.0.1",
```

```
    "port": "8080",
    "compat": ["docker2s2"]
}
```

5.2.10 Verifying the signatures of uploaded images

The `trust` extension provides a mechanism to verify image signatures using certificates and public keys.

To enable image signature verification, you must add the `trust` attribute under `extensions` in the zot configuration file and enable one or more verification tools, as shown in the following example:

```
"extensions": {
  "trust": {
    "enable": true,
    "cosign": true,
    "notation": true
  }
}
```

You must also upload public keys (for `cosign`) or certificates (for `notation`) that can be used to verify the image signatures.

For detailed information about configuring image signature verification, see Verifying image signatures.

5.2.11 Enabling the registry's graphical user interface

Using the zot graphical user interface (GUI), a user can browse a zot registry for container images and artifacts.

To configure zot's GUI, add the `ui` attribute under `extensions` in the configuration file, as shown in the following example:

```
"extensions": {
  "ui": {
    "enable": true
  }
}
```

The following table lists the configurable attributes of the `ui` extension.

Attribute	Description
`enable`	If this attribute is missing, the zot GUI is disabled by default. The GUI can be enabled by including this attribute and setting it to `true`.

5.2.12 Scrubbing the image registry

To check the integrity of the filesystem and the data in the registry, you can schedule a periodic scrub operation. The scrub process traverses the filesystem, verifying that all data blocks are readable. While running, the process may slightly reduce the registry performance.

To enable scrubbing, add the `scrub` attribute under `extensions` in the configuration file, as shown in the following example:

```
"extensions": {
  "scrub": {
    "enable": true,
    "interval": "24h"
  }
}
```

The following table lists the configurable attributes for scrubbing the registry.

Attribute	Description
`enable`	If this attribute is missing, registry scrubbing is enabled by default. Scrubbing can be disabled by setting this attribute to `false`.
`interval`	The time interval between periodic scrub operations. This value must be at least two hours (`2h`).

5.2.13 Scheduling background tasks

Some zot functions, such as garbage collection and registry synchronization, run as background tasks. These tasks are queued and executed by the scheduler.

The scheduler is by default allowed to simultaneously run a maximum number of tasks equal to four times the number of CPUs available to the zot process. For most users, there should be no need to modify this maximum number. If such a need arises, you can configure a new maximum number of simultaneous tasks in the `numWorkers` property of the `scheduler` attribute in the configuration file, as shown in the following example.

```
{
  "distSpecVersion": "1.1.0-dev",
  "scheduler": {
    "numWorkers": 3
  },
...
}
```

5.2.14 Enhanced searching and querying images

While basic searching is always enabled for images in the zot registry, you can enable enhanced registry searching and filtering using graphQL.

Add the `search` attribute under `extensions` in the configuration file to enable and configure the enhanced search extension, as shown in the following example.

```
"extensions": {
  "search": {
    "enable": true,
    "cve": {
      "updateInterval": "2h"
    }
  }
}
```

The following table lists the configurable attributes for enhanced search.

Attribute	Description
`enable`	If this attribute is missing, enhanced search is enabled by default. Enhanced search can be disabled by setting this attribute to `false`.
`cve`	Extends enhanced search to allow searching of Common Vulnerabilities and Exposures (CVE).
`updateInterval`	Sets the interval at which the searchable database of CVE items is refreshed.

5.2.15 Setting user preferences

The user preferences extension provides an API endpoint for adding configurable user preferences for a repository. This custom extension, not a part of the OCI distribution, is accessible only by authenticated users of the registry. Unauthenticated users are denied access.

The user preferences extension is enabled by default when the `search` and `ui` extensions are enabled. There are no other configuration file fields for this extension.

A `userprefs` API endpoint accepts as a query parameter an `action` to perform along with any other required parameters for the specified action. The actions currently implemented do not require an HTTP payload, nor do they return any related data other than an HTTP response code.

Current functionality

The current functions implemented by this extension include:

- Toggling the star (favorites) icon for a repository.
- Toggling the bookmark icon for a repository.

TOGGLE REPOSITORY STAR

This action sets the repository star property to `true` if it is `false`, and to `false` if it is `true`.

Action	Parameter	Parameter Description
toggleStar	repo	The name of the repository whose star is to be changed

This example toggles a star on a repository named repoName:

```
PUT
http://localhost:5000/v2/_zot/ext/userprefs?
action=toggleStar&repo=repoName
```

TOGGLE REPOSITORY BOOKMARK

This action sets the repository bookmark property to `true` if it is `false`, and to `false` if it is `true`.

Action	Parameter	Parameter Description
toggleBookmark	repo	The name of the repository whose bookmark is to be changed

This example toggles a bookmark on a repository named repoName:

```
PUT
http://localhost:5000/v2/_zot/ext/userprefs?
action=toggleBookmark&repo=repoName
```

5.2.16 Verifying the configuration file

Before launching zot, verify the syntax of your configuration file using the following command:

```
zot verify <configfile>
```

✏️ Verifying the configuration file protects against operator errors and any conflicts arising from zot release version changes.

After verifying your configuration file, you can launch zot with the following command:

```
zot serve <configfile>
```

January 22, 2025

6. Developer Guide

6.1 Onboarding zot for Development

👉 zot is a production-ready, open-source, extensible OCI-native image registry, built for developers by developers.

6.1.1 Getting Started

Supported Developer Platforms

Development is officially supported on `Linux` and `Apple MacOS` platforms. However, development should be possible on any platform that supports the `golang` toolchain.

OS	ARCH	Platform
linux	amd64	Intel-based Linux servers
linux	arm64	ARM-based servers and Raspberry Pi4
darwin	amd64	Intel-based MacOS
darwin	arm64	ARM-based MacOS (Apple M1)

Supported platforms and architectures

Prerequisites

INSTALL GOLANG

Follow the golang instructions to install the `golang` toolchain. After installation, make sure that the `path` environment variable or your IDE can find the toolchain.

✏️ You must use a golang version of at least the minimum specified in go.mod or the build will fail.

Cloning zot

The zot registry code base is hosted on GitHub at https://github.com/project-zot/zot.

To clone the zot project, use this command:

```
$ git clone https://github.com/project-zot/zot.git
```

Building zot

To build zot, execute the `make` command in the zot directory using the following general syntax:

```
$ make OS=os ARCH=architecture {binary | binary-minimal}
```

- The operating system and architecture options are listed in the Supported platforms and architectures table. If an option is not specified, the defaults are `linux` and `amd64`.
- The `binary` option builds the full zot binary image with all extensions.
- The `binary-minimal` option builds the minimal distribution-spec conformant zot binary image without extensions, reducing the attack surface.

For example, to build a zot image with extensions for an Intel-based linux server, use the following command:

```
make OS=linux ARCH=amd64 binary
```

The `make` command builds an executable image in the `zot/bin` directory. The original filename of the zot executable image will indicate the build options. For example, the filename of an Intel-based linux minimal image is `zot-minimal-linux-amd64`.

ck here to view an example of the getting started process.

Running zot

The behavior of zot is controlled via configuration only. To launch the zot server, execute the following command:

```
$ bin/zot-linux-amd64 serve examples/config-minimal.json
```

6.1.2 Debugging zot

To produce a zot binary that includes extensive debugging information, build zot with the `binary-debug` option, as shown in this example:

```
make OS=linux ARCH=amd64 binary-debug
```

You can then attach and run a debugging tool such as Delve to the running zot process.

Delve is a powerful open-source debugger for the Go programming language. Downloads and documentation for Delve are available on GitHub at https://github.com/go-delve/delve.

6.1.3 Performance profiling

Performance profiling capabilities within zot allow a zot administrator to collect and export a range of diagnostic performance data such as CPU intensive function calls, memory allocations, and execution traces. The collected data can then be analyzed using Go tools and a variety of available visualization tools.

For detailed information about performance profiling, see Performance Profiling in zot.

6.1.4 Code Organization

The zot project codebase is organized as follows:

```
/
- pkg/              # Source code for all libraries
  - api/            # Source code for HTTP APIs
    - config/       # Global configuration model
  - storage/        # Source code for storage backends
  - cli/            # Source code for command line interface (cli)
  - common/         # Source code for common utility routines
  - compliance/     # Source code for dist-spec conformance tests
  - log/            # Source code for logging framework
  - test/           # Internal test scripts/data
  - extensions/     # Source code for all extensions
    - config/
    - sync/
    - monitoring/
    - sync/
  - exporter/       # Source code for metrics exporter
- cmd/              # Source code for binary main()s
  - zot/            # Source code for zot binary
  - zli/            # Source code for zot cli
  - zb/             # Source code for zb, the dist-spec benchmarking tool
- errors/           # Source code for error codes
- examples/         # Configuration examples
- swagger/          # Swagger integration
- docs/             # Documentation
```

6.1.5 Additional state in zot

In addition to the storage of repository images, zot stores data for various processes in local and remote storage and databases. The following table shows the storage locations for different processes and types of data.

Data or Process	Storage Location	Description
images	local and AWS S3	Image blobs
repository synchronization	local / <repo_name>/.sync	The sync operation temporarily copies the upstream blobs to: /<repo_name>/.sync/${UUID}/<repo_name>, then copies to /<repo_name> and deletes the temporary directory ${UUID}
deduplication	local /cache.db	Cache for deduplication of files stored locally
	AWS DynamoDB	Cache for deduplication of files stored in AWS
CVE	local /_trivy	Database of Common Vulnerabilities and Exposures (CVE) information and scan results
user sessions	local /_sessions	zot user session authentication (for zui)
PKI authentication documents	local /_cosign	Private keys for signature verification using cosign
	local /_notation	Certificates for signature verification using notation
metadata	local /repo.db	Local storage of manifests, configurations, download counters, signature verification results
	AWS DynamoDB	Cloud storage of manifests, configurations, download counters, signature verification results

October 12, 2023

6.2 Developing New Extensions

👉 You can add new functionality to the zot registry by developing *extensions* for integration into zot.

The OCI Distribution Specification supports extending the functionality of an OCI-compliant registry implementation by adding *extensions*. Extensions are new APIs developed outside of the core OCI specs. Developers may propose their extensions to the OCI for possible future addition to the Distribution Specification.

✏️ When planning the development of a new extension, be sure to familiarize yourself with the OCI documentation and guidelines for extensions.

6.2.1 Current extensions

The following extensions are currently available in the zot project:

- metrics
- sync
- search
- scrub
- lint

You can examine the implementation of these extensions in the zot project extensions section. The operation and configuration of the current extensions is described in Configuring zot.

6.2.2 Guidelines for developing new extensions

- Each file to be included in the binary for only a specific extension must contain the following syntax at the beginning of the file. For example, a file to be included in the build for extension *foo* must begin with the following lines:

```
//go:build foo
// +build foo

package foo

...
```

```
- The first line (`//go:build foo`) is added automatically by the linter if not already present.

- The second line and the third (blank) line are mandatory.
```

- For each file that contains functions specific to the extension, create a corresponding "no-op" file that contains exactly the same function names. In this file:
- Each function is a "no-op," performing no action other than to return a "success" value if expected.
- We recommend naming this "no-op" file by appending `-disabled` to the name of the original file. For example, if the extension is implemented by `extension-foo.go`, the corresponding "no-op" file could be named `extension-foo-disabled.go`.
- The first two lines declare an "anti-tag" (for example, `!foo`). In the *foo* extension example, the "no-op" file will be included in binaries that don't implement the *foo* extension, but won't be included in binaries that implement the *foo* extension. The *foo* example "no-op" file begins with the following lines:

```
//go:build !foo
// +build !foo

package foo

...
```

See extension lint-disabled.go in the zot project for an example of a "no-op" file.

- When developing a new extension, you should create a blackbox test in which a binary containing the new extension can be tested in a usage scenario. See the test/blackbox folder in the zot project for examples of extension tests.
- Create targets in `Makefile` for newly added blackbox tests. You should also add them as GitHub Workflows in .github/workflows/ecosystem-tools.yaml in the zot project.
- When configuring multiple extensions in the `extensions` section of the zot configuration file, list new extensions after the current extensions in the recommended order, such as:

  metrics, sync, search, scrub, lint, *new_extension_1*, *new_extension_2*, ...

6.2.3 Building zot with extensions

When you build the full zot image (for example, `make binary`), all extensions listed in the **EXTENSIONS** variable in `Makefile` are included in the build. When you've created a new extension, you must modify the **EXTENSIONS** variable in `Makefile` by adding the new extension.

To build an image with only selected extensions, you can specify the desired extensions by declaring them in the build command:

```
make binary EXTENSIONS=extension1,extension2,extension3...
```

For example, to build with only sync and scrub, the command would be:

```
make binary EXTENSIONS=sync,scrub
```

September 13, 2023

6.3 Using the zot API

👉 This document describes how to use the zot REST API and provides a number of examples.

For comprehensive details of all zot API commands, see Viewing the complete zot API reference.

The zot API implements the OCI Distribution endpoints along with additional endpoints for supported extensions. You can access the REST API at the same URL and port number used by the GUI and by datapath tools.

✏️ The examples in this article assume that the zot registry is located at `localhost:8080`.

6.3.1 Supported API endpoints

The following is a list of zot API endpoints along with the conditions under which each endpoint is available.

✏️ Some API endpoints are available only when a specific extension is enabled in the zot configuration file, or when an extension build label is specified in the `make` command (for example, `make binary EXTENSIONS=ui`), or both.

For comprehensive details of the API endpoints, see Viewing the complete zot API reference.

OCI endpoints

Endpoint	Actions	Description
/v2/	GET	OCI specification endpoints
/v2/_catalog	GET	Lists repositories in a registry.
/v2/_oci/ext/discover	GET	Discover extensions per the OCI specification
/v2/{repo}/blobs/{digest}	DELETE, GET, HEAD	Lists or deletes an image's blob/layer given a digest
/v2/{repo}/blobs/uploads	POST	Creates an image blob/layer upload
/v2/{repo}/blobs/uploads/{session_id}	DELETE, GET, PATCH, PUT	Creates, lists, or deletes an image's blob/layer upload given a session_id
/v2/{repo}/manifests/{reference}	DELETE, GET, HEAD, PUT	Creates, lists, or deletes references for an image
/v2/{repo}/referrers/{digest}	GET	Lists referrers given a digest
/v2/{repo}/tags/list	GET	List all image tags in a repository

zot OCI extension endpoints

Endpoint	Actions	Description	Availability
/v2/_zot/ext/mgmt	GET	Mgmt extension endpoints	Enabled by using the `mgmt` build label and enabling the `search` extension in the configuration file.
/v2/_zot/ext/notation	POST	With query parameters, uploads certificates for signature verification	Enabled by using the `imagetrust` build label and enabling the `trust` extension with the `notation` option enabled.
/v2/_zot/ext/search		Enhanced search	Enabled by using the `search` build label and enabling the `search` extension in the configuration file.
/v2/_zot/ext/cosign	POST	Uploads keys for signature verification	Enabled by using the `imagetrust` build label and enabling the `trust` extension with the `cosign` option enabled.
/v2/_zot/ext/userprefs	PUT	User preferences endpoints	Enabled by using the `userprefs` build label and enabling both the `search` and the `ui` extensions in the configuration file.

zot auth endpoints

Endpoint	Actions	Description	Availability
/zot/auth/apikey	DELETE, GET, POST	Creates, lists, or deletes API keys	Available when API key authentication is enabled in the configuration file (`"apikey": true`).
/zot/auth/login	POST	Opens an API session	Available when authentication is available. This includes not only OpenID, but all session-based authentication.
/zot/auth/logout	POST	Ends an API session	Available when authentication is available. This includes not only OpenID, but all session-based authentication.
/zot/auth/callback/\<provider>	POST	Specifies a social authentication service provider for redirecting logins, such as Google or dex.	Enabled when an OpenID authentication service provider is specified in the configuration file.

other zot endpoints

Endpoint	Actions	Description	Availability
/v2/_zot/pprof/	GET	Returns a current HTML-format profile list along with a count of currently available records for each profile. See Performance Profiling in zot for usage details.	Always enabled.
/v2/_zot/debug/graphql-playground#		See Using GraphQL for details.	Enabled only in a `binary-debug` zot build or when the zot registry has been built with the `debug` extension label.

ORAS endpoints

Endpoint	Actions	Description	Availability
/oras/artifacts/v1/{repo}/ manifests/ {digest}/referrers	GET	OCI Registry As Storage (ORAS) endpoints	Always enabled.

prometheus endpoint

Endpoint	Actions	Description	Availability
/metrics	GET	Returns extended metrics for monitoring by prometheus	Enabled when the `metrics` extension is enabled in the configuration file.

OpenAPI (swagger) endpoints

✏️ This endpoint is accessed with a browser.

Endpoint	Action	Description	Availability
/swagger/v2/	(browser)	Displays an interactive OpenAPI (swagger) API reference.	Enabled only in a `binary-debug` zot build or when the zot registry has been built with the `debug` extension label.

6.3.2 API authentication

✏️ If zot authentication is not configured, any user can access the zot API.

When pushing and pulling images using API calls, your identity can be authenticated by either a password or an API key.

With a valid password, you can specify your credentials in a cURL request as shown in this example:

```
curl -u <user>:<password> -X GET http://<server>/<endpoint>
```

An API key has advantages in situations where the primary zot authentication does not use the command line, such as a GUI login or social login. Also, you can reduce your security exposure by using an API key instead of your broader credentials, such as an LDAP username and password.

Using API keys

ENABLING API KEYS

To enable the use of API keys, you must set the `apikey` attribute to `true` in the zot configuration file, as shown in the following example:

```
{
  "http": {
    "auth": {
      "apikey": true
    }
  }
}
```

CREATING YOUR API KEY

Before you can create or revoke an API key, you must first log in using a different authentication mechanism, such as logging in through the zot GUI. When you are logged in, you can create an API key for your identity using the following API command:

```
POST /zot/auth/apikey
```

cURL command example:

```
curl -u user:password -X POST http://localhost:8080/zot/auth/apikey -d '{"label": "myAPIKEY", "scopes": ["repo1", "repo2"], "expirationDate":
"2023-08-28T17:10:05+03:00"}'
```

✏️ The scopes and expiration date in this example are optional. By default, an API key has the same permissions as the user who created it.

Command output:

```
{
  "createdAt":"2023-08-28T17:09:59.2603515+03:00",
  "expirationDate":"2023-08-28T17:10:05+03:00",
  "isExpired":false,
  "creatorUa":"curl/7.68.0",
  "generatedBy":"manual",
  "lastUsed":"0001-01-01T00:00:00Z",
  "label":"myAPIKEY",
  "scopes": [
    "repo1",
    "repo2"
  ],
  "uuid":"c931e635-a80d-4b52-b035-6b57be5f6e74",
  "apiKey":"zak_ac55a8693d6b4370a2003fa9e10b3682"
}
```

✏️ The API key (`apiKey`) is shown to the user only in the command output when the key is created. It cannot be later retrieved from zot with any other command.

USING YOUR API KEY IN AN API COMMAND

The API key replaces a password in the API command, as shown in the following cURL example:

```
curl -u user:zak_e77bcb9e9f634f1581756abbf9ecd269 http://localhost:8080/v2/_catalog
```

REMOVING YOUR API KEY

When logged in, you can revoke your own API key with the following API command:

```
DELETE /zot/auth/apikey?id=$uuid
```

cURL command example:

```
curl -u user:password -X DELETE http://localhost:8080/v2/zot/auth/apikey?id=46a45ce7-5d92-498a-a9cb-9654b1da3da1
```

LISTING YOUR CURRENT API KEYS

When logged in, you can display a list of your API keys with the following API command:

```
GET /zot/auth/apikey
```

cURL command example:

```
curl -u user:password -X GET http://localhost:8080/zot/auth/apikey
```

Command output:

```
{
  "apiKeys": [
    {
      "createdAt": "2023-05-05T15:39:28.420926+03:00",
      "expirationDate": "0001-01-01T00:00:00Z",
      "isExpired": true,
      "creatorUa": "curl/7.68.0",
      "generatedBy": "manual",
      "lastUsed": "0001-01-01T00:00:00Z",
      "label": "git",
      "scopes": [
        "repo1",
        "repo2"
      ],
      "uuid": "46a45ce7-5d92-498a-a9cb-9654b1da3da1"
    },
    {
      "createdAt": "2023-08-11T14:43:00.6459729+03:00",
      "expirationDate": "2023-08-17T18:24:05+03:00",
      "isExpired": false,
      "creatorUa": "curl/7.68.0",
      "generatedBy": "manual",
```

```
      "lastUsed": "2023-08-11T14:43:47.5559998+03:00",
      "label": "myAPIKEY",
      "scopes": null,
      "uuid": "294abf69-b62f-4e58-b214-dad2aec0bc52"
    }
  ]
}
```


This command output example shows an expired key and a current key for this user.

✏️ The actual API key (`apiKey`) is not shown. The key is shown to the user only when it is created.

6.3.3 API examples

✏️ The following examples assume that the zot registry is located at `localhost:8080` .

Listing repositories

To get a list of all image repositories in the registry, use the following API endpoint:

```
GET /v2/_catalog
```

cURL command example:

```
curl -X GET http://localhost:8080/v2/_catalog
```

Command output:


```
{
    "repositories": ["alpine", "busybox"]
}
```


Discovering extension endpoints

To list the installed and enabled zot extensions that can be accessed through the API, use the following OCI API endpoint:

```
GET /v2/_oci/ext/discover
```

cURL command example:

```
curl -X GET http://localhost:8080/v2/_oci/ext/discover
```

Command output:


```
{
  "extensions": [
  {
    "name": "_zot",
    "url": "https://github.com/project-zot/zot/blob//pkg/extensions/_zot.md",
    "description": "zot registry extensions",
    "endpoints": ["/v2/_zot/ext/search", "/v2/_zot/ext/userprefs", "/v2/_zot/ext/mgmt"]
  }]
}
```


Uploading a certificate for Notation

To upload a certificate for notation, use the following endpoint:

```
POST /v2/_zot/ext/notation
```

cURL command example:

```
curl --data-binary @certificate.crt -X POST "http://localhost:8080/v2/_zot/ext/notation?truststoreType=ca"
```

6.3.4 Viewing the complete zot API reference

You can find comprehensive details of all zot API commands in either of the following locations:

- As text descriptions in the zot API Command Reference in the zot documentation.
- As an interactive OpenAPI (swagger) JSON file in the zot Github project.

There are many ways to view a swagger file as an interactive document. If you have the `npm` package installed, for example, you can execute the following command:

```
$ npx open-swagger-ui swagger.json
```

This command creates a local web server at `localhost:3355` where you can interact with the API reference using a browser.

November 29, 2023

6.4 zot API Command Reference

👉 This article describes the zot REST API commands, parameters, and responses.

The information presented here is adapted from the interactive OpenAPI (formerly swagger) JSON file in the zot Github project.

For instructions and examples of how to use the zot API, see Using the zot API.

6.4.1 /zot/auth/apikey

DELETE `/zot/auth/apikey`

Revokes one current user API key based on given key ID

Parameters

Name	In	Type	Required	Description
id	query	string	true	api token id (UUID)

Example responses

200 Response

```
"string"
```

Responses

Status	Meaning	Description	Schema
200	OK	ok	string
400	Bad Request	bad request	string
401	Unauthorized	unauthorized	string
500	Internal Server Error	internal server error	string

This operation does not require authentication

GET `/zot/auth/apikey`

Get list of all API keys for a logged in user

Example responses

200 Response

```
"string"
```

Responses

Status	Meaning	Description	Schema
200	OK	ok	string
401	Unauthorized	unauthorized	string
500	Internal Server Error	internal server error	string

This operation does not require authentication

POST `/zot/auth/apikey`

Can create an api key for a logged in user, based on the provided label and scopes.

Body parameter

```
{
  "expirationDate": "string",
  "label": "string",
  "scopes": [
    "string"
  ]
}
```

Parameters

Name	In	Type	Required	Description
body	body	api.APIKeyPayload	true	api token id (UUID)

Example responses

201 Response

```
"string"
```

Responses

Status	Meaning	Description	Schema
201	Created	created	string
400	Bad Request	bad request	string
401	Unauthorized	unauthorized	string
500	Internal Server Error	internal server error	string

This operation does not require authentication

6.4.2 `/zot/auth/logout`

POST `/zot/auth/logout`

Logout by removing current session

Example responses

200 Response

```
"string"
```

Responses

Status	Meaning	Description	Schema
200	OK	ok".	string
500	Internal Server Error	internal server error".	string

This operation does not require authentication

6.4.3 `/oras/artifacts/v1/{name}/manifests/{digest}/referrers`

GET `/oras/artifacts/v1/{name}/manifests/{digest}/referrers`

Get references for an image given a digest and artifact type

Parameters

Name	In	Type	Required	Description
name	path	string	true	repository name
digest	path	string	true	image digest
artifactType	query	string	true	artifact type

Example responses

200 Response

```
"string"
```

Responses

Status	Meaning	Description	Schema
200	OK	ok	string
404	Not Found	not found	string
500	Internal Server Error	internal server error	string

This operation does not require authentication

6.4.4 `/v2/`

GET `/v2/`

Check if this API version is supported

Example responses

200 Response

```
"string"
```

Responses

Status	Meaning	Description	Schema
200	OK	ok".	string

This operation does not require authentication

6.4.5 `/v2/_catalog`

GET `/v2/_catalog`

List all image repositories

Example responses

200 Response


```
{
  "repositories": [
    "string"
  ]
}
```


Responses

Status	Meaning	Description	Schema
200	OK	OK	api.RepositoryList
500	Internal Server Error	internal server error	string

This operation does not require authentication

6.4.6 `/v2/_oci/ext/discover`

GET `/v2/_oci/ext/discover`

List all extensions present on registry

Example responses

200 Response


```
{
  "extensions": [
    {
      "description": "string",
      "endpoints": [
        "string"
      ],
      "name": "string",
      "url": "string"
    }
  ]
}
```


Responses

Status	Meaning	Description	Schema
200	OK	OK	api.ExtensionList

This operation does not require authentication

6.4.7 `/v2/_zot/ext/cosign`

POST `/v2/_zot/ext/cosign`

Upload cosign public keys for verifying signatures

Body parameter


```
string
```


Parameters

Name	In	Type	Required	Description
body	body	string	true	Public key content

Example responses

200 Response

```
"string"
```

Responses

Status	Meaning	Description	Schema
200	OK	ok	string
400	Bad Request	bad request".	string
500	Internal Server Error	internal server error".	string

This operation does not require authentication

6.4.8 `/v2/_zot/ext/mgmt`

GET `/v2/_zot/ext/mgmt`

Get current server configuration

Parameters

Name	In	Type	Required	Description
resource	query	string	false	specify resource

Enumerated Values

Parameter	Value
resource	config

Example responses

200 Response

```
{
  "binaryType": "string",
  "distSpecVersion": "string",
  "http": {
    "auth": {
      "bearer": {
        "realm": "string",
        "service": "string"
      },
      "htpasswd": {
        "path": "string"
      },
      "ldap": {
        "address": "string"
      },
      "openid": {
        "providers": {
          "property1": {
            "name": "string"
          },
          "property2": {
            "name": "string"
          }
        }
      }
    }
  }
}
```

Responses

Status	Meaning	Description	Schema
200	OK	OK	extensions.StrippedConfig
500	Internal Server Error	internal server error".	string

This operation does not require authentication

6.4.9 `/v2/_zot/ext/notation`

POST `/v2/_zot/ext/notation`

Upload notation certificates for verifying signatures

Body parameter

```
string
```

Parameters

Name	In	Type	Required	Description
truststoreType	query	string	false	truststore type
body	body	string	true	Certificate content

Example responses

200 Response

```
"string"
```

Responses

Status	Meaning	Description	Schema
200	OK	ok	string
400	Bad Request	bad request".	string
500	Internal Server Error	internal server error".	string

This operation does not require authentication

6.4.10 `/v2/_zot/ext/userprefs`

PUT `/v2/_zot/ext/userprefs`

Add bookmarks/stars info

Parameters

Name	In	Type	Required	Description
action	query	string	true	specify action
repo	query	string	true	repository name

Enumerated Values

Parameter	Value
action	toggleBookmark
action	toggleStar

Example responses

200 Response

```
"string"
```

Responses

Status	Meaning	Description	Schema
200	OK	ok	string
400	Bad Request	bad request".	string
403	Forbidden	forbidden	string
404	Not Found	not found	string
500	Internal Server Error	internal server error	string

This operation does not require authentication

6.4.11 `/v2/{name}/blobs/{digest}`

DELETE `/v2/{name}/blobs/{digest}`

Delete an image's blob/layer given a digest

Parameters

Name	In	Type	Required	Description
name	path	string	true	repository name
digest	path	string	true	blob/layer digest

Example responses

202 Response

```
"string"
```

Responses

Status	Meaning	Description	Schema
202	Accepted	accepted	string

This operation does not require authentication

GET `/v2/{name}/blobs/{digest}`

Get an image's blob/layer given a digest

Parameters

Name	In	Type	Required	Description
name	path	string	true	repository name
digest	path	string	true	blob/layer digest

Example responses

200 Response

Responses

Status	Meaning	Description	Schema
200	OK	OK	api.ImageManifest

This operation does not require authentication

HEAD `/v2/{name}/blobs/{digest}`

Check an image's blob/layer given a digest

Parameters

Name	In	Type	Required	Description
name	path	string	true	repository name
digest	path	string	true	blob/layer digest

Example responses

200 Response


```
{
  "annotations": {
    "property1": "string",
    "property2": "string"
  },
  "artifactType": "string",
  "config": {
    "annotations": {
      "property1": "string",
      "property2": "string"
    },
    "artifactType": "string",
    "data": [
      0
    ],
    "digest": "string",
    "mediaType": "string",
    "platform": {
      "architecture": "string",
      "os": "string",
      "os.features": [
        "string"
      ],
      "os.version": "string",
      "variant": "string"
    },
    "size": 0,
    "urls": [
      "string"
    ]
  },
  "layers": [
    {
      "annotations": {
        "property1": "string",
        "property2": "string"
      },
      "artifactType": "string",
      "data": [
```

```
        0
      ],
      "digest": "string",
      "mediaType": "string",
      "platform": {
        "architecture": "string",
        "os": "string",
        "os.features": [
          "string"
        ],
        "os.version": "string",
        "variant": "string"
      },
      "size": 0,
      "urls": [
        "string"
      ]
    }
  ],
  "mediaType": "string",
  "schemaVersion": 0,
  "subject": {
    "annotations": {
      "property1": "string",
      "property2": "string"
    },
    "artifactType": "string",
    "data": [
      0
    ],
    "digest": "string",
    "mediaType": "string",
    "platform": {
      "architecture": "string",
      "os": "string",
      "os.features": [
        "string"
      ],
      "os.version": "string",
      "variant": "string"
    },
    "size": 0,
    "urls": [
      "string"
    ]
  }
}
```


Responses

Status	Meaning	Description	Schema
200	OK	OK	api.ImageManifest

Response Headers

Status	Header	Type	Format	Description
200	constants.DistContentDigestKey	object		none

This operation does not require authentication

6.4.12 `/v2/{name}/blobs/uploads`

POST `/v2/{name}/blobs/uploads`

Create a new image blob/layer upload

Parameters

Name	In	Type	Required	Description
name	path	string	true	repository name

Example responses

202 Response

```
"string"
```

Responses

Status	Meaning	Description	Schema
202	Accepted	accepted	string
401	Unauthorized	unauthorized	string
404	Not Found	not found	string
500	Internal Server Error	internal server error	string

Response Headers

Status	Header	Type	Format	Description
202	Location	string		/v2/{name}/blobs/uploads/{session_id}
202	Range	string		0-0

This operation does not require authentication

6.4.13 `/v2/{name}/blobs/uploads/{session_id}`

DELETE `/v2/{name}/blobs/uploads/{session_id}`

Delete an image's blob/layer given a digest

Parameters

Name	In	Type	Required	Description
name	path	string	true	repository name
session_id	path	string	true	upload session_id

Example responses

200 Response

```
"string"
```

Responses

Status	Meaning	Description	Schema
200	OK	ok	string
404	Not Found	not found	string
500	Internal Server Error	internal server error	string

This operation does not require authentication

GET `/v2/{name}/blobs/uploads/{session_id}`

*Get an image's blob/layer upload given a session_id

Parameters

Name	In	Type	Required	Description
name	path	string	true	repository name
session_id	path	string	true	upload session_id

Example responses

204 Response

```
"string"
```

Responses

Status	Meaning	Description	Schema
204	No Content	no content	string
404	Not Found	not found	string
500	Internal Server Error	internal server error	string

This operation does not require authentication

PATCH `/v2/{name}/blobs/uploads/{session_id}`

Resume an image's blob/layer upload given an session_id

Parameters

Name	In	Type	Required	Description
name	path	string	true	repository name
session_id	path	string	true	upload session_id

Example responses

202 Response

```
"string"
```

Responses

Status	Meaning	Description	Schema
202	Accepted	accepted	string
400	Bad Request	bad request	string
404	Not Found	not found	string
416	Range Not Satisfiable	range not satisfiable	string
500	Internal Server Error	internal server error	string

Response Headers

Status	Header	Type	Format	Description
202	Location	string		/v2/{name}/blobs/uploads/{session_id}
202	Range	string		0-128

This operation does not require authentication

PUT `/v2/{name}/blobs/uploads/{session_id}`

Update and finish an image's blob/layer upload given a digest

Parameters

Name	In	Type	Required	Description
name	path	string	true	repository name
session_id	path	string	true	upload session_id
digest	query	string	true	blob/layer digest

Example responses

201 Response

```
"string"
```

Responses

Status	Meaning	Description	Schema
201	Created	created	string
404	Not Found	not found	string
500	Internal Server Error	internal server error	string

This operation does not require authentication

6.4.14 `/v2/{name}/manifests/{reference}`

DELETE `/v2/{name}/manifests/{reference}`

Delete an image's manifest given a reference or a digest

Parameters

Name	In	Type	Required	Description
name	path	string	true	repository name
reference	path	string	true	image reference or digest

Example responses

200 Response

```
"string"
```

Responses

Status	Meaning	Description	Schema
200	OK	ok	string

This operation does not require authentication

GET `/v2/{name}/manifests/{reference}`

Get an image's manifest given a reference or a digest

Parameters

Name	In	Type	Required	Description
name	path	string	true	repository name
reference	path	string	true	image reference or digest

Example responses

200 Response


```
{
  "annotations": {
    "property1": "string",
    "property2": "string"
  },
  "artifactType": "string",
  "config": {
    "annotations": {
      "property1": "string",
      "property2": "string"
    },
    "artifactType": "string",
    "data": [
      0
    ],
    "digest": "string",
    "mediaType": "string",
    "platform": {
      "architecture": "string",
      "os": "string",
      "os.features": [
        "string"
      ],
      "os.version": "string",
      "variant": "string"
    },
    "size": 0,
    "urls": [
      "string"
    ]
  },
  "layers": [
    {
      "annotations": {
        "property1": "string",
        "property2": "string"
      },
      "artifactType": "string",
      "data": [
        0
      ],
      "digest": "string",
      "mediaType": "string",
      "platform": {
        "architecture": "string",
        "os": "string",
        "os.features": [
          "string"
        ],
        "os.version": "string",
        "variant": "string"
      },
      "size": 0,
      "urls": [
        "string"
      ]
    }
```

```
  ],
  "mediaType": "string",
  "schemaVersion": 0,
  "subject": {
    "annotations": {
      "property1": "string",
      "property2": "string"
    },
    "artifactType": "string",
    "data": [
      0
    ],
    "digest": "string",
    "mediaType": "string",
    "platform": {
      "architecture": "string",
      "os": "string",
      "os.features": [
        "string"
      ],
      "os.version": "string",
      "variant": "string"
    },
    "size20": 0,
    "urls": [
      "string"
    ]
  }
}
```


Responses

Status	Meaning	Description	Schema
200	OK	OK	api.ImageManifest
404	Not Found	not found	string
500	Internal Server Error	internal server error	string

Response Headers

Status	Header	Type	Format	Description
200	constants.DistContentDigestKey	object		none

This operation does not require authentication

HEAD `/v2/{name}/manifests/{reference}`

Check an image's manifest given a reference or a digest

Parameters

Name	In	Type	Required	Description
name	path	string	true	repository name
reference	path	string	true	image reference or digest

Example responses

200 Response


```
"string"
```

Responses

Status	Meaning	Description	Schema
200	OK	ok	string
404	Not Found	not found	string
500	Internal Server Error	internal server error".	string

Response Headers

Status	Header	Type	Format	Description
200	constants.DistContentDigestKey	object		none

This operation does not require authentication

PUT `/v2/{name}/manifests/{reference}`

Update an image's manifest given a reference or a digest

Parameters

Name	In	Type	Required	Description
name	path	string	true	repository name
reference	path	string	true	image reference or digest

Example responses

201 Response

```
"string"
```

Responses

Status	Meaning	Description	Schema
201	Created	created	string
400	Bad Request	bad request	string
404	Not Found	not found	string
500	Internal Server Error	internal server error	string

This operation does not require authentication

6.4.15 `/v2/{name}/referrers/{digest}`

GET `/v2/{name}/referrers/{digest}`

Get referrers given a digest

Parameters

Name	In	Type	Required	Description
name	path	string	true	repository name
digest	path	string	true	digest
artifactType	query	string	false	artifact type

Example responses

200 Response


```
{
  "annotations": {
    "property1": "string",
    "property2": "string"
  },
  "artifactType": "string",
  "manifests": [
    {
      "annotations": {
        "property1": "string",
        "property2": "string"
      },
      "artifactType": "string",
      "data": [
        0
      ],
      "digest": "string",
      "mediaType": "string",
      "platform": {
        "architecture": "string",
        "os": "string",
        "os.features": [
          "string"
        ],
        "os.version": "string",
        "variant": "string"
      },
      "size": 0,
      "urls": [
        "string"
      ]
    }
  ],
  "mediaType": "string",
  "schemaVersion": 0,
  "subject": {
    "annotations": {
      "property1": "string",
      "property2": "string"
    },
    "artifactType": "string",
    "data": [
      0
    ],
    "digest": "string",
    "mediaType": "string",
    "platform": {
      "architecture": "string",
      "os": "string",
      "os.features": [
        "string"
      ],
      "os.version": "string",
      "variant": "string"
    },
    "size": 0,
    "urls": [
      "string"
    ]
  }
}
```

Responses

Status	Meaning	Description	Schema
200	OK	OK	api.ImageIndex
404	Not Found	not found	string
500	Internal Server Error	internal server error	string

This operation does not require authentication

6.4.16 `/v2/{name}/tags/list`

GET `/v2/{name}/tags/list`

List all image tags in a repository

Parameters

Name	In	Type	Required	Description
name	path	string	true	repository name
n	query	integer	true	limit entries for pagination
last	query	string	true	last tag value for pagination

Example responses

200 Response

```
{
  "name": "string",
  "tags": [
    "string"
  ]
}
```

Responses

Status	Meaning	Description	Schema
200	OK	OK	common.ImageTags
400	Bad Request	bad request".	string
404	Not Found	not found	string

This operation does not require authentication

November 17, 2023

6.5 Contributing to zot Development

👉 The zot project is built for developers by developers. The zot project welcomes the participation of the open source community in extending and improving zot.

6.5.1 Submission Requirements

Summary: All contributions must meet these requirements:

- Adhere to the Apache license
- Be submitted by a pull request (PR) from your fork
- Commits must have a

License

zot is released under the Apache License 2.0. All contributions must adhere to this license and must explicitly state adherence.

Submitting a Pull Request (PR)

First, fork the zot project on GitHub and submit a commit to your fork. Then open a new pull request (PR) to the zot project. All pull requests must meet these requirements:

- License statement

Either the commit message or the PR description must contain the following statement:

"By submitting this pull request, I confirm that my contribution is made under the terms of the Apache 2.0 license."

- Developer Certificate of Origin (DCO) and sign-off

All commits require a Developer Certificate of Origin via the "Signed-off-by:" commit message and commit signatures using GPG keys. Include the `-s` flag in your `git commit` command.

- Commit message format

The commit message must follow the Convention Commits format. The message must begin with a keyword that categorizes the commit, followed by a colon. Validation of a commit message is determined by this expression:

```
"^((build|chore|ci|docs|feat|fix|perf|refactor|revert|style|test)(\(.+\))?(!)?(: (.*\s*)*))"
```

An example of a valid commit message is "docs: Fixes a typo in module.md."

In addition, any new PR requires a brief form to be completed by the submitter with details about the PR. Appropriate code owners are automatically identified and will be notified of the new PR.

CI/CD Checks

We take code quality very seriously. All PRs must pass various CI/CD checks that enforce code quality such as code coverage, security scanning, performance regressions, distribution spec conformance, ecosystem client tool compatibility, etc.

6.5.2 Reporting Issues

Issues are broadly classified as functional bugs and security issues. The latter is treated a little differently due to the sensitive nature.

Filing a Functional Issue

No software is perfect, and we expect users to find issues with the zot code base. First, check whether your issue has already been filed by someone else by performing an issue search. If the issue is not found, file a new issue by clicking the **New issue** button on the **zot/issues** page and answering the questions. The more information that you can provide, the easier it becomes to triage the issue.

Filing a Security Issue

Security issues are best filed by sending an email to `security@zotregistry.dev.` After 45 days, we will make the issue public and give credit to the original filer of the issue.

6.5.3 Code of Conduct

The zot project follows the CNCF Code of Conduct.

Reporting Conduct Incidents

To report a conduct-related incident occurring on the zot project, contact the zot project conduct committee by sending an email to `conduct@zotregistry.dev.` You can expect a response within three business days.

January 16, 2024

7. Articles

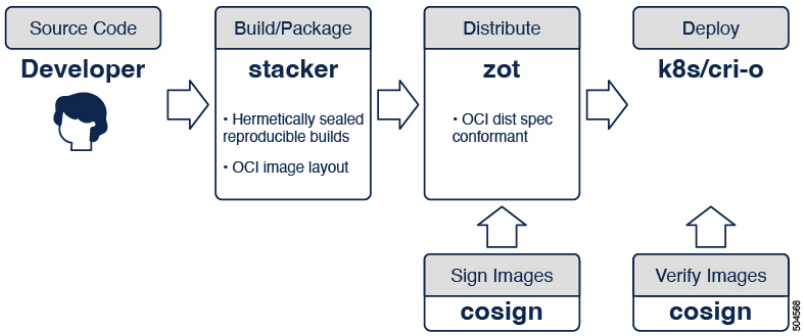
7.1 Building an OCI-native Container Image CI/CD Pipeline

👉 An **OCI-native** secure container image build/delivery pipeline using the following tools:

- `stacker` for building OCI images
- `zot` as a vendor-neutral OCI image registry server
- `skopeo` for performing repository interactions
- `cosign` for container signing and verification
- `cri-o` for deploying container images
- `cosigned` for validating container images before deployment

The Open Container Initiative (OCI) is an open governance structure for the express purpose of creating open industry standards around container formats and runtimes.

This document describes a step-by-step procedure towards achieving an OCI-native secure software supply chain using zot in collaboration with other open source tools. The following diagram shows a portion of the CI/CD pipeline.



7.1.1 Build images

`stacker` is a standalone tool for building OCI images via a declarative `yaml` format. The output of the build process is a container image in an OCI layout.

example: stacker build command

```
stacker build -f <stackerfile.yaml>
```

7.1.2 Image registry

zot is a production-ready vendor-neutral OCI image registry server purely based on the OCI Distribution Specification. If `stacker` is used to build the OCI image, it can also be used to publish the built image to an OCI registry.

example: stacker publish command

```
stacker publish --url <url> --username <user> --password <password>
```

Alternatively, you can use `skopeo`, a command line utility that performs various operations on container images and image repositories.

example: skopeo copies an image to a registry

```
skopeo copy --format=oci oci:<oci-dir>/image:tag \
  docker://<zot-server>/image:tag
```

ck here to view an example of pushing and pulling an image using skopeo.

7.1.3 Signing images

`cosign` is a tool that performs container signing, verification, and storage in an OCI registry.

example: cosign generates keys and signs an image in the registry

```
cosign generate-key-pair

cosign sign --key cosign.key <zot-server>/image:tag
```

ck here to view an example of cosign operations.

7.1.4 Deploying container images

`cri-o` is an implementation of the Kubernetes Container Runtime Interface (CRI) to enable using OCI compatible runtimes. It is a lightweight alternative to using Docker as the runtime for Kubernetes.

✏️ zot is compatible with kubernetes/cri-o using `docker://` transport, which is the default.

example: kubelet configuration file

```
apiVersion: v1
kind: Pod
metadata:
  name: example-pod
spec:
  containers:
  - name: example-container
    image: <zot-server>/image:tag
```

7.1.5 Container image verification

`cosigned` is an image admission controller that validates container images before deploying them.

example: install cosigned using Helm

```
kubectl create namespace cosign-system

kubectl create secret generic mysecret -n \
  cosign-system --from-file=cosign.pub=./cosign.pub

helm repo add sigstore https://sigstore.github.io/helm-charts

helm repo update
```

```
helm install cosigned -n cosign-system sigstore/cosigned \
  --devel --set cosign.secretKeyRef.name=mysecret
```

May 18, 2023

7.2 User Authentication and Authorization with zot

👉 A robust set of authentication/authorization options are supported:

- Authentication
- TLS, including mTLS
- Username/password or token-based user authentication
- LDAP
- htpasswd
- OAuth2 with bearer token
- Authorization
- Powerful identity-based access controls for repositories or specific repository paths
- OpenID/OAuth2 social login with Google, GitHub, GitLab, and dex

The zot configuration model supports both authentication and authorization. Authentication credentials allow access to zot HTTP APIs. Authorization policies provide fine-grained control of the actions each authenticated user can perform in the registry.

7.2.1 Authentication

TLS authentication

Because authentication credentials are passed over HTTP, it is imperative that TLS be enabled. You can enable and configure TLS authentication in the zot configuration file, as shown in the following example.

```
"http": {
...
  "tls": {
    "cert": "/etc/zot/certs/server.cert",
    "key": "/etc/zot/certs/server.key"
  }
```

See Mutual TLS authentication for additional information about TLS.

HTTP basic authentication

When basic HTTP authentication is used, the username and password credentials are joined by a colon (`:`), base64 encoded, and passed in the HTTP Authorization header.

HTTP bearer authentication

To avoid passing the username and password credentials for every HTTP request, a zot client can use bearer token-based authentication. In this method, the client first authenticates with a token server and receives a short-lived token. The client then passes this token in the HTTP Authorization header, specifying `Bearer` as the authentication scheme.

Configure bearer authentication in the zot configuration file as shown in this example.

```
"http": {
...
  "auth": {
    "bearer": {
      "realm": "https://auth.myreg.io/auth/token",
      "service": "myauth",
      "cert": "/etc/zot/auth.crt"
    }
  }
```

The following table lists the configurable attributes.

Attribute	Description
`realm`	A string typically related to the authentication scheme (*BASIC* and *BEARER*).
`service`	The name of the authentication service.
`cert`	The path and filename containing the public key to use to verify tokens. This file can either contain the public key directly, or an SSL/TLS certificate from which the key will be extracted.

7.2.2 Mutual TLS authentication

zot supports basic TLS and password-less mutual TLS authentication (mTLS). Specifying a `cacert` file in the TLS section of the zot configuration file enables mTLS. The `cacert` parameter is used to validate the client-side TLS certificates.

```
"http": {
...
  "tls": {
    "cert": "/etc/zot/certs/server.cert",
    "key": "/etc/zot/certs/server.key",
    "cacert": "/etc/zot/certs/ca.cert"
  }
```

The following table lists the configurable attributes.

Attribute	Description
`cert`	The path and filename of the server's SSL/TLS certificate.
`key`	The path and filename of the server's registry key.
`cacert`	The path and filename of the server's `cacerts` file, which contains trusted certificate authority (CA) certificates.

Preventing automated attacks with failure delay

To help prevent automated attacks, you can add a delayed response to an authentication failure. Configure the `failDelay` attribute in the configuration file as shown in the following example.

```
"http": {
  "auth": {
    "failDelay": 5
  }
}
```

The `failDelay` attribute specifies a waiting time, in seconds, before zot sends a failure notification to an authenticating user who has been denied access.

7.2.3 Server-side authentication

You can implement server-side authentication for zot using `htpasswd` or LDAP or both.

✏️ When both `htpasswd` and LDAP configuration are specified, LDAP authentication is given preference. Because `htpasswd` authentication is strictly local and requires no remote service, `htpasswd` serves as a fail-safe authentication mechanism should LDAP become unavailable.

LDAP

zot supports integration with an LDAP-based authentication service such as Microsoft Windows Active Directory (AD). Enable and configure LDAP authentication in the zot configuration file, as shown in the following example.

```
"http": {
...
  "auth": {
    "ldap": {
```

```
      "credentialsFile": "examples/config-ldap-credentials.json",
      "address": "ldap.example.org",
      "port": 389,
      "startTLS": false,
      "baseDN": "ou=Users,dc=example,dc=org",
      "userAttribute": "uid",
      "userFilter": "(!(nsaccountlock=TRUE))",
      "userGroupAttribute": "memberOf",
      "skipVerify": false,
      "subtreeSearch": true
    }
  }
}
```

The following table lists the configurable attributes for LDAP authentication.

Attribute	Description
`credentialsFile`	The path to a file containing the bind credentials for LDAP.
`address`	The IP address or hostname of the LDAP server.
`port`	The port number used by the LDAP service.
`startTLS`	Set to `true` to enable TLS communication with the LDAP server.
`baseDN`	Starting location within the LDAP directory for performing user searches.
`userAttribute`	Attribute name used to obtain the username.
`userFilter`	Optional user filter, which would be used in addition to `userAttribute`
`userGroupAttribute`	Attribute name used to obtain groups to which a user belongs.
`skipVerify`	Skip TLS verification.
`subtreeSearch`	Set to `true` to expand the scope for search to include subtrees of the base DN.

To allow for separation of configuration and credentials, the credentials for the LDAP server are specified in a separate file, as shown in the following example.

```
{
  "bindDN":"cn=ldap-searcher,ou=Users,dc=example,dc=org",
  "bindPassword":"ldap-searcher-password"
}
```

The following table lists the configurable attributes of the LDAP credentials file.

Attribute	Description
`bindDN`	Base Distinguished Name for the LDAP search.
`bindPassword`	Password of the bind LDAP user.

htpasswd

Enable and configure `htpasswd` authentication in the zot configuration file, as shown in the following example.

1. Create and store an `htpasswd` file on the server.

```
$ htpasswd -bBn <username> <password> >> /etc/zot/htpasswd
```

✏️ For strong security, make sure to use the -B option, specifying the bcrypt hashing algorithm. This is the only algorithm supported by zot for `htpasswd`.

2. Enable `htpasswd` authentication and configure the path to the `htpasswd` authentication in the zot configuration file.

```
"http": {
...
  "auth": {
      "htpasswd": {
        "path": "/etc/zot/htpasswd"
      },
```

The `path` attribute specifies the path and filename of the `htpasswd` file, which contains user names and hashed passwords.

7.2.4 Authorization

With an access scheme that relies solely on authentication, any authenticated user would be given complete access to the registry. To better control access, zot supports identity-based repository-level access control (authorization) policies. The access control policy is a function of *repository*, *user*, and the *action* being performed on that repository.

Access control policies

Five identity-based types of access control policies are supported:

Policy type	Attribute	Access allowed
Default	`defaultPolicy`	The default policy specifies what actions are allowed if a user is authenticated but does match any user-specific policy.
User-specific	`users`, `actions`	A user-specific policy specifies access and actions for explicitly named users.
Group-specific	`groups`, `actions`	A group-specific policy specifies access and actions for explicitly named groups.
Anonymous	`anonymousPolicy`	An anonymous policy specifies what an unauthenticated user is allowed to do. This is an appropriate policy when you want to grant open read-only access to one or more repositories.
Metrics	`metrics`	The metrics policy is a access control policy which grants the privilege to read data from the metrics endpoint.
Admin	`adminPolicy`	The admin policy is a global access control policy that grants privileges to perform actions on any repository.

Access control is organized by repositories, users, and their actions. Most users of a particular repository will have similar access control requirements and can be served by a repository-specific `defaultPolicy`. Should a user require an exception to the default policy, a user-specific or group-specific override policy can be configured.

With an `anonymousPolicy`, a repository can allow anonymous actions which do not require user authentication. Finally, one or more users can be designated as administrators, to whom the global `adminPolicy` applies.

A user's access to a particular repository is evaluated first by whether a user-specific policy exists, then by group-specific policies, and then (in order) by default and admin policies.

A group-specific policy can be applied within any type of access policy, including default or admin policies. The group policy name can also be used with LDAP.

CONFIGURING ACCESS CONTROL

User identity or group identity can be used as an authorization criterion for allowing actions on one or more repository paths. For specific users, you can choose to allow any combination of read, create, update, or delete actions on specific paths.

When you define policies for specific repository paths, the paths can be specified explicitly or by using `glob` patterns with simple or recursive wildcards. When a repository path matches more than one path description, authorization is granted based on the policy of the longest (most specific) path matched. For example, if policies are defined for path descriptions `**` and `repos2/repo,` the `repos2/repo` path will match both `**` and `repos2/repo` descriptions. In this case, the `repos2/repo` policy will be applied because it is longer.

Note that `**` effectively defines the default policy, as it matches any path not matched by any other per-repository policy. To override all other policies, you can specify a global admin policy.

✏️ Always include the read action in any policy that you define. The create, update, and delete actions cannot be used without the read action.

EXAMPLE: ACCESS CONTROL CONFIGURATION

Use the `accessControl` attribute in the configuration file to define a set of identity-based authorization policies, as shown in the following example.

```
"http": {
...
  "accessControl": {
    "groups": {
      "group1": {
        "users": ["bob", "mary"]
      },
      "group2": {
        "users": ["alice", "mallory", "jim"]
      }
    },
    "metrics":{
        "users": ["john"]
    },
    "repositories": {
      "**": {
        "policies": [{
          "users": ["charlie"],
          "groups": ["group2"],
          "actions": ["read", "create", "update"]
        }],
        "defaultPolicy": ["read", "create"]
      },
      "tmp/**": {
        "anonymousPolicy": ["read"],
        "defaultPolicy": ["read", "create", "update"]
      },
      "infra/*": {
        "policies": [{
            "users": ["alice", "bob"],
            "actions": ["create", "read", "update", "delete"]
          },
          {
            "users": ["mallory"],
            "groups": ["group1"],
            "actions": ["create", "read"]
          }
        ],
        "defaultPolicy": ["read"]
      },
      "repos2/repo": {
        "policies": [{
            "users": ["bob"],
            "actions": ["read", "create"]
          },
          {
            "users": ["mallory"],
            "actions": ["create", "read"]
          }
        ],
        "defaultPolicy": ["read"]
      }
    },
    "adminPolicy": {
      "users": ["admin"],
      "actions": ["read", "create", "update", "delete"]
    }
  }
```

In this example, five policies are defined:

- The default policy (`**`) gives all authenticated users the ability to read or create content, while giving user "charlie" and those in "group2" the additional ability to update content.
- The policy for `tmp/**` matches all repositories under `tmp` recursively and allows all authenticated users to read, create, or update content in those repositories. Unauthenticated users have read-only access to these repositories.
- The policy for `infra/*` matches all repositories directly under `infra.` Separate policies are defined for specific users, along with a default read-only policy for all other users.
- The policy for `repos2/repo` matches only that specific repository.
- An admin policy (`adminPolicy`) gives the user "admin" global authorization to read, create, update, or delete content in any repository, overriding all other policies.
- A metrics policy (`metrics`) gives the user "john" permissions to read data from the metrics endpoint for monitoring purposes.

✏️ In case of LDAP groups, the group FQDN must be used in the zot configuration. For example `"groups": ["cn=ldap-group,ou=Groups,dc=example,dc=org"],` .

✏️ In releases prior to zot v2.0.0, authorization policies were defined directly under the `accessControl` key in the zot configuration file. Beginning with v2.0.0, the set of authorization policies are now defined under a new `repositories` key.

Social login using OpenID/OAuth2

Social login is an authentication/authorization method in which your existing credentials for another site or service can be used to log in to a service such as zot. For example, you can log in to zot using your GitHub account credentials, and zot will contact GitHub to verify your identity using OAuth 2.0 and OpenID Connect (OIDC) protocols.

Several social login providers are supported by zot:

- github
- google
- gitlab
- oidc (for example, dex)

The following example shows the zot configuration for these providers:

```
{
  "http": {
    "auth": {
      "openid": {
        "providers": {
          "github": {
            "credentialsFile": "examples/github-oidc-credentials.json",
            "scopes": ["read:org", "user", "repo"]
          },
          "google": {
            "credentialsFile": "examples/google-oidc-credentials.json",
            "issuer": "https://accounts.google.com",
            "scopes": ["openid", "email"]
          },
          "gitlab": {
            "credentialsFile": "examples/gitlab-oidc-credentials.json",
            "issuer": "https://gitlab.com",
            "scopes": ["openid", "read_api", "read_user", "profile", "email"]
          },
          "oidc": {
            "credentialsFile": "examples/oidc-credentials.json",
            "issuer": "http://<zot-server>:5556/dex",
            "keypath": "",
            "scopes": ["openid", "profile", "email", "groups"]
          }
        }
      }
    }
  }
}
```

To allow for separation of configuration and credentials, the credentials for oidc are specified in a separate file, as shown in the following example.

```
{
    "clientid": <client_id>,
    "clientsecret": <client_secret>
}
```

The following table lists the configurable attributes of the oidc credentials file.

Attribute	Description
`clientid`	Client ID
`clientsecret`	Client Secret

USING GOOGLE, GITHUB, OR GITLAB

A client logging into zot by social login must specify a supported OpenID/OAuth2 provider as a URL query parameter. A client logging in using Google, GitHub, or GitLab must additionally specify a callback URL for redirection to a zot page after a successful authentication.

The login URL using Google, GitHub, or GitLab uses the following format:

```
http://<zot-server>/auth/login?provider=<provider>&callback_ui=<zot-server>/<page>
```

For example, a user logging in to the zot home page using GitHub as the authentication provider sends this URL:

```
http://zot.example.com:8080/auth/login?provider=github&callback_ui=http://zot.example.com:8080/home
```

Based on the specified provider, zot redirects the login to a provider service with the following URL:

```
http://<zot-server>/zot/auth/callback/<provider>
```

For the GitHub authentication example:

```
http://zot.example.com:8080/zot/auth/callback/github
```

✏️ If your network policy doesn't allow inbound connections, the callback will not work and this authentication method will fail.

USING DEX

dex is an identity service that uses OpenID Connect (OIDC) to drive authentication for client apps, such as zot. While this section shows how to use dex with zot, zot supports other OIDC services as well.

Like zot, dex uses a configuration file for setup. To specify zot as a client in dex, configure a `staticClients` entry in the dex configuration file with a zot callback, such as the following example in the dex configuration file:

```
staticClients:
  - id: zot-client
    redirectURIs:
      - 'http://zot.example.com:8080/zot/auth/callback/oidc'
    name: 'zot'
    secret: ZXhhbXBsZS1hcHAtc2VjcmV0
```

In the zot configuration file, configure dex as an OpenID auth provider as in the following example:

```
  "http": {
    "auth": {
      "openid": {
        "providers": {
          "oidc": {
            "credentialsFile": "examples/sso-oidc-credentials.json",
            "name": "Corporate SSO",
            "issuer": "http://<zot-server>:5556/dex",
            "keypath": "",
            "scopes": ["openid", "profile", "email", "groups"]
          }
        }
      }
```

```
    }
  }
```

where `examples/sso-oidc-credentials.json` contains

```
{
  "clientid": "zot-client",
  "clientsecret": "ZXhhbXBsZS1hcHAtc2VjcmV0"
}
```

A user logging in to zot using dex OpenID authentication sends a URL with dex as a URL query parameter, such as the following example:

`http://zot.example.com:8080/auth/login?provider=oidc`

For detailed information about configuring dex service, see the dex Getting Started documentation.

USING OPENID/OAUTH2 WHEN ZOT IS BEHIND A PROXY OR LOAD BALANCER

When the zot registry is running behind a proxy or load balancer, you must provide an external URL for OpenID/OAuth2 clients to redirect back to zot. This `externalUrl` attribute is the URL of the registry, as shown in this example:

```
  "http": {
    "address": "0.0.0.0",
    "port": "8080",
    "externalUrl": "https://zot.example.com",
    "auth": {
      "openid": {
        "providers": {
          "github": {
            "clientid": <client_id>,
            "clientsecret": <client_secret>,
            "scopes": ["read:org", "user", "repo"]
          }
        }
      }
    }
  }
```

July 16, 2025

7.3 Verifying image signatures

Images stored in zot can be signed with a digital signature to verify the source and integrity of the image. The digital signature can be verified by zot using public keys or certificates uploaded by the user.

To verify image signatures, zot supports the following tools:

- cosign
- notation

7.3.1 Enabling image signature verification

To enable image signature verification, add the `trust` attribute under `extensions` in the zot configuration file and enable one or more verification tools, as shown in the following example:

```
"extensions": {
  "trust": {
    "enable": true,
    "cosign": true,
    "notation": true
  }
}
```

The following table lists the configurable attributes of the `trust` extension.

Attribute	Description
`enable`	If this attribute is missing, signature verification is disabled by default. Signature verification is enabled by including this attribute and setting it to `true`. You must also enable at least one of the verification tools.
`cosign`	Set to `true` to enable signature verification using the `cosign` tool.
`notation`	Set to `true` to enable signature verification using the `notation` tool.

7.3.2 What is needed for verifying signatures

To verify the validity of a signature for an image, zot makes use of two types of files:

- A public key file that pairs with the private key used to sign an image with `cosign`
- A certificate file that is used to sign an image with `notation`

Upload these files using an extension of the zot API, as shown in the following examples:

- **To upload a public key for cosign**:

  API path

  ```
  /v2/_zot/ext/cosign"
  ```

  Example request

  ```
  curl --data-binary @file.pub -X POST "http://localhost:8080/v2/_zot/ext/cosign"
  ```

  Result

  The uploaded file is stored in the `_cosign` directory under the `rootDir` specified in the zot configuration file or in the Secrets Manager.

- **To upload a certificate for notation**:

  API path

  ```
  /v2/_zot/ext/notation?truststoreType=ca
  ```

  When uploading a certificate, you should specify the `truststoreType`. If the truststore is a certificate authority, the value is `ca`. This is the default if this attribute is omitted.

  Example request

  ```
  curl --data-binary @certificate.crt -X POST "http://localhost:8080/v2/_zot/ext/notation?truststoreType=ca"
  ```

  Result

  The uploaded file is stored in the `_notation/truststore/x509/{truststoreType}/default` directory under the `rootDir` specified in the zot configuration file or in the Secrets Manager.

7.3.3 Where needed files are stored

Uploaded public keys and certificates are stored in the local filesystem, in specific directories named `_cosign` and `_notation` under `$rootDir`, or in the Secrets Manager.

- The `_cosign` directory contains uploaded public key files in the following structure:

```
_cosign
├── $publicKey1
└── $publicKey2
```

- The `_notation` directory contains a set of files in the following structure:

```
_notation
├── trustpolicy.json
└── truststore
    └── x509
        └── $truststoreType
            └── default
                └── $certificate
```

In this directory, the `trustpolicy.json` file contains content that is updated automatically whenever a new certificate is added to a new truststore. This content cannot be changed by the user. An example of the `trustpolicy.json` file content is shown below:

```
{
 "version": "1.0",
  "trustPolicies": [
    {
      "name": "default-config",
      "registryScopes": [ "*" ],
      "signatureVerification": {
        "level" : "strict"
      },
      "trustStores": ["ca:default","signingAuthority:default"],
      "trustedIdentities": [
        "*"
      ]
    }
  ]
}
```

- By default, the `trustpolicy.json` file sets the `signatureVerification.level` property to `strict`, which enforces all validations. For example, a signature is not trusted if its certificate has expired, even if the certificate verifies the signature.
- The `trustpolicy.json` file contains two default truststores, `ca:default` and `signingAuthority:default`. This list of truststores is not updated when a new certificate is uploaded.
- The content of the `trustStores` field will match the content of the `_notation/truststore` directory.

7.3.4 How signature verification works

Based on the uploaded files and the information about images stored in zot's database, signature verification is performed for all signed images. The verification result for each signed image is stored in the database and is visible from GraphQL. The stored information about a signature includes:

- The tool that was used to generate the signature, such as `cosign` or `notation`
- The trustworthiness of the signature, such as whether a certificate or public key exists that can successfully verify the signature
- The author of the signature, which can be either:
- The public key, for signatures generated using `cosign`
- The subject of the certificate, for signatures generated using `notation`

7.3.5 Example of GraphQL output

Sample request

```
{
  Image(image: "busybox:latest") {
    Digest
    IsSigned
    Tag
    SignatureInfo {
        Tool
        IsTrusted
        Author
    }
  }
}
```

Sample response

```
{
  "data": {
    "Image": {
      "Digest":"sha256:6c19fba547b87bde9a45df2f8563e0c61826d098dd30192a2c8b86da1e1a6360",
      "IsSigned": true,
      "Tag": "latest",
      "SignatureInfo":[
        {
          "Tool":"cosign",
          "IsTrusted":false,
          "Author":""
        },
        {
          "Tool":"cosign",
          "IsTrusted":false,
          "Author":""
        },
        {
          "Tool":"cosign",
          "IsTrusted": true,
          "Author":"-----BEGIN PUBLIC KEY-----\nMFkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDQgAE9pN+/
hGcFlh4YYaNvZxNvuh8Qyhl\npURz77qScOHe3DqdmiWiuqIseyhEdjEDwpL6fHRwu3a2Nd9wbKqm0la76w==\n-----END PUBLIC KEY-----\n"
        },
        {
          "Tool":"notation",
          "IsTrusted": false,
          "Author":"CN=v4-test,O=Notary,L=Seattle,ST=WA,C=US"
        },
        {
          "Tool":"notation",
          "IsTrusted": true,
          "Author":"CN=multipleSig,O=Notary,L=Seattle,ST=WA,C=US"
        }
      ]
    }
  }
}
```

September 13, 2023

7.4 Immutable Image Tags

👉 Immutable image tag support is achieved by leveraging authorization policies.

It is considered best practice to avoid changing the content once a software version has been released. While `zot` does not have an explicit configuration flag to make image tags immutable, the same effect can be achieved with authorization as follows.

7.4.1 Immutable For All Users

By setting the `defaultPolicy` to "read" and "create" for a particular repository, images can be pushed (once) and pulled but further updates are rejected.

```
{
...
  "repositories": {
    "**": {
      "defaultPolicy": ["read", "create"]
    }
  }
...
}
```

7.4.2 Immutable With Overrides

As in the example above, with `defaultPolicy` set to "read" and "create" for a particular repository, images can be pushed (once) and pulled, but further updates are rejected. Exceptions can be made for some users, and user-specific policies can be added to allow "update" operations as shown below.

```
{
...
  "repositories": {
    "**": {
      "policies": [{
        "users": ["alice", "bob"],
        "actions": ["read", "create", "update"]
      }],
      "defaultPolicy": ["read", "create"]
    }
  }
...
}
```

March 13, 2024

7.5 Software Provenance Workflow Using OCI Artifacts

👉 This article demonstrates an end-to-end workflow for installing a zot registry, then pushing and signing an image and a related artifact, such as an SBOM.

7.5.1 Workflow

The following sections describe the step-by-step workflow. To view the steps combined into a single script, see Reference: Full workflow script.

For the workflow examples, the zot registry is assumed to be installed and running at `localhost:8080`.

Step 1: Download the client tools

This workflow uses the `regctl` registry client and the `cosign` image signing tool. As a first step, we download binaries for these tools in a tools directory.

```
TEST_TMPDIR=$(mktemp -d "${PWD}/artifact-test-${1:+-$1}.XXXXXX")

TOOLSDIR=$(pwd)/hack/tools
mkdir -p ${TOOLSDIR}/bin

REGCLIENT=${TOOLSDIR}/bin/regctl
REGCLIENT_VERSION=v0.5.1
curl -Lo ${REGCLIENT} https://github.com/regclient/regclient/releases/download/${REGCLIENT_VERSION}/regctl-linux-amd64
chmod +x ${REGCLIENT}

COSIGN=${TOOLSDIR}/bin/cosign
COSIGN_VERSION=2.1.1
curl -Lo ${COSIGN} https://github.com/sigstore/cosign/releases/download/v${COSIGN_VERSION}/cosign-linux-amd64
chmod +x ${COSIGN}
```

Step 2: Deploy an OCI registry with referrers support (zot)

Next, we execute the following tasks to deploy an OCI registry:

- Copy a zot executable binary to the server.
- Create a basic configuration file for the zot server, specifying the local root directory, the network location, and the port number.
- Launch zot with the newly-created configuration file.
- Looping with a periodic cURL query, detect when zot is up and running.

```
ZOT=${TOOLSDIR}/bin/zot
ZOT_VERSION=2.0.0-rc6
curl -Lo ${ZOT} https://github.com/project-zot/zot/releases/download/v${ZOT_VERSION}/zot-linux-amd64-minimal
chmod +x ${ZOT}

ZOT_HOST=localhost
ZOT_PORT=8080

# function to start zot and test for readiness
function zot_setup() {
cat > $TEST_TMPDIR/zot-config.json << EOF
{
  "distSpecVersion": "1.1.0-dev",
  "storage": {
      "rootDirectory": "$TEST_TMPDIR/zot"
  },
  "http": {
      "address": "$ZOT_HOST",
      "port": "$ZOT_PORT"
  },
  "log": {
      "level": "error"
  }
}
EOF
  # start zot as a background task
  ${ZOT} serve $TEST_TMPDIR/zot-config.json &
  pid=$!
        # wait until service is up
  count=5
```

```
  up=0
  while [[ $count -gt 0 ]]; do
    if [ ! -d /proc/$pid ]; then
      echo "zot failed to start or died"
      exit 1
    fi
    up=1
    curl -f http://$ZOT_HOST:$ZOT_PORT/v2/ || up=0
    if [ $up -eq 1 ]; then break; fi
    sleep 1
    count=$((count - 1))
  done
  if [ $up -eq 0 ]; then
    echo "Timed out waiting for zot"
    exit 1
  fi
  # setup an OCI client
  ${REGCLIENT} registry set --tls=disabled $ZOT_HOST:$ZOT_PORT
}

# call the function to start zot
zot_setup
```

Step 3: Copy an image to the OCI registry

This step copies a busybox container image into the registry.

```
skopeo copy --format=oci --dest-tls-verify=false docker://busybox:latest docker://${ZOT_HOST}:${ZOT_PORT}/busybox:latest
```

Step 4: Copy a related artifact to the OCI registry

This step creates a simple artifact file and associates it with the busybox image in the registry.

```
cat > ${TEST_TMPDIR}/artifact.yaml << EOF
key:
  val: artifact
EOF
${REGCLIENT} artifact put --artifact-type application/yaml -f ${TEST_TMPDIR}/artifact.yaml --subject ${ZOT_HOST}:${ZOT_PORT}/busybox:latest
```

The `--subject` command flag associates the artifact file with the specified subject (in this case, the busybox image). If no subject is specified by the command, the artifact is considered independent and not associated with any existing image.

Step 5: Display the artifact tree

This step prints the artifact tree of the busybox image. The tree includes the artifact yaml file, showing the association of the artifact to the busybox image.

These script commands define REF0 as the artifact that was uploaded referring to the first container image.

```
REF0=$(${REGCLIENT} artifact tree --format '{{jsonPretty .}}' localhost:8080/busybox:latest | jq .referrer[0].reference.Digest)
REF0="${REF0:1:-1}"
```

The following example shows the command and its output:

```
$ regctl artifact tree localhost:8080/busybox:latest

Ref: localhost:8080/busybox:latest
Digest: sha256:9172c5f692f2c65e4f773448503b21dba2de6454bd159905c4bf6d83176e4ea3
Referrers:
   - sha256:9c0655368b10ca4b2ffe39e4dd261fb89df25a46ae92d6eb4e6e1792a451883e: application/yaml
```

The displayed artifact tree shows that the original image (`localhost:8080/busybox:latest`) has one direct referrer (`sha256:9c06...883e: application/yaml`), the artifact yaml file.

Step 6: Sign the image and artifact

This step creates a key pair for cosign in a separate directory, then uses cosign to sign both the image and the artifact file. Both signatures, like the artifact file itself, are associated with the busybox image.

```
# create a key pair in a different directory
pushd ${TEST_TMPDIR}
COSIGN_PASSWORD= ${COSIGN} generate-key-pair
```

```
popd
# sign the image
COSIGN_PASSWORD= COSIGN_OCI_EXPERIMENTAL=1 COSIGN_EXPERIMENTAL=1 ${COSIGN} sign -y --key ${TEST_TMPDIR}/cosign.key --registry-referrers-mode=oci-1-1 $
{ZOT_HOST}:${ZOT_PORT}/busybox:latest
# sign the artifact referring to the image
COSIGN_PASSWORD= COSIGN_OCI_EXPERIMENTAL=1 COSIGN_EXPERIMENTAL=1 ${COSIGN} sign -y --key ${TEST_TMPDIR}/cosign.key --registry-referrers-mode=oci-1-1 $
{ZOT_HOST}:${ZOT_PORT}/busybox@${REF0}
```

Step 7: Display the artifact tree

This step again prints the artifact tree, which should now show the artifact and the two signatures, all associated to the busybox image.

```
${REGCLIENT} artifact tree localhost:8080/busybox:latest
```

The following example shows the command and its output:

```
$ regctl artifact tree localhost:8080/busybox:latest

Ref: localhost:8080/busybox:latest
Digest: sha256:9172c5f692f2c65e4f773448503b21dba2de6454bd159905c4bf6d83176e4ea3
Referrers:
  - sha256:9c0655368b10ca4b2ffe39e4dd261fb89df25a46ae92d6eb4e6e1792a451883e: application/yaml
    Referrers:
      - sha256:06792b209137486442a2b804b2225c0014e3e238d363cdbea088bbd73207fb34: application/vnd.dev.cosign.artifact.sig.v1+json
  - sha256:995b6a78bf04a7a9676dac76b4598ccb645c17e30b02f294de9fdfa2f28eb7b2: application/vnd.dev.cosign.artifact.sig.v1+json
```

The displayed artifact tree shows that the original image now has two direct referrers — the artifact and the cosign signature of the original image. In addition, there is a second-level referrer — the cosign signature of the artifact, which is a referrer of the artifact file.

Step 8: End of demonstration

This step halts the zot registry server, ending the workflow demonstration.

```
# function for stopping zot after demonstration
function zot_teardown() {
  killall zot
}

# stop zot
zot_teardown
```

7.5.2 Reference: Full workflow script

Expand the text box below to view the entire workflow as a single executable shell script.

💡 To copy the script to the clipboard, click the copy icon that appears in the upper right corner of the expanded text box.

ck here to view the all-in-one script

```
#!/bin/bash -xe

TEST_TMPDIR=$(mktemp -d "${PWD}/artifact-test-${1:+-$1}.XXXXXX")

TOOLSDIR=$(pwd)/hack/tools
mkdir -p ${TOOLSDIR}/bin

REGCLIENT=${TOOLSDIR}/bin/regctl
REGCLIENT_VERSION=v0.5.1
curl -Lo ${REGCLIENT} https://github.com/regclient/regclient/releases/download/${REGCLIENT_VERSION}/regctl-linux-amd64
chmod +x ${REGCLIENT}

COSIGN=${TOOLSDIR}/bin/cosign
COSIGN_VERSION=2.1.1
curl -Lo ${COSIGN} https://github.com/sigstore/cosign/releases/download/v${COSIGN_VERSION}/cosign-linux-amd64
chmod +x ${COSIGN}

# OCI registry
ZOT=${TOOLSDIR}/bin/zot
ZOT_VERSION=2.0.0-rc6
curl -Lo ${ZOT} https://github.com/project-zot/zot/releases/download/v${ZOT_VERSION}/zot-linux-amd64-minimal
chmod +x ${ZOT}

ZOT_HOST=localhost
ZOT_PORT=8080

function zot_setup() {
cat > $TEST_TMPDIR/zot-config.json << EOF
{
  "distSpecVersion": "1.1.0-dev",
  "storage": {
      "rootDirectory": "$TEST_TMPDIR/zot"
  },
  "http": {
      "address": "$ZOT_HOST",
      "port": "$ZOT_PORT"
  },
  "log": {
      "level": "error"
  }
}
EOF
# start as a background task
${ZOT} serve $TEST_TMPDIR/zot-config.json &
pid=$!
        # wait until service is up
count=5
up=0
while [[ $count -gt 0 ]]; do
    if [ ! -d /proc/$pid ]; then
    echo "zot failed to start or died"
    exit 1
    fi
    up=1
    curl -f http://$ZOT_HOST:$ZOT_PORT/v2/ || up=0
    if [ $up -eq 1 ]; then break; fi
    sleep 1
    count=$((count - 1))
done
if [ $up -eq 0 ]; then
    echo "Timed out waiting for zot"
    exit 1
fi
# setup a OCI client
${REGCLIENT} registry set --tls=disabled $ZOT_HOST:$ZOT_PORT
}

# function for stopping zot after demonstration
function zot_teardown() {
killall zot
}

# call the function to start zot
zot_setup

# copy an image
skopeo copy --format=oci --dest-tls-verify=false docker://busybox:latest docker://${ZOT_HOST}:${ZOT_PORT}/busybox:latest

# copy an artifact referring to the above image
cat > ${TEST_TMPDIR}/artifact.yaml << EOF
key:
val: artifact
EOF
${REGCLIENT} artifact put --artifact-type application/yaml -f ${TEST_TMPDIR}/artifact.yaml --subject ${ZOT_HOST}:${ZOT_PORT}/busybox:latest
REF0=$(${REGCLIENT} artifact tree --format '{{jsonPretty .}}' localhost:8080/busybox:latest | jq .referrer[0].reference.Digest)
REF0="${REF0:1:-1}"

# create a key pair in a different directory
pushd ${TEST_TMPDIR}
COSIGN_PASSWORD= ${COSIGN} generate-key-pair
popd
# sign the image
COSIGN_PASSWORD= COSIGN_OCI_EXPERIMENTAL=1 COSIGN_EXPERIMENTAL=1 ${COSIGN} sign -y --key ${TEST_TMPDIR}/cosign.key --registry-referrers-mode=oci-1-1 ${ZOT_HOST}:$
{ZOT_PORT}/busybox:latest
# sign the artifact referring to the image
COSIGN_PASSWORD= COSIGN_OCI_EXPERIMENTAL=1 COSIGN_EXPERIMENTAL=1 ${COSIGN} sign -y --key ${TEST_TMPDIR}/cosign.key --registry-referrers-mode=oci-1-1 ${ZOT_HOST}:$
```

```
{ZOT_PORT}/busybox@${REF0}

# list the reference tree
${REGCLIENT} artifact tree localhost:8080/busybox:latest

# stop zot
zot_teardown
```

October 6, 2023

7.6 zot Security Posture

👉 An overview of zot build-time and runtime security hardening features, including:

- Build-time hardening such as PIE-mode builds
- Minimal-build option for smaller attack surface
- Open Source Security Foundation best practices for CI/CD
- Non-root deployment
- Robust authentication/authorization options

The zot project takes a defense-in-depth approach to security, applying industry-standard best practices at various stages. Recognizing that security hardening and product features are sometimes in conflict with each other, we also provide flexibility both at build and deployment time.

7.6.1 Build-time hardening

The following are the steps taken during build-time.

PIE build-mode

The zot binary is built with PIE build-mode enabled to take advantage of ASLR support in modern operating systems such as Linux ASLR. While zot is intended to be a long-running service (without frequent restarts), it prevents attackers from developing a generic attack that depends on predictable memory addresses *across* multiple zot deployments.

Conditional builds

Functionality in zot is broadly organized as a *core* Distribution Specification implementation and additional features as *extensions*. The rationale behind this approach is to minimize or control library dependencies that get included in the binary and consequently the attack surface.

We currently build and release two image flavors:

- **minimal**, which is a minimal Distribution Specification conformant registry, and
- **full**, which incorporates the *minimal* build **and** all extensions

The *minimal* flavor is for the security-minded and minimizes the number of dependencies and libraries. The *full* flavor is for the functionality-minded with the caveat that the attack surface of the binary is potentially broader. However by no means are these the only options. Our build (via the `Makefile`) provides the flexibility to pick and choose extensions in order to build a binary between *minimal* and *full*. For example,

```
make EXTENSIONS=search binary
```

produces a zot binary with only the *search* feature enabled.

CI/CD pipeline

zot CI/CD process attempts to align with the Open Source Security Foundation (OSSF) best practices guidelines to achieve high code quality.

CODE REVIEWS

zot is an open source project and all code submissions are open and transparent. Every pull request (PR) submitted to the project repository must be reviewed by the code owners. We have additional CI/CD workflows monitoring for unreviewed commits.

CI/CD CHECKS

All PRs must pass the full CI/CD pipeline checks including unit, functional, and integration tests, code quality and style checks, and performance regressions. In addition, all binaries produced are subjected to further security scans to detect any known vulnerabilities.

7.6.2 Runtime hardening

The following steps can be taken to harden a zot deployment.

Unprivileged runtime process

Running zot doesn't require *root* privileges. In fact, the recommended approach is to create a separate user/group ID for the zot process.

Authentication

All interactions with zot are over HTTP APIs, and `htpasswd`-based local authentication, LDAP, mutual TLS, and token-based authentication mechanisms are supported. We strongly recommend enabling a suitable mechanism for your deployment use case in order to prevent unauthorized access. See the provided authentication examples.

Access control

Following authentication, it is further possible to allow or deny actions by a user on a particular repository stored on the zot registry. See the provided access control examples.

7.6.3 Vulnerability scans

Apart from hardening the deployment itself, zot also supports security scanning of stored container images.

7.6.4 Reporting security issues

We understand that no software is perfect and in spite of our best efforts, security bugs may be found. Refer to our security policy for taking a responsible course of action when reporting security bugs.

December 21, 2022

7.7 Storage Planning with zot

👉 zot supports the following features to provide OCI standards-based, vendor-agnostic image storage:

- Local and remote file storage
- Inline deduplication and garbage collection
- Data scrubbing in background

7.7.1 Storage model

Data handling in zot revolves around two main principles: that data and APIs on the wire conform to the OCI Distribution Specification and that data on the disk conforms to the OCI Image Layout Specification. As a result, any client that is compliant with the Distribution Specification can read from or write to a zot registry. Furthermore, the actual storage is simply an OCI Image Layout. With only these two specification documents in hand, the entire data flow inside can be easily understood.

✏️ zot does not implement, support, or require any vendor-specific protocols, including that of Docker.

Hosting an OCI image layout

Because zot supports the OCI image layout, it can readily host and serve any directories holding a valid OCI image layout even when those directories have been created elsewhere. This property of zot is suitable for use cases in which container images are independently built, stored, and transferred, but later need to be served over the network.

7.7.2 Storage features

Exposing flexibility in storage capabilities is a key tenet for catering to the requirements of varied environments ranging from cloud to on-premises to IoT.

COMMIT

Most modern filesystems buffer and flush RAM data to disk after a delay. The purpose of this function is to improve performance at the cost of higher disk memory usage. In embedded devices such as Raspberry Pi, for example, where RAM may be very limited and at a premium, it is desirable to flush data to disk more frequently. The zot storage configuration exposes an option called `commit` which, when enabled, causes data writes to be committed to disk immediately. This option is disabled by default.

DEDUPLICATION

Deduplication is a storage space saving feature wherein only a single copy of specific content is maintained on disk while many different image manifests may hold references to that same content. The deduplication option (`dedupe`) is also available for supported cloud storage backends.

Upon startup, zot enforces the `dedupe` status on the existing storage. If the `dedupe` status upon startup is `true` , zot deduplicates all blobs found in storage, both local and remote. If the status upon startup is `false` , zot restores cloud storage blobs to their original state. There is no need for zot to restore local filesystem storage if hard links are used.

GARBAGE COLLECTION

After an image is deleted by deleting an image manifest, the corresponding blobs can be purged to free up space. However, since Distribution Specification APIs are not transactional between blob and manifest lifecycle, care must be taken so as not to put the storage in an inconsistent state. Garbage collection in zot is an inline feature meaning that it is **not** necessary to take the registry offline. See *Configuring garbage collection* for details.

SCRUB

The `scrub` function, available as an *extension*, makes it possible to ascertain data validity by computing hashes on blobs periodically and continuously so that any bit rot is caught and reported early.

7.7.3 Storage backends

The following types of storage backends are supported.

Local filesystem

zot can store and serve files from one or more local directories. A minimum of one root directory is required for local hosting, but additional hosted directories can be added. When accessed by HTTP APIs, all directories can appear as a single data store.

✏️ Remote filesystems that are mounted and accessible locally such as `NFS` or `fuse` are treated as local filesystems.

Remote filesystem

zot can also store data remotely in the cloud, using the storage APIs of the cloud service. Currently, zot supports only the AWS s3 storage service.

EXAMPLE: CONFIGURATION FOR REMOTE (S3) STORAGE

ck here to view a sample zot configuration for remote storage.

```
{
    "distSpecVersion": "1.0.1-dev",
    "storage": {
        "rootDirectory": "/tmp/zot",
        "dedupe": true,
        "storageDriver": {
            "name": "s3",
            "rootdirectory": "/zot",
            "region": "us-east-2",
            "bucket": "zot-storage",
            "secure": true,
            "skipverify": false
        },
        "cacheDriver": {
            "name": "dynamodb",
            "endpoint": "http://localhost:4566",
            "region": "us-east-2",
            "tableName": "MainTable"
        },
    },
    "http": {
        "address": "127.0.0.1",
        "port": "8080"
    },
    "log": {
        "level": "debug"
    }
}
```

7.7.4 Configuring zot storage

Filesystem storage is configured with the `storage` attribute in the zot configuration file, as shown in the following example.

```
  "storage":{
      "rootDirectory":"/tmp/zot",
      "commit": true,
      "dedupe": true,
      "gc": true,
      "gcDelay": "1h",
      "gcInterval": "24h"
  }
```

Configurable attributes

The following table lists the attributes of the `storage` configuration.

Attribute	Description
`rootDirectory`	Location of the images stored in the server file system.
`commit`	For faster performance, data written by zot is retained in memory before being periodically committed to disk by the operating system. To eliminate this retention time and cause data to be written to disk immediately, set to `true`. This prevents data loss but reduces performance.
`dedupe`	If the server filesystem supports hard links, you can optimize storage space by enabling inline deduplication of layers and blobs that are shared among multiple container images. Deduplication is enabled by default. Set to `false` to disable deduplication.
`gc`	When an image is deleted, either by tag or by reference, orphaned blobs can lead to wasted storage. Garbage collection (gc) is enabled by default to reclaim this space. Set to `false` to disable garbage collection.
`gcDelay`	(Optional) If garbage collection is enabled, causes it to run once after the specified delay time. The default is 1 hour. Requires the `gc` attribute to be `true`.
`gcInterval`	(Optional) If garbage collection is enabled, causes periodic collection at the specified interval. Must be set based on use cases and user workloads. If no value is specified, there is no periodic collection. Requires the `gc` attribute to be `true`.
`subpaths`	You can store and serve images from multiple filesystems, each with their own repository paths and settings. The following example shows three subpaths. `"storage":{ "subPaths": { "/a": { "rootDirectory": "/tmp/zot1", "dedupe": true, "gc": true }, "/b": { "rootDirectory": "/tmp/zot2", "dedupe": true }, "/c": { "rootDirectory": "/tmp/zot3", "dedupe": false } } }`
`storageDriver`	(Remote storage only) Contains settings for a remote storage service. See *Configuring remote storage with s3* for details.
`cacheDriver`	Specifies which database is used to store duplicate blobs when deduplication is enabled. See *Cache drivers* for details.

Configuring garbage collection

The zot configuration model allows for enabling and disabling garbage collection (`gc`) and specifying a periodic interval (`gcInterval`) for collection.

`gc`	`gcInterval`	Result
false	n/a	GC disabled
omitted	n/a	GC enabled with 1 hour interval (default)
true	omitted	GC enabled with 1 hour interval
true	0	GC runs only once
true	>0	GC enabled with specified interval

The configuration model also allows the configuration of a tunable delay (`gcDelay`), which can be set depending on client network speeds and the size of blobs. The `gcDelay` attribute causes collection to run once after the specified delay time. This attribute has a default value of one hour (`1h`).

7.7.5 Configuring remote storage with s3

To configure an Amazon Simple Storage Service (s3) bucket for zot, use the `storageDriver` attribute in the zot configuration file, as shown in the following example:

```
    "storage": {
        "rootDirectory": "/tmp/zot",
        "storageDriver": {
            "name": "s3",
            "region": "us-east-2",
            "bucket": "zot-storage",
            "secure": true,
            "skipverify": false,
            "accesskey": "<YOUR_ACCESS_KEY_ID>",
            "secretkey": "<YOUR_SECRET_ACCESS_KEY>"
        }
    }
```

The following table lists the attributes of `storageDriver` when configuring s3 for remote storage:

Attribute	Required	Description
name	yes	Name of storage driver. Only `s3` is supported for now.
accesskey	no	Your AWS Access Key. If you use IAM roles, omit to fetch temporary credentials from IAM.
secretkey	no	Your AWS Secret Key. If you use IAM roles, omit to fetch temporary credentials from IAM.
region	yes	The AWS region in which your bucket exists.
regionendpoint	no	Endpoint for S3 compatible storage services (Minio, etc).
forcepathstyle	no	To enable path-style addressing when the value is set to true. The default is true.
bucket	yes	The bucket name in which you want to store the registry’s data.
encrypt	no	Specifies whether the registry stores the image in encrypted format or not. A boolean value. The default is false.
keyid	no	Optional KMS key ID to use for encryption (encrypt must be true, or this parameter is ignored). The default is none.
secure	no	Indicates whether to use HTTPS instead of HTTP. A boolean value. The default is true.
skipverify	no	Skips TLS verification when the value is set to true. The default is false.
v4auth	no	Indicates whether the registry uses Version 4 of AWS’s authentication. The default is true.
chunksize	no	The S3 API requires multipart upload chunks to be at least 5MB. This value should be a number that is larger than 5 * 1024 * 1024.
multipartcopychunksize	no	Default chunk size for all but the last S3 Multipart Upload part when copying stored objects.
multipartcopymaxconcurrency	no	Max number of concurrent S3 Multipart Upload operations when copying stored objects.
multipartcopythresholdsize	no	Default object size above which S3 Multipart Upload will be used when copying stored objects.
rootdirectory	no	This is a prefix that is applied to all S3 keys to allow you to segment data in your bucket if necessary.
storageclass	no	The S3 storage class applied to each registry file. The default is STANDARD.
useragent	no	The User-Agent header value for S3 API operations.
usedualstack	no	Use AWS dual-stack API endpoints.
accelerate	no	Enable S3 Transfer Acceleration.
objectacl	no	The S3 Canned ACL for objects. The default value is “private”.
loglevel	no	The log level for the S3 client. The default value is off.

For more information, see the s3 storage driver docs.

s3 Credentials

In the s3 configuration file example, the s3 credentials were configured with the attributes `accesskey` and `secretkey.` As an alternative, you can omit these attributes from the configuration file and you can configure them using environment variables or a credential file.

- Environment variables

  zot looks for credentials in the following environment variables:

  ```
  AWS_ACCESS_KEY_ID
  AWS_SECRET_ACCESS_KEY
  AWS_SESSION_TOKEN (optional)
  ```

- Credential file

  A credential file is a plaintext file that contains your access keys, as shown in the following example.

  ```
  [default]
  aws_access_key_id = <YOUR_DEFAULT_ACCESS_KEY_ID>
  aws_secret_access_key = <YOUR_DEFAULT_SECRET_ACCESS_KEY>

  [test-account]
  aws_access_key_id = <YOUR_TEST_ACCESS_KEY_ID>
  aws_secret_access_key = <YOUR_TEST_SECRET_ACCESS_KEY>

  [prod-account]
  ; work profile
  aws_access_key_id = <YOUR_PROD_ACCESS_KEY_ID>
  aws_secret_access_key = <YOUR_PROD_SECRET_ACCESS_KEY>
  ```

  The `[default]` heading defines credentials for the default profile, which zot will use unless you configure it to use another profile. You can specify a profile using the `AWS_PROFILE` environment variable as in this example:

  ```
  AWS_PROFILE=test-account
  ```

  The credential file must be named `credentials.` The file must be located in the `.aws/` subdirectory in the home directory of the same server that is running your zot application.

For more details about specifying s3 credentials, see the AWS documentation.

S3 permissions scopes

The following AWS policy is required by zot for push and pull.

✏️ Replace S3_BUCKET_NAME with the name of your s3 bucket.

```
[AWS CONFIGURATION]
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "s3:ListBucket",
        "s3:GetBucketLocation",
        "s3:ListBucketMultipartUploads"
      ],
      "Resource": "arn:aws:s3:::<S3_BUCKET_NAME>"
    },
    {
      "Effect": "Allow",
      "Action": [
        "s3:PutObject",
        "s3:GetObject",
        "s3:DeleteObject",
        "s3:ListMultipartUploadParts",
        "s3:AbortMultipartUpload"
      ],
      "Resource": "arn:aws:s3:::<S3_BUCKET_NAME>/*"
    }
  ]
}
```

For more details about configuring AWS policies, see the AWS documentation.

7.7.6 Cache drivers

A cache driver is used to store duplicate blobs when `dedupe` is enabled. zot supports database caching using BoltDB as the cache driver for local filesystems and DynamoDB and Redis for remote filesystems.

BoltDB

If you don't specify a cache driver, zot defaults to BoltDB. BoltDB is stored either in zot's root directory or in the subpath root directory.

```
    "storage": {
        "rootDirectory": "/tmp/zot",
        "dedupe": true
    }
```

In this example, BoltDB can be found at /tmp/zot/cache.db.

⚠️ Because BoltDB does not provide concurrent access for writes, multiple instances/replicas of zot are not supported with a BoltDB configuration.

DynamoDB

To use DynamoDB as the cache driver, the following storage configuration must be present:

- `dedupe` is enabled
- `remoteCache` is enabled
- `cacheDriver` attribute is configured as in the following example:

```
    "storage": {
        "rootDirectory": "/tmp/zot",
        "dedupe": true,
        "remoteCache": true,
        "cacheDriver": {
            "name": "dynamodb",                  // driver name
            "endpoint": "http://localhost:4566", // aws endpoint
            "region": "us-east-2",               // aws region
            "cacheTablename": "ZotBlobTable"     // table to store deduped blobs
        }
    },
```

The AWS GO SDK loads additional configuration and credentials values from your environment variables, shared credentials, and shared configuration files.

If the search extension is enabled, additional parameters are required:

```
        "cacheDriver": {
            "name": "dynamodb",
            "endpoint": "http://localhost:4566",
            "region": "us-east-2",
            "cacheTablename": "ZotBlobTable",
            // used by auth
            "userDataTablename": "ZotUserDataTable",
            "apiKeyTablename": "ZotApiKeyDataTable",
            // used by search extension
            "repoMetaTablename": "ZotRepoMetadataTable",
            "imageMetaTablename": "ZotImageMetaTable",
            "repoBlobsInfoTablename": "ZotRepoBlobsInfoTable",
            "versionTablename": "ZotVersion"
        }
```

DYNAMODB PERMISSION SCOPES

The following AWS policy is required by zot for caching blobs.

✏️ Replace DYNAMODB_TABLE with the name of your table, which should be the value of `cacheTablename` in the zot configuration.

In this case, the AWS `Resource` value would be `arn:aws:dynamodb:*:*:table/ZotBlobTable`

```
[AWS CONFIGURATION]
{
```

```
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "dynamodb:CreateTable",
        "dynamodb:DescribeTable",
        "dynamodb:DeleteTable",
        "dynamodb:Scan",
        "dynamodb:BatchGetItem",
        "dynamodb:GetItem",
        "dynamodb:UpdateItem",
        "dynamodb:DeleteItem"
      ],
      "Resource": "arn:aws:dynamodb:*:*:table/<DYNAMODB_TABLE>"
    }
  ]
}
```

Note `dynamodb:DeleteTable` is used only in running the zot tests, should not be needed in production.

For more details about configuring AWS DynamoDB, see the AWS documentation.

Redis

Redis is an alternative to BoltDB (which cannot be shared by multiple zot instances) and DynamoDB (which requires access to AWS).

To use Redis as the cache driver, the following storage configuration must be present:

- `remoteCache` is enabled
- `cacheDriver` attribute is configured as in the following example:

```
    "storage": {
        "rootDirectory": "/tmp/zot",
        "remoteCache": true,
        "cacheDriver": {
            "name": "redis",
            "url": "redis://localhost:6379",
            "keyprefix": "zot"
        }
    },
```

The cache driver name `redis` selects the Redis driver implementation.

The key `keyprefix` is a string prepended to all Redis keys created by this zot instance. If no string is specified, the default value is `zot`. If multiple zot instances share the same Redis database and storage, `keyprefix` should be the same in all their configurations. If multiple zot instances share the same Redis database, but have different storage locations containing different images, `keyprefix` should be different in all their configurations.

The `url` string can contain query parameters for various settings, and it can be used for both Redis single instance and cluster configurations. More details on how `url` is parsed are available at: - https://github.com/redis/go-redis/blob/v9.7.0/options.go#L247 - https://github.com/redis/go-redis/blob/v9.7.0/osscluster.go#L144

At this time the library we import for `url` parsing does not support Redis sentry. For this reason we also support passing the parameters individually as keys in the same `cacheDriver` map. If and only if the `url` setting is missing, the other parameters are read from `cacheDriver`. The keys are the same as the attributes that would otherwise be included in the `url`.

In the case of a Redis Sentinel setup, you would need to add each key manually in the `cacheDriver` map and make sure to specify a `master_name` key, see https://github.com/redis/go-redis/blob/v9.7.0/universal.go#L240

REDIS CLUSTER CONFIGURATION

The following storage configuration is for a Redis cluster:

```
    "storage": {
        "rootDirectory": "/tmp/zot",
        "remoteCache": true,
        "cacheDriver": {
            "name": "redis",
            "url": "redis://user:password@host1:6379?addr=host2:6379&addr=host3:6379",
            "keyprefix": "zot"
        }
    }
```

ADD REDIS CONFIGURATION PARAMETERS

The following storage configuration contains all Redis parameters:

```
    "storage": {
        "rootDirectory": "/tmp/zot",
        "remoteCache": true,
        "cacheDriver": {
            "name": "redis",
            "keyprefix": "zot",
            "addr": ["host1:6379", "host2:6379", "host3:6379"],
            "client_name": "client",
            "db": 1,
            "protocol": 3,
            "username": "user1",
            "password": "pass1",
            "sentinel_username": "user2",
            "sentinel_password": "pass2",
            "max_retries": 3,
            "min_retry_backoff": "5s",
            "max_retry_backoff": "5s",
            "dial_timeout": "5s",
            "read_timeout": "5s",
            "write_timeout": "5s",
            "context_timeout_enabled": false,
            "pool_fifo": false,
            "pool_size": 3,
            "pool_timeout": "5s",
            "min_idle_conns": 1,
            "max_idle_conns": 3,
            "max_active_conns": 3,
            "conn_max_idle_time": "5s",
            "conn_max_lifetime": "5s",
            "max_redirects": 3,
            "read_only": false,
            "route_by_latency": false,
            "route_randomly": false,
            "master_name": "zotmeta",
            "disable_identity": false,
            "identity_suffix": "zotmeta",
            "unstable_resp3": false
        },
    }
```

⚠️ setting all of the parameters above for the same use case does not make sense, as some parameters for single instance, cluster and sentry configurations exclude one another.

7.7.7 Remote storage subpaths

As in the case with local filesystem storage, you can use multiple remote storage locations using the `subpath` attribute, as in the following example.

```
"subPaths": {
    "/a": {
        "rootDirectory": "/zot-a",
        "storageDriver": {
            "name": "s3",
            "region": "us-east-2",
            "bucket": "zot-storage",
            "secure": true,
            "skipverify": false
        }
    },
    "/b": {
       .
       .
       .
    }
}
```

The `subPaths` feature ties together several separate storage filesystems and backends behind the same HTTP API interface. In the example above, both repository paths "/a" and "/b" are exposed to clients. Content on these two paths can be hosted completely separately by different storage services, locations, or filesystems, with no difference to the user interface and no perceptible difference to the user experience. This is useful if one wants to serve existing OCI images from different backends or if storage can be expanded only by using different backing stores.

💡 zot also supports different storage drivers for each subpath.

May 23, 2025

7.8 Configuring zot Tag Retention Policies

👉 To optimize image storage, you can configure tag retention policies to remove images that are no longer needed.

Tag retention policies in zot can specify how many tags of a given repository to retain or how long to retain certain tags.

You can define tag retention policies that apply one or more of the following rules:

- Top <n> tags most recently pushed
- Top <n> tags most recently pulled
- Tags pushed in the past <n> hours
- Tags pulled in the past <n> hours
- Tags matching a regular expression (regex) pattern

7.8.1 Configuring retention policies

Retention policies are configured in the `storage` section of the zot configuration file under the `retention` attribute. One or more policies can be grouped under the `policies` attribute.

By default, if no retention policies are defined, all tags are retained.

⚠️ If at least one `keepTags` policy is defined for a repository, all tags not matching those policies are removed. To avoid unintended removals, we recommend defining a default policy, as described in Configuration notes.

Configuration example

The following example is a simple retention configuration with two policies:

- The first includes all available configuration attributes.
- The second acts as a default policy.

simple policy example

```
"storage": {
  "retention": {
    "dryRun": false,
    "delay": "24h",
    "policies": [
      {
        "repositories": ["infra/*", "tmp/**"],
        "deleteReferrers": false,
        "deleteUntagged": true,
        "keepTags": [{
          "patterns": ["v2.*", ".*-prod"],
          "mostRecentlyPushedCount": 10,
          "mostRecentlyPulledCount": 10,
          "pulledWithin": "720h",
          "pushedWithin": "720h"
        }]
      },
      {
        "keepTags": [{
          "patterns": [".*"]
        }]
      }
    ]
  }
}
```

Configurable attributes

The following table lists the attributes available in the retention policy configuration.

Attribute	Value	Description
dryRun	boolean	If `true`, will log a removal action without actually removing the image. Default is `false`.
delay	time	Remove untagged and referrers only if they are older than the specified <time> hours, such as 24h.
policies	list	A list of policies.
repositories	list	A list of glob patterns to match repositories.
deleteReferrers	boolean	If true, delete manifests with a missing Subject. Default is `false`.
deleteUntagged	boolean	If true, delete untagged manifests. Default is `true`.
keepTags	list	Criteria for tags to retain always.
mostRecentlyPushedCount	count	Retains the top <count> most recently pushed tags.
mostRecentlyPulledCount	count	Retains the top <count> most recently pulled tags.
pushedWithin	time	Retains the tags pushed during the last <time> hours, such as 24h.
pulledWithin	time	Retains the tags pulled during the last <time> hours, such as 24h.
patterns	regex	See Notes.

Configuration notes

- All image retention and garbage collection processing is made per repository, not per groups of repositories. The count of retained images in one repository doesn't impact retention for another repository.
- A repository will apply the first policy it matches.
- If a repository matches no policy, the repository and all its tags are retained.
- If at least one `keepTags` policy is defined for a repository, all tags not matching those policies are removed.
- If `keepTags` is present but empty, all tags are retained.
- In general, when multiple rules are configured, a tag is retained if it meets at least one rule.
- When multiple entries are configured under the same `keepTags` list, there is a logical OR applied between them.
- When a regex pattern is combined with one or more other rules inside a single `keepTags` entry, the rules apply only to those tags matching the regex. Given a `keepTags` entry, the retained tags are: `patterns` AND (`pulledWithin` OR `pushedWithin` OR `mostRecentlyPushedCount` OR `mostRecentlyPulledCount`).
- When you specify a regex pattern with no rules other than the default, all tags matching the pattern are retained.
- In the repositories list, a single asterisk (/*) matches all first-level items in the repository. A double asterisk (/**) matches all recursively.

⚠️ We recommend defining a default `keepTags` policy, such as the following example, as the last policy in the policy list. All tags that don't match the preceding policies will be retained by this default policy:

default policy example

```
{
  "keepTags": [{
      "patterns": [".*"]
    }]
}
```

7.8.2 Complete configuration file example

The following example shows the configuration of multiple retention policies in the context of a complete configuration file.

```
{
  "distSpecVersion": "1.1.0-dev",
  "storage": {
    "rootDirectory": "/tmp/zot",
    "gc": true,
    "gcDelay": "2h",
    "gcInterval": "1h",
    "retention": {
      "dryRun": false,
      "delay": "24h",
      "policies": [
        {
          "repositories": ["infra/*", "prod/*"],
          "deleteReferrers": false,
          "keepTags": [{
            "patterns": ["v2.*", ".*-prod"]
          },
          {
            "patterns": ["v3.*", ".*-prod"],
            "pulledWithin": "168h"
          }]
        },
        {
          "repositories": ["tmp/**"],
          "deleteReferrers": true,
          "deleteUntagged": true,
          "keepTags": [{
            "patterns": ["v1.*"],
            "pulledWithin": "168h",
            "pushedWithin": "168h"
          }]
        },
        {
          "repositories": ["**"],
          "deleteReferrers": true,
          "deleteUntagged": true,
          "keepTags": [{
            "mostRecentlyPushedCount": 10,
            "mostRecentlyPulledCount": 10,
            "pulledWithin": "720h",
            "pushedWithin": "720h"
          }]
        }
      ]
    },
    "subPaths": {
      "/a": {
        "rootDirectory": "/tmp/zot1",
        "dedupe": true,
        "retention": {
          "policies": [
            {
              "repositories": ["a/infra/*", "a/prod/*"],
              "deleteReferrers": false
            }
          ]
        }
      }
    }
  },
  "http": {
    "address": "127.0.0.1",
    "port": "8080"
  },
  "log": {
    "level": "debug"
  }
}
```

Given the configuration example above, we can make the following observations.

For repositories having names starting with `infra/` and `prod/`: - Artifacts referring to missing images will be retained. - Untagged images pushed more than 24h ago (`delay`) will be deleted by default, as `deleteUntagged` is not specified. - All tags matching regex pattern `v2.*` will be retained. - All tags matching regex pattern `.*-prod` will be retained, as they match the first `keepTags` entry, so their presence in the second entry is not necessary. - Tags matching regex pattern `v3.*` will not be deleted if they were pulled within `168h`. - All other tags will be deleted.

For repositories having names starting with `tmp/`: - Artifacts pushed more than 24h ago (`delay`) referring to missing images will be deleted. - Untagged images pushed more than 24h ago (`delay`) will be deleted. - Tags matching regex pattern `v1.*` will not be deleted if they were pulled within `168h` or pushed within `168h`. - All other tags will be deleted.

For repositories having names starting with `a/infra/` and `a/prod/`: - These repositories are under a separate subpath, with an entirely different retention configuration. - Artifacts referring to missing images will be retained. - Untagged images pushed more than 24h ago will be deleted by default, as `deleteUntagged` is not specified and the default value for `delay` is 24h.

For the rest of repositories, all of them matching `**`: - Artifacts pushed more than 24h ago (`delay`) referring to missing images will be deleted. - Untagged images pushed more than 24h ago (`delay`) will be deleted. - Tags will be retained if they were pulled within `720h` or pushed within `720h` or among the 10 most recently pushed images or among the 10 most recently pulled images. - All other tags will be deleted.

⚠️ `subPaths` are a separate feature with use cases outside the scope of this article. Do NOT use `subPaths` just for the purpose of configuring retention.

March 22, 2025

7.9 OCI Registry Mirroring With zot

👉 A `zot` registry can mirror one or more upstream OCI registries, including popular cloud registries such as Docker Hub and Google Container Registry (gcr.io).

A key use case for zot is to act as a mirror for upstream registries. If an upstream registry is OCI distribution-spec conformant for pulling images, you can use zot's `sync` feature to implement a downstream mirror, synchronizing OCI images and corresponding artifacts. Because synchronized images are stored in zot's local storage, registry mirroring allows for a fully distributed disconnected container image build pipeline. Container image operations terminate in local zot storage, which may reduce network latency and costs.

⚠️ Because zot is a OCI-only registry, any upstream image stored in the Docker image format is converted to OCI format when downloading to zot. In the conversion, some non-OCI attributes may be lost and the image digest will change. Pulling with @ will not work as expected. Signatures, for example, are removed due to the mismatch between the old and the new digests.

7.9.1 Mirroring modes

For mirroring an upstream registry, two common use cases are a fully mirrored or a pull through (on-demand) cache registry.

As with git, wherein every clone is a full repository, you can configure your local zot instance to be a fully mirrored OCI registry. For this mode, configure zot for synchronization by periodic polling, not on-demand. Zot copies and caches a full copy of every image on the upstream registry, updating the cache whenever polling discovers a change in content or image version at the upstream registry.

For a pull through cache mirrored registry, configure zot for on-demand synchronization. When an image is first requested from the local zot registry, the image is downloaded from the upstream registry and cached in local storage. Subsequent requests for the same image are served from zot's cache. Images that have not been requested are not downloaded. If a polling interval is also configured, zot periodically polls the upstream registry for changes, updating any cached images if changes are detected.

✏️ Because Docker Hub rate-limits pulls and does not support catalog listing, do not use polled mirroring with Docker Hub. Use only on-demand mirroring with Docker Hub.

7.9.2 Migrating or updating a registry using mirroring

Mirroring zot using the `sync` feature allows you to easily migrate a registry. In situations such as the following, zot mirroring provides an easy solution.

- Migrating an existing zot or non-zot registry to a new location.

  Provided that the source registry is OCI-compliant for image pulls, you can mirror the registry to a new zot registry, delete the old registry, and reroute network traffic to the new registry.

- Updating (or downgrading) a zot registry.

  To minimize downtime during an update, or to avoid any incompatibilities between zot releases that would preclude an in-place update, you can bring up a new zot registry with the desired release and then migrate from the existing registry.

To ensure a complete migration of the registry contents, set a polling interval in the configuration of the new zot registry and set `prefix` to `**`, as shown in this example:

```
{
  "urls": [
      "https://registry1:5000"
  ],
  "pollInterval": "12h",
  "onDemand": true,
  "content": [
      {
          "prefix": "**"
      }
  ]
}
```

7.9.3 Basic configuration for mirroring with sync

The `sync` feature of zot is an extension of the OCI-compliant registry implementation. You can configure the `sync` feature under the `extensions` section of the zot configuration file, as shown in this example:

```
"extensions": {
  "sync": {
    "credentialsFile": "./examples/sync-auth-filepath.json",
    "registries": [
      {
        "urls": [
          "https://registry1:5000"
        ],
        "onDemand": false,
        "pollInterval": "6h",
        "tlsVerify": true,
        "certDir": "/home/user/certs",
        "maxRetries": 3,
        "retryDelay": "5m",
        "onlySigned": true,
        "content": [
          {
            "prefix": "/repo2/repo",
            "tags": {
              "regex": "4.*",
              "semver": true
            }
            "destination": "/repo2",
            "stripPrefix": true
          }
        ]
      }
    ]
  }
}
```

The following table lists the configurable attributes for the `sync` feature:

Attribute	Description

credentialsFile	The location of a local file containing credentials for other registries, as in the following example:
	
	`{`
	`  "127.0.0.1:8080": {`
	`  "username": "user",`
	`  "password": "pass"`
	`  },`
	`  "registry2:5000": {`
	`  "username": "user2",`
	`  "password": "pass2"`
	`  }`
	`}`

urls	A list of one or more URLs to an upstream image registry. If the main URL fails, the sync process will try the next URLs in the listed order.
onDemand	• `false` : Pull all images not found in the local registry. • `true` : Pull any image not found in the local registry only when the image is requested by a user.
pollInterval	The period in seconds between polling of a remote registry. If no value is specified, no periodic polling will occur. If a value is set and the **content** attributes are configured, periodic synchronization is enabled and will run at the specified value. **Note:** Because Docker Hub rate-limits pulls and does not support catalog listing, do not use polled mirroring with Docker Hub. Use only onDemand mirroring with Docker Hub.
tlsVerify	• `false` : TLS will not be verified. • `true` : (Default) The TLS connection to the destination registry will be verified.
certDir	If a path is specified, use certificates (*.crt, *.cert, *.key files) at this path when connecting to the destination registry or daemon. If no path is specified, use the default certificates directory.
maxRetries	The maximum number of retries if an error occurs during either an on-demand or periodic synchronization. If no value is specified, no retries will occur.
retryDelay	The interval in seconds between retries. This attribute is mandatory when maxRetries is configured.
onlySigned	• `false` : Synchronize signed or unsigned images. • `true` : Synchronize only signed images (either notary or cosign).
content	The included attributes in this section specify which content will be pulled. If this section is not populated, periodic polling will not occur. The included attributes can also filter which on-demand images are pulled.
prefix	On the remote registry, the path from which images will be pulled. This path can be a string that exactly matches the remote path, or it can be a glob pattern. For example, the path can include a wildcard (*) or a recursive wildcard (**).
tags	The included attributes in this optional section specify how remote images will be selected for synchronization based on image tags.
tags.regex	Specifies a regular expression for matching image tags. Images whose tags do not match the expression are not pulled.
tags.semver	Specifies whether image tags are to be filtered by semantic versioning (semver) compliance. • `false` : Do not filter by semantic versioning. • `true` : Filter by semantic versioning.

Attribute	Description
destination	Specifies the local path in which pulled images are to be stored.
stripPrefix	Specifies whether the prefix path from the remote registry will be retained or replaced when the image is stored in the zot registry. • `false`: Retain the source prefix, append it to the destination path. • `true`: Remove the source prefix. **Note:** If the source prefix was specified with meta-characters (such as **), only the prefix segments that precede the meta-characters are removed. Any remaining path segments are appended to the destination path.

7.9.4 Configuration examples for mirroring

Example: Multiple repositories with polled mirroring

The following is an example of sync configuration for mirroring multiple repositories with polled mirroring.

```
"sync": {
  "enable": true,
  "credentialsFile": "./examples/sync-auth-filepath.json",
  "registries": [
    {
      "urls": ["https://registry1:5000"],
      "onDemand": false,
      "pollInterval": "6h",
      "tlsVerify": true,
      "certDir": "/home/user/certs",
      "maxRetries": 3,
      "retryDelay": "5m",
      "onlySigned": true,
      "preserveDigest": true,
      "content": [
        {
          "prefix": "/repo1/repo",
          "tags": {
            "regex": "4.*",
            "semver": true,
            "tags": {
                            "excludeRegex": ".*-(amd64|arm64)$"
                        }
          }
        },
        {
          "prefix": "/repo2/repo",
          "destination": "/repo2",
          "stripPrefix": true
        },
        {
          "prefix": "/repo3/repo"
        }
      ]
    }
  }
```

The configuration in this example will result in the following behavior:

- Only signed images (notation and cosign) are synchronized.
- The sync communication is secured using certificates in `certDir`.
- This registry synchronizes with upstream registry every 6 hours.
- This registry preserves upstream digests instead of converting them to OCI images.
- On-demand mirroring is disabled.
- Based on the content filtering options, this registry synchronizes these images:
- From /repo1/repo, images with tags that begin with "4." and are semver compliant but excluding some tag patterns
  Files are stored locally in /repo1/repo on localhost.
- From /repo2/repo, images with all tags.
  Because `stripPrefix` is enabled, files are stored locally in /repo2. For example, docker://upstream/repo2/repo:v1 is stored as docker://local/repo2:v1.
- From /repo3/repo, images with all tags.
  Files are stored locally in /repo3/repo.

Example: Multiple registries with on-demand mirroring

The following is an example of sync configuration for mirroring multiple registries with on-demand mirroring.

```
{
  "distSpecVersion": "1.0.1",
  "storage": {
    "rootDirectory": "/tmp/zot",
    "gc": true
  },
  "http": {
    "address": "0.0.0.0",
    "port": "8080"
  },
  "log": {
    "level": "debug"
  },
  "extensions": {
    "sync": {
      "enable": true,
      "registries": [
        {
          "urls": ["https://k8s.gcr.io"],
          "content": [
            {
              "prefix": "**",
              "destination": "/k8s-images"
            }
          ],
          "onDemand": true,
          "tlsVerify": true
        },
        {
          "urls": ["https://docker.io/library"],
          "content": [
            {
              "prefix": "**",
              "destination": "/docker-images"
            }
          ],
          "onDemand": true,
          "tlsVerify": true
        }
      ]
    }
  }
}
```

With this zot configuration, the sync behavior is as follows:

1. This initial user request for content from the zot registry:
   `skopeo copy --src-tls-verify=false docker://localhost:8080/docker-images/alpine <dest>`
   causes zot to synchronize the content with the docker.io registry:
   `docker.io/library/alpine:latest`
   to the zot registry:
   `localhost:8080/docker-images/alpine:latest`
   before delivering the content to the requestor at `<dest>`.
2. This initial user request for content from the zot registry:
   `skopeo copy --src-tls-verify=false docker://localhost:8080/k8s-images/kube-proxy:v1.19.2 <dest>`
   causes zot to synchronize the content with the gcr.io registry:
   `k8s.gcr.io/kube-proxy:v1.19.2`
   to the zot registry:
   `localhost:8080/k8s-images/kube-proxy:v1.19.2`
   before delivering the content to the requestor at `<dest>`.

You can use this command:
`curl http://localhost:8080/v2/_catalog`
to display the local repositories:

```
{
  "repositories":[
    "docker-images/alpine",
    "k8s-images/kube-proxy"
  ]
}
```

Example: Multiple registries with mixed mirroring modes

The following is an example of a zot configuration file for mirroring multiple upstream registries.

```
{
  "distSpecVersion": "1.1.0-dev",
  "storage": {
    "rootDirectory": "/tmp/zot"
  },
  "http": {
    "address": "127.0.0.1",
    "port": "8080"
  },
  "log": {
    "level": "debug"
  },
  "extensions": {
    "sync": {
      "enable": true,
      "credentialsFile": "./examples/sync-auth-filepath.json",
      "registries": [
        {
          "urls": [
            "https://registry1:5000"
          ],
          "onDemand": false,
          "pollInterval": "6h",
          "tlsVerify": true,
          "certDir": "/home/user/certs",
          "maxRetries": 3,
          "retryDelay": "5m",
          "onlySigned": true,
          "content": [
            {
              "prefix": "/repo1/repo",
              "tags": {
                "regex": "4.*",
                "semver": true
              }
            },
            {
              "prefix": "/repo1/repo",
              "destination": "/repo",
              "stripPrefix": true
            },
            {
              "prefix": "/repo2/repo"
            }
```

```
          ]
        },
        {
          "urls": [
            "https://registry2:5000",
            "https://registry3:5000"
          ],
          "pollInterval": "12h",
          "tlsVerify": false,
          "onDemand": false,
          "content": [
            {
              "prefix": "/repo2",
              "tags": {
                "semver": true
              }
            }
          ]
        },
        {
          "urls": [
            "https://docker.io/library"
          ],
          "onDemand": true,
          "tlsVerify": true,
          "maxRetries": 6,
          "retryDelay": "5m"
        }
      ]
    }
  }
}
```

Example: Support for subpaths in local storage

```
{
  "distSpecVersion": "1.0.1",
  "storage": {
    "subPaths":{
      "/kube-proxy":{
        "rootDirectory": "/tmp/kube-proxy",
        "dedupe": true,
        "gc": true
       }
     },
    "rootDirectory": "/tmp/zot",
    "gc": true
  },
  "http": {
    "address": "0.0.0.0",
    "port": "8080"
  },
  "log": {
    "level": "debug"
  },
  "extensions": {
    "sync": {
      "enable": true,
      "registries": [
        {
          "urls": ["https://k8s.gcr.io"],
          "content": [
            {
              "destination": "/kube-proxy",
              "prefix": "**"
            }
          ],
          "onDemand": true,
          "tlsVerify": true,
          "maxRetries": 2,
          "retryDelay": "5m"
        }
      ]
    }
  }
}
```

With this zot configuration, the sync behavior is as follows:

- This user request for content from the zot registry:
  `skopeo copy --src-tls-verify=false docker://localhost:8080/kube-proxy/kube-proxy:v1.19.2 <dest>`
  causes zot to synchronize the content with this remote registry:
  `k8s.gcr.io/kube-proxy:v1.19.2`
  to the zot registry:
  `localhost:8080/kube-proxy/kube-proxy:v1.19.2`
  before delivering the content to the requestor at `<dest>`.

You can use this command:
`curl http://localhost:8080/v2/_catalog`
to display the local repositories:

```
{
  "repositories":[
    "docker-images/alpine",
    "k8s-images/kube-proxy",
    "kube-proxy/kube-proxy"
  ]
}
```

In zot storage, the requested content is located here:
`/tmp/zot/kube-proxy/kube-proxy/kube-proxy/`
This subpath is created from the following path components:

- `/tmp/zot` is the `rootDirectory` of the zot registry
- `kube-proxy` is the `rootDirectory` of the storage subpath
- `kube-proxy` is the sync `destination` parameter
- `kube-proxy` is the repository name

Example: Support for AWS ECR

This is an example configuration demonstrating how to use the sync extension with Amazon ECR (Elastic Container Registry) credential helper. The configuration enables zot to synchronize container images from an ECR registry.

```
"extensions": {
       "sync": {
           "credentialsFile": "",
           "DownloadDir": "/tmp/zot",
           "registries": [
               {
                   "urls": [
                       "https://ACCOUNTID.dkr.ecr.REGION.amazonaws.com"
                   ],
                   "onDemand": true,
                   "maxRetries": 5,
                   "retryDelay": "2m",
                   "credentialHelper": "ecr"
               }
           ]
       }
   }
```

May 23, 2025

7.10 zot Clustering

👉 High availability of the zot registry is supported by the following features:

- Stateless zot instances to simplify scale out
- Shared remote storage
- Bare-metal and Kubernetes deployments

To ensure high availability of the registry, zot supports a clustering scheme with stateless zot instances/replicas fronted by a load balancer and a shared remote backend storage. This scheme allows the registry service to remain available even if a few replicas fail or become unavailable. Load balancing across many zot replicas can also increase aggregate network throughput.

✏️ Beginning with zot release v2.1.0, you can design a highly scalable cluster that does not require configuring the load balancer to direct repository queries to specific zot instances within the cluster. See Scale-out clustering. Scale-out clustering is the preferred method if you are running v2.1.0 or later.

Clustering is supported in both bare-metal and Kubernetes environments.

✏️ The remote backend storage must be S3 API-compatible.

7.10.1 Bare-metal deployment

Prerequisites

- A highly-available load balancer such as HAProxy configured to direct traffic to zot replicas
- Multiple zot replicas as `systemd` services hosted on multiple hosts or VMs
- AWS S3 API-compatible remote backend storage

7.10.2 Kubernetes deployment

Prerequisites

- A zot Kubernetes Deployment with required number of replicas
- AWS S3 API-compatible remote backend storage.
- A zot Kubernetes Service
- A zot Kubernetes Ingress Gateway if the service needs to be exposed outside

7.10.3 Implementing stateless zot

zot maintains two types of durable state:

- the actual image data itself
- the image metadata in the registry's cache

In a stateless clustering scheme, the image data is stored in the remote storage backend and the registry cache is disabled by turning off deduplication.

7.10.4 Ecosystem tools

The OCI Distribution Specification imposes certain rules about the HTTP URI paths to which various ecosystem tools must conform. Consider these rules when setting the HTTP prefixes during load balancing and ingress gateway configuration.

7.10.5 Examples

Clustering is supported by using multiple stateless zot replicas with shared S3 storage and an HAProxy (with sticky session) load balancing traffic to the replicas. Each replica is responsible for one or more repositories.

HAProxy configuration

ck here to view a sample HAProxy configuration.

```
global
        log /dev/log    local0
        log /dev/log    local1 notice
        chroot /var/lib/haproxy
        maxconn 2000
        stats socket /run/haproxy/admin.sock mode 660 level admin expose-fd listeners
        stats timeout 30s
        user haproxy
        group haproxy
        daemon

        # Default SSL material locations
        ca-base /etc/ssl/certs
        crt-base /etc/ssl/private

        # See: https://ssl-config.mozilla.org/#server=haproxy&server-version=2.0.3&config=intermediate
        ssl-default-bind-ciphers ECDHE-ECDSA-AES128-GCM-SHA256:ECDHE-RSA-AES128-GCM-SHA256:ECDHE-ECDSA-AES256-GCM-SHA384:ECDHE-RSA-AES256-GCM-SHA384:ECDHE-ECDSA-
CHACHA20-POLY1305:ECDHE-RSA-CHACHA20-POLY1305:DHE-RSA-AES128-GCM-SHA256:DHE-RSA-AES256-GCM-SHA384
        ssl-default-bind-ciphersuites TLS_AES_128_GCM_SHA256:TLS_AES_256_GCM_SHA384:TLS_CHACHA20_POLY1305_SHA256
        ssl-default-bind-options ssl-min-ver TLSv1.2 no-tls-tickets

defaults
        log     global
        mode    http
        option  httplog
        option  dontlognull
        timeout connect 5000
        timeout client  50000
        timeout server  50000
        errorfile 400 /etc/haproxy/errors/400.http
        errorfile 403 /etc/haproxy/errors/403.http
        errorfile 408 /etc/haproxy/errors/408.http
        errorfile 500 /etc/haproxy/errors/500.http
        errorfile 502 /etc/haproxy/errors/502.http
        errorfile 503 /etc/haproxy/errors/503.http
        errorfile 504 /etc/haproxy/errors/504.http

frontend zot
    bind *:8080
    mode http
    use_backend zot-instance1 if { path_beg /v2/repo1/ }
    use_backend zot-instance2 if { path_beg /v2/repo2/ }
    use_backend zot-instance3 if { path_beg /v2/repo3/ }
    default_backend zot-cluster

backend zot-cluster
    mode http
    balance roundrobin
    cookie SERVER insert indirect nocache
    server zot-server1 127.0.0.1:9000 check cookie zot-server1
    server zot-server2 127.0.0.2:9000 check cookie zot-server2
    server zot-server3 127.0.0.3:9000 check cookie zot-server3

backend zot-instance1
    server zot-server1 127.0.0.1:9000 check maxconn 30

backend zot-instance2
    server zot-server2 127.0.0.2:9000 check maxconn 30

backend zot-instance3
    server zot-server3 127.0.0.3:9000 check maxconn 30
```

zot S3 configuration with DynamoDB

ck here to view a sample zot configuration for S3 and DynamoDB.

```
{
    "distSpecVersion": "1.0.1-dev",
    "storage": {
        "rootDirectory": "/tmp/zot",
        "dedupe": false,
        "remoteCache": true,
        "storageDriver": {
            "name": "s3",
            "rootdirectory": "/zot",
            "region": "us-east-2",
            "bucket": "zot-storage",
            "secure": true,
            "skipverify": false
        },
        "cacheDriver": {
            "name": "dynamodb",
            "endpoint": "http://localhost:4566",
            "region": "us-east-2",
            "tableName": "MainTable"
        }
    },
    "http": {
        "address": "127.0.0.1",
        "port": "8080"
    },
    "log": {
        "level": "debug"
    }
}
```

zot S3 configuration with Redis

Multiple zot instances can share the same S3 `storageDriver` and Redis `cacheDriver` configurations. The Redis server, DB, and `keyprefix` must match for all zot instances.

While the Redis `cacheDriver` implementation does support local storage, zot clustering with Redis is only supported for S3.

ck here to view a sample zot configuration for S3 and Redis.

```
{
    "distSpecVersion": "1.0.1-dev",
    "storage": {
        "rootDirectory": "/tmp/zot",
        "dedupe": false,
        "remoteCache": true,
        "storageDriver": {
            "name": "s3",
            "rootdirectory": "/zot",
            "region": "us-east-2",
            "bucket": "zot-storage",
            "secure": true,
            "skipverify": false
        },
        "cacheDriver": {
            "name": "redis",
            "url": "redis://host:6379",
            "keyprefix": "zot"
        }
    },
    "http": {
        "address": "127.0.0.1",
        "port": "8080"
    },
    "log": {
        "level": "debug"
    }
}
```

February 11, 2025

7.11 Scale-out clustering

👉 A cluster of zot instances can be easily scaled with no repo-specific intelligence in the load balancing scheme, using:

- Stateless zot instances to simplify scale out
- Shared remote storage
- zot release v2.1.0 or later

Beginning with zot release v2.1.0, a new "scale-out" architecture greatly reduces the configuration required when deploying large numbers of zot instances. As before, multiple identical zot instances run simultaneously using the same shared reliable storage, but with improved scale and performance in large deployments. A highly scalable cluster can be architected by automatically sharding based on repository name so that each zot instance is responsible for a subset of repositories.

In a cloud deployment, the shared backend storage (such as AWS S3) and metadata storage (such as DynamoDB) can also be easily scaled along with the zot instances.

✏️ For high availability clustering with earlier zot releases, see zot Clustering.

7.11.1 Prerequisites

For easy scaling of instances (replicas), the following conditions must be met:

- All zot replicas must be running zot release v2.1.0 (or later) with identical configurations.
- All zot replicas in the cluster use remote storage at a single shared S3 backend. There is no local caching in the zot replicas.
- Each zot replica in the cluster has its own IP address, but all replicas use the same port number.

7.11.2 How it works

Each repo is served by one zot replica, and that replica is solely responsible for serving all images of that repo. A repo in storage can be written to only by the zot replica responsible for that repo.

When a zot replica in the cluster receives an image push or pull request for a repo, the receiving replica hashes the repo path and consults a hash table to determine which replica is responsible for the repo.

- If the hash indicates that another replica is responsible, the receiving replica forwards the request to the responsible replica and then acts as a proxy, returning the response to the requestor.
- If the hash indicates that the current (receiving) replica is responsible, the request is handled locally.

✏️ For better resistance to collisions and preimage attacks, zot uses SipHash as the hashing algorithm.

Either of the following two schemes can be used to reach the cluster.

Using a single entry point load balancer

When a single entry point load balancer such as HAProxy is deployed, the number of zot replicas can easily be expanded by simply adding the IP addresses of the new replicas in the load balancer configuration.

When the load balancer receives an image push or pull request for a repo, it forwards the request to any replica in the cluster. No repo-specific programming of the load balancer is needed because the load balancer does not need to know which replica owns which repo. The replicas themselves can determine this.

Using DNS-based load balancing

Because the scale-out architecture greatly simplifies the role of the load balancer, it may be possible to eliminate the load balancer entirely. A scheme such as DNS-based routing can be implemented, exposing the zot replicas directly to the clients.

7.11.3 Configuration examples

In these examples, clustering is supported by using multiple stateless zot replicas with shared S3 storage and an HAProxy (with sticky session) load balancer forwarding traffic to the replicas.

Cluster member configuration

In the replica configuration, each replica must have a list of its peers configured in the "members" section of the JSON structure. This is a list of reachable addresses or hostnames. Each replica owns one of these addresses.

The replica must also have a hash key for hashing the repo path of the image request and a TLS certificate for authenticating with its peers.

ck here to view a sample cluster configuration for each replica. See the "cluster" section in the JSON structure.

```
{
  "distSpecVersion": "1.1.0",
  "storage": {
    "rootDirectory": "/tmp/zot",
    "dedupe": false,
    "remoteCache": true,
    "storageDriver": {
      "name": "s3",
      "rootdirectory": "/zot",
      "region": "us-east-1",
      "regionendpoint": "localhost:4566",
      "bucket": "zot-storage",
      "secure": false,
      "skipverify": false
    },
    "cacheDriver": {
      "name": "dynamodb",
      "endpoint": "http://localhost:4566",
      "region": "us-east-1",
      "cacheTablename": "ZotBlobTable",
      "repoMetaTablename": "ZotRepoMetadataTable",
      "imageMetaTablename": "ZotImageMetaTable",
      "repoBlobsInfoTablename": "ZotRepoBlobsInfoTable",
      "userDataTablename": "ZotUserDataTable",
      "versionTablename": "ZotVersion",
      "apiKeyTablename": "ZotApiKeyTable"
    }
  },
  "http": {
    "address": "0.0.0.0",
    "port": "9000",
    "tls": {
      "cert": "test/data/server.cert",
      "key": "test/data/server.key"
    }
  },
  "log": {
    "level": "debug"
  },
  "cluster": {
    "members": [
      "zot-server1:9000",
      "zot-server2:9000",
      "zot-server3:9000"
    ],
    "hashKey": "loremipsumdolors",
    "tls": {
      "cacert": "test/data/ca.crt"
    }
  }
}
```

HAProxy configuration

The HAProxy load balancer uses a simple round-robin balancing scheme and delivers a cookie to the requestor to maintain a sticky session connection to the assigned replica.

ck here to view a sample HAProxy configuration.

```
global
        log /dev/log    local0
        log /dev/log    local1 notice
        chroot /var/lib/haproxy
        maxconn 2000
        stats timeout 30s

defaults
        log     global
        mode    tcp
        option  tcplog
        option  dontlognull
        timeout connect 5000
        timeout client  50000
        timeout server  50000

frontend zot
    bind *:8080
    default_backend zot-cluster

backend zot-cluster
    mode http
    balance roundrobin
    cookie SERVER insert indirect nocache
    server zot-server1 127.0.0.1:9000 check cookie zot-server1
    server zot-server2 127.0.0.2:9000 check cookie zot-server2
    server zot-server3 127.0.0.3:9000 check cookie zot-server3
```

7.11.4 When a replica fails

The scale-out clustering scheme described in this article is not self-healing when a replica fails. In case of a replica failure, only those repositories that are mapped to the failed replica are affected. If the error is not transient, the cluster must be resized and restarted to exclude that replica.

💡 With an HAProxy load balancer, we recommend implementing an HAProxy circuit breaker to monitor and protect the cluster.

7.11.5 CVE repository in a zot cluster environment

CVE scanning is not supported for cloud deployments. In the scale-out clustering scheme described in this article, CVE scanning is disabled. In this case, we recommend implementing a CVE repository with a zot instance outside of the cluster using a local disk for storage and Trivy as the detection engine.

7.11.6 Registry sync

The sync feature of zot, either on demand or periodic, is compatible with scale-out clustering. In this case, the repo names are hashed to a particular replica and only that replica will perform the sync.

May 31, 2024

7.12 Deploying a Highly Available zot Registry

👉 A highly available zot registry can be easily implemented using zot's registry synchronization feature.

In the zot configuration, the `sync` extension allows a zot instance to mirror another zot instance with various container image download policies, including on-demand and periodic downloads. You can use the zot `sync` function combined with a load balancer such as HAProxy to implement a highly available registry.

Two failover configurations are possible:

- Active/standby

  Registry requests are sent by the load balancer to the active zot instance, while a standby instance mirrors the active. If the load balancer detects a failure of the active instance, it then sends requests to the standby instance.

- Active/active

  Registry requests are load-balanced between two zot instances, each of which mirrors the other.

✏️ The highly available zot registry described in this article differs from zot clustering. Although zot clustering provides a level of high availability, the instances share common storage, whose failure would affect all instances. In the method described in this article, each instance has its own storage, providing an additional level of safety.

For details of configuring the `sync` extension, see OCI Registry Mirroring With zot.

7.12.1 Configuring an active/standby registry

An active/standby zot registry can be implemented between two zot instances by configuring the `sync` extension in the standby instance to mirror the other instance. In this scheme:

- a load balancer such as HAProxy is deployed for active/passive load balancing of the zot instances
- each zot instance is configured as a standalone registry with its own storage
- the standby zot instance has its `sync` extension enabled to periodically synchronize with (mirror) the active instance

With periodic synchronization, a window of failure exists between synchronization actions. For example, if an image is posted to the active instance soon after the standby has synchronized with the active, and then the active fails, the standby will not have the new image. To minimize this exposure, we recommend keeping the synchronization period as small as practical.

7.12.2 Configuring an active/active registry

An active/active zot registry can be implemented between two zot instances by configuring the `sync` extension in each instance to point to the other instance. In this scheme:

- a load balancer such as HAProxy or a DNS-based routing scheme is deployed for load balancing between zot instances
- path-based routing must be implemented
- each zot instance is configured as a standalone registry with its own storage
- each zot instance has its `sync` extension enabled to periodically synchronize with the other instance

With periodic synchronization, a window of failure exists between synchronization actions. For example, if an image is posted to instance A soon after instance B has synchronized with instance A, and then instance A fails, instance B will not have the new image. To minimize this exposure, we recommend keeping the synchronization period as small as practical.

June 24, 2024

7.13 Monitoring the registry

👉 **zot** supports a range of monitoring tools including logging, metrics, and benchmarking.

The following sections describe how to configure logging and monitoring with zot. You can use zot's benchmarking tool to test your configuration and deployment, as described in Benchmarking zot with zb.

7.13.1 Logging

Logging for zot operations is configured with the `log` attribute in the configuration file, as shown in the following example.

```
"log":{
  "level":"debug",
  "output":"/tmp/zot.log",
  "audit": "/tmp/zot-audit.log"
}
```

The following table lists the configurable attributes.

Attribute	Description
`level`	The minimum level for logged events. The levels are: `panic`, `fatal`, `error`, `warn`, `info`, `debug`, and `trace`.
`output`	The filesystem path for the log output file. The default is `stdout`.
`audit`	(Optional) If a filesystem path is specified for audit logging, an audit log is enabled and will be stored at the specified path.

7.13.2 Metrics

The available methods for collecting metrics varies depending on whether your zot installation is a minimal (distribution-spec-only) image or a full image including extensions.

Enabling metrics for a full zot image with extensions

Add the `metrics` attribute under `extensions` in the configuration file to enable and configure metrics, as shown in the following example.

```
"extensions": {
    "metrics": {
        "enable": true,
        "prometheus": {
            "path": "/metrics"
        }
    }
}
```

The following table lists the configurable attributes for metrics collection.

Attribute	Description
`enable`	If this attribute is missing, metrics collection is enabled by default. Metrics collection can be disabled by setting this attribute to `false`.
`prometheus`	Attributes under `prometheus` contain configuration settings for the Prometheus node exporter.
`path`	The server path on which metrics will be exposed.

Collecting metrics from a minimal zot image using a node exporter

Although a minimal zot image does not contain a node exporter, it exposes internal metrics in a Prometheus format for collection by a separate node exporter tool such as zxp. The zot companion binary `zxp` is a node exporter that can be deployed with a minimal zot image in order to scrape metrics from the zot server.

Metrics are automatically enabled in the zot server upon first scrape from the node exporter and the metrics are automatically disabled when the node exporter has not performed any scraping for some period. No extra zot configuration is needed for this behavior.

You can download the zxp executable binary for your server platform and architecture under "Assets" on the GitHub zot releases page.

The binary image is named using the target platform and architecture. For example, the binary for an Intel-based MacOS server is `zxp-darwin-amd64`. To configure the zxp example image, run this command:

```
zxp-darwin-amd64 config zxp-config-file
```

💡 For convenience, you can rename the binary image file to simply `zxp`.

💡 A sample Dockerfile for zxp is available at Dockerfile-zxp.

The configuration file of zxp contains connection details for the zot server from which it will scrape metrics. The following JSON structure is an example of the `zxp-config-file` contents:

```
{
    "Server": {
        "protocol": "http",
        "host": "127.0.0.1",
        "port": "8080"
    },
    "Exporter": {
        "port": "8081",
        "log": {
            "level": "debug"
        }
    }
}
```

💡 The zxp module does not have Prometheus integration.

The zxp module is not needed with a full zot image.

Authorization

The server supports configuring authorization for the endpoint used for metrics, in both the zot minimal image, and the image including the metrics extension. For more details see the authentication and authorization article.

May 13, 2025

7.14 Using GraphQL for Enhanced Searches

👉 A GraphQL backend server within zot's registry search engine provides efficient and enhanced search capabilities. You can submit a GraphQL structured query as an API call or you can use a browser to access the GraphQL Playground, an interactive graphical environment for GraphQL queries.

7.14.1 How to use GraphQL for search queries

GraphQL is a query language for APIs. A GraphQL server, as implemented in zot's registry search engine, executes GraphQL queries that match schema recognized by the server. In response, the server returns a structure containing the requested information. The schema currently recognized by zot are those that correspond to the queries listed in What GraphQL queries are supported.

💡 To learn more about GraphQL, see these resources:

- Introduction to GraphQL
- The Fullstack Tutorial for GraphQL

To perform a search, compose a GraphQL structured query for a specific search and deliver it to zot using one of the methods described in the following sections.

For examples of GraphQL queries supported in zot, see Examples of zot searches using GraphQL.

Using the search API directly

You can submit a GraphQL structured query as the HTML data payload in a direct API call using a shell tool such as cURL or Postman. GraphQL queries are sent to the zot `search` extension API:

```
/v2/_zot/ext/search
```

The following example submits a zot GraphQL query using cURL:

```
curl -X POST -H "Content-Type: application/json" --data '{ "query": "{ ImageListForCVE (id:\"CVE-2002-1119\") { Results { Name Tags } } }" }' http://
localhost:8080/v2/_zot/ext/search
```

The reply to your query is returned as a JSON payload in the HTML response.

Using the GraphQL Playground

✏️ The GraphQL Playground feature is available only in a `binary-debug` zot build or when the zot registry was built with the `debug` extension label.

The GraphQL Playground is an interactive graphical web interface for GraphQL hosted by the zot registry server.

The GraphQL Playground is reachable by a browser at the following zot API:

```
/v2/_zot/debug/graphql-playground#
```

For example, if your zot server is located at `http://localhost:8080`, the GraphQL Playground can be accessed by your browser at this URL:

```
http://localhost:8080/v2/_zot/debug/graphql-playground#
```

In the GraphQL Playground, you can construct and submit a query structure and you can view the query response in a graphical environment in the browser. You can also inspect the schema.

7.14.2 What GraphQL queries are supported

Supported queries	graphQL query	Input	Output	Description
Search images by digest	ImageListForDigest	digest	image list	Searches all repositories in the registry and returns list of images that matches given digest (manifest, config or layers)
Search images affected by a given CVE id	CVEListForImage	CVE id	image list	Searches the entire registry and returns list of images affected by given CVE
List CVEs for a given image	CVEListForImage	image	CVE list	Scans given image and returns list of CVEs affecting the image
List images not affected by a given CVE id	ImagesListWithCVEFixed	repository, CVE id	image list	Scans all images in a given repository and returns list of latest (by date) images not affected by the given CVE
Latest image from all repos	RepoListWithNewestImage	none	repo summary list	Returns the latest image from all the repos in the registry
List all images with expanded information for a given repository	ExpandedRepoInfo	repository	repo info	List expanded repo information for all images in repo, alongside a repo summary
All images in repo	ImageList	repository	image list	Returns all images in the specified repo
Global search	GlobalSearch	query	image summary / repo summary / layer summary	Will return what's requested in the query argument
Derived image list	DerivedImageList	image	image list	Returns a list of images that depend on the image specified in the argument
Base image list	BaseImageList	image	image list	Returns a list of images that the specified image depends on
Get details of a specific image	Image	image	image summary	Returns details about a specific image
Get referrers of a specific image	Referrers	repo, digest, type	artifact manifests	Returns a list of artifacts of given

Supported queries	graphQL query	Input	Output	Description
				type referring to a specific repo and digests

7.14.3 Examples of zot searches using GraphQL

✏️ These examples show only the GraphQL query without details on how to send them to a server. See How to use GraphQL for search queries.

The query structures shown in these examples request all fields allowed by the schema for the particular query type. The schema allows you to request a subset of the data, if desired, omitting any fields that you don't need.

List CVEs of given image

Sample request

```
{
  CVEListForImage(
    image: "alpine:3.17"
    requestedPage: {limit: 1, offset:1, sortBy: SEVERITY}
  ) {
    Tag
    Page {
      TotalCount
      ItemCount
    }
    CVEList {
      Id
      Title
      Description
      Severity
      PackageList {
        Name
        InstalledVersion
        FixedVersion
      }
    }
  }
}
```

Sample response

```
{
  "data": {
    "CVEListForImage": {
      "Tag": "3.17",
      "Page": {
        "TotalCount": 9,
        "ItemCount": 1
      },
      "CVEList": [
        {
          "Id": "CVE-2023-5363",
          "Title": "openssl: Incorrect cipher key and IV length processing",
          "Description": "Issue summary: A bug has been identified in the processing of key and\ninitialisation vector (IV) lengths.  This can lead to
potential truncation\nor overruns during the initialisation of some symmetric ciphers.\n\nImpact summary: A truncation in the IV can result in non-uniqueness,
\nwhich could result in loss of confidentiality for some cipher modes.\n\nWhen calling EVP_EncryptInit_ex2(), EVP_DecryptInit_ex2() or\nEVP_CipherInit_ex2()
the provided OSSL_PARAM array is processed after\nthe key and IV have been established.  Any alterations to the key length,\nvia the \"keylen\" parameter or
the IV length, via the \"ivlen\" parameter,\nwithin the OSSL_PARAM array will not take effect as intended, potentially\ncausing truncation or overreading of
these values.  The following ciphers\nand cipher modes are impacted: RC2, RC4, RC5, CCM, GCM and OCB.\n\nFor the CCM, GCM and OCB cipher modes, truncation of
the IV can result in\nloss of confidentiality.  For example, when following NIST's SP 800-38D\nsection 8.2.1 guidance for constructing a deterministic IV for
AES in\nGCM mode, truncation of the counter portion could lead to IV reuse.\n\nBoth truncations and overruns of the key and overruns of the IV will\nproduce
incorrect results and could, in some cases, trigger a memory\nexception.  However, these issues are not currently assessed as security\ncritical.\n\nChanging
the key and/or IV lengths is not considered to be a common operation\nand the vulnerable API was recently introduced. Furthermore it is likely
that\napplication developers will have spotted this problem during testing since\ndecryption would fail unless both peers in the communication were
similarly\nvulnerable. For these reasons we expect the probability of an application being\nvulnerable to this to be quite low. However if an application is
vulnerable then\nthis issue is considered very serious. For these reasons we have assessed this\nissue as Moderate severity overall.\n\nThe OpenSSL SSL/TLS
implementation is not affected by this issue.\n\nThe OpenSSL 3.0 and 3.1 FIPS providers are not affected by this because\nthe issue lies outside of the FIPS
provider boundary.\n\nOpenSSL 3.1 and 3.0 are vulnerable to this issue.",
          "Severity": "HIGH",
          "PackageList": [
            {
              "Name": "libcrypto3",
              "InstalledVersion": "3.0.8-r0",
              "FixedVersion": "3.0.12-r0"
            },
            {
              "Name": "libssl3",
```

```
              "InstalledVersion": "3.0.8-r0",
              "FixedVersion": "3.0.12-r0"
            }
          ]
        }
      ]
    }
  }
}
```

Search images affected by a given CVE id

Sample request

```
{
  ImageListForCVE(id: "CVE-2023-0464") {
    Results{
      RepoName
      Tag
      Digest
      LastUpdated
      IsSigned
      Size
      Vendor
      DownloadCount
      Licenses
      Title
      Manifests {
        Digest
        ConfigDigest
        Platform {
          Os
          Arch
        }
      }
    }
  }
}
```

Sample response

```
{
  "data": {
    "ImageListForCVE": {
      "Results": [
        {
          "RepoName": "alpine",
          "Tag": "3.17",
          "Digest": "sha256:75bfe77c8d5a76b4421cfcebbd62a28ae70d10147578d0cda45820e99b0ef1d8",
          "LastUpdated": "2023-02-11T04:46:42.5583430682",
          "IsSigned": true,
          "Size": "3375436",
          "Vendor": "",
          "DownloadCount": 0,
          "Licenses": "",
          "Title": "",
          "Manifests": [
            {
              "Digest": "sha256:75bfe77c8d5a76b4421cfcebbd62a28ae70d10147578d0cda45820e99b0ef1d8",
              "ConfigDigest": "sha256:6a2bcc1c7b4c9207f791a4512d7f2fa8fc2daeae58dbc51cb2797b05415f082a",
              "Platform": {
                "Os": "linux",
                "Arch": "amd64"
              }
            }
          ]
        },
      ]
    }
  }
}
```

List images not affected by a given CVE id

Sample request

```
{
  ImageListWithCVEFixed(id: "CVE-2023-0464", image: "ubuntu") {
    Results {
      RepoName
      Tag
      Digest
      LastUpdated
      Manifests {
        Digest
```

```
        ConfigDigest
      }
    }
  }
}
```

Sample response

```
{
  "data": {
    "ImageListWithCVEFixed": {
      "Results": [
        {
          "RepoName": "ubuntu",
          "Tag": "kinetic",
          "Digest": "sha256:1ac35e499e330f6520e80e91b29a55ff298077211f5ed66aff5cb357cca4a28f",
          "LastUpdated": "2022-10-14T15:28:55.0263968Z",
          "Manifests": [
            {
              "Digest": "sha256:1ac35e499e330f6520e80e91b29a55ff298077211f5ed66aff5cb357cca4a28f",
              "ConfigDigest": "sha256:824c0269745923afceb9765ae24f5b331bb6fcf2a82f7eba98b3cfd543afb41e"
            }
          ]
        },
        {
          "RepoName": "ubuntu",
          "Tag": "kinetic-20220922",
          "Digest": "sha256:79eae04a0e32878fef3f8c5f901c32f6704c4a80b7f3fd9d89629e15867acfff",
          "LastUpdated": "2022-10-14T15:27:41.2144454Z",
          "Manifests": [
            {
              "Digest": "sha256:79eae04a0e32878fef3f8c5f901c32f6704c4a80b7f3fd9d89629e15867acfff",
              "ConfigDigest": "sha256:15c8dcf63970bb14ea36e41aa001b87d8d31e25a082bf6f659d12489d3e53d90"
            }
          ]
        }
      ]
    }
  }
}
```

Search images by digest

Sample request

```
{
  ImageListForDigest(
    id: "79eae04a0e32878fef3f8c5f901c32f6704c4a80b7f3fd9d89629e15867acfff"
  ) {
    Results{
      RepoName
      Tag
      Title
    }
  }
}
```

Sample response

```
{
  "data": {
    "ImageListForDigest": {
      "Results": [
        {
          "RepoName": "ubuntu",
          "Tag": "kinetic-20220922",
          "Title": "ubuntu"
        }
      ]
    }
  }
}
```

List the latest image across every repository

Sample request

```
{
  RepoListWithNewestImage(requestedPage: {limit: 2, offset:0, sortBy: ALPHABETIC_ASC}) {
    Page {
      TotalCount
      ItemCount
    }
```

```
    Results {
      Name
      LastUpdated
      Size
      Platforms {
        Os
        Arch
      }
      NewestImage {
        Digest
        Tag
      }
    }
  }
}
```

Sample response

```
{
  "data": {
    "RepoListWithNewestImage": {
      "Page": {
        "TotalCount": 30,
        "ItemCount": 2
      },
      "Results": [
        {
          "Name": "mariadb",
          "LastUpdated": "2022-10-18T14:56:33.1993083+03:00",
          "Size": "124116964",
          "Platforms": [
            {
              "Os": "linux",
              "Arch": "amd64"
            }
          ],
          "NewestImage": {
            "Digest": "sha256:49a299f5c4b1af5bc2aa6cf8e50ab5bad85db4d0095745369acfc1934ece99d0",
            "Tag": "latest"
          }
        },
        {
          "Name": "tomcat",
          "LastUpdated": "2022-10-18T14:55:13.8303866+03:00",
          "Size": "311658063",
          "Platforms": [
            {
              "Os": "linux",
              "Arch": "amd64"
            }
          ],
          "NewestImage": {
            "Digest": "sha256:bbc5a3912b568fbfb5912beaf25054f1f407c32a53acae29f19ad97485731a78",
            "Tag": "jre17"
          }
        }
      ]
    }
  }
}
```

All images in repo

Sample request

```
{
  ImageList (repo: "ubuntu") {
    Results {
      Tag
      Digest
      LastUpdated
      Size
    }
  }
}
```

Sample response

```
{
  "data": {
    "ImageList": {
      "Results": [
        {
          "Tag": "jammy",
          "Digest": "sha256:f96fcb040c7ee00c037c758cf0ab40638e6ee89b03a9d639178fcbd0e7f96d27",
          "LastUpdated": "2022-10-14T15:29:18.0325322Z",
          "Size": "30472739"
```

```
        },
        {
          "Tag": "jammy-20221003",
          "Digest": "sha256:86681debca1719dff33f426a0f5c41792ebc52496c5d78a93b655b8b48fb71b2",
          "LastUpdated": "2022-10-14T15:29:07.0004587Z",
          "Size": "30472748"
        },
        {
          "Tag": "kinetic",
          "Digest": "sha256:1ac35e499e330f6520e80e91b29a55ff298077211f5ed66aff5cb357cca4a28f",
          "LastUpdated": "2022-10-14T15:28:55.0263968Z",
          "Size": "27498890"
        },
        {
          "Tag": "kinetic-20220922",
          "Digest": "sha256:79eae04a0e32878fef3f8c5f901c32f6704c4a80b7f3fd9d89629e15867acfff",
          "LastUpdated": "2022-10-14T15:27:41.2144454Z",
          "Size": "27498899"
        },
        {
          "Tag": "latest",
          "Digest": "sha256:9bc6d811431613bf2fd8bf3565b319af9998fc5c46304022b647c63e1165657c",
          "LastUpdated": "2022-10-14T15:26:59.6707939Z",
          "Size": "30472740"
        },
        {
          "Tag": "rolling",
          "Digest": "sha256:72e75626c5068b9d9a462c4fc80a29787d0cf61c8abc81bfd5ea69f6248d56fc",
          "LastUpdated": "2022-10-14T15:27:21.2441356Z",
          "Size": "30472741"
        }
      ]
    }
  }
}
```


List all images with expanded information for a given repository

Sample request

```
{
  ExpandedRepoInfo(repo: "ubuntu") {
    Images {
      Tag
      Digest
    }
    Summary {
      LastUpdated
      Size
      NewestImage {
        Tag
        LastUpdated
        Digest
      }
    }
  }
}
```

Sample response


```
{
  "data": {
    "ExpandedRepoInfo": {
      "Images": [
        {
          "Tag": "jammy",
          "Digest": "sha256:f96fcb040c7ee00c037c758cf0ab40638e6ee89b03a9d639178fcbd0e7f96d27"
        },
        {
          "Tag": "jammy-20221003",
          "Digest": "sha256:86681debca1719dff33f426a0f5c41792ebc52496c5d78a93b655b8b48fb71b2"
        },
        {
          "Tag": "kinetic",
          "Digest": "sha256:1ac35e499e330f6520e80e91b29a55ff298077211f5ed66aff5cb357cca4a28f"
        },
        {
          "Tag": "kinetic-20220922",
          "Digest": "sha256:79eae04a0e32878fef3f8c5f901c32f6704c4a80b7f3fd9d89629e15867acfff"
        },
        {
          "Tag": "rolling",
          "Digest": "sha256:72e75626c5068b9d9a462c4fc80a29787d0cf61c8abc81bfd5ea69f6248d56fc"
        },
        {
          "Tag": "latest",
          "Digest": "sha256:9bc6d811431613bf2fd8bf3565b319af9998fc5c46304022b647c63e1165657c"
        }
      ],
```

```
      "Summary": {
        "LastUpdated": "2022-10-14T15:29:18.0325322Z",
        "Size": "58146896",
        "NewestImage": {
          "Tag": "jammy",
          "LastUpdated": "2022-10-14T15:29:18.0325322Z",
          "Digest": "sha256:f96fcb040c7ee00c037c758cf0ab40638e6ee89b03a9d639178fcbd0e7f96d27"
        }
      }
    }
  }
}
```

Global search

Sample request

```
{
  GlobalSearch(query: "ubuntu:latest") {
    Page {
      ItemCount
      TotalCount
    }
    Images {
      RepoName
      Tag
      LastUpdated
      Manifests {
        Digest
        Layers {
          Size
          Digest
        }
      }
    }
  }
}
```

Sample response

```
{
  "data": {
    "GlobalSearch": {
      "Page": {
        "ItemCount": 1,
        "TotalCount": 1
      },
      "Images": [
        {
          "RepoName": "ubuntu",
          "Tag": "latest",
          "LastUpdated": "2022-10-14T15:26:59.6707939Z",
          "Manifests": [
            {
              "Digest": "sha256:9bc6d811431613bf2fd8bf3565b319af9998fc5c46304022b647c63e1165657c",
              "Layers": [
                {
                  "Size": "30428928",
                  "Digest": "sha256:cf92e523b49ea3d1fae59f5f082437a5f96c244fda6697995920142ff31d59cf"
                }
              ]
            }
          ]
        }
      ]
    }
  }
}
```

Sample request

```
{
  GlobalSearch(query: "") {
    Repos {
      Name
    }
  }
}
```

Sample response

```
{
  "data": {
    "GlobalSearch": {
      "Repos": [
```

```
        {
          "Name": "centos"
        },
        {
          "Name": "ubuntu"
        }
      ]
    }
  }
}
```

Search derived images

Sample query

```
{
  DerivedImageList(image: "ubuntu:latest", requestedPage: {offset: 0, limit: 10}) {
    Page {
      TotalCount
      ItemCount
    }
    Results {
      RepoName
      Tag
      LastUpdated
    }
  }
}
```

Sample response

```
{
  "data": {
    "DerivedImageList": {
      "Page": {
        "TotalCount": 9,
        "ItemCount": 9
      },
      "Results": [
        {
          "RepoName": "mariadb",
          "Tag": "latest",
          "LastUpdated": "2022-10-18T14:56:33.1993083+03:00"
        },
        {
          "RepoName": "maven",
          "Tag": "latest",
          "LastUpdated": "2022-10-14T18:30:12.0929807+03:00"
        },
        {
          "RepoName": "tomcat",
          "Tag": "latest",
          "LastUpdated": "2022-10-18T14:50:09.7229959+03:00"
        },
        {
          "RepoName": "tomcat",
          "Tag": "jre17",
          "LastUpdated": "2022-10-18T14:55:13.8303866+03:00"
        },
        {
          "RepoName": "tomcat",
          "Tag": "jre17-temurin",
          "LastUpdated": "2022-10-18T14:54:46.4133521+03:00"
        },
        {
          "RepoName": "tomcat",
          "Tag": "jre17-temurin-jammy",
          "LastUpdated": "2022-10-18T14:51:12.235475+03:00"
        }
      ]
    }
  }
}
```

Search base images

Sample query

```
{
  BaseImageList(image: "mariadb:latest", requestedPage: {offset: 0, limit: 10}) {
    Page {
      TotalCount
      ItemCount
    }
    Results {
```

```
      RepoName
      Tag
      LastUpdated
    }
  }
}
```

Sample response

```
{
  "data": {
    "BaseImageList": {
      "Page": {
        "TotalCount": 4,
        "ItemCount": 4
      },
      "Results": [
        {
          "RepoName": "ubuntu",
          "Tag": "jammy",
          "LastUpdated": "2022-10-14T18:29:18.0325322+03:00"
        },
        {
          "RepoName": "ubuntu",
          "Tag": "jammy-20221003",
          "LastUpdated": "2022-10-14T18:29:07.0004587+03:00"
        },
        {
          "RepoName": "ubuntu",
          "Tag": "latest",
          "LastUpdated": "2022-10-14T18:26:59.6707939+03:00"
        },
        {
          "RepoName": "ubuntu",
          "Tag": "rolling",
          "LastUpdated": "2022-10-14T18:27:21.2441356+03:00"
        }
      ]
    }
  }
}
```

Get details of a specific image

Sample query

```
{
  Image(image: "mariadb:latest") {
    RepoName
    Tag
    LastUpdated
    Digest
    Description
  }
}
```

Sample response

```
{
  "data": {
    "Image": {
      "RepoName": "mariadb",
      "Tag": "latest",
      "LastUpdated": "2022-10-18T14:56:33.1993083+03:00",
      "Digest": "sha256:49a299f5c4b1af5bc2aa6cf8e50ab5bad85db4d0095745369acfc1934ece99d0",
      "Description": "MariaDB Server is a high performing open source relational database, forked from MySQL."
    }
  }
}
```

Get referrers of a specific image

Sample query

```
{
  Referrers(
    repo: "golang"
    digest: "sha256:fed08b0eaea00aab17f82ecbb78675919d216c72eea985581758191f694aeaf7"
    type: "application/vnd.example.icecream.v1"
  ) {
    MediaType
    ArtifactType
    Digest
```

```
    Annotations {
      Key
      Value
    }
  }
}
```

Sample response


```
{
  "data": {
    "Referrers": [
      {
        "MediaType": "application/vnd.oci.artifact.manifest.v1+json",
        "ArtifactType": "application/vnd.example.icecream.v1",
        "Digest": "sha256:be7a3d01c35a2cf53c502e9dc50cdf36b15d9361c81c63bf319f1d5cbe44ab7c",
        "Annotations": [
          {
            "Key": "format",
            "Value": "oci"
          },
          {
            "Key": "demo",
            "Value": "true"
          }
        ]
      },
      {
        "MediaType": "application/vnd.oci.artifact.manifest.v1+json",
        "ArtifactType": "application/vnd.example.icecream.v1",
        "Digest": "sha256:d9ad22f41d9cb9797c134401416eee2a70446cee1a8eb76fc6b191f4320dade2",
        "Annotations": [
          {
            "Key": "demo",
            "Value": "true"
          },
          {
            "Key": "format",
            "Value": "oci"
          }
        ]
      }
    ]
  }
}
```


November 15, 2023

7.15 Benchmarking zot with zb

👉 The **zb** tool is useful for benchmarking OCI registry workloads in scenarios such as the following:

- comparing configuration changes
- comparing software versions
- comparing hardware/deployment environments
- comparing with other registries

With the zb tool, you can benchmark a zot registry or any other container image registry that conforms to the OCI Distribution Specification published by the Open Container Initiative (OCI).

💡 We recommend installing and benchmarking with zb when you install zot.

7.15.1 How to get zb

The zb project is hosted with zot on GitHub at project-zot. From GitHub, you can download the zb binary or you can build zb from the source. You can also directly run the released docker image.

7.15.2 Supported platforms and architectures

zb is supported for the following operating systems and platform architectures:

OS	ARCH	Platform
linux	amd64	Intel-based Linux servers
linux	arm64	ARM-based servers and Raspberry Pi4
darwin	amd64	Intel-based MacOS
darwin	arm64	ARM-based MacOS

7.15.3 Downloading zb binaries

Download the executable binary for your server platform and architecture under "Assets" on the GitHub zot releases page.

The binary image is named using the target platform and architecture from the Supported platforms and architectures table. For example, the binary for an Intel-based MacOS server is `zb-darwin-amd64`.

7.15.4 Building zb from source

To build the zb binary, copy or clone the zot project from GitHub and execute the `make bench` command in the `zot` directory. Use the same command options that you used to build zot, as shown:

```
make OS=os ARCH=architecture bench
```

For example, the following command builds zb for an Intel-based MacOS server:

```
make OS=darwin ARCH=amd64 bench
```

In this example, the resulting executable file is `zb-darwin-amd64` in the `zot/bin` directory.

💡 A sample Dockerfile for zb is available at Dockerfile-zb.

7.15.5 Running zb

The original filename of the executable file will reflect the build options, such as `zb-linux-amd64.` For convenience, you can rename the executable to simply `zb`.

💡 The instructions and examples in this guide use `zb` as the name of the executable file.

Usage

To view the usage and options of `zb`, run the command with the `--help` option:

```
bin/zb --help
```

Command output:

```
Usage:
  zb <url> [flags]

Flags:
  -A, --auth-creds string      Use colon-separated BASIC auth creds
  -c, --concurrency int        Number of multiple requests to make at a time (default 1)
  -h, --help                   help for zb
  -o, --output-format string   Output format of test results: stdout (default), json, ci-cd
  -r, --repo string            Use specified repo on remote registry for test data
  -n, --requests int           Number of requests to perform (default 1)
  -s, --src-cidr string        Use specified cidr to obtain ips to make requests from, src-ips and src-cidr are mutually exclusive
  -i, --src-ips string         Use colon-separated ips to make requests from, src-ips and src-cidr are mutually exclusive
  -v, --version                Show the version and exit
  -d, --working-dir string     Use specified directory to store test data
```

Example

The following example executes a benchmark operation using zb.

```
bin/zb -c 10 -s 127.0.10.0/24 -n 1000 http://localhost:8080
```

You can also run the released docker image.

```
docker run --net=host -it ghcr.io/project-zot/zb-linux-amd64:latest -c 10 -n 1000 -s 127.0.10.0/24 http://localhost:8080
```

Command output:

```
Registry URL: http://localhost:8080

Concurrency Level: 2
Total requests:    100
Working dir:

========
Test name:            Get Catalog
Time taken for tests: 45.397205ms
Complete requests:    100
Failed requests:      0
Requests per second:  2202.7788

2xx responses: 100

min: 402.259µs
max: 3.295887ms
p50: 855.045µs
p75: 971.709µs
p90: 1.127389ms
p99: 3.295887ms

========
Test name:            Push Monolith 1MB
Time taken for tests: 952.336383ms
Complete requests:    100
Failed requests:      0
Requests per second:  105.00491

2xx responses: 100

min: 11.125673ms
max: 26.375356ms
p50: 18.917253ms
p75: 21.753441ms
p90: 24.02137ms
p99: 26.375356ms
```

```
...
```

December 21, 2022

7.16 Performance Profiling in zot

👉 Use zot's built-in profiling tools to collect and analyze runtime performance.

The profiling capabilities within zot allow a zot administrator to collect and export a range of diagnostic performance data such as CPU intensive function calls, memory allocations, and execution traces. The collected data can then be analyzed using Go tools and a variety of available visualization tools.

✏️ If authentication is enabled, only a zot admin user can access the APIs for profiling.

✏️ All examples in this article assume that the zot registry is running at `localhost:8080`.

7.16.1 What data is available?

The zot source code incorporates golang's pprof package of runtime analysis tools to collect data for the following performance-related profiles:

Profile	Description
allocs	A sampling of all past memory allocations.
block	Stack traces that led to blocking on synchronization primitives.
cmdline	The command line invocation of the current program.
goroutine	Stack traces of all current goroutines. Use `debug=2` as a URL query parameter to export in the same format as an unrecovered panic.
heap	A sampling of memory allocations of live objects. You can specify the `gc` GET parameter to run GC before taking the heap sample.
mutex	Stack traces of holders of contended mutexes.
profile	CPU usage profile. You can specify the duration in the `seconds` URL query parameter. After receiving the profile file, use the `go tool pprof` command to investigate the profile.
threadcreate	Stack traces that led to the creation of new OS threads.
trace	A trace of execution of the current program. You can specify the duration in the `seconds` URL query parameter. After you get the trace file, use the `go tool trace` command to investigate the trace.

To return a current HTML-format profile list along with a count of currently available records for each profile, use the following API command:

```
/v2/_zot/pprof/
```

✏️ If authentication is enabled, only an admin user can access this API.

7.16.2 How do I export profile data?

To collect and export any available profile, use the following API command format:

```
/v2/_zot/pprof/<profile-type>[?<query-parameters>]
```

The following example shows an API request for the CPU usage profile named `profile` using a collection window of 30 seconds:

```
$ curl -s http://localhost:8080/v2/_zot/pprof/profile?seconds=30 > cpu.prof
```

This command example creates an output data file named "cpu.prof".

- The query parameter `?seconds=<number>` specifies the number of seconds to gather the profile data. If this parameter is not specified, the default is 30 seconds.
- In this example, the raw output data is redirected to a file named "cpu.prof". Alternatively, you can use `curl -O` to create a file with the default profile name (in this case, "profile"). If no output file is specified by either a cURL flag or an output redirection, the cURL command fails with "Failure writing output to destination".
- The command output file is in a machine-readable format that can be interpreted by performance analyzers.

7.16.3 Analyzing the CPU usage profile using `go tool pprof`

Go's pprof package provides a variety of presentation formats for analyzing runtime performance.

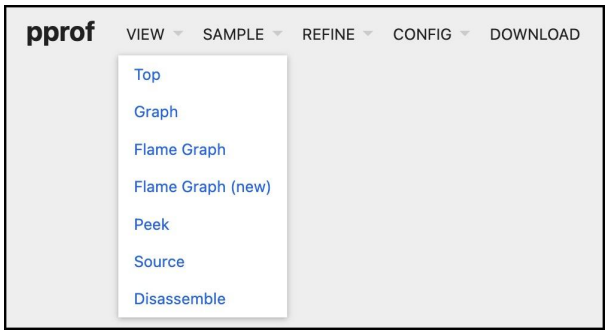
For detailed information, see the pprof documentation.

Generating a pprof web presentation

When an HTTP port is specified as a command flag, the `go tool pprof` command installs and opens a local web server that provides a web interface for viewing and analyzing the profile data. This example opens a localhost page at port 9090 for viewing the CPU usage data captured in the profile file named "cpu.prof".

```
$ go tool pprof -http=:9090 cpu.prof
Serving web UI on http://localhost:9090
```

The pprof web view offers several options for viewing and interpreting the collected performance data. Select VIEW to see the available options:

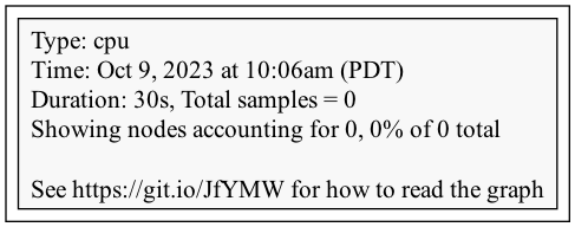


A Flame Graph can be very useful for analyzing CPU usage:

Generating a graphic image

The pprof package can generate graphic representations of profile data in many formats. This example generates a PNG file representing the CPU usage in the "cpu.prof" file.

```
$ go tool pprof -png cpu.prof
Generating report in profile001.png
```



Opening a pprof interactive session

This example opens an interactive session with pprof and executes the pprof `top` command, which displays the top ten modules by CPU usage during the profiling capture window.

```
$ go tool pprof cpu.prof
Type: cpu
Time: Sep 26, 2023 at 10:01am (PDT)
Duration: 30s, Total samples = 10ms (  0.1%)
Entering interactive mode (type "help" for commands, "o" for options)
(pprof) top
Showing nodes accounting for 10ms, 100% of 10ms total
    flat  flat%   sum%        cum   cum%
    10ms   100%   100%       10ms   100%  runtime.pthread_cond_signal
         0     0%   100%       10ms   100%  runtime.findRunnable
         0     0%   100%       10ms   100%  runtime.mcall
         0     0%   100%       10ms   100%  runtime.notewakeup
         0     0%   100%       10ms   100%  runtime.park_m
         0     0%   100%       10ms   100%  runtime.runSafePointFn
         0     0%   100%       10ms   100%  runtime.schedule
         0     0%   100%       10ms   100%  runtime.semawakeup
(pprof)
```

7.16.4 Analyzing the trace profile using `go tool trace`

You can collect trace data with the `trace` profile, as in this example:

```
  $ curl -s -v http://localhost:8080/v2/_zot/pprof/trace?seconds=30 > trace.prof
```

Using the `go tool trace` package, you can analyze the trace data captured in the "trace.prof" example file:

```
$ go tool trace trace.prof
2023/09/21 16:58:58 Parsing trace...
2023/09/21 16:58:58 Splitting trace...
2023/09/21 16:58:58 Opening browser. Trace viewer is listening on http://127.0.0.1:62606
```

The `go tool trace` command installs and opens a local web server that provides a web interface for viewing and analyzing the trace data.

As an alternative, you can generate a pprof-like profile from the trace file using the following command:

```
$ go tool trace -pprof=[net|sync|syscall|sched] <filename>
```

For example:

```
$ go tool trace -pprof=net trace.prof
```

October 12, 2023

7.17 Using `kind` for Deployment Testing

👉 Use `kind` to try out zot deployment with Kubernetes.

This article describes how to create a `kind` cluster that includes a local zot registry.

7.17.1 Deploying the cluster and registry

The procedure described installs a `kind` cluster with a zot registry at `localhost:5001` and then loads and runs a "hello" app to test the installation. Although the procedure is given as a series of steps, you can find a complete shell script to perform these steps at the end of this article.

✏️ This article is based on Create A Cluster And Registry, which you can find on the `kind` website.

Step 1: Prepare the environment

The following packages must be installed:

- `docker`
- `kubernetes`
- `kind`
- `containerd`
- `skopeo`

Execute the following shell commands to set environment variables.

```
set -o errexit

# set no_proxy if applicable
if [ ! -z "${no_proxy}" ]; then
  echo "Updating no_proxy environment variables";
  export no_proxy=${no_proxy},kind-registry;
  export NO_PROXY=${no_proxy};
fi
```

Step 2: Create a registry container

Create a `kind-registry` container, pulling a zot binary from the GitHub Container Registry (ghcr.io).

✏️ This example pulls `zot-minimal-linux-amd64:latest`,
a minimal (no extensions) zot image for an AMD-based linux server.

Other available images are described at the zot releases page in GitHub.
You can also specify a release by replacing `latest` with an available release number.

```
# create registry container unless it already exists
reg_name='kind-registry'
reg_port='5001'
if [ "$(docker inspect -f '{{.State.Running}}' "${reg_name}" 2>/dev/null || true)" != 'true' ]; then
  docker run \
    -d --restart=always -p "127.0.0.1:${reg_port}:5000" --name "${reg_name}" \
    ghcr.io/project-zot/zot-minimal-linux-amd64:latest
fi
```

Step 3: Create the `kind` cluster

Create a cluster with the local registry enabled.

```
# enable the local registry in containerd
cat <<EOF | kind create cluster --config=-
kind: Cluster
apiVersion: kind.x-k8s.io/v1alpha4
containerdConfigPatches:
- |-
```

```
  [plugins."io.containerd.grpc.v1.cri".registry.mirrors."localhost:${reg_port}"]
    endpoint = ["http://${reg_name}:5000"]
EOF
```

Step 4: Connect the registry to the cluster network

Connect the registry to the "kind" network so that it can communicate with other resources in the same network.

```
# check whether already connected to the network
if [ "$(docker inspect -f='{{json .NetworkSettings.Networks.kind}}' \
"${reg_name}")" = 'null' ]; then
  docker network connect "kind" "${reg_name}"
fi
```

Step 5: Document the local registry

Create a ConfigMap that specifies how to interact with the local registry. This ConfigMap follows the KEP-1755 Standard for communicating a local registry.

```
cat <<EOF | kubectl apply -f -
apiVersion: v1
kind: ConfigMap
metadata:
  name: local-registry-hosting
  namespace: kube-public
data:
  localRegistryHosting.v1: |
    host: "localhost:${reg_port}"
    help: "https://kind.sigs.k8s.io/docs/user/local-registry/"
EOF
```

Step 6: Deploy and test

Use `skopeo` to copy (pull) a "hello" app from the Google Container Registry (gcr.io) into the new zot registry. Using `kubectl`, deploy the app from the new local zot registry as "hello-server" and monitor the deployment for initial availability.

```
# copy an image
skopeo copy --format=oci --dest-tls-verify=false \
docker://gcr.io/google-samples/hello-app:1.0 \
docker://localhost:5001/hello-app:1.0

# deploy the image
kubectl create deployment hello-server --image=localhost:5001/hello-app:1.0

# check for availability
echo "Waiting for deployment/hello-server to be ready ..."
kubectl wait deployment -n default hello-server \
  --for condition=Available=True --timeout=90s
```

7.17.2 Clean up

To clean up after testing, run the following commands to delete the `kind` cluster and registry.

```
kind delete cluster
docker stop kind-registry
docker rm kind-registry
```

7.17.3 Reference: A complete script

The following script executes all of the preceding steps.

ck here to view the entire script

```
#!/bin/sh
set -o errexit

# Reference: https://kind.sigs.k8s.io/docs/user/local-registry/

# set no_proxy if applicable
if [ ! -z "${no_proxy}" ]; then
  echo "Updating no_proxy env var";
  export no_proxy=${no_proxy},kind-registry;
  export NO_PROXY=${no_proxy};
fi

# create registry container unless it already exists
reg_name='kind-registry'
reg_port='5001'
if [ "$(docker inspect -f '{{.State.Running}}' "${reg_name}" 2>/dev/null || true)" != 'true' ]; then
  docker run \
    -d --restart=always -p "127.0.0.1:${reg_port}:5000" --name "${reg_name}" \
    ghcr.io/project-zot/zot-minimal-linux-amd64:latest
fi

# create a cluster with the local registry enabled in containerd
cat <<EOF | kind create cluster --config=-
kind: Cluster
apiVersion: kind.x-k8s.io/v1alpha4
containerdConfigPatches:
- |-
  [plugins."io.containerd.grpc.v1.cri".registry.mirrors."localhost:${reg_port}"]
    endpoint = ["http://${reg_name}:5000"]
EOF

# connect the registry to the cluster network if not already connected
if [ "$(docker inspect -f='{{json .NetworkSettings.Networks.kind}}' "${reg_name}")" = 'null' ]; then
  docker network connect "kind" "${reg_name}"
fi

# https://github.com/kubernetes/enhancements/tree/master/keps/sig-cluster-lifecycle/generic/1755-communicating-a-local-registry
#
# document the local registry
cat <<EOF | kubectl apply -f -
apiVersion: v1
kind: ConfigMap
metadata:
  name: local-registry-hosting
  namespace: kube-public
data:
  localRegistryHosting.v1: |
    host: "localhost:${reg_port}"
    help: "https://kind.sigs.k8s.io/docs/user/local-registry/"
EOF

# copy an image
skopeo copy --format=oci --dest-tls-verify=false docker://gcr.io/google-samples/hello-app:1.0 docker://localhost:5001/hello-app:1.0

# deploy the image
kubectl create deployment hello-server --image=localhost:5001/hello-app:1.0

# check for availability
echo "Waiting for deployment/hello-server to be ready ..."
kubectl wait deployment -n default hello-server --for condition=Available=True --timeout=90s

# cleanup
echo "Press a key to begin cleanup ..."
read KEYPRESS
kind delete cluster
docker stop kind-registry
docker rm kind-registry
```

September 13, 2023

7.18 Mirroring From *zot*

👉 *containerd* supports registry mirroring and *zot* can be used as an upstream registry to mirror from.

If your images are in a repository named *docker* in the *zot* registry and you want *containerd* to pull and mirror images from this repository, then your *containerd* configuration would look like the following.

7.18.1 *containerd* v1.x

The registry mirror configuration is specified inline as the plugin.

```
version = 2

[plugins."io.containerd.grpc.v1.cri".registry.mirrors."docker.io"]
endpoint = ['https://<my zot instance>:8080/v2/docker']
```

Note that */v2* is required for this to work.

7.18.2 *containerd* v2.x

The registry mirror configuration has changed in v2.x and now follows a directory structure. The following example shows how *zot* can be used as the default mirror registry.

```
version = 3

[plugins."io.containerd.cri.v1.images".registry]
  config_path = "/etc/containerd/certs.d"
```

```
$ tree /etc/containerd/certs.d
```

```
/etc/containerd/certs.d/
└── _default
    └── hosts.toml
```

```
$ cat /etc/containerd/certs.d/_default/hosts.toml
```

```
# zot is running at localhost:8080
server = "http://localhost:8080"
```

More information about various mirror registry configuration options is available here.

April 11, 2025