THE LINUX FOUNDATION
OPEN SOURCE SUMMIT
NORTH AMERICA

Embedded Linux Conference

OCI Images: not just for containers anymore

Austin Abro

Austin Abro

OSS Maintainer

Defense Unicorns

austinabro321

The OCI Specification

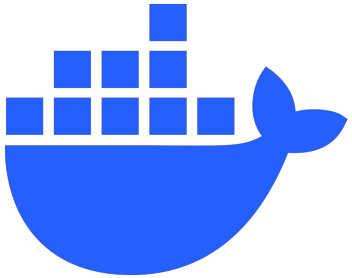
There are three different OCI specifications:

- **Image** - the structure of an OCI image
- **Runtime** - running an Image
- **Distribution** - pushing and pulling to a registry

Today we'll be focused on the **image** specification

OCI Adopters

Design inspired by:



Docker

Now used by:

sigstore
cosign

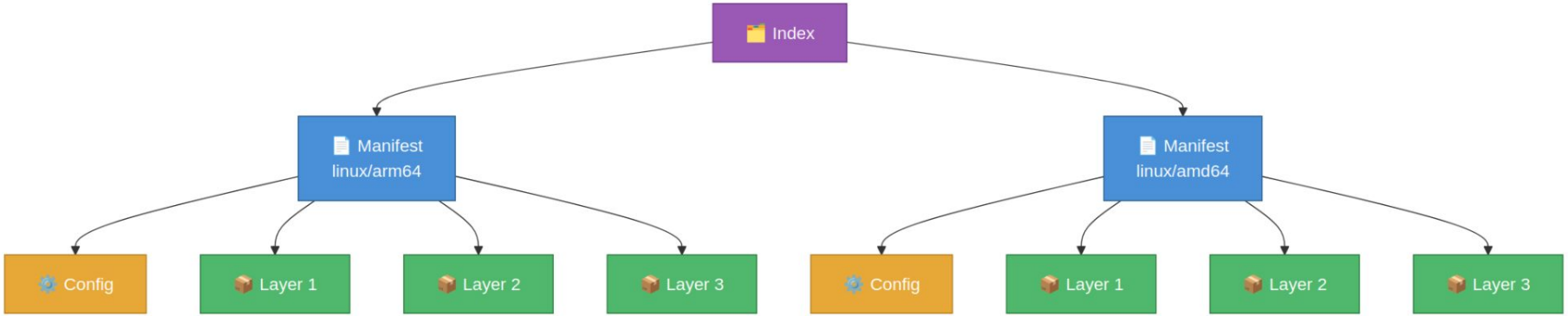
(And many more)

Why have so many tools adopted OCI? Should you?

THE LINUX FOUNDATION
OPEN SOURCE SUMMIT
NORTH AMERICA
Embedded Linux Conference

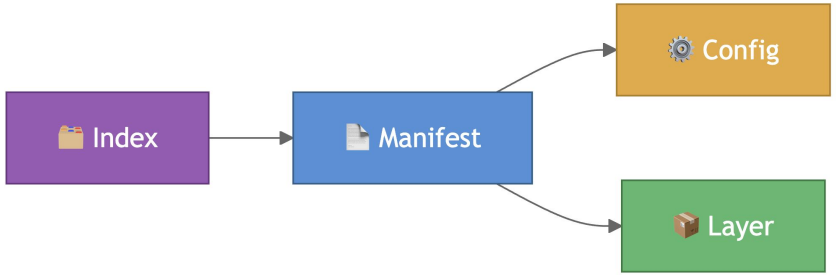
First, let’s see what’s under the hood of the OCI spec

OCI Image structure



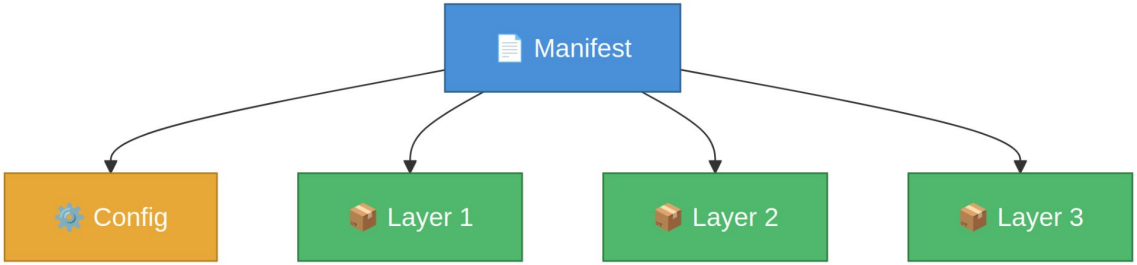
OCI Image structure

- **Index** - Points to one or more manifests
- **Manifest** - Lists the config and layers, often for a specific platform
- **Config** - Metadata describing how to use the image
- **Layers** - The actual data (filesystem changes, archived files, etc.)



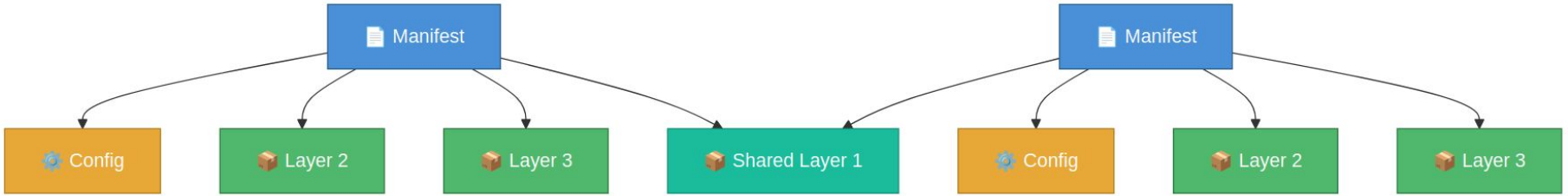
Layers

- Hold all the image content
- Container image layers hold the filesystem
- Helm chart artifacts, always contain a single layer - the archived chart



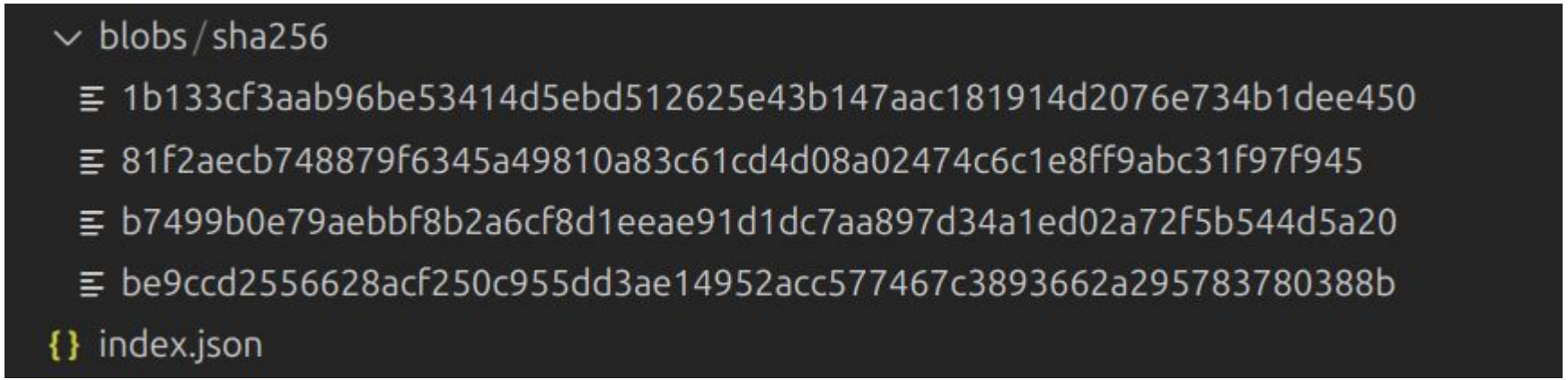
Layers

- Unrelated images can share layers
- base images: `FROM: python:3.11`



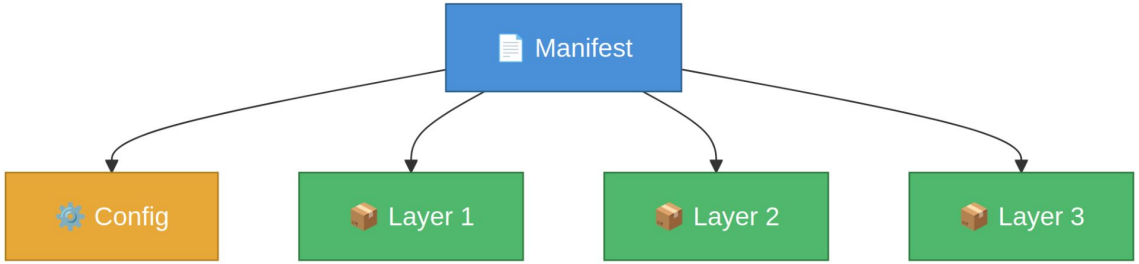
Layers

- Stored in the filesystem by sha256 hash
- Intuitive to find, impossible to duplicate



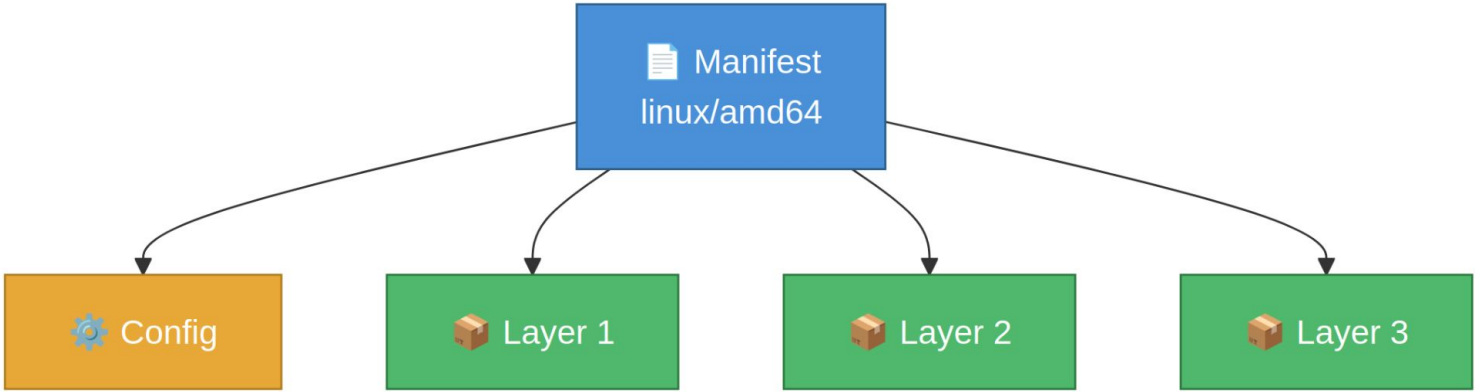
Manifest Config

- Metadata, about the image
- Container images have instructions on how to build the filesystem using the layers
- Helm's manifest config is a json representation of the chart.yaml
- Can be empty



Manifests

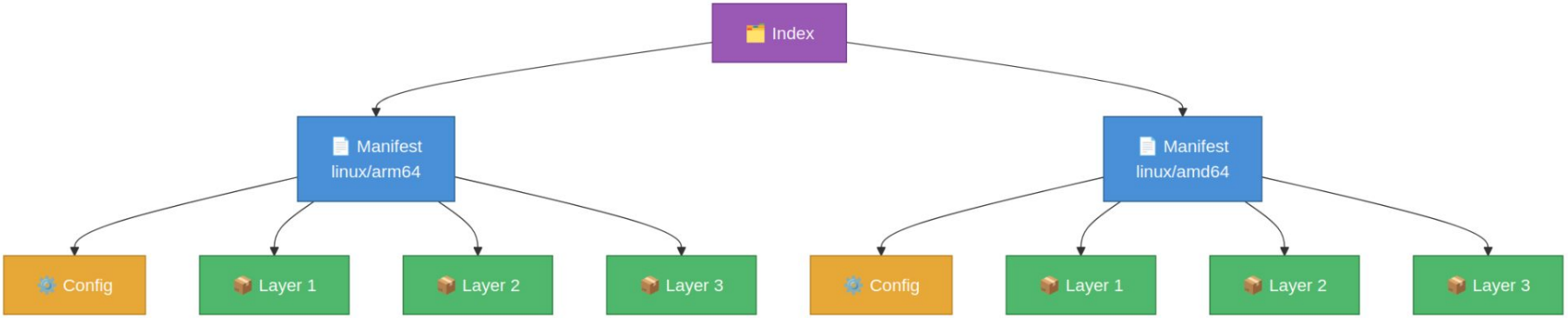
- Manifests reference the config and layers
- Can be built to a platform (OS/architecture)



```json
{
  "schemaVersion": 2,
  "mediaType": "application/vnd.oci.image.manifest.v1+json",
  "config": {
    "mediaType": "application/vnd.oci.image.config.v1+json",
    "digest": "sha256:4752b809555b8767401dfd39638f256c2373763a1f2cc421012301bb48481e6d",
    "size": 1084
  },
  "layers": [
    {
      "mediaType": "application/vnd.oci.image.layer.v1.tar+gzip",
      "digest": "sha256:e6886dc0b01f09d19480a3270fd9e7c4b261346cee9490b881b36edf21c7e722",
      "size": 3192931
    },
    {
      "mediaType": "application/vnd.oci.image.layer.v1.tar+gzip",
      "digest": "sha256:cc1421ef2ded4a559feaefe8dc266488d60675fcc995db7e51f0b0a6d893e010",
      "size": 280131
    },
    {
      "mediaType": "application/vnd.oci.image.layer.v1.tar+gzip",
      "digest": "sha256:4f4fb700ef54461cfa02571ae0db9a0dc1e0cdb5577484a6d75e68dc38e8acc1",
      "size": 32
    }
  ]
}
```

Indexes

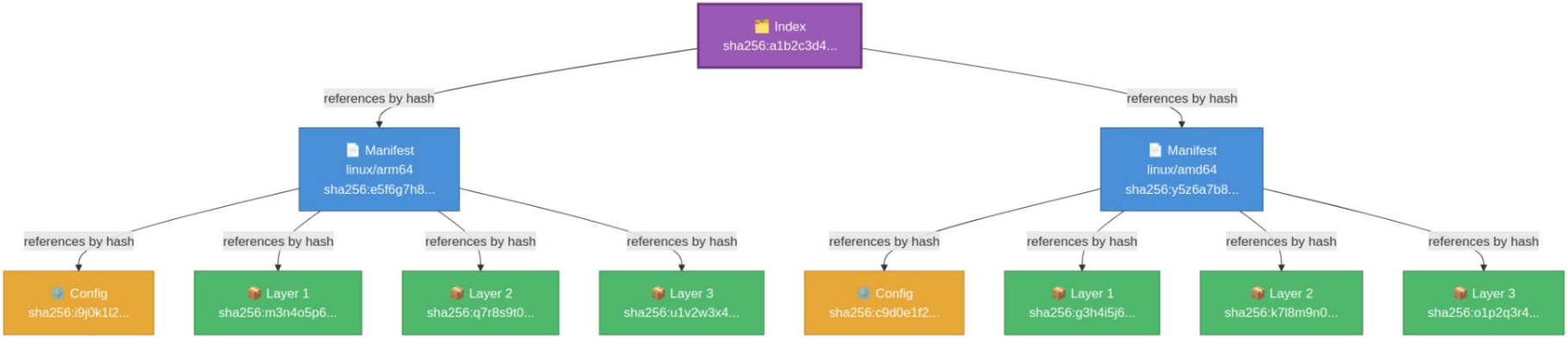
- Reference one or more manifest
- All manifests for a single image or all manifests stored in a OCI layout




{
"schemaVersion": 2,
"mediaType": "application/vnd.oci.image.index.v1+json",
"manifests": [
{
"mediaType": "application/vnd.oci.image.manifest.v1+json",
"digest": "sha256:227aa7b4f3d89833db58676eacdbe9a49b5d5e4748e0ec3f05005335fa73aaf9",
"size": 2107,
"platform": {
"architecture": "amd64",
"os": "linux"
}
},
{
"mediaType": "application/vnd.oci.image.manifest.v1+json",
"digest": "sha256:19392769a90aa8ba542f9c0f0ccaba21d350535f68063ca62ec7ac35f46258e8",
"size": 2107,
"platform": {
"architecture": "arm64",
"os": "linux",
"variant": "v8"
}
}
]
}

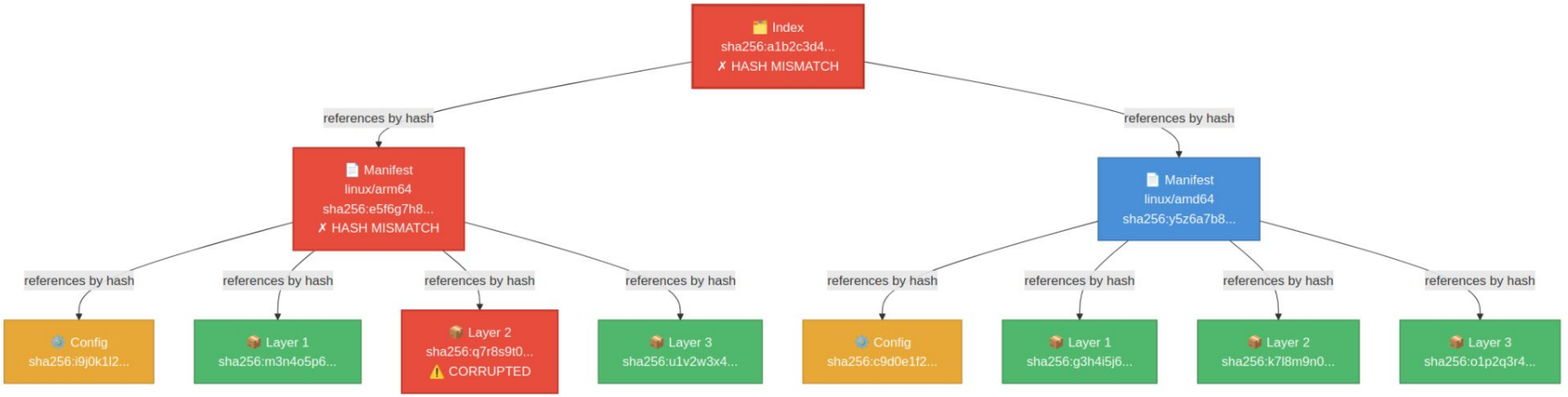
Merkle Tree

Each parent node is comprised of cryptographic hashes of it's child nodes



Merkle Tree

When any child node is corrupted it invalidates all of its parent's hashes



Why should you use OCI

- **Security** - corruption prevention
- **Storage efficiencies** - de-duplicated layers
- **User Experience** - users interact with existing registries

Let’s view some community image spec implementations

ORAS

- Similar to Docker CLI; built for generic OCI images
- login, push, pull, tag, etc
- Fantastic backend oras-go to ease maintenance of OCI objects

Cosign

- Tool to sign and verify OCI images
- Cosign artifacts are simple - single layer of signing data + an empty config
- Lives right next to the image it's signing, uses the referrers API to reference it
- Registry Explorer

sigstore

cosign

Zarf

- Zarf is an air-gap Kubernetes packaging tool
- It stores images, files, and Git repositories into one artifact
  - Files and repositories get archived into tars and stored as layers.
  - Images are stored in an OCI-layout, allowing for layer de-duplication
- OCI provides simple image caching
- Registry Explorer

Helm

- Helm charts OCI artifacts are always one layer, an archived chart
- Helm config is a json representation of the chart.yaml
- Helm users already have access to a registry, so it’s a natural place to put charts
- Registry Explorer

Flux

- Similar to Helm - one layer of archived manifests
- Flux references the OCI artifact during deploy, can pull it using existing registry secrets
- The Flux OCIRepo CRD has built in cosign support
- Registry Explorer

Why should you use OCI

- **Security** - corruption prevention
- **Storage efficiencies** - de-duplicated layers
- **User Experience** - users interact with existing registries
- **signing** - Cosign
- **air-gap packaging** - ORAS, Zarf

Getting meta: slideshows as OCI objects

THE LINUX FOUNDATION
OPEN SOURCE SUMMIT
NORTH AMERICA
Embedded Linux Conference

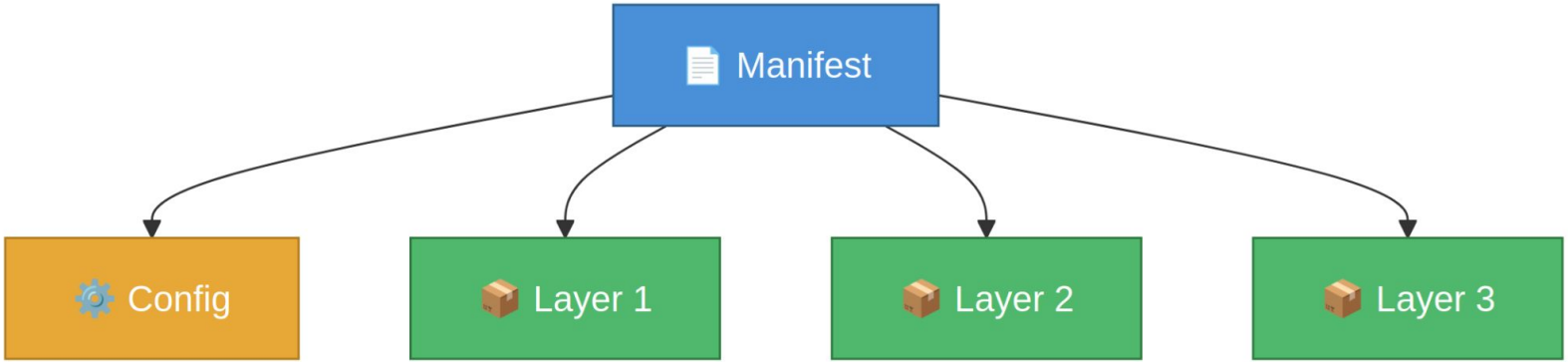
OCI Slideshow

- Let's take a custom format for OCI objects
- Simple, just a title and images

```
title: "OCI Images: not just for containers anymore"
images:
  - slides/slide-01.png
  - slides/slide-02.png
  - slides/slide-03.png
  - slides/slide-04.png
  - slides/slide-05.png
```

Goal

- Config is the yaml specification
- Each slide is a layer



```go
func Build(s *slideshow.Slideshow, yamlDir string) (*Artifact, error) {
  configBytes, err := s.ConfigJSON()
  if err != nil {
    return nil, fmt.Errorf("marshaling config: %w", err)
  }
  config := Blob{Descriptor: descriptor(MediaTypeConfig, configBytes), Data: configBytes}

  var layers []Blob
  for _, imgPath := range s.Images {
    data, err := os.ReadFile(filepath.Join(yamlDir, imgPath))
    if err != nil {
      return nil, fmt.Errorf("reading image %s: %w", imgPath, err)
    }
    layers = append(layers, Blob{Descriptor: descriptor(MediaTypeImage, data), Data: data})
  }
  layerDescs := make([]ocispec.Descriptor, len(layers))
  for i, l := range layers {
    layerDescs[i] = l.Descriptor
  }
  manifest := ocispec.Manifest{
    Versioned: specs.Versioned{SchemaVersion: 2},
    MediaType: ocispec.MediaTypeImageManifest,
    Config:    config.Descriptor,
    Layers:    layerDescs,
  }
```

Pushing the image

- Push the config
- Push the layers
- Push the manifest

```
if err := pushBlob(ctx, repo, art.Config.Descriptor, art.Config.Data); err != nil {
  return fmt.Errorf("pushing config: %w", err)
}
for _, layer := range art.Layers {
  if err := pushBlob(ctx, repo, layer.Descriptor, layer.Data); err != nil {
    return fmt.Errorf("pushing layer: %w", err)
  }
}
if err := repo.PushReference(ctx, art.Manifest.Descriptor, bytes.NewReader(art.Manifest.Data), tag); err != nil {
  return fmt.Errorf("pushing manifest: %w", err)
}
```

Pulling the image

- Copy from the registry to the filesystem

```
store, err := oci.New(outputDir)
if err != nil {
  return fmt.Errorf("opening oci store at %s: %w", outputDir, err)
}

if _, err := oras.Copy(ctx, repo, tag, store, ref, oras.DefaultCopyOptions); err != nil {
  return fmt.Errorf("pulling %s: %w", ref, err)
}
```

Let see it in action

- Layer de-duplication
- Corruption prevention
- Signing

Questions?

Thank You!

THE LINUX FOUNDATION
OPEN SOURCE SUMMIT
NORTH AMERICA

Embedded Linux Conference