Inside The Indexer

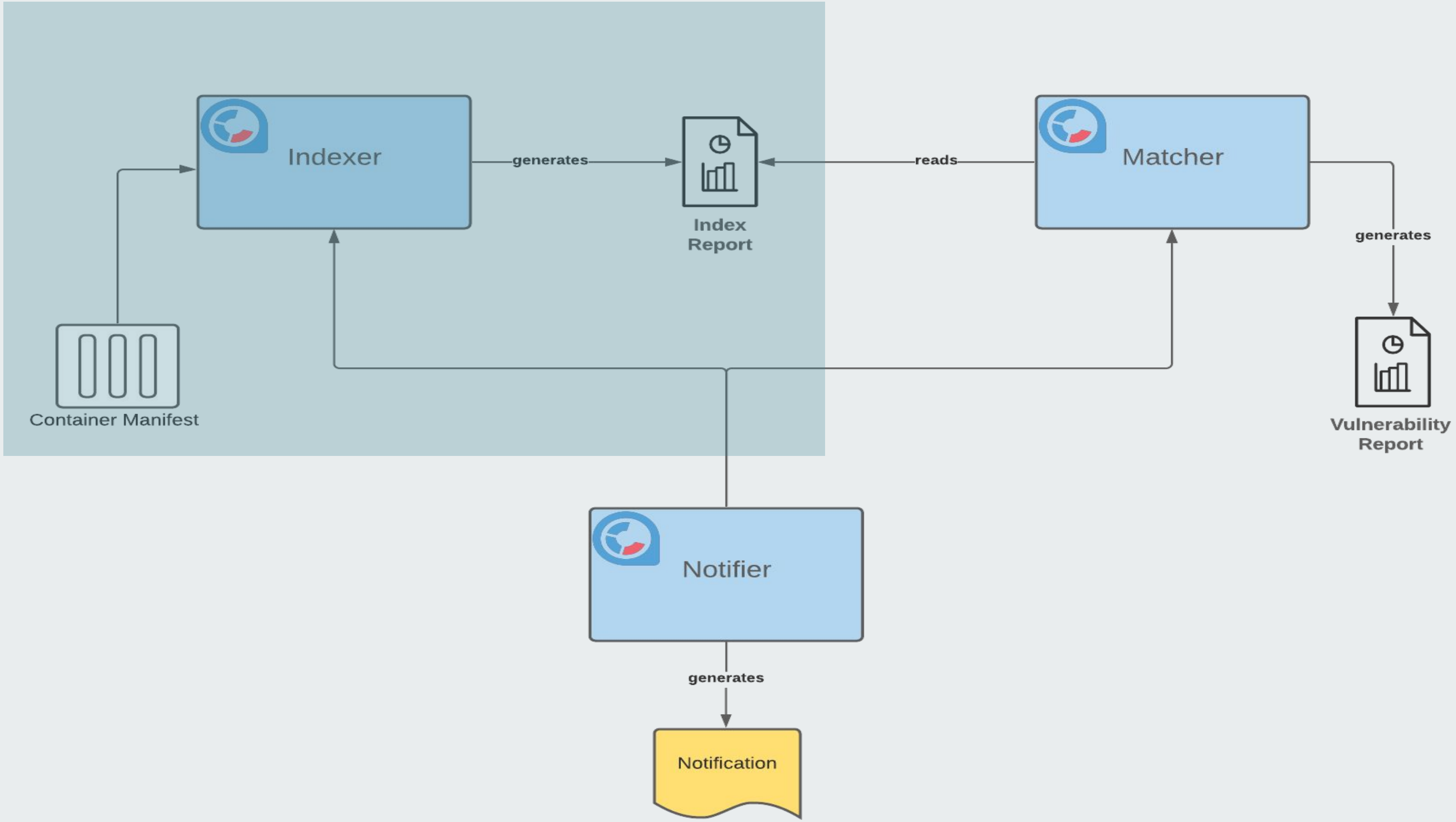
How Clair V4 extracts and persists the contents of containers.

By: Louis DeLosSantos
Principal Software Engineer, Clair

Introduction

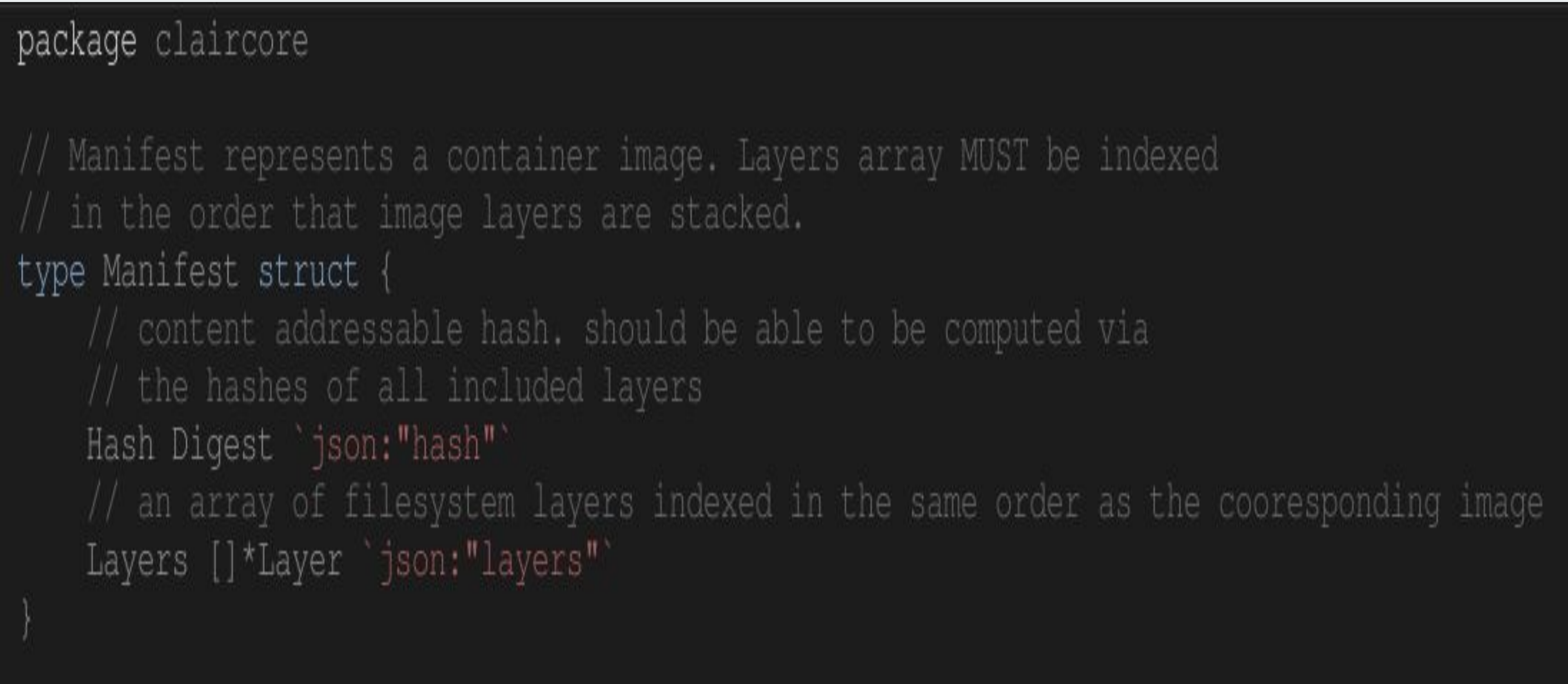
What is "Indexing"

- First step in Clair's analysis pipeline
- Responsible for Index Report creation


Indexer
generates
Index Report
reads
Matcher
generates
Vulnerability Report
Container Manifest
Notifier
generates
Notification

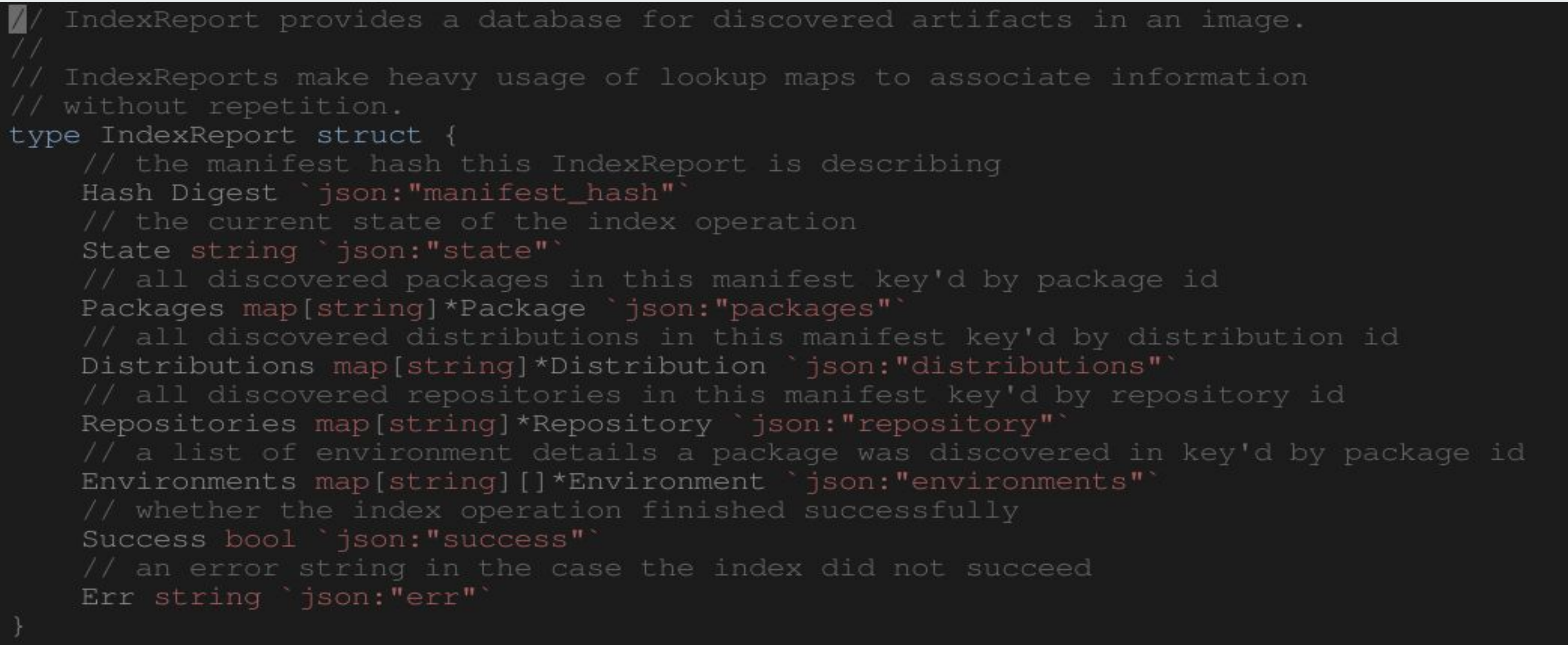
Key Components

Manifests



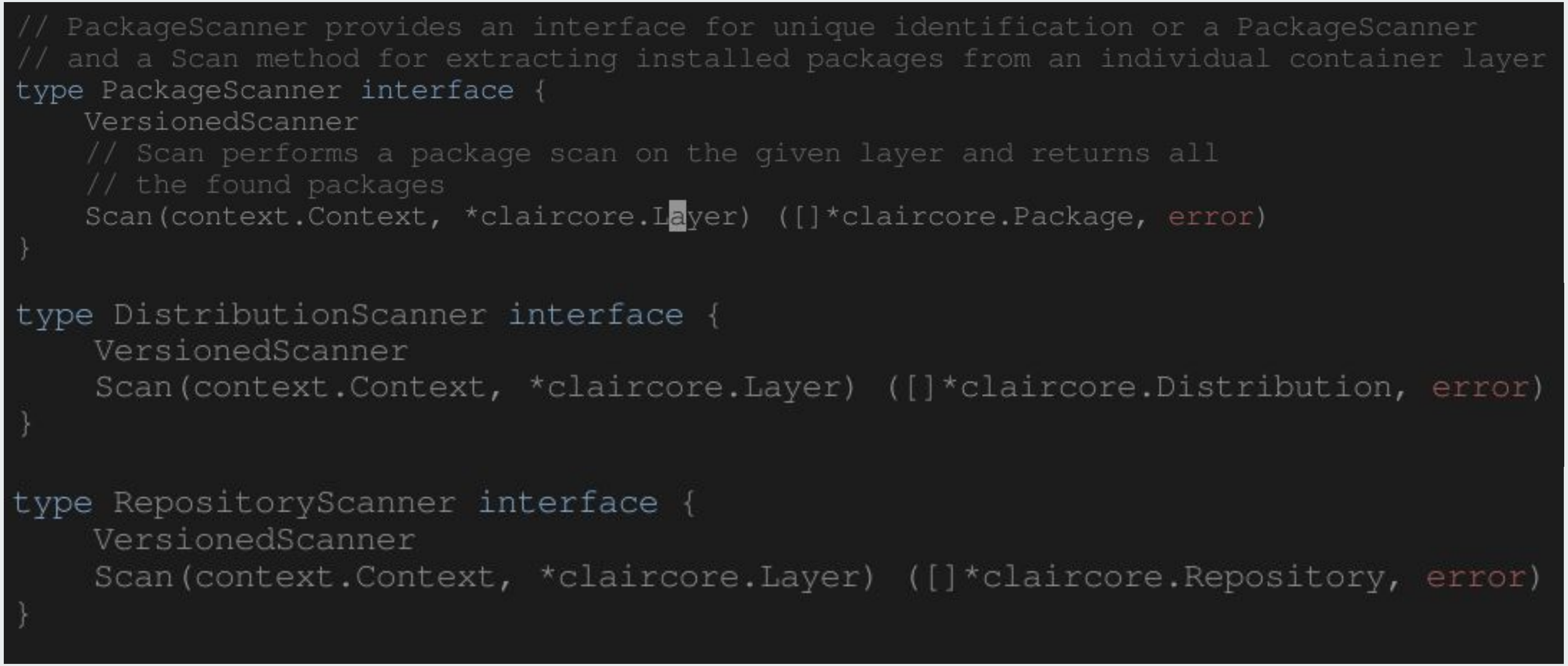
```go
package claircore

// Manifest represents a container image. Layers array MUST be indexed
// in the order that image layers are stacked.
type Manifest struct {
	// content addressable hash. should be able to be computed via
	// the hashes of all included layers
	Hash Digest `json:"hash"`
	// an array of filesystem layers indexed in the same order as the cooresponding image
	Layers []*Layer `json:"layers"`
}
```

Index Reports



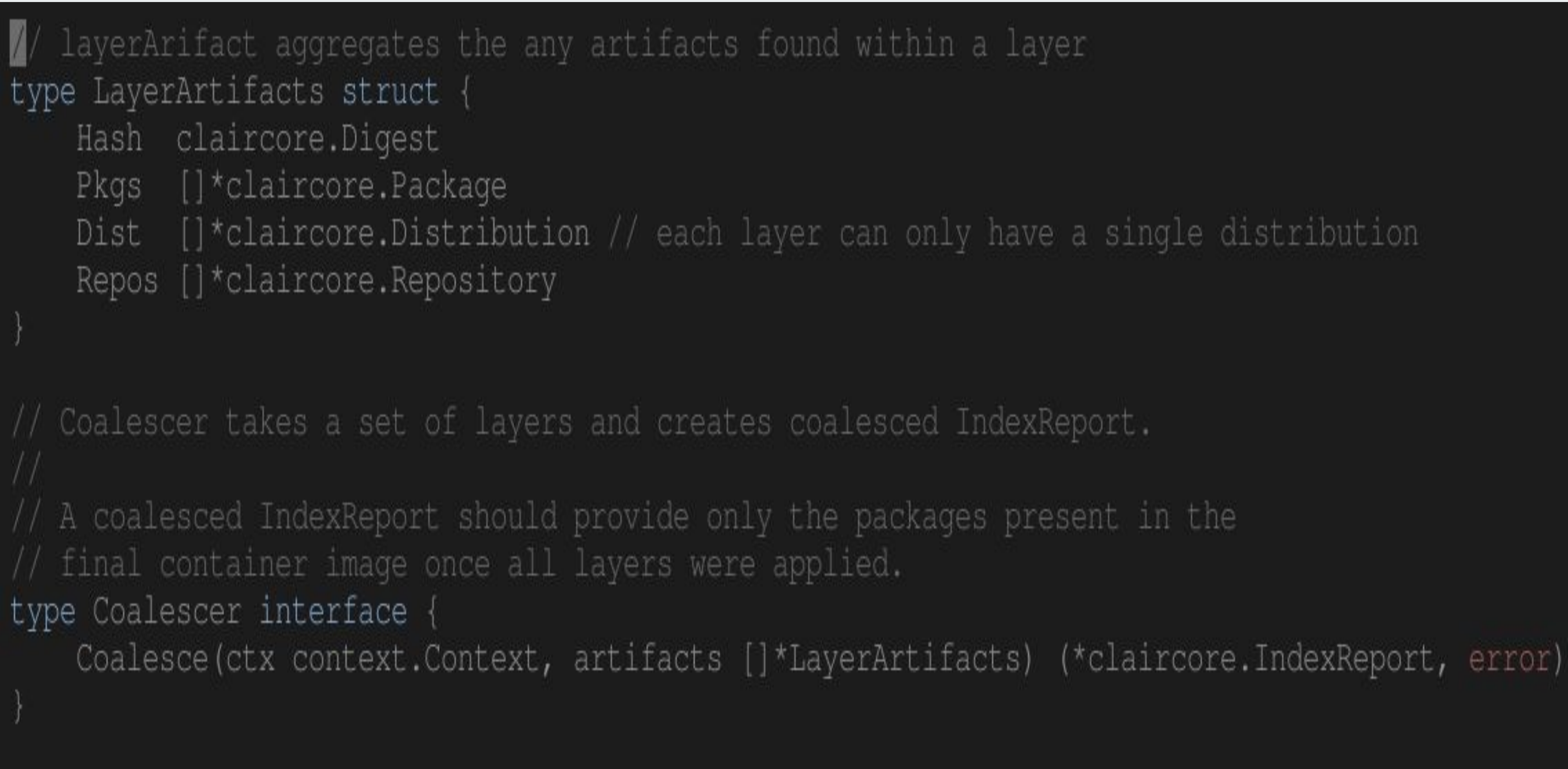
```go
// IndexReport provides a database for discovered artifacts in an image.
//
// IndexReports make heavy usage of lookup maps to associate information
// without repetition.
type IndexReport struct {
    // the manifest hash this IndexReport is describing
    Hash Digest `json:"manifest_hash"`
    // the current state of the index operation
    State string `json:"state"`
    // all discovered packages in this manifest key'd by package id
    Packages map[string]*Package `json:"packages"`
    // all discovered distributions in this manifest key'd by distribution id
    Distributions map[string]*Distribution `json:"distributions"`
    // all discovered repositories in this manifest key'd by repository id
    Repositories map[string]*Repository `json:"repository"`
    // a list of environment details a package was discovered in key'd by package id
    Environments map[string][]*Environment `json:"environments"`
    // whether the index operation finished successfully
    Success bool `json:"success"`
    // an error string in the case the index did not succeed
    Err string `json:"err"`
}
```

Scanner Interfaces



```go
// PackageScanner provides an interface for unique identification or a PackageScanner
// and a Scan method for extracting installed packages from an individual container layer
type PackageScanner interface {
    VersionedScanner
    // Scan performs a package scan on the given layer and returns all
    // the found packages
    Scan(context.Context, *claircore.Layer) ([]*claircore.Package, error)
}

type DistributionScanner interface {
    VersionedScanner
    Scan(context.Context, *claircore.Layer) ([]*claircore.Distribution, error)
}

type RepositoryScanner interface {
    VersionedScanner
    Scan(context.Context, *claircore.Layer) ([]*claircore.Repository, error)
}
```

Coalescer

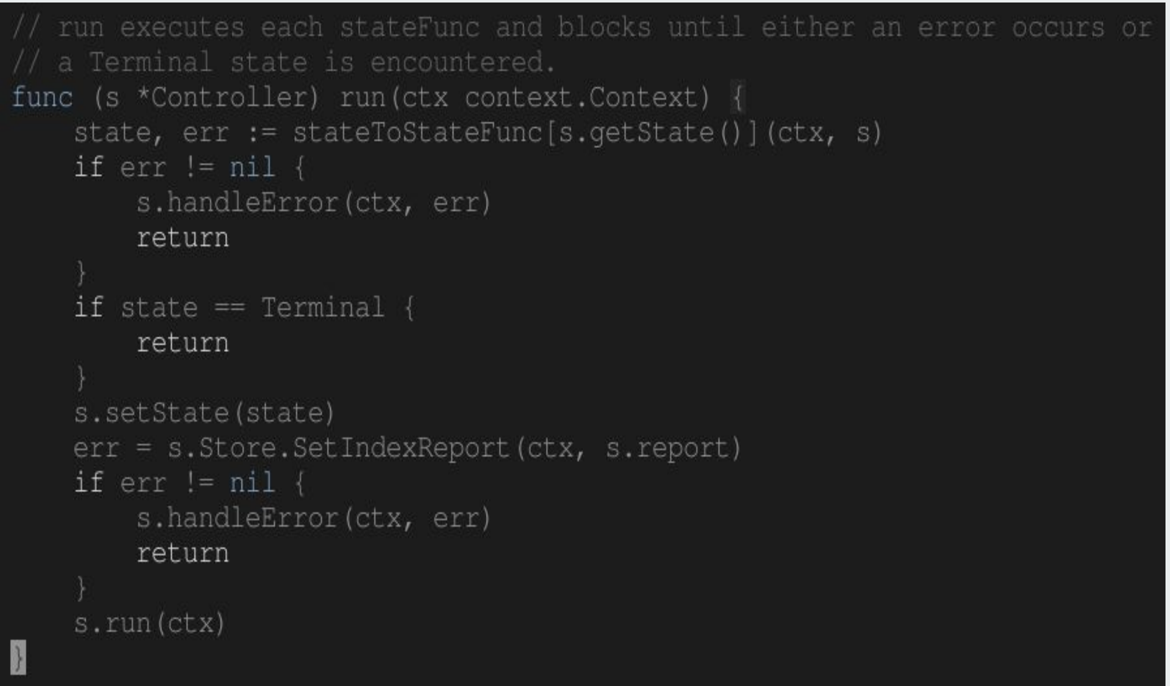


```go
// layerArifact aggregates the any artifacts found within a layer
type LayerArtifacts struct {
    Hash  claircore.Digest
    Pkgs  []*claircore.Package
    Dist  []*claircore.Distribution // each layer can only have a single distribution
    Repos []*claircore.Repository
}

// Coalescer takes a set of layers and creates coalesced IndexReport.
//
// A coalesced IndexReport should provide only the packages present in the
// final container image once all layers were applied.
type Coalescer interface {
    Coalesce(ctx context.Context, artifacts []*LayerArtifacts) (*claircore.IndexReport, error)
}
```

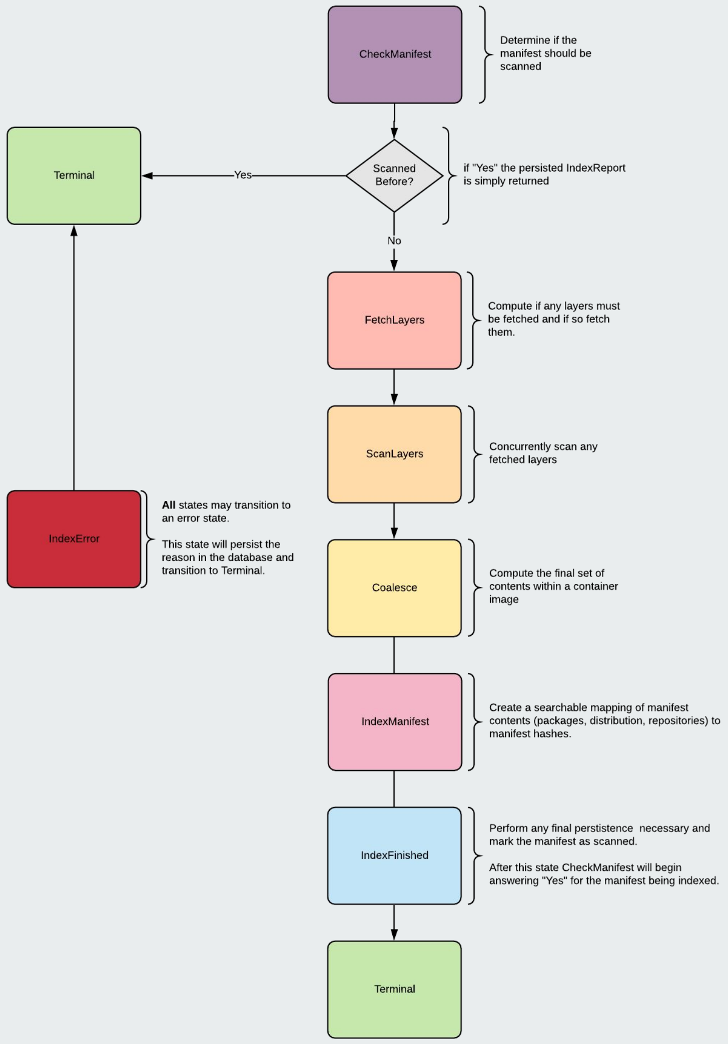
Architecture

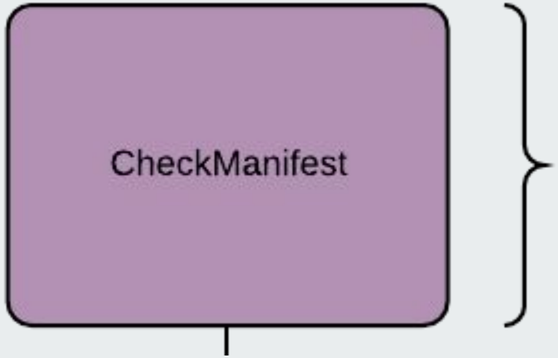
- RESTful HTTP API
- Finite State Machine

Finite State Machine

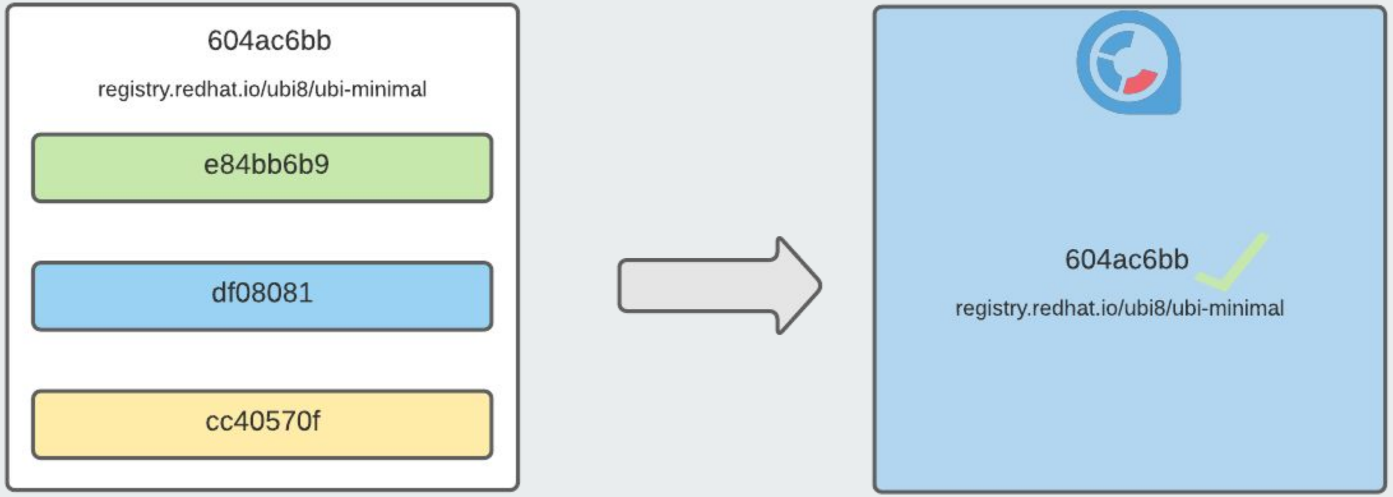


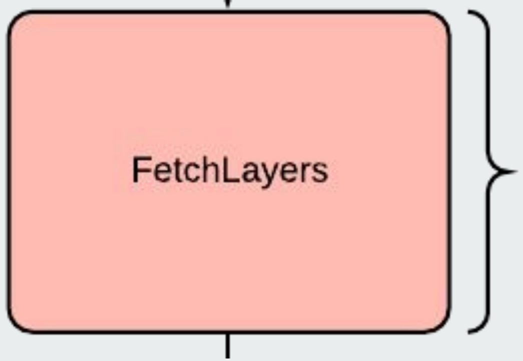
```
// run executes each stateFunc and blocks until either an error occurs or
// a Terminal state is encountered.
func (s *Controller) run(ctx context.Context) {
    state, err := stateToStateFunc[s.getState()](ctx, s)
    if err != nil {
        s.handleError(ctx, err)
        return
    }
    if state == Terminal {
        return
    }
    s.setState(state)
    err = s.Store.SetIndexReport(ctx, s.report)
    if err != nil {
        s.handleError(ctx, err)
        return
    }
    s.run(ctx)
}
```


CheckManifest
Determine if the manifest should be scanned
Scanned Before?
Yes
Terminal
if "Yes" the persisted IndexReport is simply returned
No
FetchLayers
Compute if any layers must be fetched and if so fetch them.
ScanLayers
Concurrently scan any fetched layers
IndexError
All states may transition to an error state.
This state will persist the reason in the database and transition to Terminal.
Coalesce
Compute the final set of contents within a container image
IndexManifest
Create a searchable mapping of manifest contents (packages, distribution, repositories) to manifest hashes.
IndexFinished
Perform any final perstistence necessary and mark the manifest as scanned.
After this state CheckManifest will begin answering "Yes" for the manifest being indexed.
Terminal

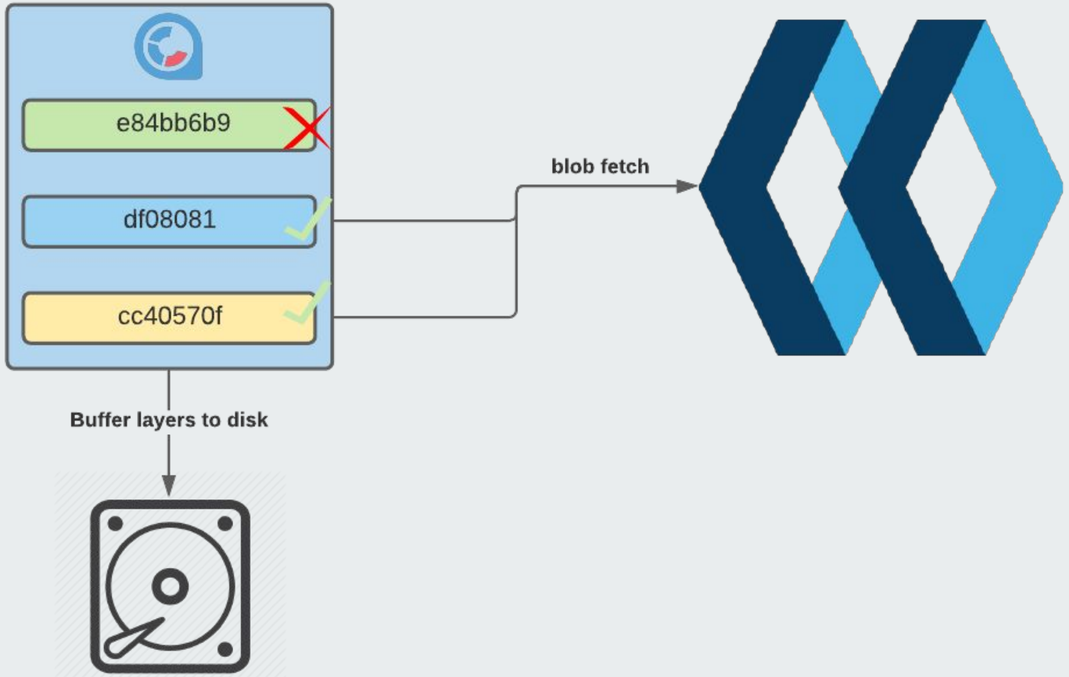


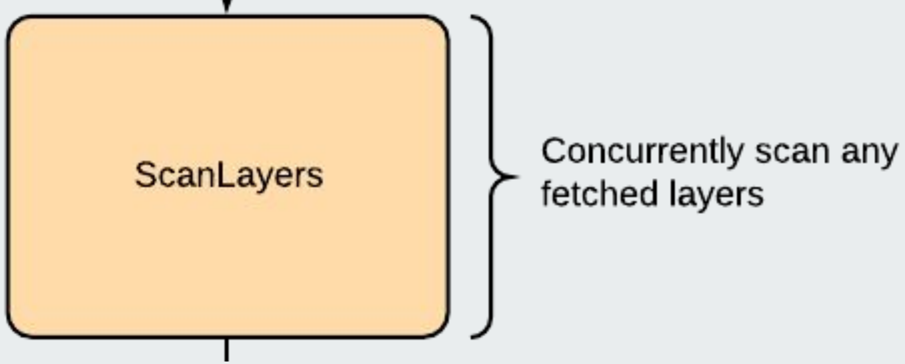
Determine if the manifest should be scanned

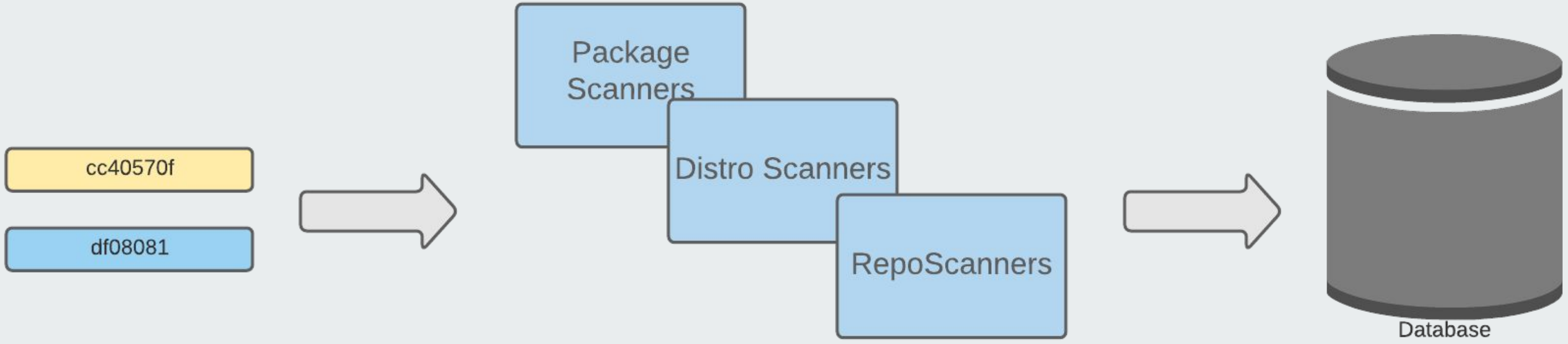

604ac6bb
registry.redhat.io/ubi8/ubi-minimal
e84bb6b9
df08081
cc40570f
604ac6bb
registry.redhat.io/ubi8/ubi-minimal



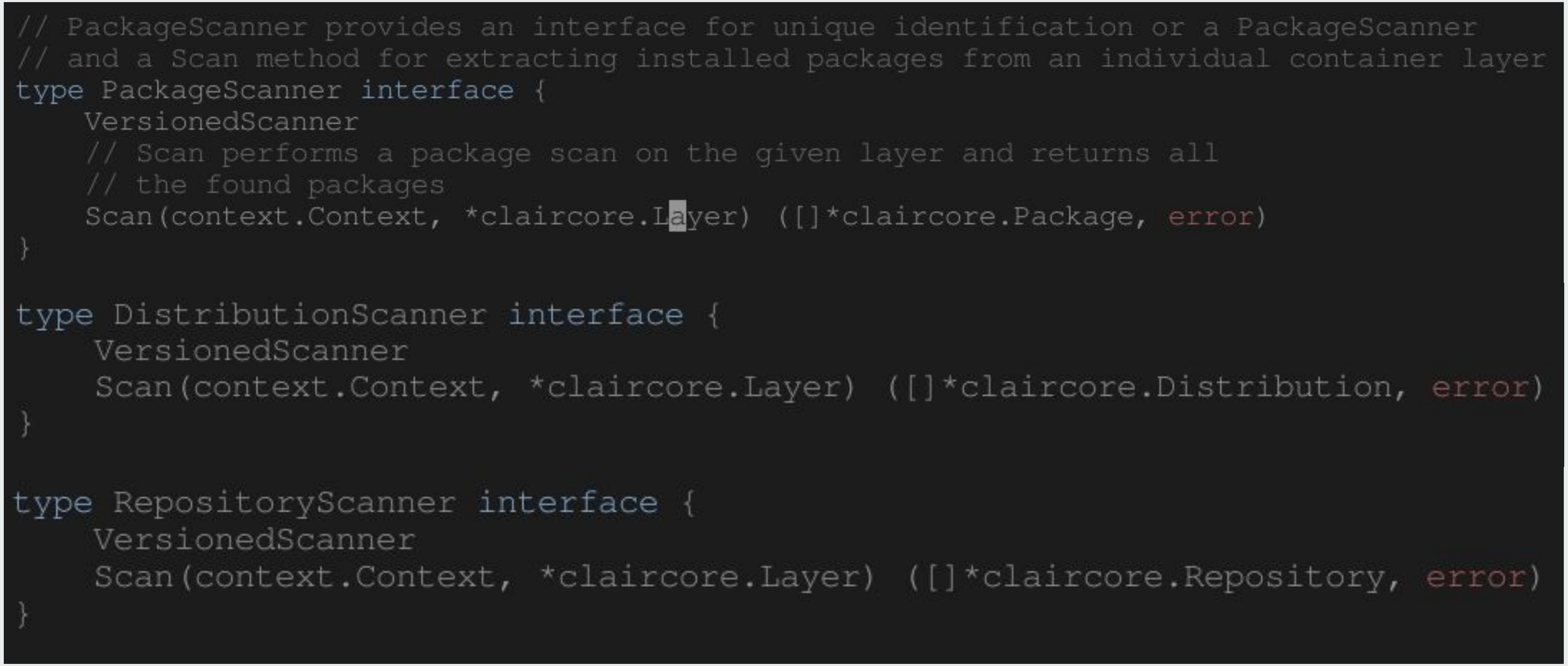
Compute if any layers must be fetched and if so fetch them.




ScanLayers
Concurrently scan any fetched layers

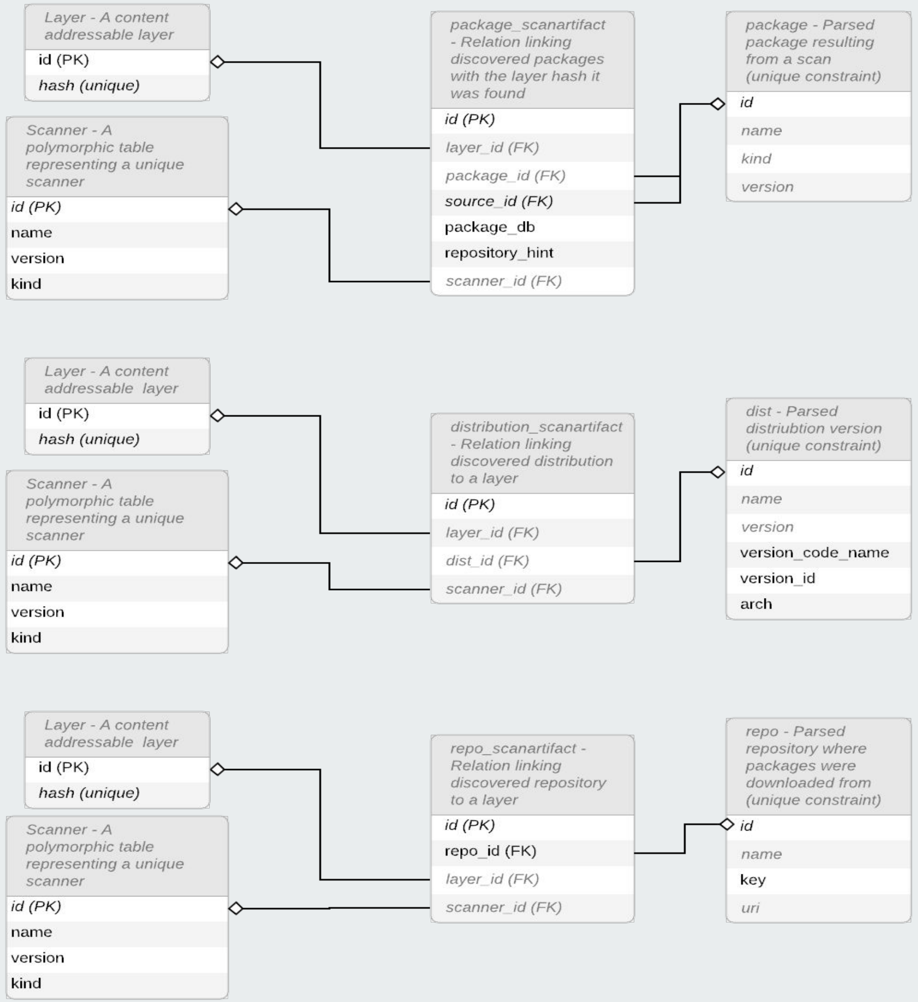

cc40570f
df08081
Package Scanners
Distro Scanners
RepoScanners
Database

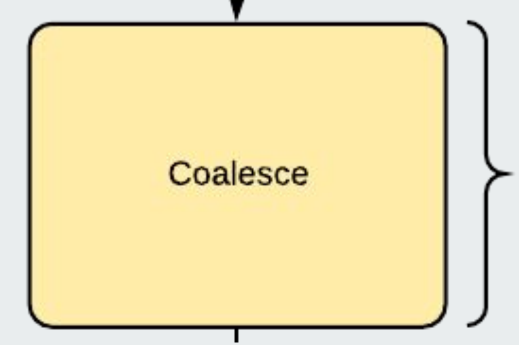
Scanner Interfaces



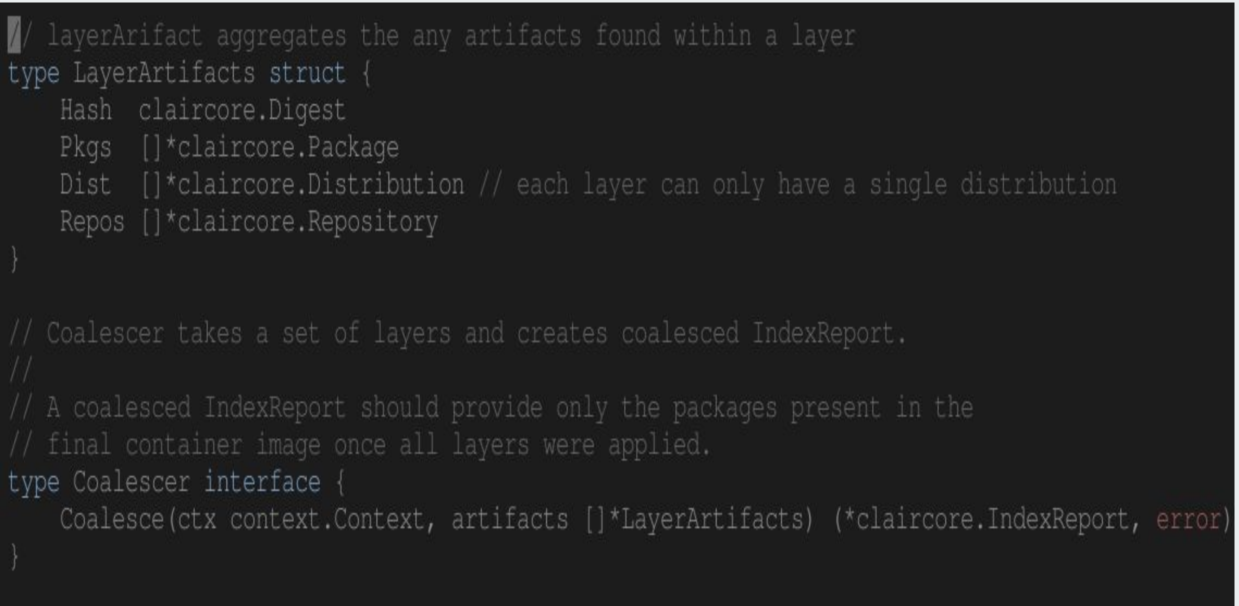
```go
// PackageScanner provides an interface for unique identification or a PackageScanner
// and a Scan method for extracting installed packages from an individual container layer
type PackageScanner interface {
    VersionedScanner
    // Scan performs a package scan on the given layer and returns all
    // the found packages
    Scan(context.Context, *claircore.Layer) ([]*claircore.Package, error)
}

type DistributionScanner interface {
    VersionedScanner
    Scan(context.Context, *claircore.Layer) ([]*claircore.Distribution, error)
}

type RepositoryScanner interface {
    VersionedScanner
    Scan(context.Context, *claircore.Layer) ([]*claircore.Repository, error)
}
```

Scan Artifacts

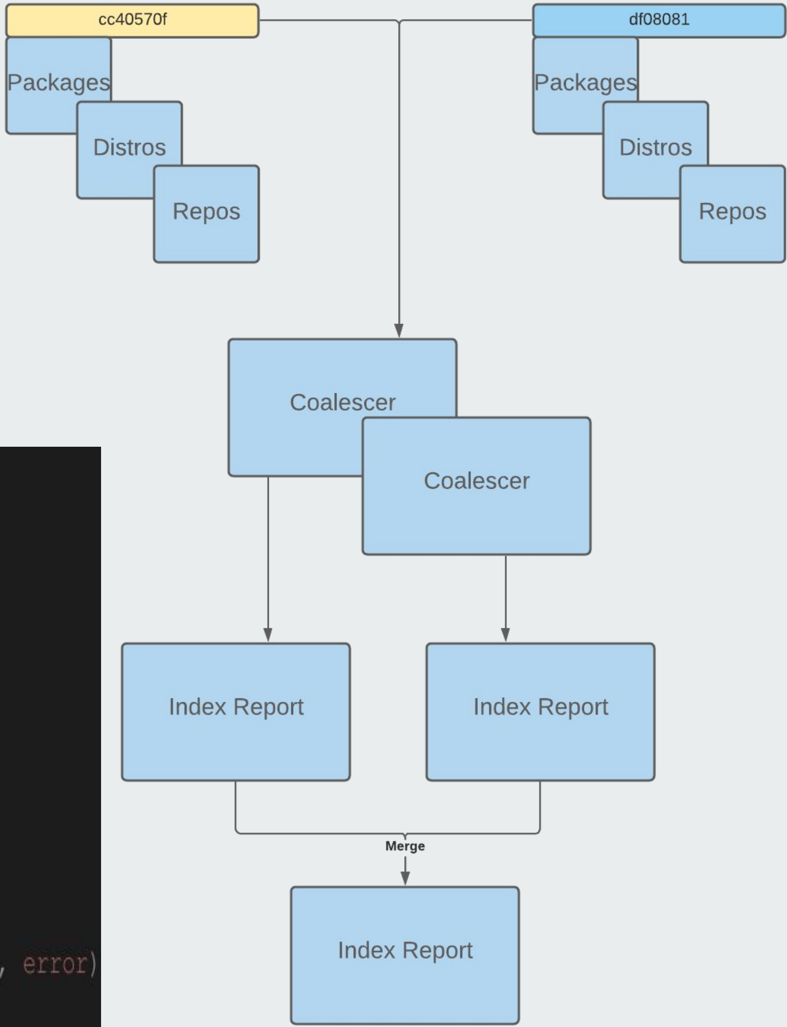

Layer - A content addressable layer
id (PK)
hash (unique)
Scanner - A polymorphic table representing a unique scanner
id (PK)
name
version
kind
package_scanartifact - Relation linking discovered packages with the layer hash it was found
id (PK)
layer_id (FK)
package_id (FK)
source_id (FK)
package_db
repository_hint
scanner_id (FK)
package - Parsed package resulting from a scan (unique constraint)
id
name
kind
version
Layer - A content addressable layer
id (PK)
hash (unique)
Scanner - A polymorphic table representing a unique scanner
id (PK)
name
version
kind
distribution_scanartifact - Relation linking discovered distribution to a layer
id (PK)
layer_id (FK)
dist_id (FK)
scanner_id (FK)
dist - Parsed distriubtion version (unique constraint)
id
name
version
version_code_name
version_id
arch
Layer - A content addressable layer
id (PK)
hash (unique)
Scanner - A polymorphic table representing a unique scanner
id (PK)
name
version
kind
repo_scanartifact - Relation linking discovered repository to a layer
id (PK)
repo_id (FK)
layer_id (FK)
scanner_id (FK)
repo - Parsed repository where packages were downloaded from (unique constraint)
id
name
key
uri

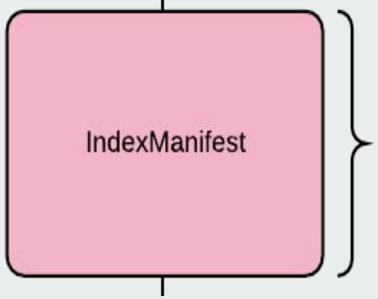


Compute the final set of contents within a container image

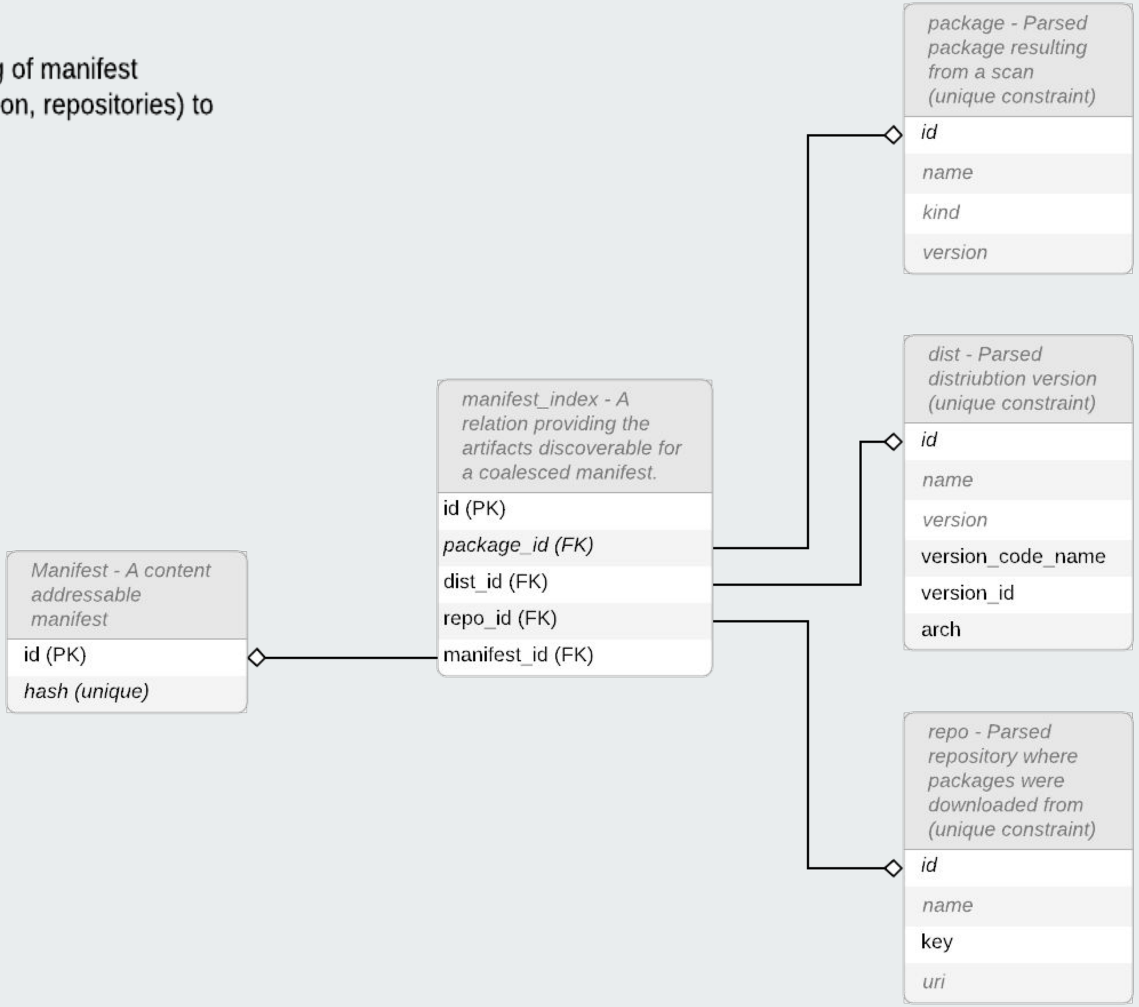


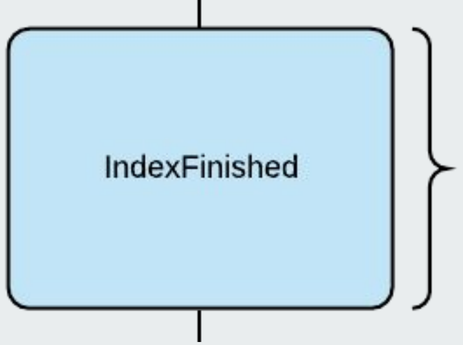
```
// layerArifact aggregates the any artifacts found within a layer
type LayerArtifacts struct {
    Hash  claircore.Digest
    Pkgs  []*claircore.Package
    Dist  []*claircore.Distribution // each layer can only have a single distribution
    Repos []*claircore.Repository
}

// Coalescer takes a set of layers and creates coalesced IndexReport.
//
// A coalesced IndexReport should provide only the packages present in the
// final container image once all layers were applied.
type Coalescer interface {
    Coalesce(ctx context.Context, artifacts []*LayerArtifacts) (*claircore.IndexReport, error)
}
```





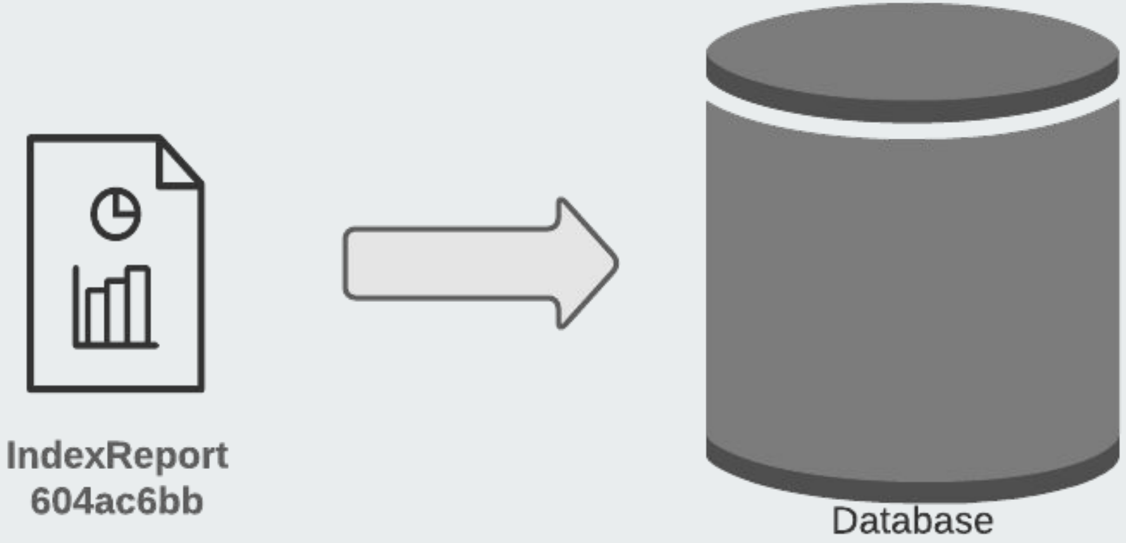
Create a searchable mapping of manifest contents (packages, distribution, repositories) to manifest hashes.


Manifest - A content addressable manifest
id (PK)
hash (unique)
manifest_index - A relation providing the artifacts discoverable for a coalesced manifest.
id (PK)
package_id (FK)
dist_id (FK)
repo_id (FK)
manifest_id (FK)
package - Parsed package resulting from a scan (unique constraint)
id
name
kind
version
dist - Parsed distriubtion version (unique constraint)
id
name
version
version_code_name
version_id
arch
repo - Parsed repository where packages were downloaded from (unique constraint)
id
name
key
uri



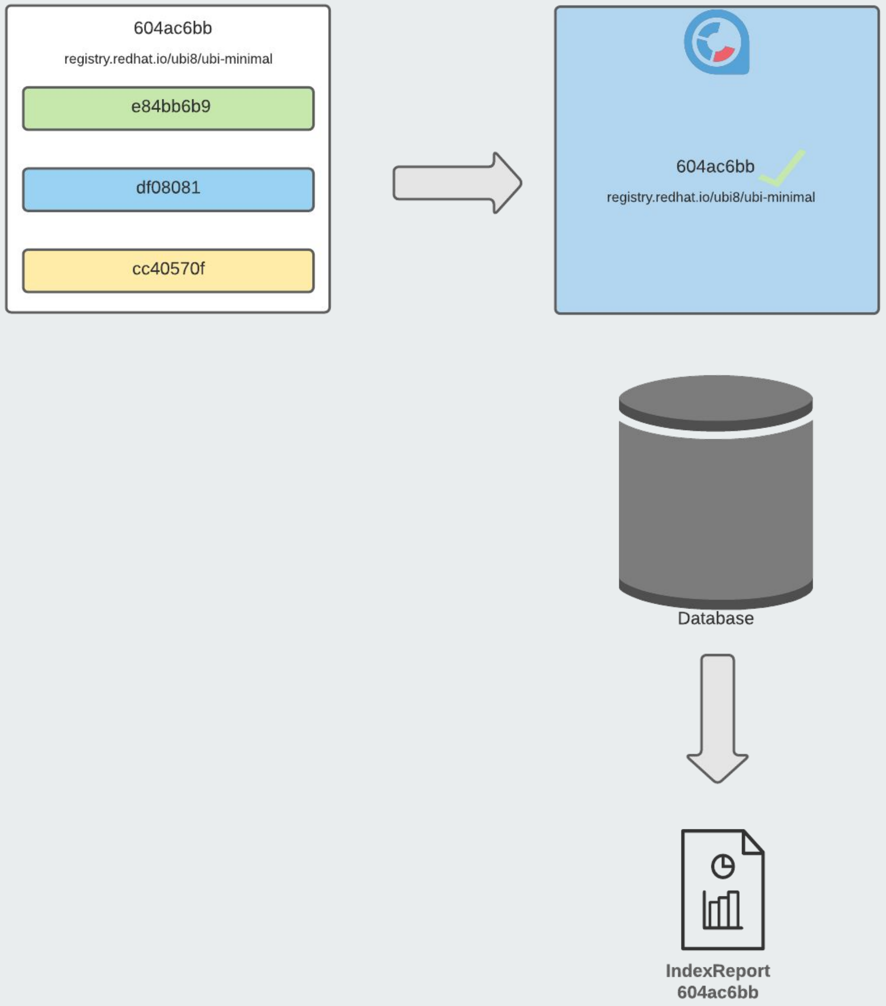
Perform any final perstistence necessary and mark the manifest as scanned.

After this state CheckManifest will begin answering "Yes" for the manifest being indexed.

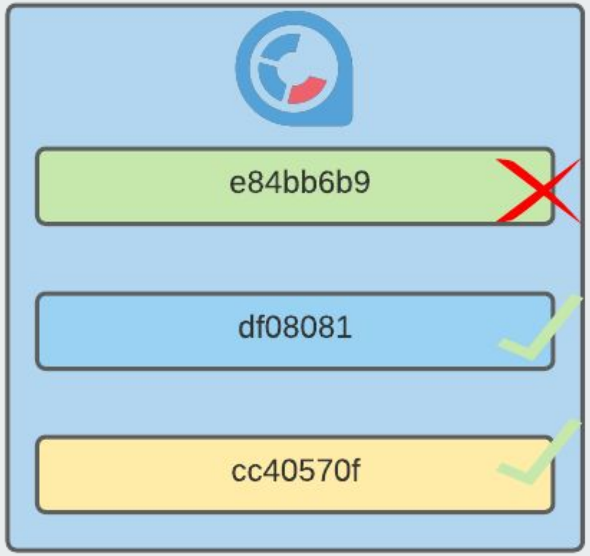


Deferring Work

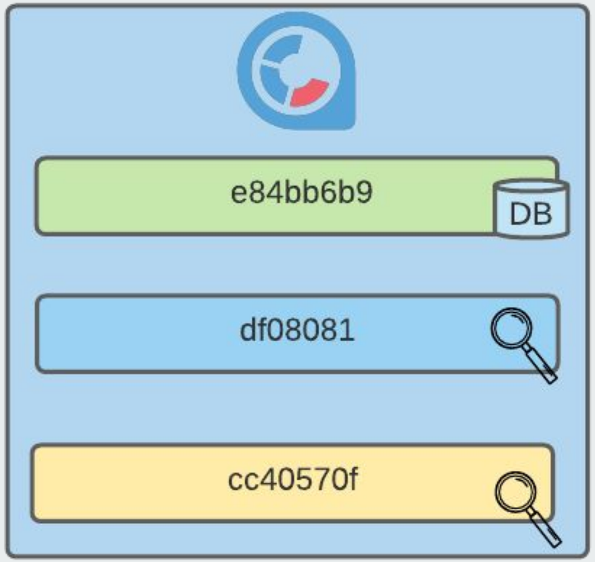
Manifest Seen


604ac6bb
registry.redhat.io/ubi8/ubi-minimal
e84bb6b9
df08081
cc40570f
604ac6bb
registry.redhat.io/ubi8/ubi-minimal
Database
IndexReport
604ac6bb

Determine layers to scan



Scan only necessary layers



Info

- ldelossa@redhat.com
- https://github.com/quay/clair
- https://github.com/quay/claircore/