

deploy-sourcegraph in hall

proof of concept

$f(\text{base dhall, overlay dhall}) \rightarrow \text{customized deployment dhall}$

presentation in four parts

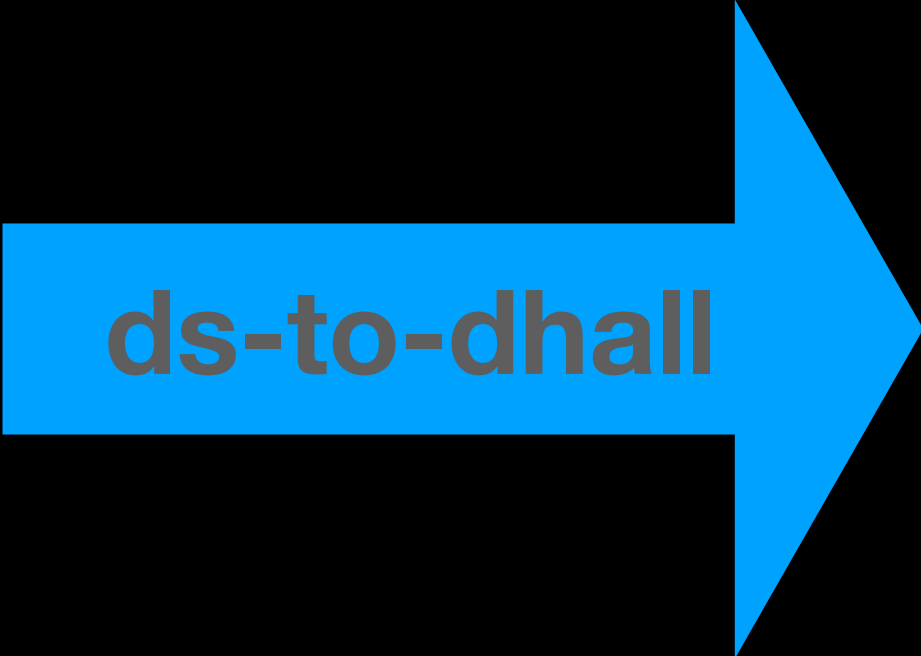
- base dhall
- overlay dhall
- function f
- implementation

part one: base

```
backend.Service.yaml
cadvisor
├── README.md
├── cadvisor.ClusterRole.yaml
├── cadvisor.ClusterRoleBinding.yaml
├── cadvisor.DaemonSet.yaml
├── cadvisor.PodSecurityPolicy.yaml
├── cadvisor.ServiceAccount.yaml
frontend
├── sourcegraph-frontend-internal.Service.yaml
├── sourcegraph-frontend.Deployment.yaml
├── sourcegraph-frontend.Ingress.yaml
├── sourcegraph-frontend.Role.yaml
├── sourcegraph-frontend.RoleBinding.yaml
├── sourcegraph-frontend.Service.yaml
├── sourcegraph-frontend.ServiceAccount.yaml
github-proxy
├── github-proxy.Deployment.yaml
├── github-proxy.Service.yaml
gitserver
├── gitserver.Service.yaml
├── gitserver.StatefulSet.yaml
grafana
├── README.md
├── grafana.ConfigMap.yaml
├── grafana.Service.yaml
├── grafana.ServiceAccount.yaml
├── grafana.StatefulSet.yaml
..
redis
├── redis-cache.Deployment.yaml
├── redis-cache.PersistentVolumeClaim.yaml
├── redis-cache.Service.yaml
├── redis-store.Deployment.yaml
├── redis-store.PersistentVolumeClaim.yaml
├── redis-store.Service.yaml
repo-updater
├── repo-updater.Deployment.yaml
├── repo-updater.Service.yaml
searcher
├── searcher.Deployment.yaml
├── searcher.Service.yaml
sourcegraph.StorageClass.yaml
symbols
├── symbols.Deployment.yaml
├── symbols.Service.yaml
syntect-server
├── syntect-server.Deployment.yaml
├── syntect-server.Service.yaml
```

base : Base

```
Frontend :
{ Deployment :
  { sourcegraph-frontend :
    { apiVersion : Text
    , kind : Text
    , metadata :
      { annotations : { description : Text }
      , labels :
        { `app.kubernetes.io/component` : Text
        , deploy : Text
        , sourcegraph-resource-requires : Text
        }
      , name : Text
      , namespace : Optional Text
    }
    , spec :
      { minReadySeconds : Natural
      , replicas : Natural
      , revisionHistoryLimit : Natural
      , selector : { matchLabels : { app : Text } }
      , strategy :
        { rollingUpdate :
          { maxSurge : Natural, maxUnavailable : Natural }
          , type : Text
        }
      , template :
        { metadata :
          { labels : { app : Text, deploy : Text }
          , namespace : Optional Text
        }
        , spec :
          { containers :
            { frontend :
```



Component :> Kind :> Name :> contents from manifest

base : Base

```
Frontend :
  { Deployment :
    { sourcegraph-frontend :
      { spec :
        { template :
          spec :
            { containers :
              { frontend :
                { args : List Text
                , env :
                  { CACHE_DIR :
                    Optional { name : Text, value : Text }
                  , GRAFANA_SERVER_URL :
                    Optional { name : Text, value : Text }
                  , JAEGER_SERVER_URL :
                    Optional { name : Text, value : Text }
                }
              , image :
                < asText : Text
                | asRecord :
                  { name : Text
                  , registry : Text
                  , sha256 : Text
                  , version : Text
                  }
                >
              , jaeger-agent :
```

schema improvements:

- convert lists into records with keys from list member's name field
- make image a union type of Text and record of image parts
- add a few Optionals for fields not declared in the original base yaml manifests
- add additionalEnv field for additional env vars
- make env values Optional

type.dhall

```
, Frontend :
{ Deployment :
  { sourcegraph-frontend :
    { apiVersion : Text
    , kind : Text
    , metadata :
      { annotations : { description : Text }
      , labels :
        { `app.kubernetes.io/component` : Text
        , deploy : Text
        , sourcegraph-resource-requires : Text
        }
      , name : Text
      , namespace : Optional Text
    }
  , spec :
    { minReadySeconds : Natural
    , replicas : Natural
    , revisionHistoryLimit : Natural
    , selector : { matchLabels : { app : Text } }
    , strategy :
      { rollingUpdate :
        { maxSurge : Natural, maxUnavailable : Natural }
        , type : Text
      }
    , template :
      { metadata :
        { labels : { app : Text, deploy : Text }
        , namespace : Optional Text
      }
      , spec :
        { containers :
          { frontend :
            { args : List Text
            , env :
              { CACHE_DIR :
                Optional { name : Text, value : Text }
              , GRAFANA_SERVER_URL :
                Optional { name : Text, value : Text }
              , JAEGER_SERVER_URL :
                Optional { name : Text, value : Text }
              Optional { name : Text, value : Text }
            }
          }
        , image :
          < asText : Text
          | asRecord :
            { name : Text
            , registry : Text
            , sha256 : Text
            , version : Text
            }
          >
        }
      }
    }
  }
}
```

ds-to-dhall

schema.dhall

{ Type = ./type.dhall, default = ./record.dhall }

record.dhall

```
, Frontend =
{ Deployment.sourcegraph-frontend =
  { apiVersion = "apps/v1"
  , kind = "Deployment"
  , metadata =
    { annotations.description =
      "Serves the frontend of Sourcegraph via HTTP(S)."
    , labels =
      { `app.kubernetes.io/component` = "frontend"
      , deploy = "sourcegraph"
      , sourcegraph-resource-requires = "no-cluster-admin"
      }
    , name = "sourcegraph-frontend"
    , namespace = None Text
  }
, spec =
  { minReadySeconds = 10
  , replicas = 1
  , revisionHistoryLimit = 10
  , selector.matchLabels.app = "sourcegraph-frontend"
  , strategy =
    { rollingUpdate = { maxSurge = 2, maxUnavailable = 0 }
    , type = "RollingUpdate"
    }
  , template =
    { metadata =
      { labels = { app = "sourcegraph-frontend", deploy = "sourcegraph" }
      , namespace = None Text
    }
    , spec =
      { containers =
        { frontend =
          { additionalEnv = None (List { name : Text, value : Text })
          , args = [ "serve" ]
          , env =
            { CACHE_DIR = Some
              { name = "CACHE_DIR", value = "/mnt/cache/$(POD_NAME)" }
            , GRAFANA_SERVER_URL = Some
              { name = "GRAFANA_SERVER_URL"
              , value = "http://grafana:30070"
              }
            , JAEGER_SERVER_URL = Some
              { name = "JAEGER_SERVER_URL"
              , value = "http://jaeger-query:16686"
              }
            }
          }
        }
      , image =
        < asRecord :
          { name : Text
          , registry : Text
          , sha256 : Text
          , version : Text
          }
        | asText : Text
      >.asRecord
      { name = "sourcegraph/frontend"
      , registry = "index.docker.io"
      , sha256 =
        "776606b680d7ce4a5d37451831ef2414ab10414b5e945ed5f50fe768f898d23f"
      , version = "3.19.2"
      }
    }
  }
}
```

part two: overlay

overlay : Overlay

```
let customization/container = ./container.dhall
```

```
let container/image = ./image.dhall
```

```
let schema =
  { Type =
    { Shared :
      { namespace : Optional Text,
        imageMods : container/image.Type }
    }
  ⋈ { Precise-Code-Intel :
    { Deployment :
      { precise-code-intel-bundle-manager :
        { containers :
          { precise-code-intel-bundle-manager :
            customization/container.Type
          }
        }
      }
    }
  ⋈ { Precise-Code-Intel :
    { Deployment :
      { precise-code-intel-worker :
        { containers :
          { precise-code-intel-worker :
            customization/container.Type
          }
        }
      }
    }
  ⋈ { Redis :
```

```
let container/resources = ../combinators/container-resources.dhall
```

```
let vsType = { version : Text, sha256 : Text }
```

```
let container =
  { Type =
    { resources : container/resources.Type
      , vs : Optional vsType
      , additionalEnv : Optional (List { name : Text, value : Text })
    }
  , default =
    { resources = container/resources.default
      , vs = None vsType
      , additionalEnv = None (List { name : Text, value : Text })
    }
  }
```

```
in container
```

```
let image =
  { Type =
    { prefix : Optional Text
      , suffix : Optional Text
      , registry : Optional Text
      , omitRegistry : Bool
      , omitSha256 : Bool
    }
  , default =
    { prefix = None Text
      , suffix = None Text
      , registry = None Text
      , omitRegistry = False
      , omitSha256 = False
    }
  }
```

```
in image
```



```
let overlay = overlaySchema::{=}
```

```
let overlay = overlay with Shared.imageMods.registry = Some "google.io"
```

```
let overlay = overlay with Shared.namespace = Some "ns-sourcegraph"
```

```
let overlay = overlay  
  with Symbols.Deployment.symbols.containers.jaeger-agent.resources.limits.cpu = Some "500m"
```

```
in overlay
```

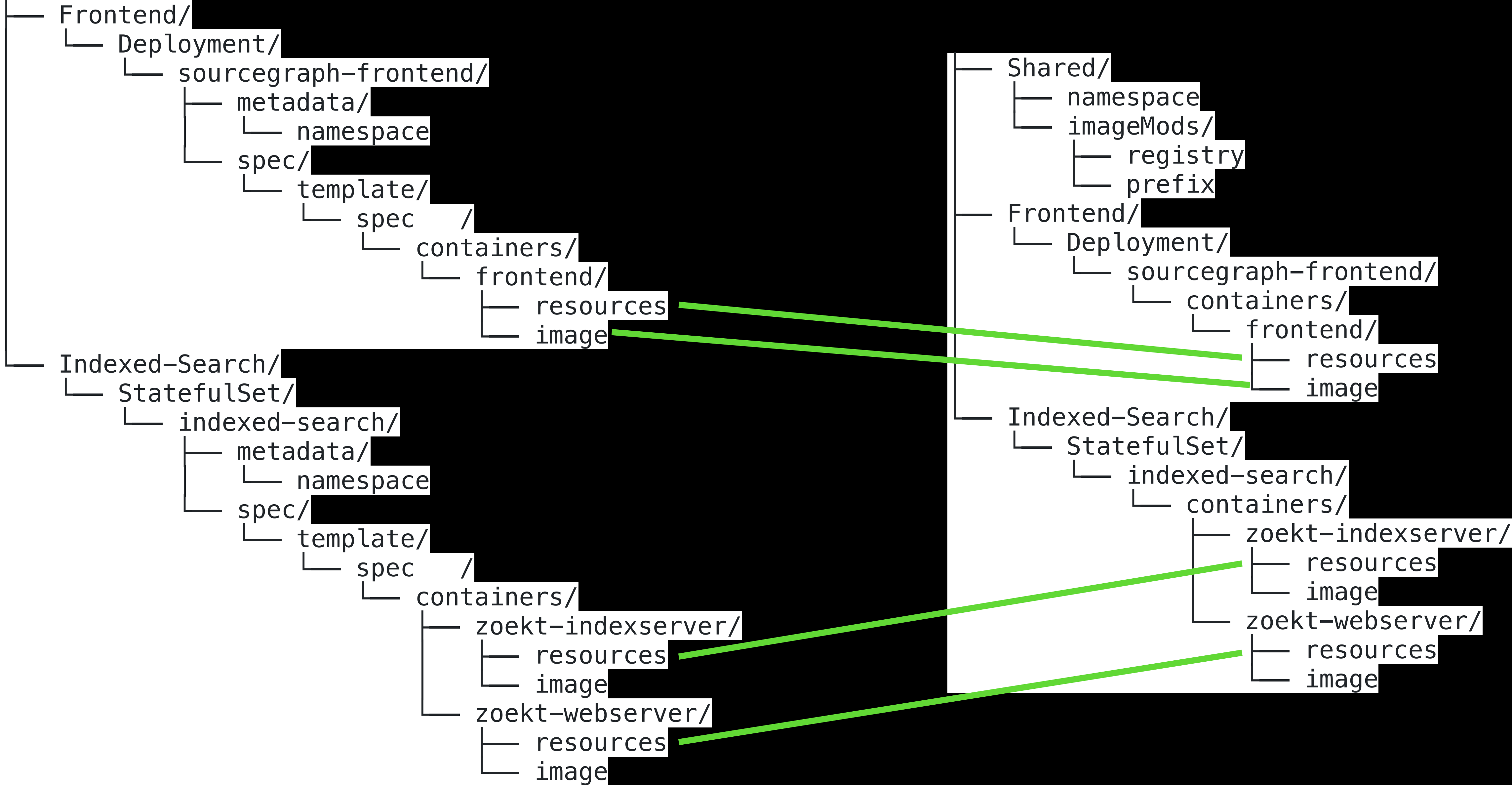
part three: function f

function f combines $base$ with $overlay$
into customized deployment d_{hall}
with the same type as $base$

$$f : (\forall \text{ Base}, \forall \text{ Overlay}) \rightarrow \text{Base}$$

base

overlay



- extract resources from base record tree
- extract resources from overlay record tree
- combine them
- return base record with modified resources

foreach Component
foreach Deployment
for each container

```
baseResources = Component.Deployment.Name....containers.name.resources  
overlayResources = Component.Deployment.Name.containers.name.resources  
finalResources = combineResources(baseResources, overlayResources)  
return base with Component.Deployment.Name....containers.name.resources = finalResources
```

paths in base tree:

Frontend/Deployment/sourcegraph-frontend/spec/template/spec/containers/frontend/resources
Indexed-Search/StatefulSet/indexed-search/spec/template/spec/containers/zoekt-indexserver/resources
Indexed-Search/StatefulSet/indexed-search/spec/template/spec/containers/zoekt-webserver/resources

paths in overlay tree:

Frontend/Deployment/sourcegraph-frontend/containers/frontend/resources
Indexed-Search/StatefulSet/indexed-search/containers/zoekt-indexserver/resources
Indexed-Search/StatefulSet/indexed-search/containers/zoekt-webserver/resources

combineResources is the same for all the possible paths
in base and overlay trees so **IDEA**

**create one function for each Component.Deployment.container
tuple from a template. each one of these functions has the same
type = $(\forall \text{ Base}, \forall \text{ Overlay}) \rightarrow \text{Base}$, so they can be chained.**

```
let baseSchema = ../base/schema.dhall
```

```
let overlaySchema = ../customizations/schema.dhall
```

```
let resourceCombinator = ../combinators/container-resources.dhall
```

```
{{range .DeploymentTuples}}
```

```
let applyResources{.Identifier}}
```

```
  : baseSchema.Type → overlaySchema.Type → baseSchema.Type
```

```
  = λ(base : baseSchema.Type) →
```

```
    λ(overlay : overlaySchema.Type) →
```

```
      let baseResources =
```

```
        base.{.Component}.Deployment.{.Name}.spec.template.spec.containers.{.ContainerName}.resources
```

```
      let overlayResources =
```

```
        overlay.{.Component}.Deployment.{.Name}.containers.{.ContainerName}.resources
```

```
      let finalResources =
```

```
        resourceCombinator.overlayMerge
```

```
        baseResources
```

```
        overlayResources
```

```
    in base
```

```
      with {.Component}.Deployment.{.Name}.spec.template.spec.containers.{.ContainerName}.resources =
```

```
        finalResources
```

```
{{end}}
```

```
let applyAll
```

```
  : baseSchema.Type → overlaySchema.Type → baseSchema.Type
```

```
  = λ(base : baseSchema.Type) →
```

```
    λ(overlay : overlaySchema.Type) →
```

```
      let r = base
```

```
        {{range .DeploymentTuples}}
```

```
          let r = applyResources{.Identifier} r overlay
```

```
        {{end}}
```

```
      in r
```

```
in applyAll
```

chaining

$f_i : (\forall \text{ Base}, \forall \text{ Overlay}) \rightarrow \text{Base}, \quad 0 \leq i < N$

let b = base

let o = overlay

let r = b

for i in range [0, N)

 let r = $f_i(r, o)$

endfor

in r

f is the chaining of all of the functions created from all of the templates plus specialized functions that were not templated and cover specific paths into the base and overlay trees.

part four: implementation

Tooling

ds-to-dhall

<https://github.com/sourcegraph/ds-to-dhall>

ds-to-dhall

```
-o dhall-base/record.dhall \  
-t dhall-base/type.dhall \  
-s dhall-base/schema.dhall \  
-c dhall-base/components.yaml \  
--strengthenSchema \  
deploy-sourcegraph/base
```

Tooling <https://github.com/uwedeportivo/dhallie>

dhallie (template engine)

dhallie

```
-c ds-dhall/dhall-base/components.yaml \
ds-dhall
```

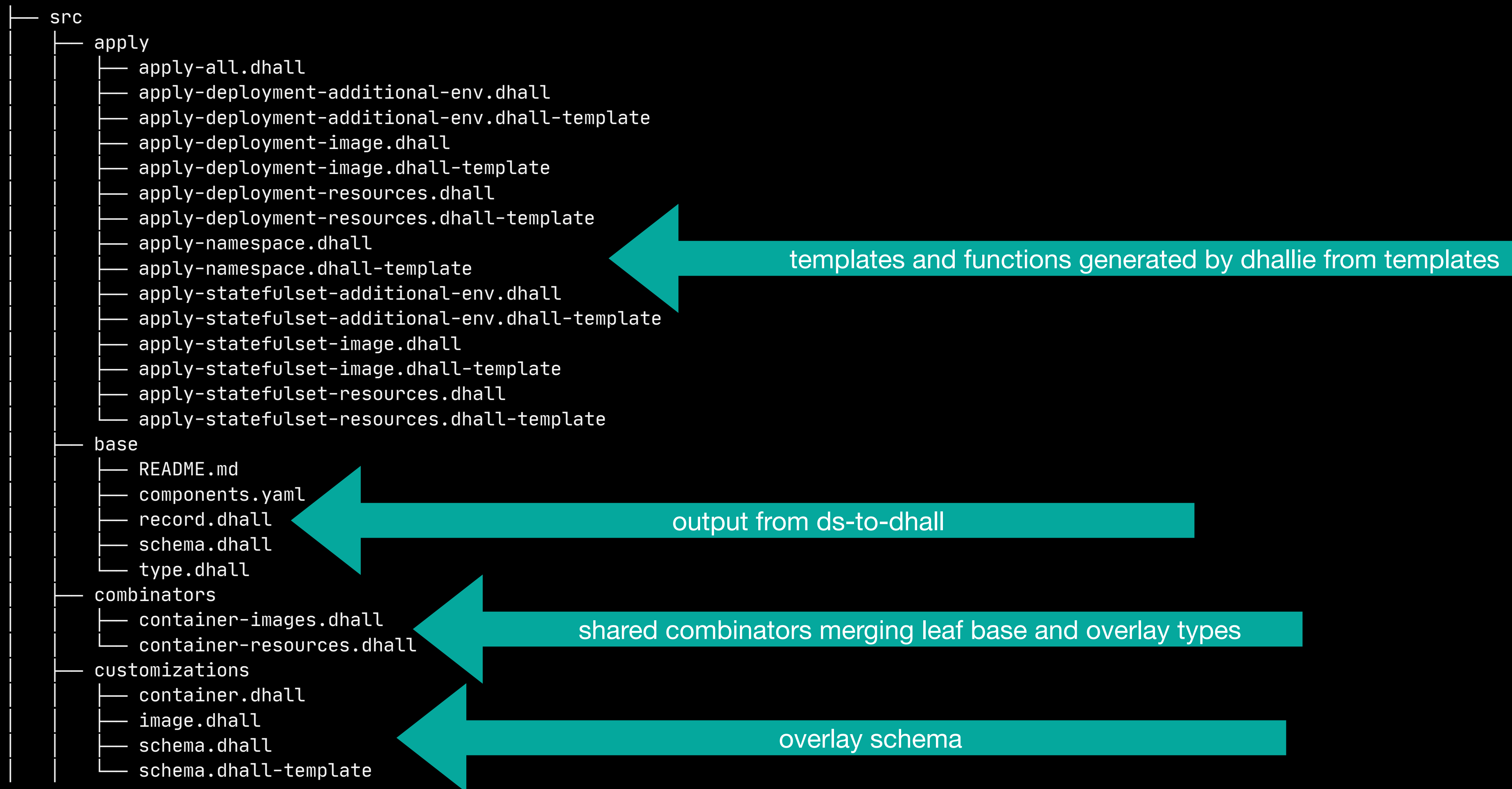
components.yaml:

```
Frontend:
  Deployment:
    sourcegraph-frontend:
      containers:
        frontend: {}
        jaeger-agent: {}
  Ingress:
    sourcegraph-frontend: {}
  Role:
    sourcegraph-frontend: {}
  RoleBinding:
    sourcegraph-frontend: {}
  Service:
    sourcegraph-frontend: {}
    sourcegraph-frontend-internal: {}
  ServiceAccount:
    sourcegraph-frontend: {}
Github-Proxy:
  Deployment:
    github-proxy:
      containers:
        github-proxy: {}
        jaeger-agent: {}
  Service:
    github-proxy: {}
Gitserver:
  Service:
    gitserver: {}
  StatefulSet:
    gitserver:
      containers:
        gitserver: {}
        jaeger-agent: {}
```

proof of concept repo

ds-dhall

<https://github.com/uwedeportivo/ds-dhall>



FAQ (continued)

- **Q:** Why all this complication ?

Why not just do a recursive merge of base and overlay records ?

A: This would almost work. Unfortunately some leaf types are not easily merged and don't fit the builtin recursive record merge operation. There are also shared customizations to consider and customizations affecting multiple leaves. It might make sense to separate everything that can be covered by the builtin recursive merge and use this approach for only the customizations that don't fit in.