Working with Modular Configuration

Nigel Brown
Freelance Technical Author

@n_brownuk | @nigelb@fosstodon.org | windsock.io

Module Outline

Coming up:

- Replace SQLite DB with a MySQL DB
- The need for modular configuration
- Creating modules using Kustomizations
- Introducing the Kustomize Component
- Using Components to create modules
- Components in action with our app

Demo

Introducing a New Overlay

- SQLite swapped for MySQL database
- Additional resources with new patches
- Generation of ConfigMaps and Secrets
- Deploy and test the app in the cluster

Reuse

Will we end up repeating ourselves again?

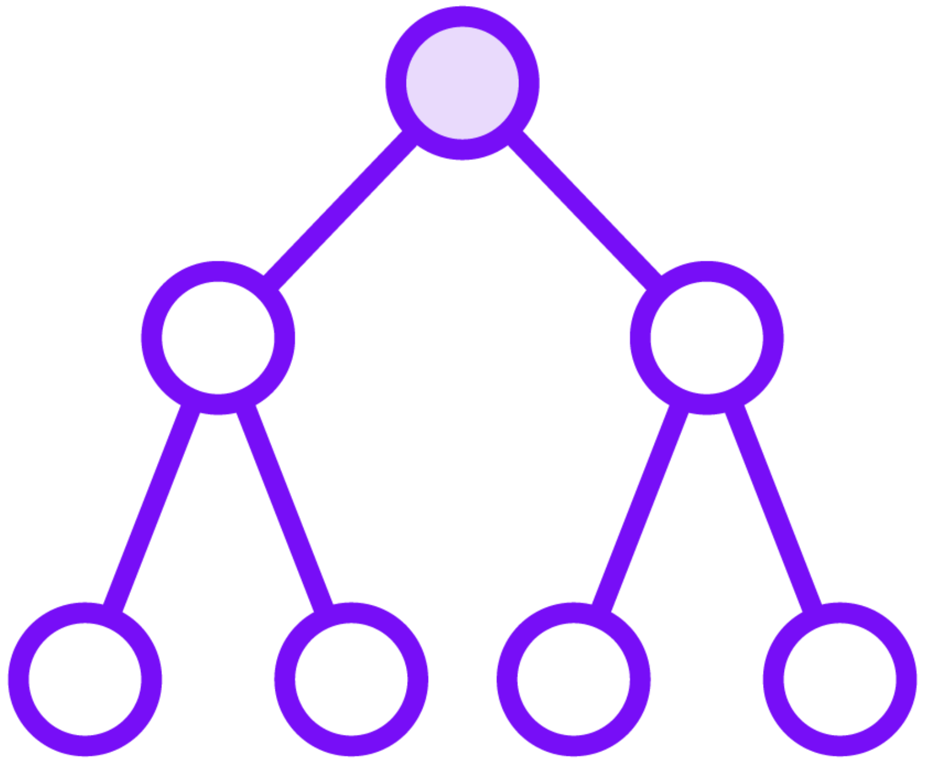
What about configuring common overlays?

Can we manage multiple, optional config items?

Reusable Configuration

Scenarios may necessitate the provision of optional configuration, for use in some variants and not others.

Modular Configuration

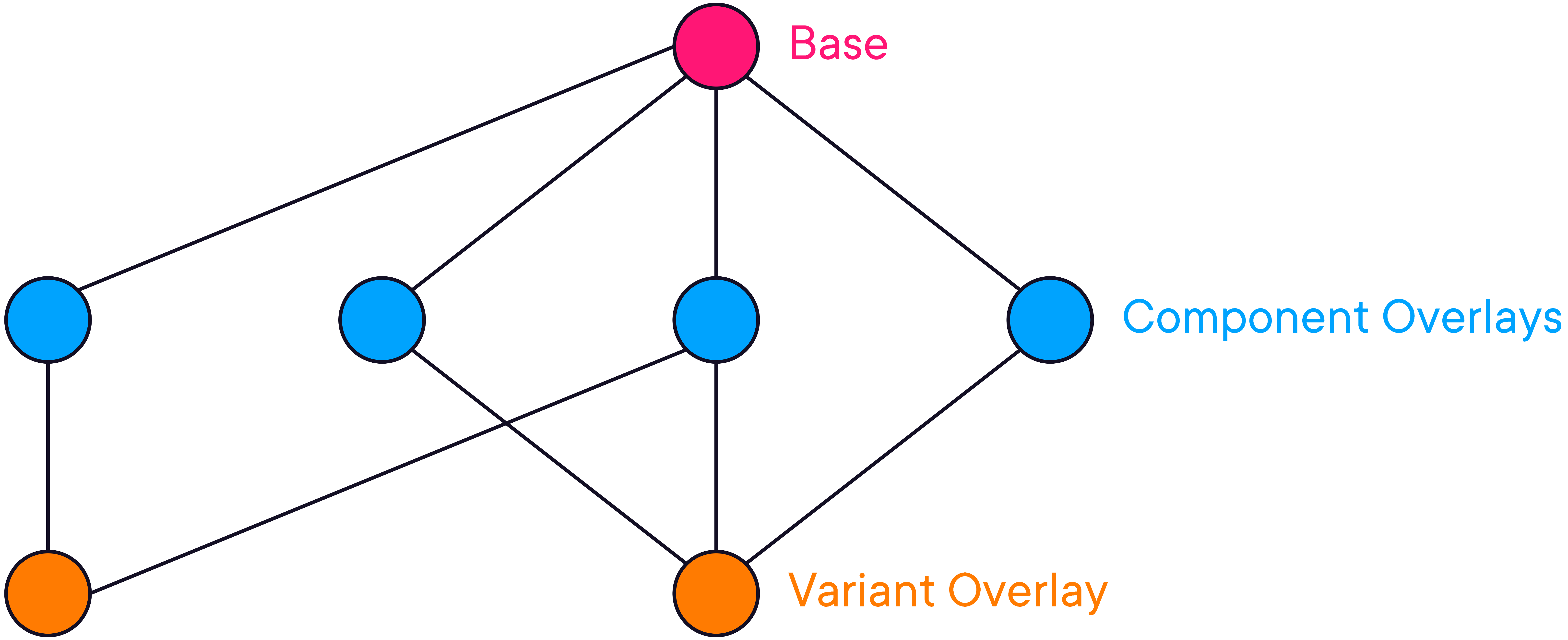


How do you use Kustomize to work with optional modular configuration?

The problem:

- Variants are created through inheritance
- Stacking overlays is not very scalable
- Begin to contravene the DRY principle

Component Overlays



Sibling Overlays

Sibling overlay Kustomizations are processed in parallel

Patching common bases causes conflicts during builds

Components

Kustomize provides a variation of a Kustomization called a Component, to cater for modular configuration.

Defining Components

Kustomize Type

A Component is a type expressed in KRM form

Kustomization Field

A ‘components’ field is used to refer to Components

Characteristics of Components

Behave in a similar way to regular Kustomizations, but with a key difference

Components are processed in a sequential fashion rather than in parallel

The ordering of Components defined in Kustomizations can be important

Order of Processing

1. Resources
2. Generators
3. Components

Exercise Caution

The Component type is ‘alpha’ in terms of maturity

Its interface and semantics may be subject to change

Referencing a Component

overlay/kustomization.yaml

```yaml
apiVersion: kustomize.config.k8s.io/v1beta1
kind: Kustomization
resources:
  - ../base
  - namespace.yaml
components:
  - ../component

<snip>
```

Referencing a Component

overlay/kustomization.yaml

```yaml
apiVersion: kustomize.config.k8s.io/v1beta1
kind: Kustomization
resources:
  - ../base
  - namespace.yaml
components:
  - ../components/patchA
  - ../components/patchB

<snip>
```

KRM Syntax for Components

Components are defined in files called ‘kustomization.yaml’.

component/kustomization.yaml

```
apiVersion: kustomize.config.k8s.io/v1alpha1
kind: Component

<snip>
```

KRM Syntax for Components

Components are defined in files called ‘kustomization.yaml’.

component/kustomization.yaml

```yaml
apiVersion: kustomize.config.k8s.io/v1alpha1
kind: Component

<snip>
```

KRM Syntax for Components

Components are defined in files called ‘kustomization.yaml’.

component/kustomization.yaml

```
apiVersion: kustomize.config.k8s.io/v1alpha1
kind: Component

<snip>
```

Demo

Modularizing an App’s Configuration

- Refactor previous ‘qa’ Kustomization
- Inspect the Component definitions
- Compare ‘before’ and ‘after’ builds

Demo

Using Optional Components in an Overlay

- Define an Ingress Component
- Build Kustomization with Ingress Component
- Apply new overlay to a cluster

Module Summary

What we covered:

- Configuration reuse helps us keep DRY
- Difficult to achieve with Kustomizations
- Components aid modular configuration

thanks!
Tombow ABT
Acid Free
N15

Useful References

Kustomize Reference, https://bit.ly/3ZG8iR7

Kustomize on GitHub, https://bit.ly/3LNV7Yy

Kubernetes Slack (#kustomize), http://slack.k8s.io/

Final Words

Feedback

Discussion