

Шаблонизация манифестов Kubernetes

Не забудь включить запись!



План занятия

- CI/CD и возможные проблемы
- Helm - концепция, архитектура
- Helm 2 и Helm 3
- Недостатки Helm
- Использование Helm (helm-hooks, helm-secrets, helmfile)
- Jsonnet
- Kustomize

Возможные проблемы с CI/CD

Возможные проблемы с CI/CD

```
1 apiVersion: apps/v1
2 kind: Deployment
3 metadata:
4   name: myapp
5 spec:
6   replicas: 1
7   template:
8     metadata:
9       labels:
10      app: myapp
11   spec:
12     containers:
13       - name: myapp
14         image: myapp:v1
15     env:
16       - name: DEBUG
17         value: ???
```

DEBUG: true

DEV namespace

DEBUG: false

PROD namespace

Возможные проблемы с CI/CD

```
1 apiVersion: apps/v1
2 kind: Deployment
3 metadata:
4   name: myapp
5 spec:
6   replicas: 1
7   template:
8     metadata:
9       labels:
10      app: myapp
11   spec:
12     containers:
13       - name: myapp
14         image: myapp:v1
15       env:
16         - name: DEBUG
17           value: {{ .Values.debug.enabled }}
```

DEBUG: true

DEV namespace

DEBUG: false

PROD namespace

Возможные проблемы с CI/CD

```
1 apiVersion: apps/v1
2 kind: Deployment
3 metadata:
4   name: myapp
5 spec:
6   replicas: 1
7   template:
8     metadata:
9       labels:
10      app: myapp
11   spec:
12     containers:
13       - name: myapp
14         image: myapp:v1
15       volumeMounts:
16         - name: ...
17       volumes:
18         - name: ...
```

DEV namespace

PROD namespace

PV

Возможные проблемы с CI/CD

```
1 apiVersion: apps/v1
2 kind: Deployment
3 metadata:
4   name: myapp
5 spec:
6   replicas: 1
7   template:
8     metadata:
9       labels:
10      app: myapp
11   spec:
12     containers:
13       - name: myapp
14         image: myapp:v1
15 {{- if .Values.persistentVolume.enabled
16   }}
17   volumeMounts:
18     - name: ...
19   volumes:
20     - name: ...
21 {{ end }}
```

DEV namespace

PROD namespace

PV

Возможные проблемы с CI/CD

Как выкапывать новую версию приложения?

```
1 apiVersion: apps/v1
2 kind: Deployment
3 metadata:
4   name: myapp
5 spec:
6   replicas: 1
7   template:
8     metadata:
9       labels:
10      app: myapp
11   spec:
12     containers:
13       - name: myapp
14         image: myapp:latest
```

Возможные проблемы с CI/CD

Как выкапывать новую версию приложения?

```
1 apiVersion: apps/v1
2 kind: Deployment
3 metadata:
4   name: myapp
5 spec:
6   replicas: 1
7   template:
8     metadata:
9       labels:
10      app: myapp
11   spec:
12     containers:
13       - name: myapp
14         image: myapp:{{ .Values.image.tag }}
```

image.tag=\${CI_COMMIT_SHA}

Helm



Избавляем бизнес от ИТ-зависимости

Helm

Причины использования

- Шаблонизация
- Откат
- Версионирование
- Переиспользование манифестов (в том числе публичных)

Helm

Возможности

- Упаковка нескольких манифестов Kubernetes в пакет - **Chart**
- Установка пакета в Kubernetes (установленный **Chart** называется **Release**)
- Шаблонизация во время установки пакета
- **Upgrade** (обновления) и **Rollback** (откаты) установленных пакетов
- Управление зависимостями между пакетами
- Хранение пакетов в удаленных репозиториях

Helm

Основные концепции

- Chart
- Values
- Release

Helm

Chart - пакет

- Метаданные
- Шаблоны описания ресурсов Kubernetes
- Конфигурация установки (**values.yaml**)
- Документация
- Список зависимостей

```
example/
  Chart.yaml          # описание пакета
  README.md
  requirements.yaml   # список зависимостей
  values.yaml         # переменные
  charts/              # загруженные зависимости
  templates/           # шаблоны
```

Helm

Chart.yaml - описание пакета

```
apiVersion: v1
description: A Helm chart for ELK
home: https://www.elastic.co/products # Информация для community
name: elastic-stack # Название Chart
version: 1.7.0 # Версия Chart
appVersion: 6.0 # Версия приложения
maintainers: # Информация для community
- name: rendhalver
  email: pete.brown@powerhrg.com
- name: jar361
  email: jrodgers@powerhrg.com
- name: christian-roggia
  email: christian.roggia@gmail.com
```

Helm

Values - переменные

- Конфигурация для **Chart**
- Используются в шаблонах манифестов
- У пакета бывают значения по умолчанию
- Можно перезаписывать во время установки пакета
- Иерархическая структура

Helm

values.yaml

```
# Default values for elk.
# This is a YAML-formatted file.
# Declare variables to be passed into your templates.

elasticsearch:
  enabled: true

kibana:
  enabled: true
  env:
    ELASTICSEARCH_URL: http://http.default.svc.cluster.local:9200
```

Helm

Release

- Установленный в Kubernetes **Chart**
- Хранятся в configMaps и Secrets
- **Chart + Values = Release**
- **1 Upgrade = 1 Release**

Helm

Workflow

- Установка **Helm** на локальную машину (или на CI агента)
- **Для Helm 2:** Инициализация **Helm** в кластере Kubernetes (установка **Tiller**)
- Создание **Chart** или загрузка с **Helm Hub**
- Установка **Chart** в кластер Kubernetes (создание **Release**)
- Поддержание жизненного цикла приложения (upgrade, rollback, delete, etc.)

Helm

Workflow

```
$ helm install <chart_name> --name=<release_name> --namespace=<namespace> # Создание release
$ kubectl get secrets -n <namespace> | grep <release_name>
sh.helm.release.v1.<release_name>.v1      helm.sh/release.v1    1    115m
```

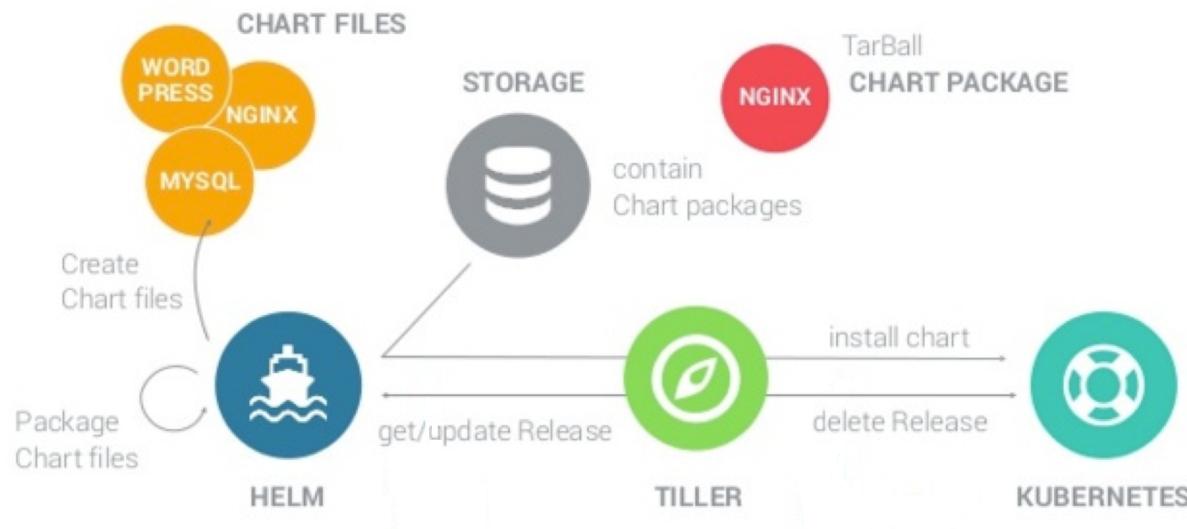
```
$ helm upgrade <release_name> <chart_name> --namespace=<namespace> # Обновление release
$ kubectl get secrets -n <namespace> | grep <release_name>
sh.helm.release.v1.<release_name>.v1      helm.sh/release.v1    1    115m
sh.helm.release.v1.<release_name>.v2      helm.sh/release.v1    1    56m
```

```
$ helm upgrade --install <release_name> <chart_name> --namespace=<namespace> #
Создание или обновление release
$ kubectl get secrets -n <namespace> | grep <release_name>
sh.helm.release.v1.<release_name>.v1      helm.sh/release.v1    1    115m
sh.helm.release.v1.<release_name>.v2      helm.sh/release.v1    1    56m
sh.helm.release.v1.<release_name>.v3      helm.sh/release.v1    1    5s
```

Helm

Архитектура (v2)

- Helm client (**helm**) - CLI клиент на локальной машине
- **Tiller** server - сервер (pod) внутри кластера Kubernetes



Helm v2

Helm Client

Используется для:

- Локальной работы с **Chart**
- Управления репозиториями
- Взаимодействия с **Tiller**
- Запроса:
 - Установки **Chart**
 - Обновления и удаления **Release**
 - Информации о текущих **Release**



Helm v2

Tiller

Используется для:

- Ожидания входящих запросов от Helm Client
- Установки **Chart** в Kubernetes
- Обновления и удаления **Release**
- Взаимодействия с Kubernetes API



Хранит информацию о релизах в **configMaps** в соответствующем namespace

Helm v2

Tiller

Установка Tiller в кластер:

```
helm init
```

Установка Tiller с правами определенного service-account:

```
helm init --service-account=tiller
```

Проверка:

```
helm version
```

Helm 3

Что нового в Helm 3 (текущая версия - v3.3.0)

- Убран Tiller
- Release'ы теперь хранятся в namespace'ах (и namespace необходимо создавать)
- Зависимости указываются в Chart.yaml
- Экспериментальная поддержка OCI-совместимых registry для хранения Helm Charts
- Lua как альтернативный язык написания скриптов
- Переименованы некоторые команды клиента:
 - helm inspect -> helm show
 - helm fetch -> helm pull

Миграция v2 -> v3

Избавляем бизнес от ИТ-зависимости

Недостатки Helm

Helm 2: Tiller

В большинстве случаев устанавливается с правами `cluster-admin`

Как обойти проблему:

- Плагин **helm-tiller**
- Tiller per namespace
- Helm 3

К чему может привести установка Tiller "самым простым путем"

Неприятные баги

Error: UPGRADE FAILED: "chart" has no deployed releases

Если первый release установился неуспешно - последующие также не будут установлены

- `helm upgrade --install --atomic` - при установке первого release
- `helm upgrade --install --force` - если первый release уже установлен
- `helm delete --purge && helm upgrade --install` - универсальный* способ (replaced with `helm uninstall`)
- **Helm 3** - появится команда `helm deploy` (сейчас нет)

Прочее

- Go/YAML Templating - на любителя
- Отступы (indent)
- `helm delete --purge` replaced by `helm uninstall`

Полезные ссылки:

- <https://habr.com/ru/company/southbridge/blog/429340/>
- <https://habr.com/ru/company/flant/blog/438814/>

Демо. Устанавливаем community helm chart

Templating

Templating | Манифест

```
1 apiVersion: apps/v1
2 kind: Deployment
3 metadata:
4   name: {{ .Release.name }}
5   labels:
6     app: nginx
7 spec:
8   replicas: {{ .Values.nginx.replicas }}
9   selector:
10    matchLabels:
11      app: nginx
12   template:
13     metadata:
14       labels:
15         app: nginx
16     spec:
17       containers:
18         - name: nginx
19           image: {{ .Values.nginx.image.repository }}:{{ .Values.nginx.image.tag }}
20       ports:
21         - containerPort: 80
```

Templating

Встроенные переменные

- **Release** - информация об устанавливаемом release
- **Chart** - информация о chart, из которого происходит установка
- **Files** - возможность загружать в шаблон данные из файлов (например, в `configMap`)
- **Capabilities** - информация о возможностях кластера (например, версия Kubernetes)
- **Templates** - информация о манифесте, из которого был создан ресурс

Подробное описание доступно по [ссылке](#)

Templating

Определяемые переменные

Задаются в файле **values.yaml**

```
1 nginx:  
2   replicas: 1  
3   image:  
4     repository: my.private.registry/nginx  
5     tag: latest
```

Helm

Workflow

Создание release с использованием переменных из файла `values.yaml`:

```
$ helm upgrade --install <release_name> <chart_name> -f values.yaml
```

Переопределение переменной в момент установки:

```
$ helm upgrade --install <release_name> <chart_name> -f values.yaml --set nginx.image.tag=${CI_COMMIT_SHA}
```

Рекомендуется использовать ключ `--wait` при создании или обновлении release. [Подробное описание ключа](#)

Templating

В основе Helm лежит шаблонизатор Go с 50+ встроенными функциями
[\(документация\)](#)

Условия:

```
{-{ if .Values.server.persistentVolume.enabled } }
  persistentVolumeClaim:
    ...
{-{ else } }
```

Циклы:

```
{-{ range $key, $value := .Values.server.annotations } }
  {{ $key}} {{ $value}}
{-{ end } }
```

Templating

Более сложный пример

```
1 {{- $files := .Files.Glob "dashboards/*.json" --}}
2 {{- if $files --}}
3 apiVersion: v1
4 kind: ConfigMapList
5 items:
6 {{- range $path, $fileContents := $files --}}
7 {{- $dashboardName := regexReplaceAll "^(.*\/)(.*)\\.json$" $path "${2}" --}}
8 - apiVersion: v1
9   kind: ConfigMap
10  metadata:
11    name: {{ $dashboardName | trunc 63 | trimSuffix "-" }}
12    labels:
13      grafana_dashboard: "1"
14  data:
15    {{ $dashboardName }}.json: {{ $.Files.Get $path | toJson }}
16 {{- end --}}
17 {{- end --}}
```

Полезные советы

- Указывайте все используемые в шаблонах переменные в `values.yaml`, выбирайте адекватные значения по умолчанию
- Используйте команду `helm create` для генерации структуры своего chart
- Пользуйтесь плагином `helm docs` для документирования своего chart

Больше советов по написанию helm charts от разработчиков
https://helm.sh/docs/chart_best_practices/

Хранение Helm Charts

В репозитории с приложением

- Удобно если манифесты пишут разработчики продукта
- Манифесты версионируются по тому же flow что и приложение
- Самый простой способ

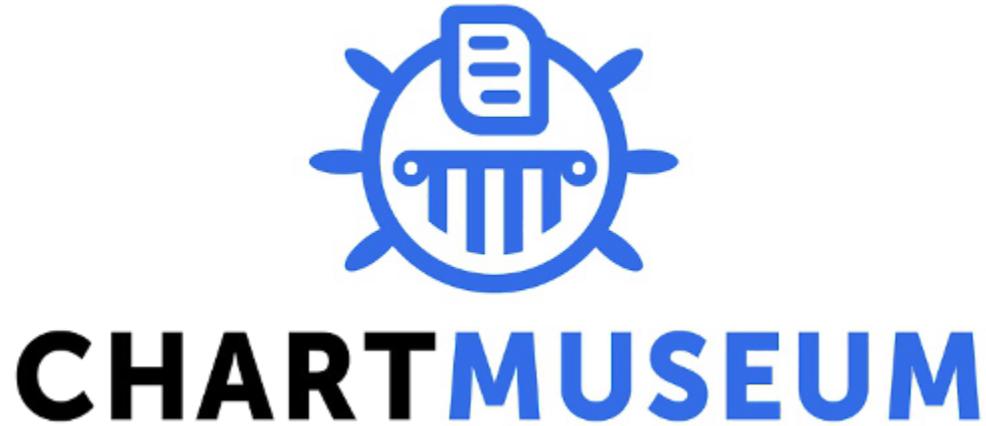
Внешний репозиторий

GitHub/GitLab/etc, либо хранилища артефактов

- Удобно если манифесты пишет выделенная команда (Infrastructure team)
- Проще переиспользовать наработки
- Полезно если есть необходимость в meta pipelines
- Сложнее привязать flow работы с кодом продукта к инфраструктурному коду (манифестам)

Хранилища артефактов

- ChartMuseum
- Harbor (внутри - ChartMuseum)
- Artifactory
- ...



git

- Необходим `index.yaml` файл с описанием всех chart в репозитории
- Необходима сборка в tgz перед публикацией (в идеале tgz архивы должны храниться вне репозитория)

```
helm package charts/
helm repo index .
helm-docs charts/
```

Пример репозитория

**Демо. Пишем свой helm
chart. Используем
harbor для хранения
charts.**

Helm Hooks

Hooks

Определенные действия, выполняемые в различные моменты жизненного цикла поставки. Hook, как правило, запускает Job (но это не обязательно).

Виды hooks:

- pre/post-install
- pre/post-delete
- pre/post-upgrade
- pre/post-rollback

| Документация

Hooks | Описание

Обычный манифест, но требуется добавить annotations.

Например, **pre-install** hook:

```
apiVersion: ...
kind: ....
metadata:
  annotations:
    "helm.sh/hook": "pre-install"
```

Hooks | Пример из жизни

Для развертывания продукта из feature branch во временное окружение (namespace) необходимо сделать следующие действия:

1. Добавить в namespace секреты для подключения к внешнему инстансу БД
2. Создать в инстансе новую БД
3. Выкатить миграции для БД (база используется более чем одним компонентом)
4. Установить Helm Chart с приложением

Hooks | Weight

Позволяют определить порядок выполнения одинаковых типов hooks

```
apiVersion: ...
kind: ....
metadata:
  annotations:
    "helm.sh/hook": "pre-install"
    "helm.sh/hook-weight": "10"
```

Чем меньше вес - тем раньше будет выполнен hook 🤝

Hooks | Удаление

После выполнения hook можно (и в большинстве случаев нужно) удалять артефакты после него (например - **job**). [Подробнее](#)

Для этого придумана специальная annotation - **"helm.sh/hook-delete-policy"** Могут использоваться следующие [значения](#):

- "hook-succeeded"
- "hook-failed"
- "before-hook-creation"

```
apiVersion: ...
kind: ....
metadata:
  annotations:
    "helm.sh/hook": "pre-install"
    "helm.sh/hook-weight": "10"
    "helm.sh/hook-delete-policy": "hook-succeeded"
```

Helm Secrets

Helm Secrets

- Плагин для Helm
- Механизм удобного* хранения и деплоя секретов для тех, у кого нет HashiCorp Vault
- Реализован поверх другого решения - Mozilla Sops
- Возможность сохранить структуру зашифрованного файла (YAML, JSON)
- Поддержка PGP и KMS (AWS, GCP)

Helm Secrets

Workflow (PGP)

- Создание PGP ключа
- Создание незашифрованного файла `secrets.yaml` (по аналогии с `values.yaml`)
- Шифрование файла
- Создание шаблона `Secret`, в который будут подставляться расшифрованные значения:

```
1 apiVersion: v1
2 kind: Secret
3 metadata:
4   name: database-password
5 type: Opaque
6 data:
7   DB_PASSWORD: {{ .Values.DATABASE_PASSWORD | b64enc | quote }}
```

Helm Secrets

Workflow (PGP)

- `git push`
- В ходе выполнения pipeline:

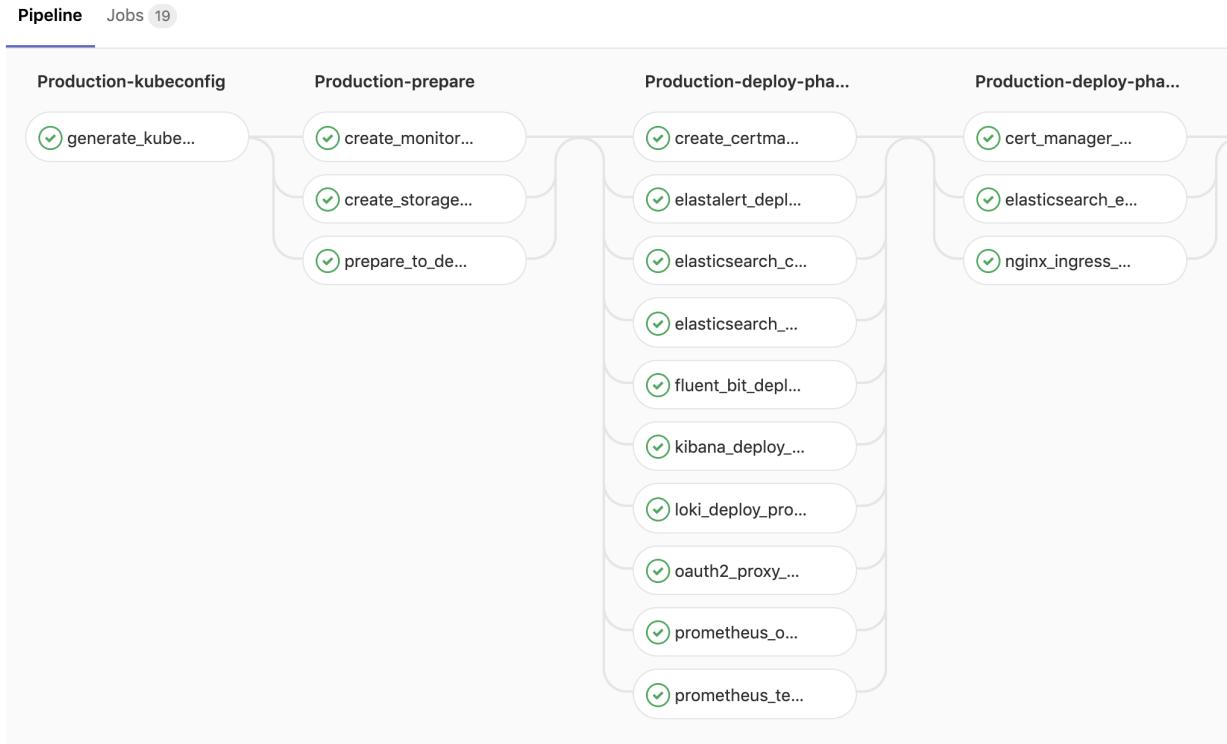
```
helm secrets upgrade --install -f values.yaml -f secret.yaml ...
```

Подробности в домашнем задании

Helmfile

Pipeline курильщика

Как мы делали раньше



Pipeline здорового человека

Как мы делаем сейчас

Pipeline Jobs 2

Prepare

Deploy



production-kub...



production-de...



Helmfile

Helm для Helm

- Управление развертыванием нескольких Helm Charts на нескольких окружениях
- Возможность устанавливать Helm Charts в необходимом порядке
- Больше шаблонизации, в том числе и в `values.yaml`
- Поддержка различных плагинов (**helm-tiller, helm-secret, helm-diff**)
- Главное - не увлечься шаблонизацией и сохранять прозрачность решения

Официальный репозиторий

Полезный обзор

Helmfile | Описываем конфигурацию окружений

```
1 environments:  
2   develop:  
3     values:  
4       - domain: develop.example.com  
5   production:  
6     values:  
7       - domain: production.example.com
```

Helmfile | Описываем конфигурацию Release

Репозитории из которых будем устанавливать Charts:

```
1 repositories:
2   - name: stable
3     url: https://kubernetes-charts.storage.googleapis.com
4   - name: express42
5     url: https://raw.githubusercontent.com/express42/helm-charts/master
```

Общий шаблон для release (не обязательно):

```
1 templates:
2   template: &template
3     missingFileHandler: Info
4     namespace: monitoring
5     values:
6       - ./values/{{`{{ .Release.Name }}`}}.yaml.gotmpl
```

Helmfile | Описываем в Helmfile конфигурацию Release

Описываем сами Release (наследуем от шаблона):

```
1 releases:
2   - name: prometheus-operator
3     chart: stable/prometheus-operator
4     version: 6.11.0
5     <<: *template
6
7   - name: prometheus-telegram-bot
8     chart: express42/prometheus-telegram-bot
9     version: 0.0.1
10    <<: *template
```

Тонкости конфигурации

- Определение порядка накатки чартов
- Декомпозиция на несколько файлов и порядок их рендеринга
- Базовые файлы

Ссылка на документацию конфига

Helmfile | Итоговый helmfile

```
1 environments:
2   develop:
3     values:
4       - domain: develop.example.com
5   production:
6     values:
7       - domain: production.example.com
8
9 repositories:
10  - name: stable
11    url: https://kubernetes-charts.storage.googleapis.com
12  - name: express42
13    url: https://raw.githubusercontent.com/express42/helm-charts/master
14
15 templates:
16  template: &template
17  missingFileHandler: Info
18  namespace: monitoring
19  values:
20  - ./values/{{`{{ .Release.Name }}`}}.yaml.gotmpl
21
22 releases:
23  - name: prometheus-operator
24    chart: stable/prometheus-operator
25    version: 6.11.0
```

Helmfile | Values

Добавим values:

```
1 prometheus:
2   ingress:
3     enabled: true
4     hosts:
5       - prometheus.{{ .Values | get "domain" }}
6   tls:
7     - secretName: prometheus-tls
8       hosts:
9         - prometheus.{{ .Values | get "domain" }}
```

Helmfile

И попробуем посмотреть что применится на описанные окружения.

Develop:

```
$ helmfile --environment develop template | grep example.com

  - host: prometheus.develop.example.com
    - prometheus.develop.example.com
  externalUrl: "http://prometheus.develop.example.com/"
```

Production:

```
$ helmfile --environment production template | grep example.com

  - host: prometheus.production.example.com
    - prometheus.production.example.com
  externalUrl: "http://prometheus.production.example.com/"
```

Helmfile | Структура репозитория

Один из способов организации структуры репозитория при использовании helmfile:

```
.  
├── charts  
├── commons  
├── helmfile.yaml  
└── helmfiles
```

Helmfile | Структура репозитория

Helmfiles

```
helmfiles
├── cert-manager-clusterissuers.yaml
├── cert-manager-crd.yaml
├── cert-manager.yaml
├── grafana-dashboards.yaml
├── logging.yaml
├── monitoring.yaml
├── nginx-ingress.yaml
├── oauth2-proxy.yaml
├── polaris.yaml
├── prometheus-rules.yaml
├── storage-classes.yaml
└── values
```

Helmfile | Структура репозитория

Values

```
helmfiles/values
├── elastalert.yaml.gotmpl
├── elasticsearch-curator.yaml.gotmpl
├── elasticsearch-exporter.yaml.gotmpl
├── elasticsearch.yaml.gotmpl
├── fluent-bit.yaml.gotmpl
├── kibana.yaml.gotmpl
├── nginx-ingress.yaml.gotmpl
├── oauth2-proxy.yaml.gotmpl
├── polaris.yaml.gotmpl
├── prometheus-operator.yaml.gotmpl
└── prometheus-telegram-bot.yaml.gotmpl
```

Еще один подход

Демо. Используем helmfile

Jsonnet и решения на его основе

Что такое Jsonnet

- Продукт от Google
- Расширение JSON (**как YAML**)
- Любой валидный JSON - валидный Jsonnet (**как YAML**)
- Полноценный язык программирования* (заточенный под шаблонизацию)
(не как YAML)

Tutorial

Зачем использовать Jsonnet (в Kubernetes)

- Для генерации и применения манифестов множества однотипных ресурсов, отличающихся несколькими параметрами
- Если есть ненависть к YAML, многострочным портнякам на YAML и отступам в YAML
- Для генерации YAML и передачи его в другие утилиты (например - kubectl):

```
kubecfg show workers.jsonnet | kubectl apply -f -
```

Откуда начинать

Ksonnet

- Раньше - наиболее популярное решение на основе Jsonnet
- Сейчас - разработка прекращена, репозиторий не поддерживается, ssl сертификат на сайте ksonnet.io просрочен



Kubecfg

Продукт от Bitnami - компании, которая занимается упаковкой различных продуктов под различные платформы



Простой и развивающийся* инструмент-обертка над jsonnet

Общий workflow следующий:

1. Импортируем подготовленную библиотеку с описанием ресурсов
2. Пишем общий для сервисов шаблон
3. Наследуемся от шаблона, указывая конкретные параметры

| Вебинар CNCF

Другие варианты

- Qbec
- Kapitan
- Tanka
- ...

Демо. Шаблонизация с Jsonnet и kubecfg

Kustomize

Kustomize

- Поддержка встроена в **kubectl**
- Кастомизация готовых манифестов
- Все больше приложений начинают использовать **kustomize** как альтернативный вариант поставки (istio, nginx-ingress, etc...)
- Почти как Jsonnet, только YAML (но **kustomize** - это не templating)
- Нет параметризации при вызове, но можно делать так:
`kustomize edit
set image ...`

"Книга" про kustomize

Kustomize

Общая логика работы:

- Создаем базовые манифесты ресурсов
- Создаем файл `kustomization.yaml` с описанием общих значений
- Кастомизируем манифесты и применяем их
- Отлично подходит для `labels`, `environment variables` и многое еще

| Поля, которые можно кастомизировать или трансформировать

Kustomize | Использование

Существует три варианта попадания манифестов в кластер

- Установить манифести с кастомизацией:

```
kubectl apply -k .
```

- Аналогичная команда:

```
kubectl kustomize . | kubectl apply -f -
```

- Использовать отдельную утилиту kustomize в которой **больше функционала**:

```
kustomize build . | kubectl apply -f -
```

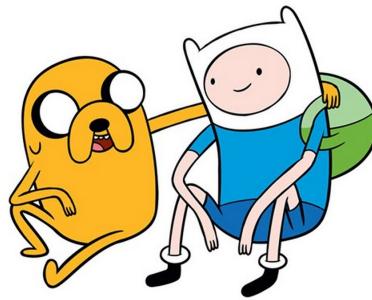
Kustomize | Базовая структура

Наиболее простой случай - установка из кастомизированных манифестов на два окружения:

```
.  
├── base  
│   ├── deployment.yaml  
│   ├── kustomization.yaml  
│   └── service.yaml  
└── overlays  
    ├── dev  
    │   └── kustomization.yaml  
    └── prod  
        └── kustomization.yaml
```

Пример более сложной структуры

Демо. Используем Kustomize



Спасибо за внимание!

Время для ваших вопросов!