Автоматизация разработки и эксплуатации программного обеспечения (осень 2023 года)

ИУ-5, бакалавриат, курс по выбору

Паттерны использования Kubernetes

Лекция №8

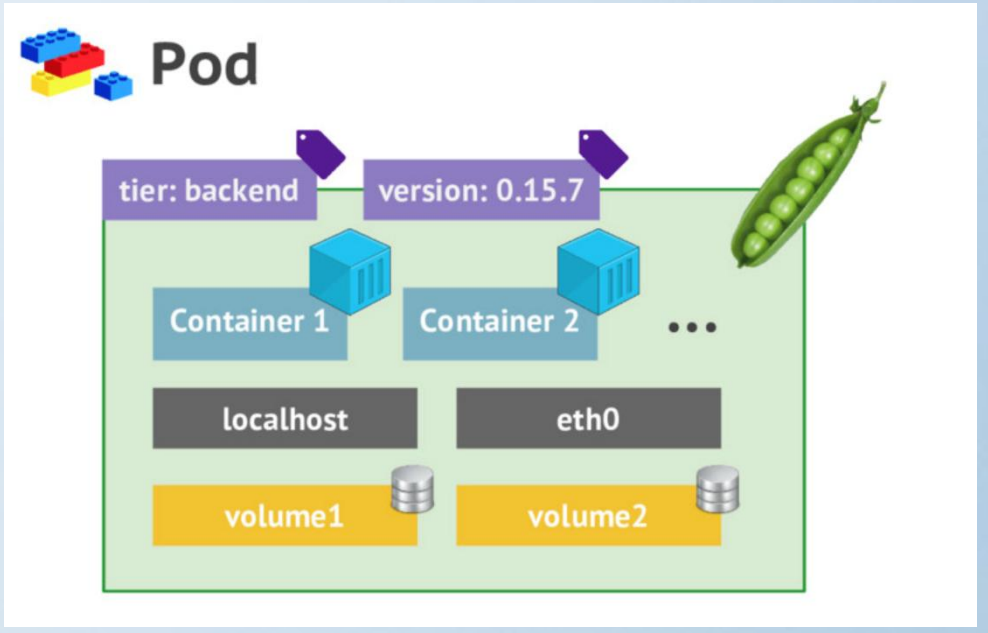
В предыдущих сериях

- Что такое докер?
- Что такое контейнер?
- Что такое Kubernetes?
- Что такое Pod?

Итого

- Под (Pod) — это атомарная единица планирования
- Под гарантирует совместное размещение контейнеров
- Под имеет IP-адрес, имя и диапазон портов, общих для всех контейнеров, входящих в группу

Ссылка на презентацию про сущности k8s из доклада



Ресурсы

Определение контейнера может задавать необходимую долю процессорного времени и объем памяти в форме запроса
и лимита. В общих чертах идея запросов/лимитов напоминает мягкие/ жесткие лимиты.

```
apiVersion: v1
kind: Pod
metadata:
 name: server
spec:
 containers:
 - image: nginx:1.18
   name: nginx
   resources:
     requests:
       cpu: 100m
       memory: 100Mi
     limits:
       cpu: 200m
       memory: 200Mi
```

QoS (Quality of Service)

- Guaranteed (Гарантированный)
- Burstable (С переменным качеством)
- BestEffort (Без гарантий)

Качество обслуживания влияет на распределение и вытеснение подов на нодах.

Класс приоритета

Определение контейнера может задавать необходимую долю процессорного времени и объем памяти в форме запроса и лимита. В общих чертах идея запросов/лимитов напоминает мягкие/жесткие лимиты.

```
apiVersion: scheduling.k8s.io/v1
kind: PriorityClass
metadata:
  name: high-priority
value: 1000
preemptionPolicy: Never
globalDefault: false
---
apiVersion: v1
kind: Pod
metadata:
  name: nginx
  labels:
    env: test
spec:
  containers:
  - name: nginx
    image: nginx
    imagePullPolicy: IfNotPresent
  priorityClassName: high-priority
```

Проверка работоспособности

Проверка работоспособности подразумевает использование периодических проверок для определения состояния.

Проверкой может быть:

- HTTP-запрос GET на IP-адрес контейнера и в ожидании HTTP-ответа с кодом от 200 до 399.
- Установка соединения через сокет TCP предполагает благополучный обмен через соединение TCP.
- Выполнение произвольной команды в контейнере и в ожидании кода благополучного ее завершения.

Liveness Probe

Liveness проба используется для проверки работоспособности пода, если проба завершилась с ошибкой, то Kubernetes перезапустит Pod.

```
apiVersion: v1
kind: Pod
metadata:
 name: nginx-liveness
spec:
 containers:
 - image: nginx:1.18
   name: nginx
ports:
   - containerPort: 8080
     protocol: TCP
     livenessProbe:
       exec:
         command: ["rm", "/var/run/liveness" ]
     initialDelaySeconds: 30
     periodSeconds: 5
```

Readiness Probe

Readiness проба используется для определения готовности пода к приему трафика, если проба завершилась с ошибкой, ip пода удаляется из сервиса.

```
apiVersion: v1
kind: Pod
metadata:
 name: nginx-readiness
spec:
 containers:
 - image: nginx:1.18
   name: nginx
ports:
   - containerPort: 8080
     protocol: TCP
   readinessProbe:
     httpGet:
       path: /healthz
       port: 8080
     periodSeconds: 5
```

Startup Probe

Startup проба используется первоначальной проверки приложения при запуске, если приложение долго инициализируется.

```
ports:
- name: liveness-port
  containerPort: 8080
  hostPort: 8080

livenessProbe:
  httpGet:
    path: /healthz
    port: liveness-port
  failureThreshold: 1
  periodSeconds: 10

startupProbe:
  httpGet:
    path: /healthz
    port: liveness-port
  failureThreshold: 30
  periodSeconds: 10
```

Управляемый жизненный цикл

В Kubernetes существуют дополнительные обработчики жизненного цикла пода, такие как сигналы postStart и preStop

PostStart – выполняется сразу как создан контейнер и не дает гарантий, что Entrypoint выполнен.

PreStop – выполняется сразу перед остановкой контейнера. Сигнал перехватывает SIGTERM, но не спасает от SIGKILL, поэтому надо настроить у пода параметр terminationGracePeriodSeconds

```
lifecycle:
 postStart:
   exec:
     command:
     - sh
     - -c
     - sleep 30 && echo "Wake up!" > /tmp/postStart
---
lifecycle:
 preStop:
   httpGet:
     port: 8080
     path: /shutdown
```

Node Selector

Задает правила выбора ноды по лейблам на ноде **kubectl label nodes <your-node-name> disktype=ssd**

```
…
spec:
  containers:
  - name: nginx
    image: nginx
    imagePullPolicy: IfNotPresent
  nodeSelector:
    disktype: ssd
…
```

Affinity / Anti-affinity

Правила сопоставления объектов и узлов:
рядом – affinity
раздельно – anti-affinity

Можно задавать по селекторам для отдельных типов подов

```
…
 spec:
      containers:
      - image: nginx
        name: nginx
      affinity:
        podAntiAffinity:
           requiredDuringSchedulingIgnoredDuringExecution:
             - labelSelector:
                 matchExpressions:
                   - key: "app"
                     operator: In
                     values:
                     - nginx
               topologyKey: "kubernetes.io/hostname”
…
```

Taints and Tolerations

Правила задают, может ли под быть запущен на той или иной «испорченной» ноде

Эффекты:

NoExecute – аффектит уже
работающие поды

NoSchedule – аффектит только новые поды

PreferNoSchedule – пытается следовать, но не гарантирует

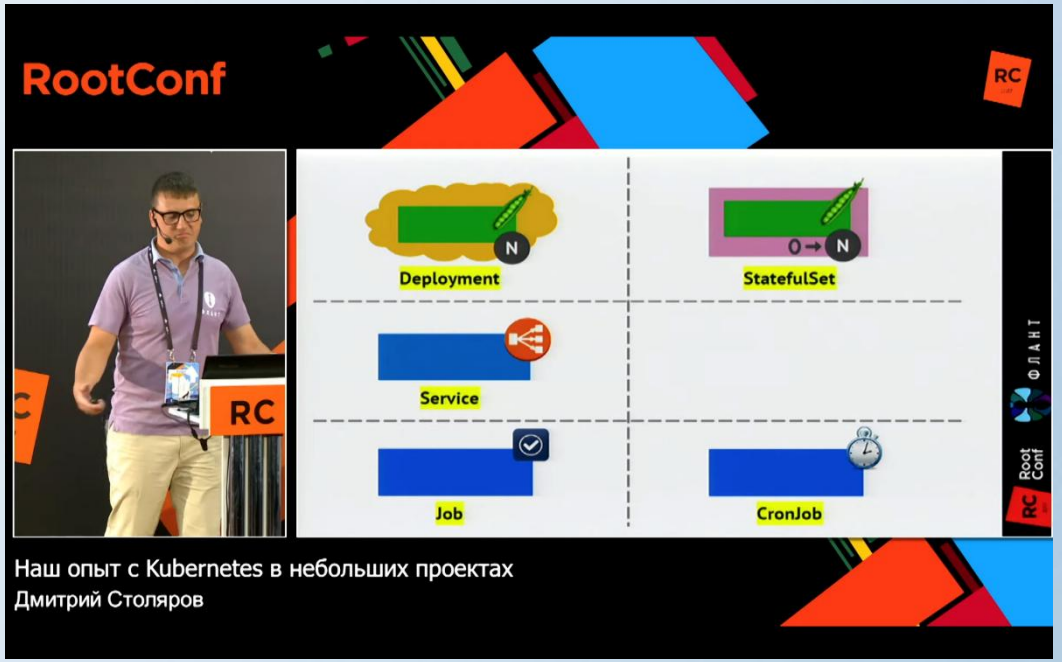
```
…
брес:
  containers:
  - name: nginx
    image: nginx
    imagePullPolicy: IfNotPresent
  tolerations:
  - key: "example-key"
    operator: "Exists"
    effect: "NoSchedule"
…
```

Поведенческие паттерны

Эти паттерны предназначены для создания коммуникационных механизмов и организации взаимодействия между подами и управляющей платформой

В зависимости от типа управляющего контроллера под может выполняться до завершения или запускаться и останавливаться по расписанию. Он может работать как *фоновая служба* или предоставлять гарантии уникальности своим репликам.

Советуем видео для лучшего понимания:
[Наш опыт с Kubernetes в небольших проектах / Дмитрий Столяров (Флант)]



StatefulSet

- Идет в разрез с привычным пониманием контейнера (stateless) – т.е. под имеет “состояние”
- Каждый Pod из StatefulSet обращается к своему volume, они создаются по шаблону под каждый Pod автоматически
- Имеют постоянные имена и hostname (app-0, app-1, app-2)
- Используется для кластерных stateful-приложений, например:
  - Кэши и KV-хранилища (Redis, Memcached, …)
  - СУБД (MySQL, PostgreSQL, Cassandra, Mongo, …)
  - Наши stateful-приложения, требующие сохранения состояния

DaemonSet

- Как ReplicaSet, но запускает по одному поду на узле
- Можно ограничить число узлов nodeSelector-ом
- Не требует планировщика (подходит для модулей k8s)
- Используется для запуска инфраструктурных элементов:
  - агенты журналирования
  - экспортеры метрик
  - Kube-proxy

Job

Job – разовая задача на выполнение

Job – после завершения не перезапускается (done)

```
apiVersion: v1
kind: Job
metadata:
  name: random-generator
spec:
  completions: 5
  parallelism: 2
  template:
    metadata:
      name: random-generator
    spec:
      restartPolicy: OnFailure
      containers:
        - image: k8spatterns/random-generator:1.0
          name: random-generator
          command: [ "gen.py", "numbers.txt" ]
```

CronJob

- Распределенный
- Отказоустойчивый
- Cron (регулярная задача)

```
apiVersion: v1
kind: CronJob
metadata:
  name: random-generator
spec:
  # Через каждые три минуты
  schedule: "*/3 * * * *"
  jobTemplate:
  spec:
    template:
      spec:
        containers:
          - image: k8spatterns/random-generator:1.0
            name: random-generator
            command: [ "gen.py", "numbers.txt" ]
            restartPolicy: OnFailure
```

Структурные паттерны

Контейнеры в Kubernetes не работают по отдельности; они объединяются в абстракции более высокого уровня — поды (группы контейнеров), которые предоставляют уникальные возможности времени выполнения

Еще раз про Volumes

Позволяет подключать к контейнерам в поде внешние данные

```
apiVersion: v1
kind: Pod
metadata:
  name: test-pd
spec:
  containers:
  - image: registry.k8s.io/test-webserver
    name: test-container
    volumeMounts:
    - mountPath: /cache
      name: cache-volume
  volumes:
  - name: cache-volume
    emptyDir:
      sizeLimit: 500Mi
```

Init контейнер

Init контейнеры запускаются по порядку до запуска основных контейнеров.

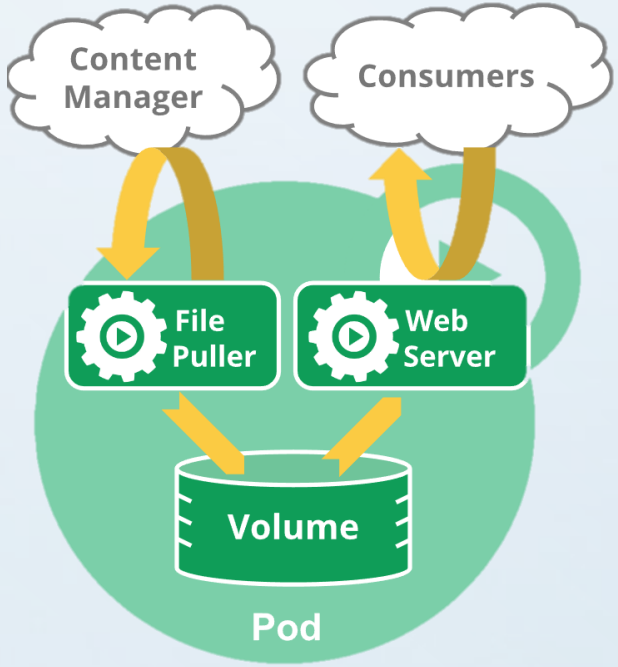
Основные задачи, решаемые Init контейнерами:

- Подготовка конфигурации основного приложения
- Задержка запуска основного приложения

```
apiVersion: v1
kind: Pod
metadata:
 name: www
 labels:
 app: www
spec:
 initContainers:
 - name: download
   image: agit
   command:
   - git
   - clone
   - https://github.com/mdn/beginner-html-site-scripted
   - /var/lib/data
   volumeMounts:
   - mountPath: /var/lib/data
 name: source
 containers:
 - name: run
   image: docker.io/centos/httpd
   ports:
   - containerPort: 80
   volumeMounts:
   - mountPath: /var/www/html
     name: source
 volumes:
 - emptyDir: {}
   name: source
```

Sidacar

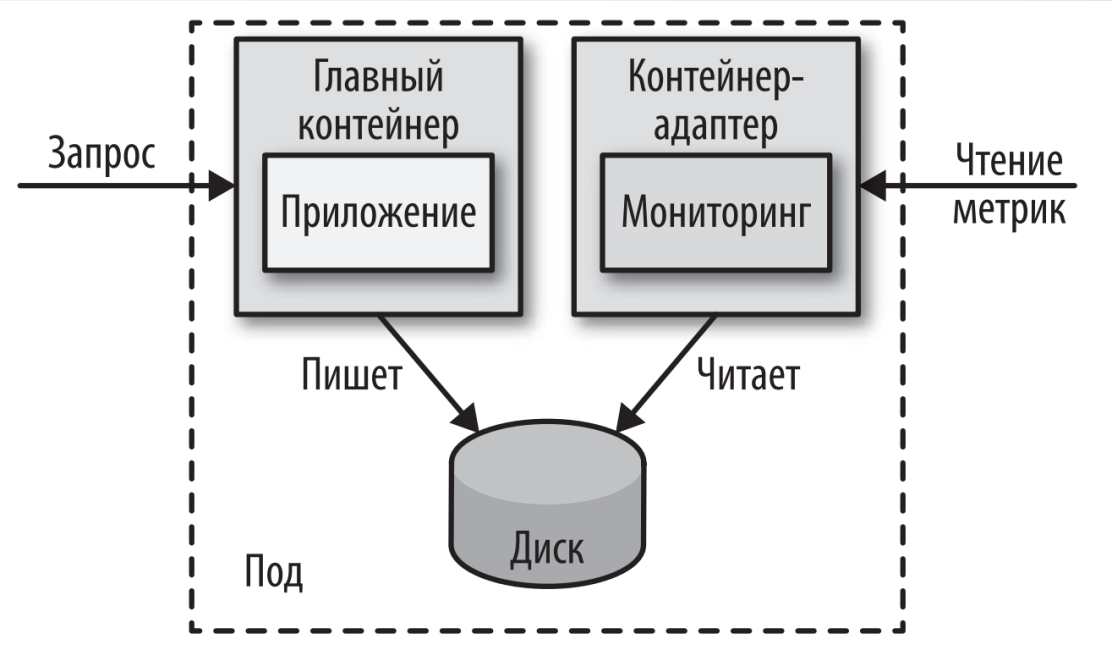
Sidecar контейнеры – вспомогательные контейнеры, которые работают на протяжении работы основного приложения для реализации какого либо дополнительного функционала.




```
apiVersion: v1
kind: Pod
metadata:
  name: sidecar-container-demo
spec:
  containers:
  - image: busybox
    command: ["/bin/sh"]
    args: ["-c", "while true; do echo echo $(date -u)
'Hi I am from Sidecar container' >>
/var/log/index.html; sleep 5;done"]
    name: sidecar-container
    resources: {}
    volumeMounts:
    - name: var-logs
      mountPath: /var/log
  - image: nginx
    name: main-container
    resources: {}
    ports:
      - containerPort: 80
    volumeMounts:
    - name: var-logs
      mountPath: /usr/share/nginx/html
  dnsPolicy: Default
  volumes:
  - name: var-logs
    emptyDir: {}
```

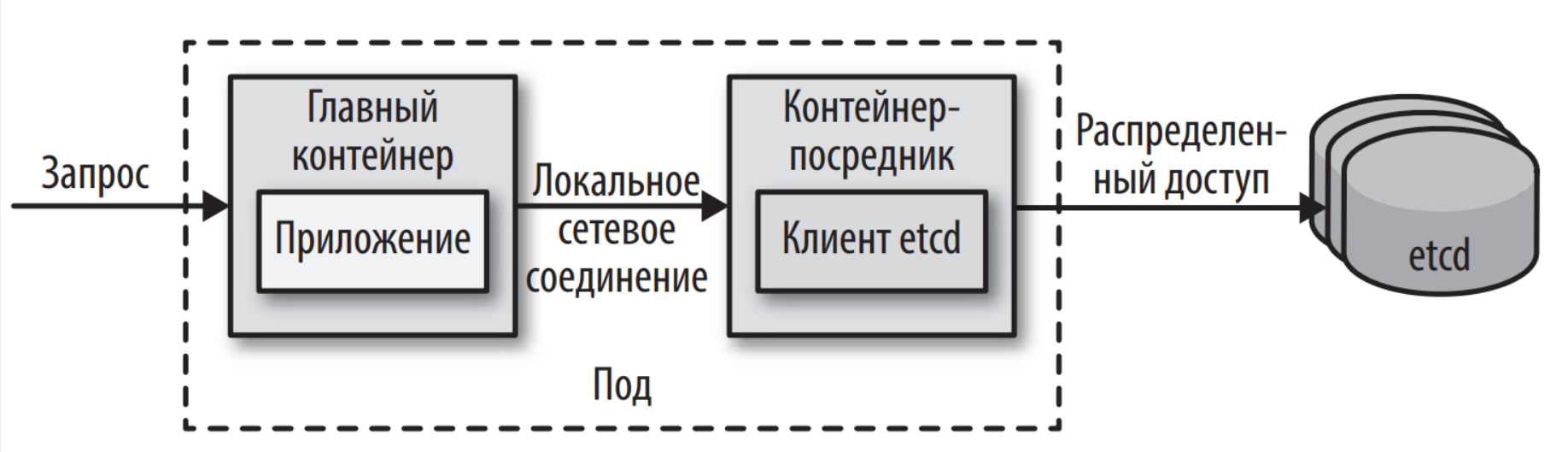
Паттерн Адаптер

Паттерн Adapter (Адаптер) предлагает решение, помогающее скрыть сложность системы и предоставить унифицированный доступ к ней.



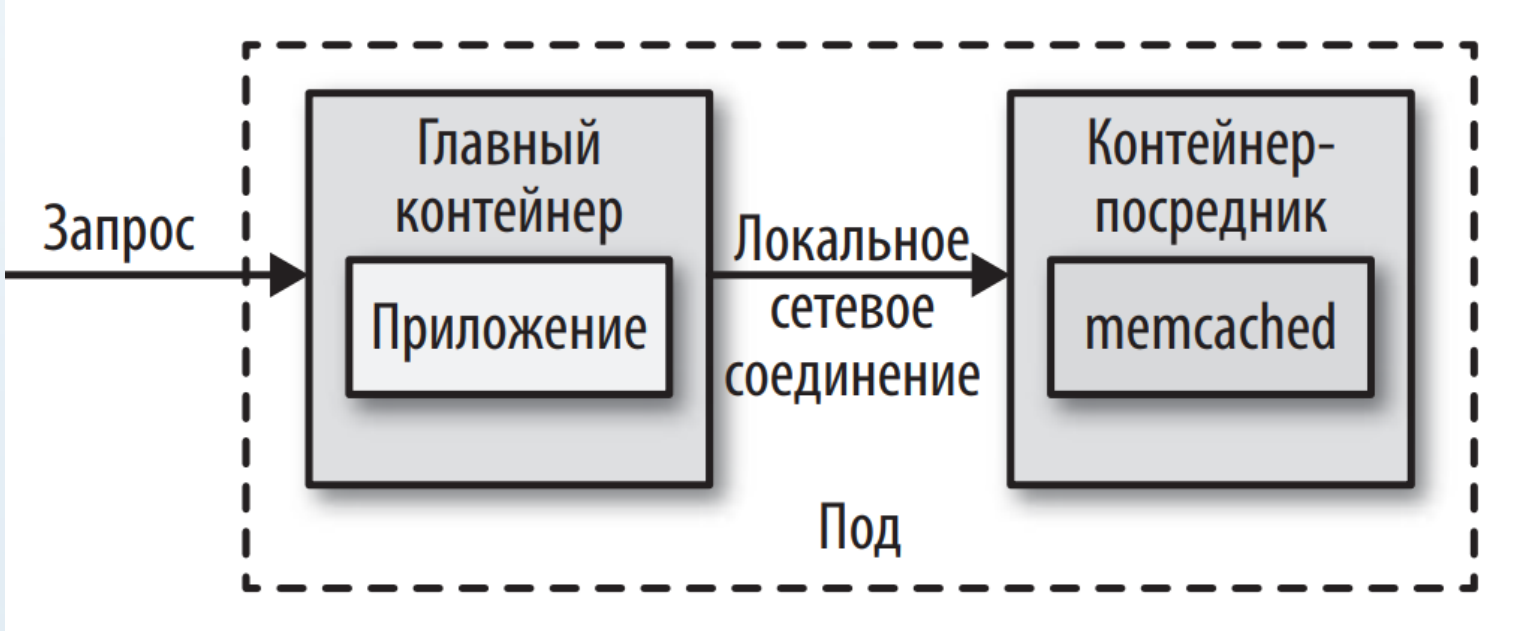
Паттерн Посредник

Паттерн Ambassador (Посредник) — это специализированный вариант паттерна Sidecar, отвечающий за сокрытие сложности и предоставляющий унифицированный интерфейс для доступа к службам за пределами пода



Паттерн Посредник для локального хранилища данных

Паттерн Посредник можно применить и для размещения дополнительного функционала в самом поде, без использования внешних сервисов.



Конфигурация приложения

Любое приложение должно иметь возможность быть перенастроено в зависимости от окружения, в котором оно запускается.

Хранить конфигурацию вместе с кодом приложения считается антипатерном в методике непрерывной доставки кода

Конфигурация через переменные окружения

Простейшим примером конфигурации приложения отдельно от кода является конфигурация через переменные окружения, которые можно задать у пода.

```
apiVersion: v1
kind: Pod
metadata:
  name: print-greeting
spec:
  containers:
  - name: env-print-demo
    image: bash
    env:
    - name: GREETING
      value: "Warm greetings to"
    - name: HONORIFIC
      value: "The Most Honorable"
    - name: NAME
      value: "Kubernetes"
    command: ["echo"]
    args: ["${GREETING} ${HONORIFIC} ${NAME}"]
```

Переменные окружения через ConfigMap

Чтобы не перечислять все переменные окружения в манифесте пода, можно сохранить переменные в отдельный ресурс ConfigMap и подключить его к контейнеру

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: special-config
  namespace: default
data:
  GREETING: "Warm greetings to"
  HONORIFIC : "The Most Honorable"
  NAME : "Kubernetes"
---
apiVersion: v1
kind: Pod
metadata:
  name: demo
spec:
  containers:
  - name: demo-container
    image: gcr.io/google-samples/node-hello:1.0
    command: ["echo"]
    args: ["${GREETING} ${HONORIFIC} ${NAME}"]
    envFrom:
      - configMapRef:
          name: special-config
```

Файлы конфигурации через ConfigMap

Через ConfigMap можно передавать и сами файлы конфигурации, которые можно подключить к контейнеру в виде тома (volume)

```
apiVersion: v1
kind: ConfigMap
metadata:
name: game-config
data:
  game-special-key: |
    enemies=aliens
    lives=3
    enemies.cheat=true
    enemies.cheat.level=noGoodRotten
---
apiVersion: v1
kind: Pod
metadata:
  name: dapi-test-pod
spec:
  containers:
    - name: test-container
      image: registry.k8s.io/busybox
      command: [ "/bin/sh", "-c", "ls /etc/config/" ]
      volumeMounts:
      - name: config-volume
        mountPath: /etc/config
  volumes:
    - name: config-volume
      configMap:
         name: special-config
```

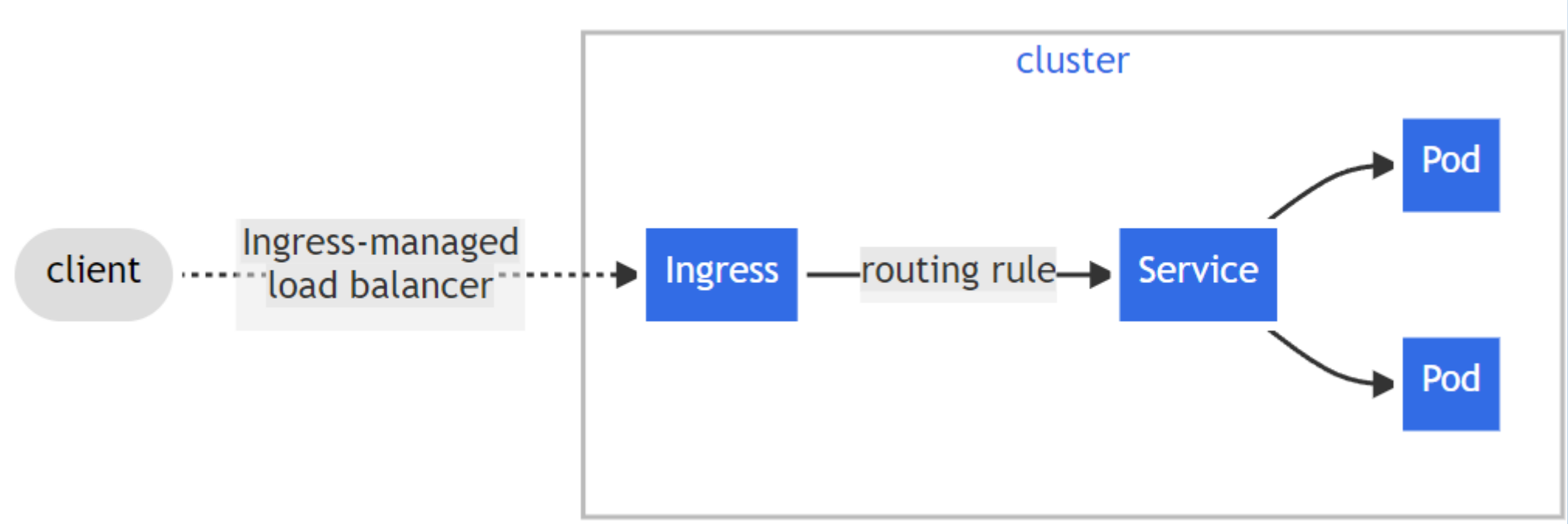
Конфигурация с помощью Secrets

Secrets аналогичны ConfigMap, но хранятся в зашифрованном виде и монтируются к приложению через tmpfs (не хранятся на диске)

```
apiVersion: v1
kind: Secret
metadata:
  name: mysecret
type: Opaque
data:
  username: YWRtaW4=
  password: MWYyZDFlMmU2N2Rm
---
apiVersion: v1
kind: Pod
metadata:
  name: mypod
spec:
  containers:
  - name: mypod
    image: redis
    volumeMounts:
    - name: foo
      mountPath: "/etc/foo"
      readOnly: true
  volumes:
  - name: foo
    secret:
      secretName: mysecret
```

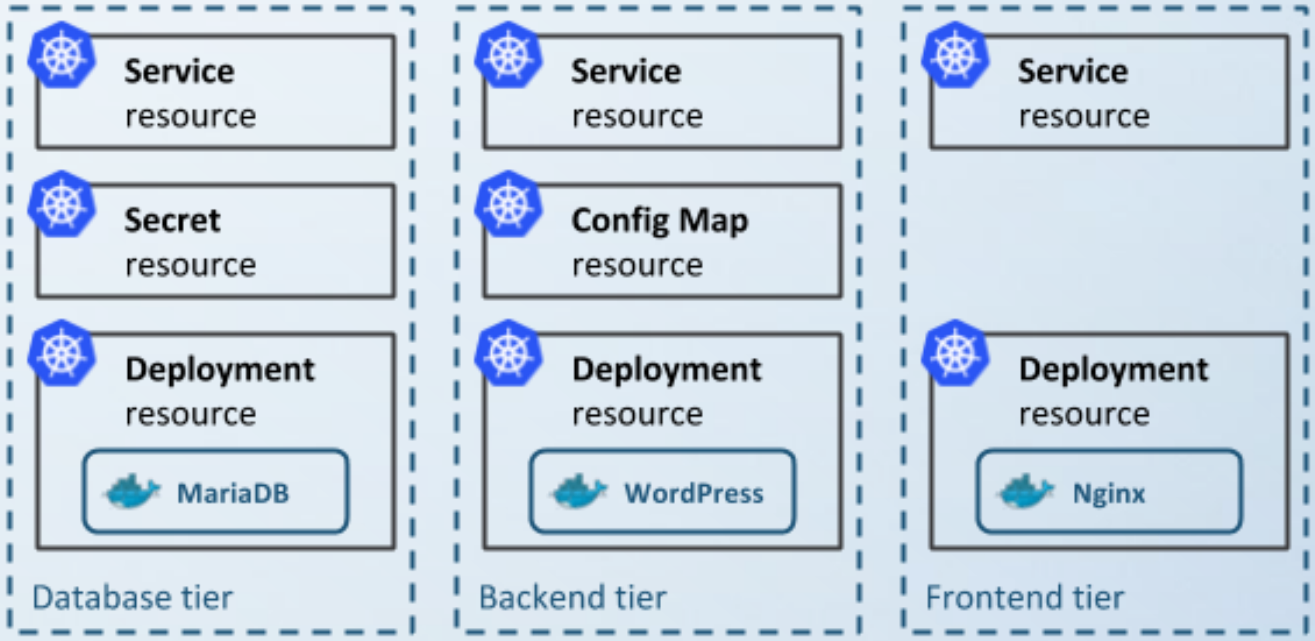
Ingress

Ingress позволяет распределять внешний трафик по сервисам в кластере с помощью правил



Распространение приложения

Современные приложения состоят из множеств компонентов, которые необходимо развернуть, сконфигурировать и настроить на работу друг с другом.

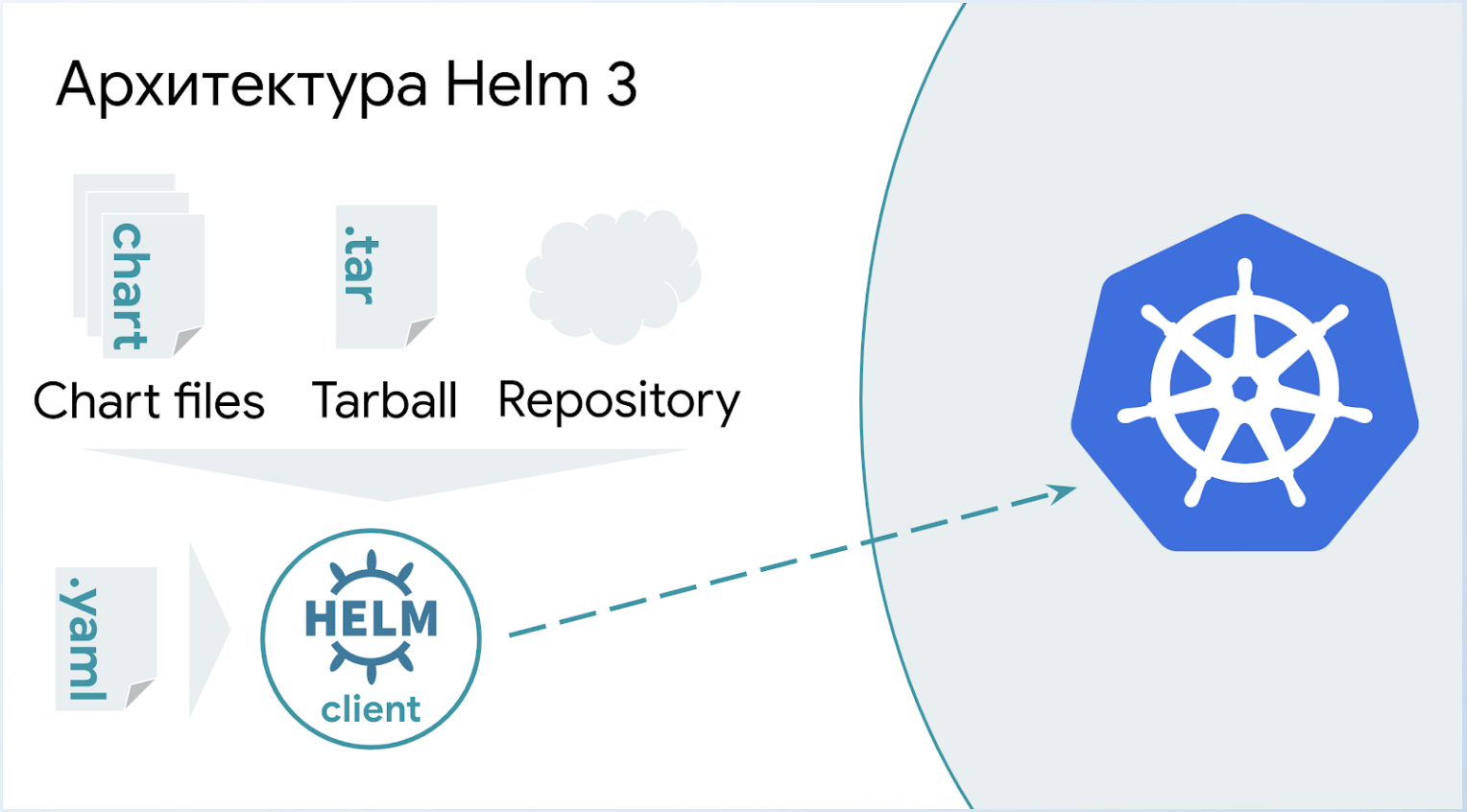


Helm

Пакетный менеджер для манифестов Helm позволяет разворачивать готовые приложения в несколько команд.

Helm позволяет хранить, управлять версиями и распространять шаблоны манифестов для деплоя приложений.

Архитектура Helm 3



Использование Helm

1. Смотрим стандартные переменные шаблонизации values.yaml
2. Переопределяем нужные нам параметры под наше окружение
3. helm install my-release grafana/Grafana –f my_values.yaml
4. helm upgrade my-release grafana/Grafana –f my_new_values.yaml
5. helm delete my-release

Время для вопросов