Диагностика, отладка и тестирование в Kubernetes

Не забудь включить запись!

План занятия

- Часть 1. Ops. `superkubectl`
- Часть 2. Dev. Локальная разработка
- Часть 3. О тестировании: `kubetest` и немного хаоса
- Часть 4. А если найду?

Часть 1. Ops

kubectl для самых маленьких

- Главный инструмент администратора - это `kubectl`
- Помимо базовых команд, поддерживает установку плагинов
  - самый простой способ установки (и, вдобавок, официальный) - `krew`
  - Это как `brew`, только для `kubectl`
  - Инсталляция - тут
  - `kubectl krew search` - список доступных плагинов

kubectl | top

```
$ kubectl top node
NAME                                          CPU(cores)   CPU%   MEMORY(bytes)
MEMORY%
gke-hipster-shop-default-pool-6c8b1884-23q7   80m          8%     1095Mi          94%
gke-hipster-shop-default-pool-6c8b1884-2bwh   279m         29%    955Mi           82%

$ kubectl top pod
NAME                                     CPU(cores)   MEMORY(bytes)
adservice-7fddbcf6f9-l92df               24m          193Mi
cartservice-86f7c988dd-zpdsf             7m           30Mi
```

Выводит список нод или подов и потребляемых ресурсов.

- `--selector` - задать фильтр по меткам
- `--sort-by` - отсортировать по `cpu|memory`

kubectl | describe

`kubectl describe` - выводит полную информацию о запрошенном объекте:

- текущую конфигурацию
- связанные события (events)
- текущее состояние (`PodStatus`, IP-адреса и подобное)

kubectl | describe

```
$ kubectl describe pod checkoutservice-845777b5f9-mb268

Name:           checkoutservice-845777b5f9-mb268
Namespace:      default
Priority:       0
Node:           gke-hipster-shop-default-pool-6c8b1884-416f/10.132.15.229
Start Time:     Fri, 23 Aug 2019 11:36:32 +0300
Labels:         app=checkoutservice
                pod-template-hash=845777b5f9
Annotations:    <none>
Status:         Running
IP:             10.36.4.4
Controlled By:  ReplicaSet/checkoutservice-845777b5f9
Containers:
  server:
    Container ID:   docker://b10258e24dff7ea869cbc69e1f467190179024e004090c47da0075e18c0e92a2
    Image:          gcr.io/google-samples/microservices-demo/checkoutservice:v0.1.2
    Image ID:       docker-pullable://gcr.io/google-samples/microservices-demo/checkoutservice@sha256:b6ec530b4ef237600a2ecb04054fe7d5d1b377c113cc927c87c5f8cd734a73fe
    Port:           5050/TCP
    Host Port:      0/TCP
    State:          Running
      Started:      Fri, 23 Aug 2019 11:36:37 +0300
    Ready:          True
    Restart Count:  0
    Limits:
      cpu:     200m
      memory:  128Mi
    Requests:
      cpu:      100m
      memory:   64Mi
    Liveness:   exec [/bin/grpc_health_probe -addr=:5050] delay=0s timeout=1s period=10s #success=1 #failure=3
    Readiness:  exec [/bin/grpc_health_probe -addr=:5050] delay=0s timeout=1s period=10s #success=1 #failure=3
    Environment:
      PORT:                          5050
      PRODUCT_CATALOG_SERVICE_ADDR:  productcatalogservice:3550
      SHIPPING_SERVICE_ADDR:         shippingservice:50051
      PAYMENT_SERVICE_ADDR:          paymentservice:50051
      EMAIL_SERVICE_ADDR:            emailservice:5000
      CURRENCY_SERVICE_ADDR:         currencyservice:7000
      CART_SERVICE_ADDR:             cartservice:7070
    Mounts:
      /var/run/secrets/kubernetes.io/serviceaccount from default-token-sflrb (ro)
Conditions:
  Type              Status
  Initialized       True
  Ready             True
  ContainersReady   True
  PodScheduled      True
Volumes:
  default-token-sflrb:
    Type:        Secret (a volume populated by a Secret)
    SecretName:  default-token-sflrb
    Optional:    false
QoS Class:       Burstable
Node-Selectors:  <none>
Tolerations:     node.kubernetes.io/not-ready:NoExecute for 300s
                 node.kubernetes.io/unreachable:NoExecute for 300s
Events:          <none>
```

kubectl | edit

Чтобы "быстро поправить на проде" можно использовать команду

```
kubectl edit ...
```

Это для экстренных ситуаций или dev-окружений:

- Противоречит идее Infrastructure As a Code
- Легко забыть, кто, что делал и почему не закоммитил в репозиторий
- В манифесте будет много лишнего (т.к. редактируем "живой" объект из базы K8s)

kubectl | port-forward

С помощью этой команды можно быстро "потыкать палочкой" нужный Pod.

```
kubectl port-forward service/myservice 5000 6000
```

- Проброс трафика происходит через `kube-apiserver`, поэтому не надо highload
- Можно указать не только под, но и сервис (ReplicaSet или Deployment), но проброс порта все равно произойдет на один конкретный под (**под упал == forward упал** и никакой балансировки).

kubectl | logs

Показывает логи контейнеров в поде.

Удобно делать так:

```
# Показать последние 10 строк лога всех контейнеров пода и затем выводить новые сообщения
kubectl logs --all-containers=true --follow --tail=10
```

- `--since` или `--since-time` - показать логи с определенного периода
- `--selector` - показать логи всех подов с определенными лейблами
- `--previous` - посмотреть логи старого контейнера, если был рестарт

kubectl || get

- `get events --watch` - посмотреть за event-объектами в кластере (изменения состояний объектов, фейлы проверок и т.п.)
- `get type/name -o yaml` - получить YAML с объектом из Kubernetes, если вдруг его потеряли, создали оператором или вручную и т.п.

kubectl || exec, attach, cp

- `exec` - выполняет команду в контейнере, опционально - в интерактивном режиме
- `attach` - подключиться к stdin/stdout основного процесса в контейнере
- `cp` - скопировать файл в контейнер

kubectl debug

- Это отдельный плагин, позволяет запустить специальный контейнер внутри Pod для отладки и диагностики.
- Работает в двух режимах - agentless и в режиме с агентом
  - *agentless* - экономит немного ресурсов на нодах. Но иногда может не хватить ресурсов на ноде для дебага
  - В режиме с агентом создается DaemonSet с предсозданным Podom
    - Экономит время при запуске отладки
    - Увеличвает шансы на выделение ресурсов debug-поду

kubectl debug

- Документация и установка плагина - тут
- Для Kubernetes планируется нативная поддержка debug-контейнеров - Ephemeral containers KEP
  - Пока даже не Alpha - код не принят в проект
  - планируются изменения в PodSpec
  - Будут отдельные сущности RBAC и еще много всего

kubectl debug

Как это работает:

- Агент - это специальный Pod, в который пробрасывается `docker.sock` с хост-системы и настроен проброс TTY-over-IP
- При запуске `kubectl debug` создается контейнер в namespace нужного нам Pod и stdin/stdout перенаправляются в сокет
- Агент по умолчанию использует сеть хоста, но если прямого доступа к ноде у нас нет (GKE) - выручает опция `--port-forward`
- Пока поддерживается только Docker-runtime

kubectl debug

```
$ kubectl debug emailservice-8d9d8bbcf-l827b --port-forward
pod emailservice-8d9d8bbcf-l827b PodIP 10.36.2.3, agentPodIP 10.132.15.227
wait for forward port to debug agent ready...
Forwarding from 127.0.0.1:10027 -> 10027
Forwarding from [::1]:10027 -> 10027
Handling connection for 10027
pulling image nicolaka/netshoot:latest...
latest: Pulling from nicolaka/netshoot
Digest: sha256:8b020dc72d8ef07663e44c449f1294fc47c81a10ef5303dc8c2d9635e8ca22b1
Status: Downloaded newer image for nicolaka/netshoot:latest
starting debug container...
container created, open tty...
bash-5.0# ps aux
PID   USER     TIME  COMMAND
    1 root     16:36 python email_server.py
1566892 root      0:00 bash
1566908 root      0:00 /bin/grpc_health_probe -addr=:8080
1566920 root      0:00 ps aux
```

kubectl debug

```
bash-5.0# cat /proc/1/root/email_server/email_server.py

#!/usr/bin/python
#
# Copyright 2018 Google LLC

from concurrent import futures
import argparse
import os
import sys
import time
import grpc
from jinja2 import Environment, FileSystemLoader, select_autoescape, TemplateError
from google.api_core.exceptions import GoogleAPICallError

import demo_pb2
import demo_pb2_grpc
from grpc_health.v1 import health_pb2
from grpc_health.v1 import health_pb2_grpc
```

kubectl debug

```
bash-5.0# strace -p 7 -c
strace: Process 7 attached
^Cstrace: Process 7 detached
% time     seconds  usecs/call     calls    errors syscall
------ ----------- ----------- --------- --------- ----------------
 27.90    0.000639           3       172           epoll_wait
 26.94    0.000617           4       137        52 recvfrom
 20.13    0.000461           7        61           sendto
  7.64    0.000175           1       102       102 stat
  7.29    0.000167           0       401           getpid
  6.68    0.000153           0       228           write
  2.18    0.000050           0        61           getsockopt
  1.22    0.000028           0        51        51 epoll_ctl
------ ----------- ----------- --------- --------- ----------------
100.00    0.002290                  1213       205 total
```

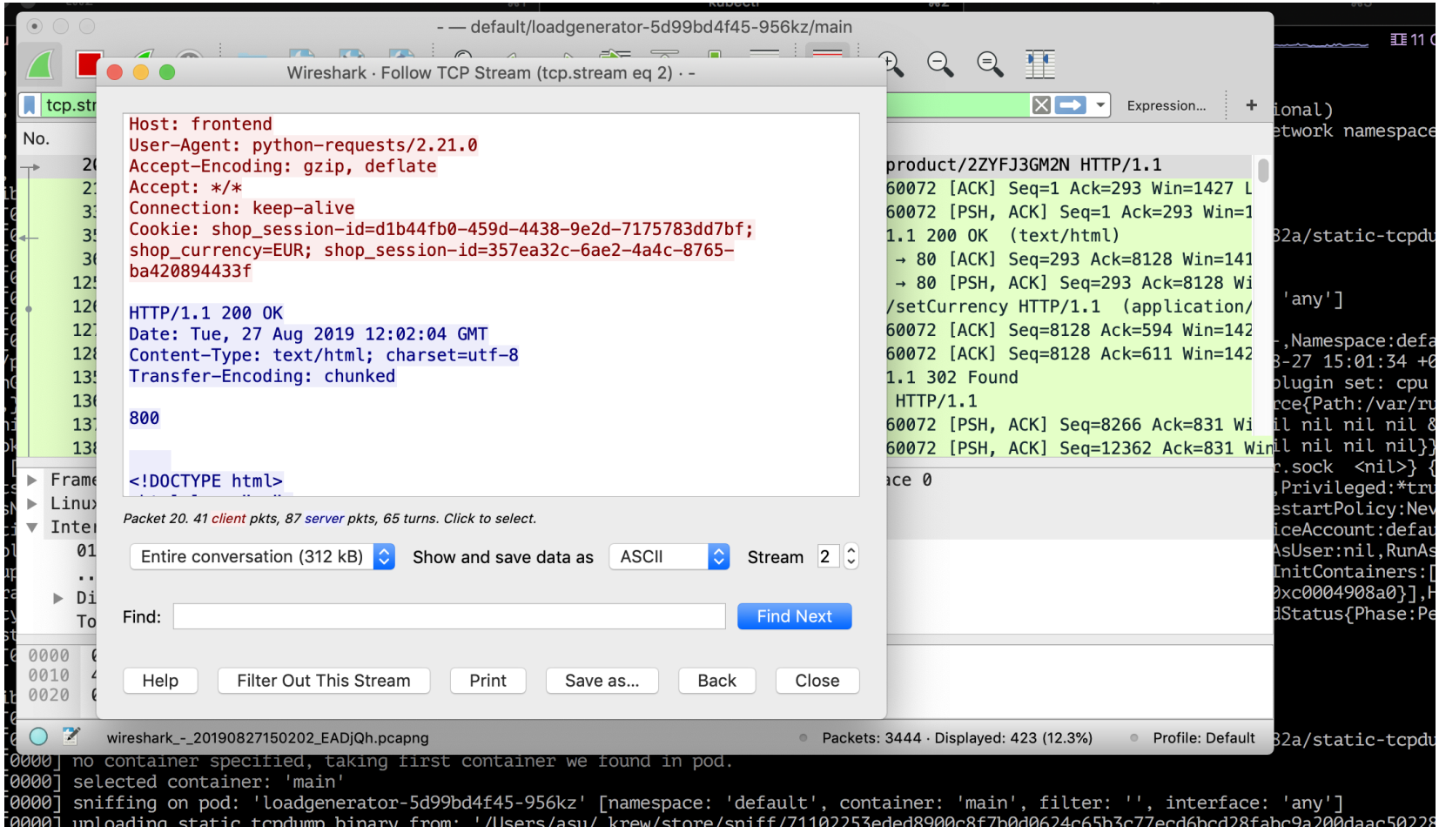
Немного про strace от Brendan Gregg

ksniff

Это снова плагин для `kubectl`, который позволяет посмотреть сетевой трафик в любимом Wireshark

- Инсталляция тут
- Запуск - `kubectl sniff <pod_name>`
- `ksniff` копирует статически слинкованный `tcpdump` в целевой контейнер (или подцепляет готовый образ к нужному Pod)
- Собранный `tcpdump`-ом трафик сливается в stdout и затем в Wireshark или файл на локальную машину
- Если Pod использует минимальный образ или запущен без привилегий, то добавляем флаг `-p` (создаст контейнер с нужными привилегиями в network namespace целевого Pod)

ksniff


- — default/loadgenerator-5d99bd4f45-956kz/main
Wireshark · Follow TCP Stream (tcp.stream eq 2) · -
Host: frontend
User-Agent: python-requests/2.21.0
Accept-Encoding: gzip, deflate
Accept: */*
Connection: keep-alive
Cookie: shop_session-id=d1b44fb0-459d-4438-9e2d-7175783dd7bf;
shop_currency=EUR; shop_session-id=357ea32c-6ae2-4a4c-8765-
ba420894433f
HTTP/1.1 200 OK
Date: Tue, 27 Aug 2019 12:02:04 GMT
Content-Type: text/html; charset=utf-8
Transfer-Encoding: chunked
800
<!DOCTYPE html>
Packet 20. 41 client pkts, 87 server pkts, 65 turns. Click to select.
Entire conversation (312 kB)
Show and save data as
ASCII
Stream
2
Find:
Find Next
Help
Filter Out This Stream
Print
Save as...
Back
Close
wireshark_-_20190827150202_EADjQh.pcapng
Packets: 3444 · Displayed: 423 (12.3%)
Profile: Default
[0000] no container specified, taking first container we found in pod.
[0000] selected container: 'main'
[0000] sniffing on pod: 'loadgenerator-5d99bd4f45-956kz' [namespace: 'default', container: 'main', filter: '', interface: 'any']

inspektor-gadget

Целая пачка полезных инструментов от компании Kinvolk. Заточена под их дистрибутив Linux, но может запускаться не только на нем.

Инсталляция требует чуть сложнее, чем через `krew`. Также потребуются телодвижения по настройке нод кластера и пересборка `runc`.

Подробности тут

inspektor-gadget

Те, кто поставят это чудо в кластер, обретут суперсилы, использующие BPF:

- Просмотр открываемых файлов, сетевых сокетов и запускаемых процессов в Pod
- Запрошенные Linux capabilites
- Трейсинг системных вызовов

> В целом, с помощью `kubectl-debug + strace` и `ksniff` можно получить примерно то же самое.

Часть 2. Dev

Разработка в Kubernetes

- В идеале, разрабатывать код, запускаемый в кластере, должно быть так же быстро и удобно, как и запускаемый локально.
- В идеале, мы должны тестировать еще и манифесты
- В идеале, мы не должны запускать весь стек локально

Скажем нет локальному окружению

А почему не Minikube, Kind и whatever?

- Что делать с облачными API и ресурсами? (или с bare-metal и злыми безопасниками?)
- Как тестировать распределенные системы? Множество инстансов, сетевые проблемы и тормозные распределенные ФС ждут нас!
- Где взять столько памяти, чтобы все наши микросервисы влезли в один ноутбук?

Ну давайте в CI засунем...

- Коммит сборка, unit-тесты, Security-тесты, деплой, забылопятьпробелвentrypoint.sh, начинаем все сначала
- Как это дебажить, если я - тут, а оно - там?
- У меня на `localhost` работало, а тут - не работает. Давайте выпилим Kubernetes. Жили же раньше.

Ksync

Это удобный инструмент для синхронизации файлов между машиной разработчика и подом в кластере K8s. Никаких пересборок и прочего.

Подробные инструкции по запуску и настройке есть в репозитории проекта

```
➜  ksync-test git:(master) ✗ curl https://webapp.34-77-181-114.nip.io/index.html
- hi!
➜  ksync-test git:(master) ✗ echo "- howdy?" >> ./index.html
➜  ksync-test git:(master) ✗ curl https://webapp.34-77-181-114.nip.io/index.html
- hi!
- howdy?
➜  ksync-test git:(master) ✗ echo "- fine. bye." >> ./index.html
➜  ksync-test git:(master) ✗ curl https://webapp.34-77-181-114.nip.io/index.html
- hi!
- howdy?
- fine. bye.
```

Ksync

- Синхронизация работает в обе стороны
- Состоит из DaemonSet в кластере и процесса `ksync watch` на локальной машине
- Под капотом - Syncthing (и всякие `.stignore` тоже работают)
- Работает с Docker-runtime и только на `overlay2`-драйвере
- Не работает с Volumes, даже если это `EmptyDir`
- Не работает, если сделать слэш к конце пути (`/app/`)
- Иногда просто не работает (до рестарта `ksync watch`)

Ksync

1. `ksync init`
2. `ksync watch`
3. `ksync create ./dirtosync ./poddir`
    - `--reload=true/false` - рестартовать ли под
    - `--name` - если рандомные `wild-octopus` не нравятся
    - `--selector` - куда синкаемся, назовите лейблы
    - `--(remote|local)-read-only` - исходная или конечная папка как read-only

Telepresence

Telepresence - это двухсторонний вариант прокси прямо в кластер. Позволяет запустить приложение или Docker-контейнер локально и...

- Получить доступ к сервисам в кластере (как будто приложение запущено там)
- Получить доступ к секретам, переменным окружения и прочему с локальной машины
- Обращаться к локальному процессу из кластера
- Подменять запущенные Deployments локальным инстансом

Если это перебор, есть `kubefwd`, `kuttle` и `warp`

Squash

А мне бы просто подебажить....

Для удаленной отладки уже запущенных приложений есть Squash

- Запускает привилегированные поды (как и все), которые используются для отладки
- Поддерживаются `gdb`, Java-отладчик и `dlv` (для Golang).
  - В планах - полноценная поддержка Python и Node.js
  - Сейчас есть legacy-способы (`ptvsd`)
- Основной IDE - Visual Studio Code. В планах IntelliJ и Eclipse

Squash

- В теории, Squash можно использовать и для отладки продуктивных приложений
- Для этого предусмотрен Safe Mode, который с помощью RBAC ограничивает доступ к отладке

Часть 3. Немного о тестировании

Kubetest

Надстройка на официальным Python-клиентом для Kubernetes и фреймворком `py.test`.

- Предоставляет `fixture` (ресурс для проведения теста) - динамический namespace, в котором будут развернуты нужные нам ресурсы (удаляется после завершения теста)
- Подгружает манифесты из YAMLов и создает абстракцию над ними (состояние, общие методы и т.п.)
- Это pure Python, а не YAML, можно делать что угодно. Например, "усилить" Kubetest с помощью TestInfra (или добавить Machine Learning 😄)

Kubetest

А зачем?

1. Тестирование - часть практики Infrastructure as a Code
2. Мы проверяем работу наших манифестов, получаем обратную связь, если что-то сломали
3. Проверяем работу приложения, до того как выкатим его куда-нибудь
4. Проверяем кластер на возможность запуска приложения (Pre-flight) - версии API, доступность ресурсов и т.п.
5. Можно провести End-to-End тестирование компонентов

Kubetest

Что не надо тестировать?

1. Перепроверять создание ресурсов - если манифест принят, то объект создан и надо смотреть за его состоянием
2. Валидировать YAMLики, если это могут сделать AdmissionControllers
3. Проверять то, что можно проверить через `liveness/readynessProbe`

Kubetest

- Код проекта доступен на Github, кстати это авторы ksync
- Часть ДЗ с помощью Kubetest
- Если вам нравится Golang, можно использовать Ginkgo как в тестах ingress-nginx и вообще в проекте Kubernetes
- Если Python - OK, а Kubetest - не OK, можно тестировать непосредственно Python-клиентом, как в тестах nginx-ingress

Немножко хаоса

Хаос-тестирование - горячая тема для всех, кто занят в разработке распределенных систем.

- Это намеренное внесение проблем в работу инфраструктуры и приложений, чтобы проверить их работу в условиях отказа.
- Правильное хаос-тестирование - это контролируемый эксперимент, проверяющий гипотезу.
- Бессмысленно создавать отказы там, где мы с ними ничего не делаем (например, ронять единственный инстанс БД)
- Результаты эксперимента должны приводить не к увольнениям, а к повышению надежности системы

Немножко хаоса

Что ломаем?

- Убиваем контейнеры, поды, ноды или дата-центры (как Netflix)
- Портим сетевую связность (ошибки, задержки, полные прерывания)
- Доводим нагрузку до предела (проверяем auto-scaling, мониторинг и балансировку)

Инструменты для хаос-тестирования | Pumba

Репозиторий проекта

Относительно простой инструмент, но создает весьма изощренные варианты отказа:

- Ставит на паузу процессы в контейнерах
- Эмулирует сетевые проблемы (джиттер, потери, порча, перестановки пакетов) с помощью `netem`
- Банальное убийство отдельных контейнеров

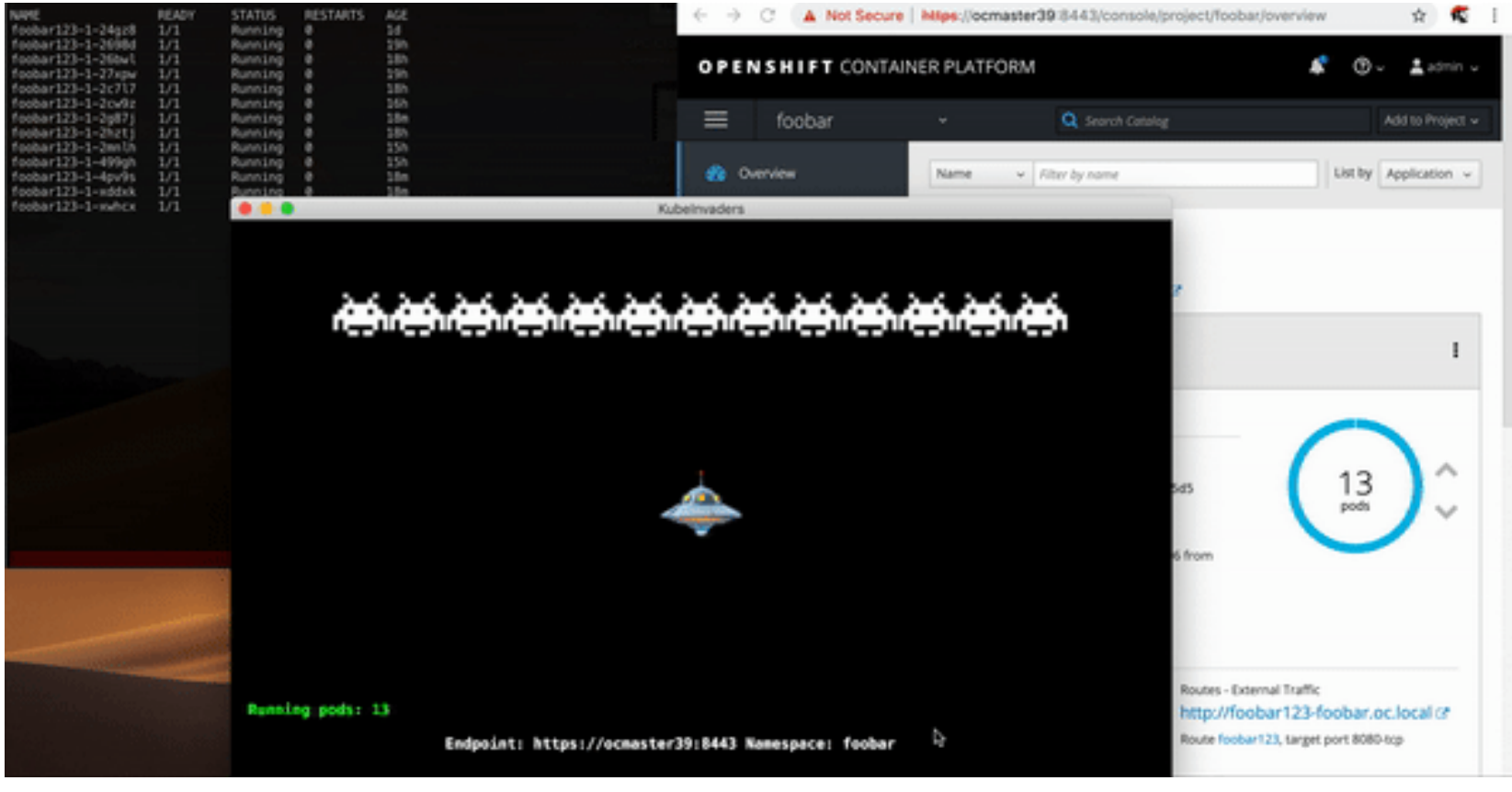
Инструменты для хаос-тестирования | Litmus Chaos Operator

Репозиторий проекта

- Проект заточен под тестирование приложений в Kubernetes
- Это полноценный оператор с CRDшками и блэкджеком
- Каждый эксперимент описывается манифестом
- Под капотом - Ansible и существующие инструменты для хаос-тестов
- Litmus обеспечивает полный цикл для проведения экспериментов: описание эксперимента, подготовка тестовой инфраструктуры, запуск эксперимента, сбор результатов

Инструменты для хаос-тестирования | KubeInvaders

Чисто ради веселья в пятницу вечером



Часть 4. Уязвимости есть? А если найду?

Что и зачем проверяем?

Что:

- K8s сложный, дырок много
- Linux сложный, дырок много
- Docker сложный, дырок много

Зачем:

- Нужно знать где и как (и почему) мы отступили от лучших практик
- Нужно знать, что мы запускаем
- Надо учить команды разработки хорошим практикам

Polaris

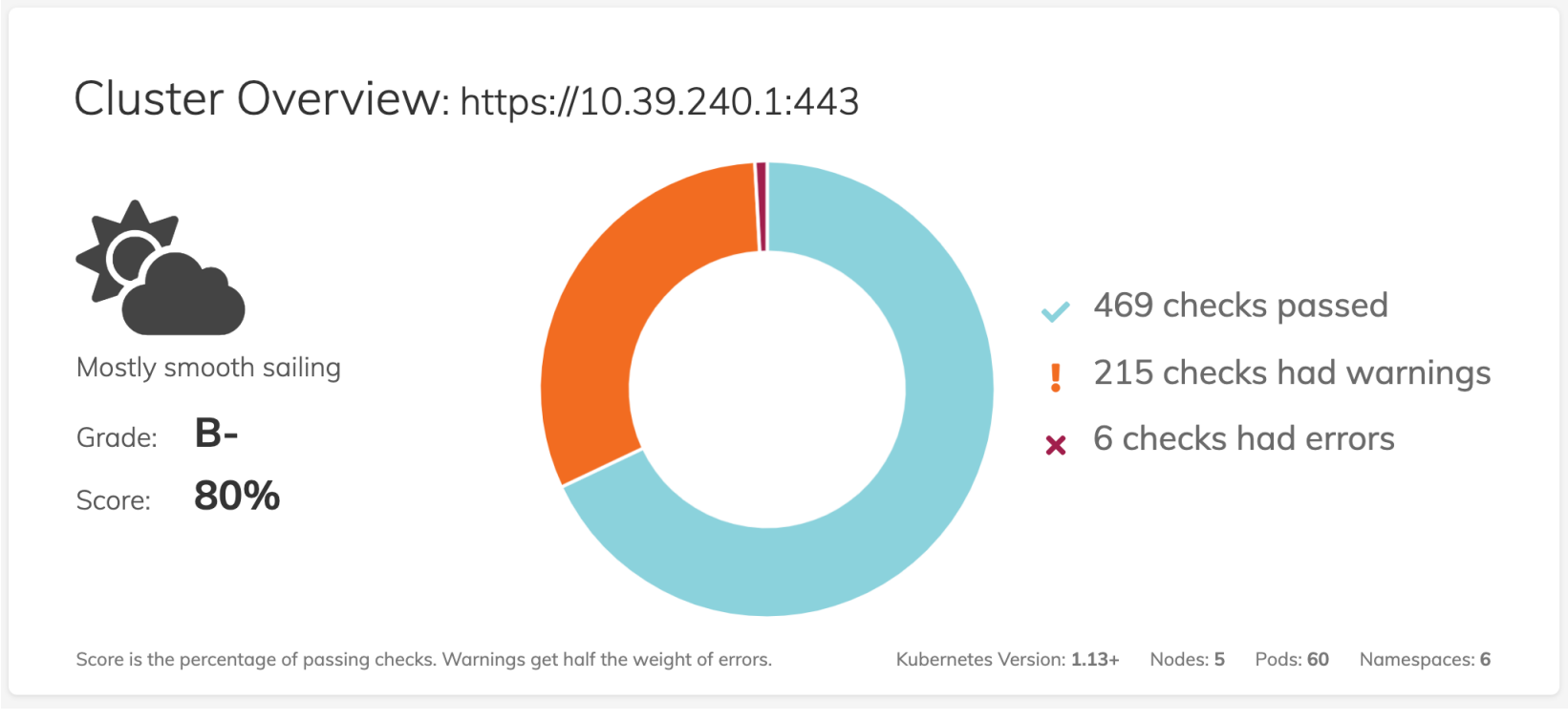
Самый простой "изкоробочный" инструмент - Polaris

Можно использовать в трех вариантах:

- Dashboard - симпатичный UI, в который нестыдно отправлять разработчиков манифестов
- CLI - в меру информативный инструмент для разового аудита (зато хорошо смотрится в CI)
  - Можно натравить на пачку YAMLов
  - Можно натравить на кластер
- Webhook - AdmissionController, который будет бить по рукам за нарушение правил

Polaris

```
$ polaris -audit -output-format score -set-exit-code-below-score 85 -set-exit-code-on-error
42
INFO[0001] 6 errors found in audit
```



Aquasecurity kube-bench

kube-bench - один из самых известных тестов на соответствие рекомендациям CIS

- Основной документ лежит здесь
- Другой известный инструмент для выполнения проверок, использует InSpec и несколько тяжелее в запуске, если ноды недоступны по SSH

Aquasecurity kube-bench

```
[FAIL] 2.1.1 Ensure that the --anonymous-auth argument is set to false (Scored)
[FAIL] 2.1.2 Ensure that the --authorization-mode argument is not set to AlwaysAllow
(Scored)
[FAIL] 2.1.3 Ensure that the --client-ca-file argument is set as appropriate (Scored)
[FAIL] 2.1.4 Ensure that the --read-only-port argument is set to 0 (Scored)
[FAIL] 2.1.5 Ensure that the --streaming-connection-idle-timeout argument is not set to
0 (Scored)
[FAIL] 2.1.6 Ensure that the --protect-kernel-defaults argument is set to true (Scored)
[FAIL] 2.1.9 Ensure that the --event-qps argument is set to 0 (Scored)
[FAIL] 2.1.10 Ensure that the --tls-cert-file and --tls-private-key-file arguments are
set as appropriate (Scored)
[FAIL] 2.1.12 Ensure that the --rotate-certificates argument is not set to false
(Scored)
[FAIL] 2.1.13 Ensure that the RotateKubeletServerCertificate argument is set to true
(Scored)
[FAIL] 2.2.8 Ensure that the client certificate authorities file ownership is set to
root:root (Scored)
[FAIL] 2.2.9 Ensure that the kubelet configuration file ownership is set to root:root
(Scored)
[FAIL] 2.2.10 Ensure that the kubelet configuration file has permissions set to 644 or
more restrictive (Scored)
```

Тестирование образов

Помимо тестирования манифестов и конфигурации кластера, надо следить, чтобы образы контейнеров тоже хоть чему-то соответствовали.

Есть два инструмента, которые удобно встраивать в CI и запускать периодически, чтобы не выпускать в прод "бракованные" контейнеры:

- Dockle - с уклоном на best-practices
- Trivy - с уклоном на поиск CVE

Dockle

```
$ dockle python:3.7-alpine
WARN  - CIS-DI-0001: Create a user for the container
        * Last user should not be root
INFO    - CIS-DI-0005: Enable Content trust for Docker
        * export DOCKER_CONTENT_TRUST=1 before docker pull/build
WARN    - CIS-DI-0006: Add HEALTHCHECK instruction to the container image
        * not found HEALTHCHECK statement
PASS    - CIS-DI-0007: Do not use update instructions alone in the Dockerfile
PASS    - CIS-DI-0008: Remove setuid and setgid permissions in the images
PASS    - CIS-DI-0009: Use COPY instead of ADD in Dockerfile
PASS    - CIS-DI-0010: Do not store secrets in ENVIRONMENT variables
PASS    - CIS-DI-0010: Do not store secret files
PASS    - DKL-DI-0001: Avoid sudo command
PASS    - DKL-DI-0002: Avoid sensitive directory mounting
PASS    - DKL-DI-0003: Avoid apt-get/apk/dist-upgrade
PASS    - DKL-DI-0004: Use apk add with --no-cache
PASS    - DKL-DI-0005: Clear apt-get caches
PASS    - DKL-DI-0006: Avoid latest tag
PASS    - DKL-LI-0001: Avoid empty password
PASS    - DKL-LI-0002: Be unique UID
PASS    - DKL-LI-0002: Be unique GROUP
```

Trivy

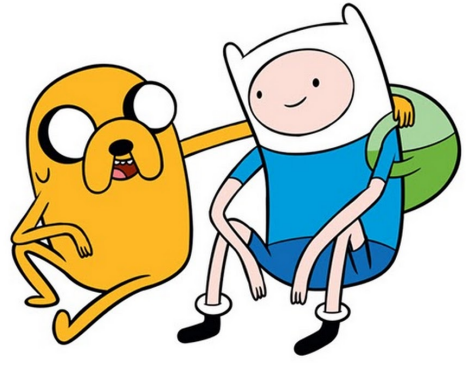
```
$ trivy python:3.7-alpine
2019-08-29T18:41:26.350+0300       INFO     Updating vulnerability database...
2019-08-29T18:41:28.803+0300    INFO     Detecting Alpine vulnerabilities...

python:3.7-alpine (alpine 3.10.2)
=================================
Total: 2 (UNKNOWN: 0, LOW: 1, MEDIUM: 1, HIGH: 0, CRITICAL: 0)

+----------+------------------+----------+-------------------+---------------+--------------------------------+
| LIBRARY  | VULNERABILITY ID | SEVERITY | INSTALLED VERSION | FIXED VERSION |             TITLE              |
+----------+------------------+----------+-------------------+---------------+--------------------------------+
| openldap | CVE-2019-13565   | MEDIUM   | 2.4.47-r2         | 2.4.48-r0     | openldap: ACL restrictions     |
|          |                  |          |                   |               | bypass due to sasl_ssf value   |
|          |                  |          |                   |               | being set permanently          |
+          +------------------+----------+                   +               +--------------------------------+
|          | CVE-2019-13057   | LOW      |                   |               | openldap: Information          |
|          |                  |          |                   |               | disclosure issue in slapd      |
|          |                  |          |                   |               | component                      |
+----------+------------------+----------+-------------------+---------------+--------------------------------+
```

README

1. Официальная документация K8s по отладке
2. Поучительные истории о проблемах с Kubernetes



Спасибо за внимание!

Время для ваших вопросов!