运维笔记

adamanteye

1. 概念

1.1. 开发

- The Twelve-Factor App

1.2. 部署

水平伸缩 提高副本数量

垂直伸缩 提高单实例资源

Rolling Update 逐个更新

Canary Deployment 只向部分用户推送新版,渐进更新

Infrastructure as Code 基础设施即代码(IsC)

1.3. 监控

2. CI/CD

2.1. GitHub Security

2.1.1. Dependabot

- 配置 Dependabot 版本更新 - GitHub 文档
- Dependabot 选项参考 - GitHub 文档

2.2. GitHub Actions

- GitHub Actions 的工作流语法 - GitHub 文档

2.2.1. 代码检出

默认的检出方式是浅克隆,即只包含最新提交,如果需要根据提交历史生成修改变更等,应当禁用浅克隆:

```yaml
- uses: actions/checkout@v4
  with:
    fetch-depth: 0
```

2.2.2. Self-hosted Runner

`systemd` 系统服务例子:

```
# /etc/systemd/system/github-actions-runner.service
[Unit]
Description=GitHub Actions Runner Service
After=network-online.target

[Service]
ExecStart=/home/ci-user/actions-runner/run.sh
WorkingDirectory=/home/ci-user/actions-runner
Restart=on-failure
RestartSec=5
User=ci-user
Group=ci-user
Environment="PATH=/home/ci-user/.local/bin:/usr/local/bin:/usr/bin:/bin"
Environment="all_proxy=http://[::1]:10801"

[Install]
WantedBy=multi-user.target
```

允许在 Self-hosted Runner 上运行容器内的作业,前提是容器内安装有对应的软件.例如 cloudflare/wrangler-action 需要 `npm` 以及 `node`.

2.2.3. 复用工作流

- 复用工作流 - GitHub 文档

2.2.4. 其他参考

- The Pain That is Github Actions

3. 自动化测试

3.1. 单元测试

原则

- Protection against regressions
- Resistance to refactoring
- Fast feedback
- Maintainbility

4. 网络基础设施

4.1. Cloudflare

4.1.1. Tunnel

CF Tunnel 可以方便地在中国地区搭建服务,例如:

```yaml
services:
  nginx:
    image: nginx:1.27.5-alpine
    volumes:
      - ./nginx.conf:/etc/nginx/nginx.conf:ro
      - /srv/:/srv/:ro
    restart: unless-stopped
    networks:
      - tunnel-net

  cloudflared:
    image: cloudflare/cloudflared:latest
    restart: unless-stopped
    command: [ "tunnel", "--no-autoupdate", "run", "--token", "your-token" ]
```

```
    depends_on:
      - nginx
    networks:
      - tunnel-net

networks:
  tunnel-net:
```

注意在面板中为服务 `http://nginx:80` 配置相应的域名.

4.1.2. Workers and Wrangler

更新 wrangler 版本时注意查看 Compatibility flags 以了解是否出现兼容性问题.

5. 监控与日志

Grafana 是开源的数据可视化平台,用于创建仪表盘,展示来自各种数据源的指标.结合数据收集代理 Telegraf 与高性能的时序数据库 InfluxDB,可以完成数据的采集,存储,查询展示的流程.

首先配置 Telegraf 采集数据,例如采集 CPU 负载:

```toml
[agent]
collection_jitter = "20s"
debug = false
hostname = "heloise"
interval = "20s"

# Read metrics about cpu usage
[[inputs.cpu]]
## Whether to report per-cpu stats or not
percpu = false
## Whether to report total system cpu stats or not
totalcpu = true
## Comment this line if you want the raw CPU time metrics
fielddrop = ["time_*"]
```

连接到 InfluxDB:

```toml
[[outputs.influxdb_v2]]
## The URLs of the InfluxDB cluster nodes.
##
## Multiple URLs can be specified for a single cluster, only ONE of the
## urls will be written to each interval.
##   ex: urls = ["https://us-west-2-1.aws.cloud2.influxdata.com"]
urls = ["http://influxdb2:8086"]

## Local address to bind when connecting to the server
## If empty or not set, the local address is automatically chosen.
# local_address = ""

## Token for authentication.
token = ""

## Organization is the name of the organization you wish to write to.
organization = ""

## Destination bucket to write into.
bucket = ""
```

最终在 Grafana 中添加数据源,配置仪表盘.

6. 证书签发

6.1. acme.sh

通过 Cloudflare 提供的 API 创建 TXT 记录以签发证书:

```shell
./acme.sh --issue --dns dns_cf -d thudep.com -d '*.thudep.com'
```

完成签发后,安装到指定位置:

```shell
./acme.sh --install-cert -d 'thudep.com' --fullchain-file /srv/cert/all.thudep.com.pem --key-file /srv/cert/all.thudep.com.key
```

这里拿到的证书是同样适用于二级域名的:

```
X509v3 Subject Alternative Name:
  DNS:thudep.com, DNS:*.thudep.com
```

可以通过查看 PEM 文件验证之:

```shell
openssl x509 -in <cert> -text -noout
```

此外,注意检查 `acme.sh` 是否创建了 crontab.

Let's Encrypt 并不支持签发多级通配符证书,这是出于防止滥用的考虑.

7. Podman

推荐 crun 作为运行时.

编辑`/etc/containers/registries.conf` 以添加镜像站:

```toml
unqualified-search-registries = ["docker.io"]
[[registry]]
location = "docker.io"
[[registry.mirror]]
```

```
location = "docker.thudep.com"
```

8. Docker

8.1. 文件系统

在 Btrfs 上的 Docker 会消耗大量 metadata.

8.2. 构建镜像

从含 `Dockerfile` 的路径构建镜像:

```shell
docker buildx build --tag nvchecker:latest .
```

8.2.1. 最小化打包

使用 alpine 镜像,并且从 alpine 安装软件包时使用`--no-cache` 选项:

```shell
apk add --no-cache bash
# 等价于
apk update && apk add bash && rm -rf /var/cache/apk/*
```

如果使用 debian 镜像,类似的减少体积的方式是

```shell
apt-get update && \
apt-get install -y --no-install-recommends bash && \
rm -rf /var/lib/apt/lists/*
```

- Super small Docker image based on Alpine Linux | Hacker News

8.2.2. 可复现构建

可复现构建的源头是保证上游的一致不变,在镜像打包的语义下,上游不仅仅是指开发者作为上游,从发行版安装的软件也是上游.

对于发行版,使用对应快照来锁定版本.

Debian 发行版: 仅仅依赖稳定版冻结软件包版本是不可靠的,因为还有可能从 debian-security 接受更新.

```shell
sed -i 's|http://deb.debian.org/debian-security|http://snapshot.debian.org/archive/debian-security/20250717T060459Z|g' /etc/apt/sources.list.d/debian.sources
sed -i 's|http://deb.debian.org/debian|http://snapshot.debian.org/archive/debian/20250718T082802Z|g' /etc/apt/sources.list.d/debian.sources
echo "Acquire::Check-Valid-Until false;" >> /etc/apt/apt.conf.d/no-check-valid-until
```

8.3. 运行容器

交互式运行容器,设置代理,挂载本地路径:

```shell
docker run -it --rm --name debian --network host -e HTTP_PROXY=http://[::1]:10801 -v "$HOME/Documents/debian:/home/debian:rw" -v "/etc/wgetrc:/etc/wgetrc:ro" -e DEBFULLNAME -e DEBEMAIL ghcr.io/adamanteye/debian-builder:master
```

`docker compose down && docker compose up -d` 与 `docker compose restart` 存在差别,推荐始终使用前一种.

- 后者不会更新环境变量

8.4. 清理

删除所有未使用镜像(不止 dangling):

```shell
docker image prune -a
```

8.5. 配置网络

在中国,免不了需要配置各种代理,为 docker daemon 配置代理可以通过编写`/etc/docker/daemon.json` 实现:

```json
{
  "proxies": {
    "http-proxy": "http://[::1]:10801",
    "https-proxy": "http://[::1]:10801"
  }
}
```

如果要为 docker container 配置代理,可以使用`--add-host` 配置容器与宿主机网络的映射,例如:

```shell
docker run -it --rm --add-host=host.docker.internal:host-gateway --name typst ghcr.io/adamanteye/typst:latest
```

进入容器后查看`/etc/hosts`,发现域名已经解析到宿主机地址:

```
172.17.0.1 host.docker.internal
```

在宿主机上,可以监听 `172.17.0.1` 这个地址并提供代理.而在容器中指定相应的环境变量:

```shell
export http_proxy='http://host.docker.internal:10801'
```

8.6. 托管镜像

托管镜像站时注意,如果开启了 htpasswd 鉴权,那么试图通过该镜像站拉取镜像将不再成功,因为 `docker pull` 并不会使用 `docker login https://docker.thudep.com` 时保存的凭据.相反,需要指定源:

```
docker pull docker.thudep.com/library/
nginx:alpine
```
Shell

而如果镜像站没有任何鉴权,那么在 `daemon.json` 里面配置的镜像站就可以在默认拉取 DockerHub 时生效了.

9. Kubernetes

Kubernetes 是源自 Google 的内部工具 Borg,现在成为了企业级容器和集群调度的工具.

Kubernetes 提供的能力有:

- 服务发现和负载均衡
- 存储编排
- 自动部署和回滚
- 水平扩展

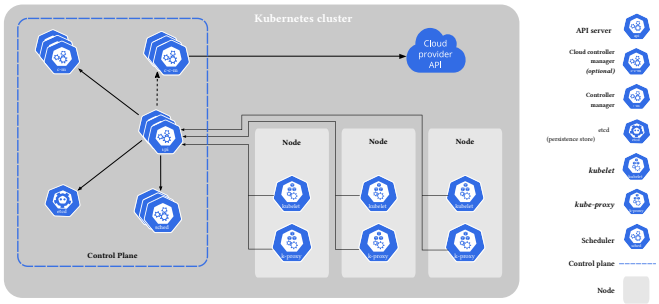

图 1　Components of Kubernetes

Pod　一组容器,被作为一个单元放置在相同节点上.例如一个数据库容器与后端服务容器,它们可以组成一个 Pod.

Label　每个 Pod 都有若干键值对,一个键值对就是一个 Label.可以通过 Label 来选中一组 Pod.

Scaling　根据 Label 以及期望的 Pod 数量,Kubernetes 中的 Replication Controller 可以缩放 Pod.

参考:

- Kubernetes 架构 | Kubernetes
- Hello Minikube | Kubernetes
- *B. Burns, J. Beda, K. Hightower, and L. Evenson, Kubernetes: up and running: dive into the future of infrastructure, Third edition. Beijing Boston Farnham Sebastopol Tokyo: O'Reilly, 2022.*
- *B. Burns, E. Villalba, D. Strebel, and L. Evenson, Kubernetes best practices: blueprints for building successful applications on Kubernetes, Second edition. Sebastopol, CA: O'Reilly Media, Inc, 2023.*

9.1. 配额

Kubernetes 利用 cgroup 来限制 Pod 的资源使用.最小的 CPU 单位是 `1m`,即千分之一个 CPU 核心.

- Resource Management for Pods and Containers | Kubernetes

9.2. Etcd

- Etcd - Wikipedia

9.3. 节点

节点(Node)可以是物理机或虚拟机,每个节点至少运行:

- kubelet
- 容器运行时

节点必须有不同的主机名.

9.4. Pod

每个 Pod 在集群范围内,都会分配到独立的 `IP`,例如 `10.x.x.x`.集群内的访问是不需要 NAT 的.

在同一 Pod 内的所有容器都可以在 `localhost` 上访问彼此的端口,代价是降低了容器间的隔离性,不同容器的端口可能冲突.

以上性质与 Docker 不同.

9.5. 容器

Kubernetes 支持的容器运行时为 containerd 与 CRI-O,以及兼容 CRI 的任何实现.

9.6. Gateway

2023 年末,Kubernetes 发布了 Gateway API,它旨在成为 Kubernetes 中暴露服务的新标准,替代 Ingress.

9.7. 卷

PersistentVolume　访问模式分为:

ReadWriteOnce　被一个 Node 以读写方式挂载,并且允许该 Node 上的多个 Pod 读写该卷.

ReadOnlyMany　被多个 Node 以只读方式挂载.

ReadWriteMany　被多个 Node 以读写模式挂载.

StorageClass 是集群管理员提前配置的,例如阿里云 ACK 集群上通过 csi-provisioner 提供不同类型的 StorageClass.开发人员声明 PersistentVolumeClaim,并且配置对应的 StorageClassName 后即可动态从中创建 PersistentVolume.

9.8. 网络

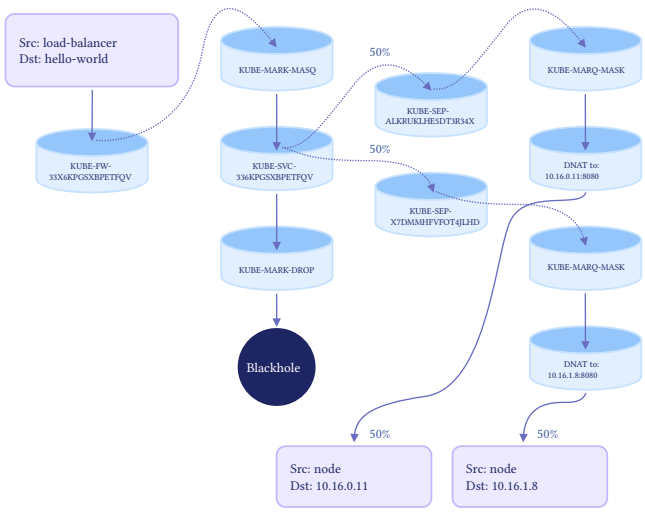

图 2 Kubernetes Networking Demystified

Flannel 是流行的 Kubernetes 容器网络插件,通过 VXLAN 创建 Overlay 网络.

9.8.1. DNS

在集群内部,服务被解析为 `<servicename>.<namespace>.svc.cluster.local`.

9.9. CRD & Controller

k8s 允许创建自定义资源(Custom Resource),注意 CRD 不是 ConfigMap,它的使用方法不是记录静态数据.相反,CRD 最常见的用法是声明式的集群对象,配合自己编写的 Controller,当 CRD 被改变时,Controller 对集群施加动作,使得集群对象变为期望的状态.

Pod,Deployment,StatefulSet 都是原生的 Kubernetes 资源.它们都有对应的 Controller 管理.

- Controller Overview
- Custom Resources

9.10. Operator

- Introducing Operators: Putting Operational Knowledge into Software

10. Kubernetes 部署

- What're people using as self-hoted/on-prem K8 distributions in 2025? : r/kubernetes

10.1. k3s

必须安装 iptables,只有 nft 是不够的.

默认的 kubeconfig 文件位于`/etc/rancher/k3s/k3s.yaml`.

Registry 配置位于`/etc/rancher/k3s/registries.yaml`.

```yaml
mirrors:
  docker.io:
    endpoint:
      - https://mirror.ccs.tencentyun.com
```

为了配置 server 的 `tls-san` 等参数,修改`/etc/rancher/k3s/config.yaml`:

```yaml
tls-san:
  - "10.0.0.1"
  - "k3s.adamanteye.cc"
cluster-cidr: "10.16.0.0/12"
service-cidr: "10.32.0.0/12"
```

agent 上不用做配置.

10.2. Flux

Flux 提供了可选组件,需要通过`--components-extra=image-reflector-controller,image-automation-controller` 安装.如果第一次未安装,附加该选项后再次 bootstrap.

- Building with FluxCD and Kubernetes - YouTube (精炼,适合入门)
- Connecting a Kubernetes cluster to GitLab (只参考创建 GitLab Agent 之前的内容,因为 GitLab Agent 和 Flux 没什么关系,也不好用)
- What is Flux CD & How Does It Work | Tutorial

11. Kubernetes 管理

11.1. kubectl

最常用的操作

- `kubectl top` 使用情况
  - `1m` 代表 1 millicore
- `kubectl get` 列出资源
- `kubectl describe` 显示资源详细信息
- `kubectl logs` 打印 Pod 中容器的日志
- `kubectl exec` 在 Pod 中的容器上执行命令

检查 `kubectl` 是否能与集群通信:

```shell
kubectl cluster-info
```

查看集群配置

```shell
kubectl config view
```

- Kubectl Commands - The Blue Book

11.2. Flux

查看 Kustomizations:

```shell
kubectl get kustomizations.kustomize.toolkit.fluxcd.io -A
```

或

```shell
flux get kustomizations
```

12. GitOps

Weaveworks 推广了 GitOps 的概念.GitOps 旨在通过声明式的配置文件维持集群.

常用的拉取-应用更改的工具有 Flux 和 ArgoCD.

`kustomization.yaml` 告诉了 Kubernetes 从哪里组装 `yaml` 文件,运行以下命令查看组装的结果:

```shell
kubectl kustomize .
```

常见的 `kustomazation.yaml` 为:

```yaml
apiVersion: kustomize.config.k8s.io/v1beta1
kind: Kustomization
resources:
- bar.yaml
- foo/
```

13. 邮件

MUA 邮件用户代理(Mail User Agent)是通常所说的邮件客户端,常见的 MUA 有 `mutt`,`thunderbird` 等.

MTA 邮件传输代理(Mail Transfer Agent)是邮件中继,专门用于接收已提交邮件并进行转发.

MDA 邮件投递代理(Mail Delivery Agent)负责接收来自 MTA 的电子邮件并放进收件人的信箱中.收件人通过 IMAP(S)同步邮箱状态.

邮件投递的要点是,当用户 Alice(例如 outlook)向用户 Bob(例如 gmail)发送邮件时,Alice 不会直接向 Bob 的 MTA 投递,而始终首先发到自己的 MTA.

13.1. 反垃圾邮件

What are DMARC, DKIM, and SPF? | Cloudflare 介绍了提高 MTA 可信任性的方法.

14. 跨域

15. 服务

15.1. Git

Git 可视化页面,以便在无 CI/CD,用户管理,Issues 等需求的情况下使用.

- mlan/gitweb - Docker Image | Docker Hub
- Gitolite

15.2. nginx

15.2.1. 参数

`client_max_body_size 1000M;` 调整大小,以满足反向代理 registry 的 Docker 镜像上传的需求,否则 `413 Request Entity Too Large`.

15.3. GitLab

通过 Docker 部署 GitLab

```yaml
services:
  gitlab:
    image: gitlab/gitlab-ce:18.2.1-ce.0
    container_name: gitlab
    restart: unless-stopped
    environment:
      GITLAB_OMNIBUS_CONFIG: |
        # Add any other gitlab.rb configuration here,
        each on its own line
        external_url 'https://gitlab.example.com'
        letsencrypt['enabled'] = false
        nginx['listen_https'] = false
        nginx['listen_port'] = 81
        registry_external_url 'https://registry.
        example.com'
        registry_nginx['listen_https'] = false
        registry_nginx['listen_port'] = 82
    ports:
      - '22:22'
    volumes:
      - '/srv/gitlab/config:/etc/gitlab'
      - '/srv/gitlab/logs:/var/log/gitlab'
      - '/srv/gitlab/data:/var/opt/gitlab'
```

此外一并配置反向代理.

15.4. GitLab Runner

```yaml
services:
  gitlab-runner:
    image: gitlab/gitlab-runner
    container_name: gitlab-runner
    restart: unless-stopped
    volumes:
      - /cache:/srv/gitlab-runner/cache
      - /var/run/docker.sock:/var/run/docker.sock
      - ./config.toml:/etc/gitlab-runner/config.toml
```

注意 GitLab 的文档提到了多种在 Docker Container 中配置 Docker Executor 的办法(Docker in Docker),最简单的是挂载 socks:

```toml
[[runners]]
  executor = "docker"
  [runners.cache]
    MaxUploadedArchiveSize = 0
    [runners.cache.s3]
    [runners.cache.gcs]
    [runners.cache.azure]
  [runners.docker]
    tls_verify = false
    image = "docker:28.3.2"
    privileged = false
    disable_entrypoint_overwrite = false
    oom_kill_disable = false
    disable_cache = false
    volumes = ["/cache", "/var/run/docker.sock:/var/
    run/docker.sock"]
    shm_size = 0
    network_mtu = 0
```

如果要在 Kubernetes 中部署,参考 Docker-in-Docker with TLS disabled in Kubernetes.

runner 应配置为 privileged 模式.

```toml
[[runners]]
[runners.kubernetes]
image = "ubuntu:24.04"
privileged = true
```

还可以配置缓存

```toml
[[runners.kubernetes.volumes.pvc]]
name = "docker-cache"
mount_path = "/var/lib/docker"
```

此外,参考 Enable registry mirror for docker:dind service 配置 CI 当中的镜像.

```toml
[[runners.kubernetes.volumes.config_map]]
name = "docker-daemon"
mount_path = "/etc/docker/daemon.json"
sub_path = "daemon.json"
```

16. 安全