# 第一章 中间件

## 1.1 ElasticSearch

### ES Info

Elastic-Compose

[Reference](https://juejin.im/post/5ba4c8976fb9a05cec4da9f5)

[es6.2.4为什么移除了Type](https://www.jianshu.com/p/2be99e253662)

#### Shard

[ES 如何进行分片、分表、副本数评估](https://oa.kedacom.com/confluence/pages/viewpage.action?pageId=28347395)

一个节点(Node)一般会管理多个分片，这些分片可能是属于同一份索引，也有可能属于不同索引。为了可靠性和可用性，同一个索引的分片尽量会分布在不同节点(Node)上。

自定义动态映射

在运行时增加新的字段

dynamic\_templates 完全控制新检测生成字段的映射

Index Template&Dynamic Template

#### Cluster info

Cluster Health

GET /\_cat/health?v



Pri：primary shard

Status

* green：每个index的primary shard和replica shard都是 active
* yellow：每个index的pri都是 active，部分rep不是active，即不可用。
* red：不是所有索引的pri都是active，即部分索引有数据丢失了

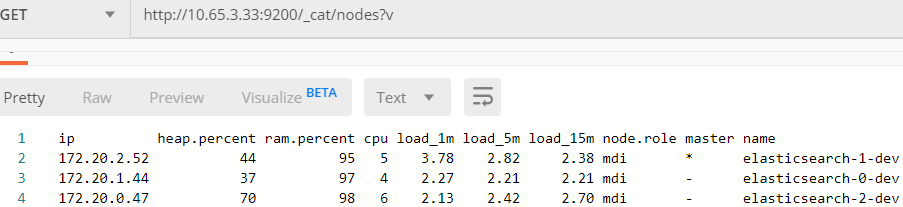
查看dev es

GET <http://10.65.3.17:9200>

1. {
2. "name": "elasticsearch-2-dev",
3. "cluster\_name": "es-dev",
4. "version": {
5. "number": "5.6.6",
6. "build\_date": "2018-01-09T23:55:47.880Z",
7. "lucene\_version": "6.6.1"
8. }
9. }

#### Nodes

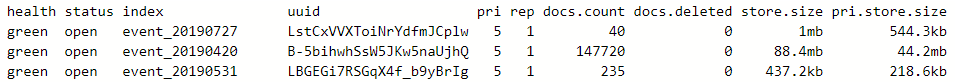
GET /\_cat/nodes?v



#### Indices

<http://10.65.3.17:9200/_cat/indices?v>

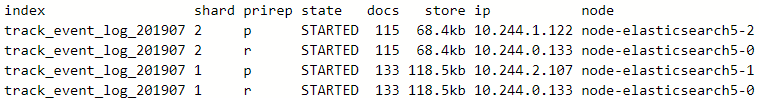
[http://10.65.3.17:9200/\_cat/indices/event\_\*?v](http://10.65.3.17:9200/_cat/indices/event_*?v)



* store.size: 索引总大小
* pri.store.size: 索引主分片大小

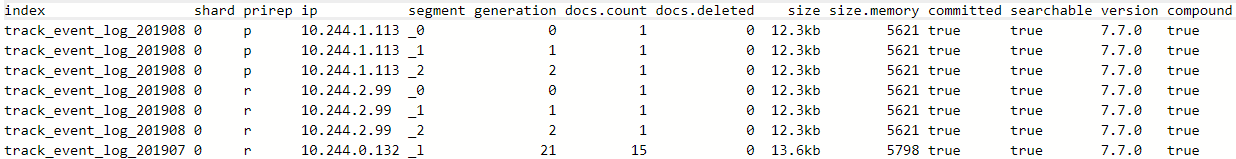
#### Shards

GET /\_cat/shards/track\_event\_log\_\*?v



#### segments

GET /\_cat/segments/track\_event\_log\_\*?v



### Index APIS

#### Settings

PUT /my\_temp\_index

{

"settings": {

"number\_of\_shards" : 1,

"number\_of\_replicas" : 0

}

}

用 update-index-settings API 动态修改副本数：

PUT /my\_temp\_index/\_settings

{

"number\_of\_replicas": 1

}

DELETE /my\_index

#### \_reindex

### Document APIS

#### Delete

POST /index-name/\_delete\_by\_query

可以加过滤参数，通query

1. {
2. "query": {
3. "term": {"\_id" : "AWvWAGMOJ7AmwS\_GxknK"}
4. }
5. }

#### Create

POST /index-name/index-type

Param

1. {
2. "name": "first",
3. "clientTime": 1567733117271,
4. "address":"苏州",
5. "age": 18,
6. "detail":"this is fine day"
7. }

#### Update

1.update by id

PUT /index/type/id

{

Filed:value

}

2.update by query

POST /index/\_update\_by\_query

{

"script": {

"source": "ctx.\_source.del = 1;",

"lang": "painless"

},

"query" : { {"term" : {"xxjsxtmc" : "xtmc1"}}}

}

### Query DSL

查询和过滤

"filter" 表示不评分、只过滤的查询，从 es2.0 开始，过滤（filters）已从技术上被排除，查询（queries）拥有不评分查询的能力。

setPostFilter

SearchRequestBuilder.setPostFilter(QueryBuilder postFilter).

Sets a filter that will be executed after the query has been executed and only has affect on the search hits(not aggregations).

This filter is always executed as last filtering mechanism.

query影响hits和aggs

{

"size": 10,

"query": {"match":{"\_id":"AWuiIP2jL\_yXvRf0iFHS"}},

"aggs": {

"avg\_age": { "avg": {"field": "age"}},

"max\_age": {"max": {"field": "age" }}

}

}

filter只影响aggs

{

"aggs": {

"age\_terms": {

"filter": {"match":{"\_id":"AWuiIP2jL\_yXvRf0iFHS"}},

"aggs": {

"max\_age": {

"max": {

"field": "age"

}

}

}

}

}

}

#### 精确值查找

不会计算相关度（直接跳过了整个评分阶段）且很容易被缓存。

url: [http://10.65.3.17:9200/event\_\*/\_search](http://10.65.3.17:9200/event_*/_search)

method: GET

1. {
2. "query" : {
3. .....json....
4. }
5. }

分页/排序

1. {
2. "from": 0,
3. "size": 10,
4. "sort": [{ "age": "desc" }],
5. }

term

用处理数字（numbers）、布尔值（Booleans）、日期（dates）以及文本（text）。

term vs match

match会对搜索关键字进行分词

1. {
2. "query": {"term": {"detail" : "here"} } ,
3. "sort": [{ "age": "desc" }],
4. "size":  100
5. }

terms

一个字段多个值

1. {
2. "query" : {
3. "terms" : {"action" : ["OPERATE\_LOG","OPEN\_APP"]}
4. }
5. }

#### 组合过滤器(bool)

bool

它接受多个其他过滤器作为参数，并这些过滤器结合成各式各样的布尔（逻辑）组合，构造复杂逻辑条件，

1. {
2. "bool" : {
3. "must" :     [],
4. "should" :   [],
5. "must\_not" : [],
6. }
7. }

must-and,must\_not-not,should-OR

**注意must和should之前是and的关系**

##### must/must\_not/should

1. "bool": {
2. "must":{
3. "term" : {"pttStatus" : "1"}
4. },
5. "should": [
6. { "match": { "action": "LOGIN" }}
7. ]
8. }

must有多个条件时可用 "must":[{ "term" : {"pttStatus" : "1"} ]

组合

1. "bool": {
2. "must": {
3. "term" : {"action" : "LOGIN"}
4. },
5. "filter": {
6. "range" : { "status" : { "gt" : 0 }}
7. }
8. }

##### term

1. {
2. "query" : {
3. "bool": {
4. "must":[
5. {"term":{"category": "APP\_MANAGER"}}
6. ]
7. }
8. }
9. }

##### filter

1. {
2. "query" : {
3. "bool": {
4. "filter": [
5. { "term":  { "action": "LOGIN" }},
6. {"range" : {"age" : { "gt" : 0 } }}
7. ]
8. }
9. }
10. }

##### exists

1. {
2. "query" : {
3. "bool": {
4. "must":[
5. {"term":{"category": "APP\_MANAGER"}},
6. {"exists" : {"field" : "userCode"}}
7. ],
8. "must\_not":[
9. {"exists": {"field": "value.userName"}}
10. ]
11. }
12. }
13. }

#### range

1. {
2. "query": {
3. "range" : { "age" : {"from":"3","to":10"} },
4. "range" : { "code" : { "gt" : 0 } }
5. }
6. }

#### full-text search

全文搜索

match

1. {
2. "query" : {
3. "match" : {"detail" : "here1 detail1"}
4. }
5. }

multi\_match

1. {
2. "query" : {
3. "multi\_match": {
4. "query":    "0",
5. "fields":   [ "userCode", "intoTimeStr" ]
6. }
7. }
8. }

fuzzy

f1

1. {
2. "query": {
3. "fuzzy" : { "detail" : "heba" }
4. }
5. }

f2

1. {
2. "query": {
3. "fuzzy" : {
4. "detail" : {
5. "value" :         "detail",
6. "boost" :         1.0,
7. "fuzziness" :     2,
8. "prefix\_length" : 0,
9. "max\_expansions": 100
10. }
11. }
12. }
13. }

分词 vs 模糊查找

#### Indices APIS

查看mapping [http://10.65.3.17:9200/event\_\*/\_mapping](http://10.65.3.17:9200/event_*/_mapping)

create

URL: http://ip:port/index-name

Method: PUT

Body

1. {
2. "settings" : {
3. "index" : {
4. "number\_of\_shards" : 3,
5. "number\_of\_replicas" : 1
6. }
7. },
8. "mappings":{
9. "test\_index\_type":{
10. "properties":{
11. "appId":{"type":"keyword"},
12. "name":{"type":"text"},
13. "age":{"type":"integer"},
14. "address":{"type":"keyword"},
15. "time":{"type":"date"},
16. "detail":{"type":"text"}
17. }
18. }
19. }
20. }

delete

URL: http://ip:port/index-name

Method: DELETE

hits

1. "hits": {
2. "total": 47158,
3. "max\_score": 6.8834624,
4. "hits": [
5. {
6. "\_index": "event\_20190725",
7. "\_type": "event",
8. "\_id": "AWwn6RtM4U5VpdVg4\_f-",
9. "\_score": 6.8834624,
10. "\_source": {
11. "intoTime": 1564037678643,
12. "action": "LOGIN",
13. "category": "APP\_MANAGER"
14. }
15. }
16. ]
17. }

total

max\_score

Query String&Simple Query String

q

**URL**: http://10.65.3.17:9200/event\_\*/\_search

Query Params

Key:q

Value:action:LOGIN

#### 部分匹配

prefix

1. {
2. "query": { "prefix": {"name": "a"}}
3. }

wildcard

范围查找

1. {
2. "query": {"wildcard": {"name": "dong\*"}}
3. }

支持模糊匹配

#### 控制相关度

### Aggs

聚合

高阶概念

桶（Buckets）：满足特定条件的文档的集合。

指标（Metrics）：对bucket内的文档进行统计计算。metric很像SQL中的avg、max、min等方法

1.分组

Buckets，类似Sql的group by

DSL

1. {
2. "aggs": {
3. "terms\_app\_id": {
4. "terms": {"field": "value.appId"}
5. }
6. }

aggs返回结构

1. {
2. "aggregations": {
3. "terms\_app\_id": {
4. "buckets": [
5. {
6. "key": "P8OMfaCYNMJEnuJuTbJI",
7. "doc\_count": 1
8. }...
9. ]
10. }
11. }

Aggregate

1. {
2. "aggs": {
3. "terms\_app\_id": {
4. "terms": {
5. "field": "value.appId",
6. "size": 10,
7. "order" : { "\_count" : "desc" }
8. }
9. }
10. }
11. }

Size控制返回值数量

Order排序

2.分组聚合

1. {
2. "aggs": {
3. "terms\_app\_id": {
4. "terms": {
5. "field": "value.appId"
6. },
7. "aggs": {
8. "app\_count": {"cardinality": { "field": "value.appId"} }
9. }
11. }
12. }

#### metrics

stats

1. "aggs": {
2. "age\_stats": {"stats": {"field": "age"}}
3. }

结果包含 count/min/max/avg/sum

cardinality

cardinality相当于sql的distinct

1. {
2. "size": 0,
3. "aggs": {
4. "app\_count": {"cardinality": { "field": "appId"}}
5. }
6. }

#### histogram

### 性能分析

#### health

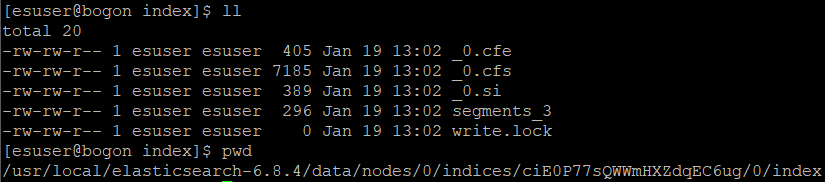
GET /\_cat/health?v



shards

1. {
2. "took": 77,
3. "timed\_out": false,
4. "\_shards": {
5. "total": 400,
6. "successful": 400,
7. "skipped": 0,
8. "failed": 0
9. },
10. "hits": {...}
11. }

Data文件



#### 监控

[监控指标](https://oa.kedacom.com/confluence/pages/viewpage.action?pageId=61768226)(confluence)

Solr

页面搜索严禁左模糊或者全模糊，如果需要请走搜索引擎来解决

Java REST Client

[Reference](https://www.elastic.co/guide/en/elasticsearch/client/java-rest/5.6/java-rest-overview.html)

## 1.2 Kafka

docker run -d --name kafka -p 9092:9092 -e KAFKA\_BROKER\_ID=0 -e KAFKA\_ZOOKEEPER\_CONNECT=172.17.0.1:2181 -e KAFKA\_ADVERTISED\_LISTENERS=PLAINTEXT://172.17.0.1:9092 -e KAFKA\_LISTENERS=PLAINTEXT://0.0.0.0:9092 -t wurstmeister/kafka

#### 客户端

启动Producer

./bin/kafka-console-producer.sh --broker-list localhost:9092 --topic test

启动Consumer

0.10.0版本：/bin/kafka-console-consumer.sh --zookeeper localhost:2181 --topic test [--from-beginning]

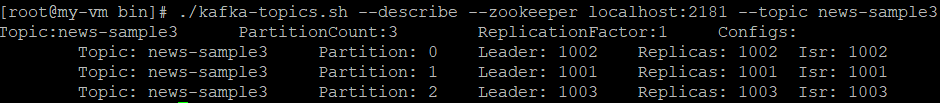
1.1.0版本：./bin/kafka-console-consumer.sh --bootstrap-server localhost:9092 --topic test --from-beginning

#### Topic

* Topic列表： bin/kafka-topics.sh -list -zookeeper localhost:2181
* 创建topic：bin/kafka-topics.sh --create --zookeeper localhost:2181 --replication-factor 1 --partitions 1 --topic test
* 删除topic： bin/kafka-topics.sh --zookeeper 127.0.0.1:2181 --delete --topic test

话题描述

bin/kafka-topics.sh --describe --zookeeper localhost:2181 --topic <topic>



* Leader：该partitons所在的所有broker中担任leader的broker id，每个broker都有可能成为leader
* Replicas：显示该partiton所有副本所在的broker列表，包括leader，不管该broker是否是存活，不管是否和leader保持了同步
* Isr：in-sync replicas的简写，表示存活且副本都已同步的的broker集合，是replicas的子集

# 第二章 部署平台

[Reference](https://zh.wikipedia.org/wiki/DevOps)

* 促成开发与运维的协作
* 让开发人员更多地控制生产环境
* 尽可能地自动化
* 更小、更频繁的变更──意味着更少的风险
* 定义简洁明了的流程
* 更多地以应用程序为中心来理解基础设施

[DevOps工程师到底是做什么的？](https://mp.weixin.qq.com/s?__biz=MzU0MzQ5MDA0Mw==&mid=2247486262&idx=1&sn=96412d0f34c6e3f38a160424c0c969c7&chksm=fb0be1a2cc7c68b44d82301c231f4d97e1d95609c70055da1884c005d0c8b33433cdfabddbe8&mpshare=1&scene=1&srcid=&sharer_sharetime=1563889304936&sharer_shareid=ff601b700721a407cdee60a9c63c1b87&key=4bf9c06e1cf58bb52765f6c3ca302fb7752d283fe6f7e070bf2f8dc6bdf2071be30c50bf07c768334a8917fa89dfc5f00893296ae18800b80e8c3b1ace381d53eed7ec886391048aa865d09ef7aeb1c8&ascene=1&uin=Mjc3ODQ1MTk0MA%3D%3D&devicetype=Windows+10&version=62060833&lang=zh_CN&pass_ticket=VMfG5kwDNh8gQEvN4QTgqw5sleXmlBtD1s0J%2FHWo1xPoU%2BhWNYjuoxI8qzHgm%2Fx7)

Java Client

[Reference](https://blog.csdn.net/Ay_Ly/article/details/90404445)

## 2.1 Docker

### 镜像

创建镜像：docker build -t ndong211/my:v1 .（注意这里有点）

查看镜像 docker images [name]

删除：docker rmi <image-id>

Dockerfile

1. FROM java:8
2. VOLUME /tmp
3. COPY ./my-1.0-SNAPSHOT.jar my.jar
4. RUN bash -c "touch /my.jar"
5. EXPOSE 8080
6. ENTRYPOINT ["java","-jar","my.jar"]

### 容器

创建容器 docker run --name names -d -p 80:80 repository:tag

Docker run = 用镜像创建容器 + docker start。

* -d：后台运行
* -p：端口映射 -P容器端口随机
* -i：以交互模式运行容器，常与-t同时使用；
* -t：为容器重新分配一个伪输入终端，常与-i同时使用。

示例：

docker run --name my-test -d -p 8080:8080 ndong211/my:v2

启动容器

查看所有容器：docker ps [-a]

启动：docker start/restart container\_id

停止：docker stop container\_id或container\_name

删除：docker rm container\_id

容器shell：docker exec -it <contain-id/name> /bin/bash

查看容器信息：docker inspect <container\_id?

Logs

查看内部标准输出 docker logs [–f] bf08b7f2cd89

### Docker compose

docker-compose.yml

1. version: '3'
2. services:
3. serviceName:
4. image: imageName
5. ......
6. networks:
7. ......

Image

### Repository

docker hub

[Reference](https://yeasy.gitbooks.io/docker_practice/repository/dockerhub.html)

1.登陆 docker hub

docker login

Username:ndong211

Password: ccx3801=123

2.docker pull/push <image-name>

push：docker pull ndong211/my:v2

搜索：docker search <name>

CMD

docker cp ./myhome <container-id>:/

## 2.2 kubernetes

[Reference](https://jimmysong.io/kubernetes-handbook/concepts/deployment.html)

### 2.1.1 Ops

Kubectl get <resource> [-o/output=json/yaml/wide]

yaml生成yaml

logs

kubectl logs <pod-name> [-n kube-system/ --namespace=kube-system]

### 2.1.2 管理对象

#### Objects

|  |  |
| --- | --- |
| 类别 | 名称 |
| 资源对象 | Pod、ReplicaSet、ReplicationController、Deployment、StatefulSet、DaemonSet、Job、CronJob、HorizontalPodAutoscaling、Node、Namespace、Service、Ingress、Label、CustomResourceDefinition |
| 存储对象 | Volume、PersistentVolume、Secret、ConfigMap |
| 策略对象 | SecurityContext、ResourceQuota、LimitRange |
| 身份对象 | ServiceAccount、Role、ClusterRole |

* Pod：一组关联的容器集合
* Namespace：对一组资源和对象的抽象集合
* Node：Pod真正运行的主机。K8s为了管理Pod，
* Service：对一组提供相同功能的Pods的抽象
* DaemonSet 守护进程集。DaemonSet保证在特定或所有Node节点上都运行一个Pod实例，常用来部署一些集群的日志采集、监控或者其他系统管理应用。典型的应用包括:

日志收集，比如fluentd，logstash等

系统监控，比如Prometheus Node Exporter，collectd等

系统程序，比如kube-proxy, kube-dns, glusterd, ceph，ingress-controller等

创建对象

通过配置文件名创建一个集群资源对象。支持json和yaml格式的文件。

kubectl create -f fileName

示例

kubectl create -f nginx-deploy.yaml

1. apiVersion: apps/v1beta1
2. kind: Deployment
3. metadata:
4. name: nginx-deployment
5. spec: …

Required Fields

* apiVersion：创建该对象所使用的 Kubernetes API 的版本
* kind：对象的类型
* metadata：帮助识别对象唯一性的数据，包括name,UID 和可选的 namespace

spec

1. spec:
2. replicas: 3
3. template:
4. metadata:
5. labels:
6. app: nginx
7. spec:
8. containers:
9. - name: nginx
10. image: nginx:1.7.9
11. ports:
12. - containerPort: 80

replicas:运行的副本数。

Selector:

template：定义Pod，包括Pod的名字，Pod拥有的label以及Pod中运行的应用。

#### Node

Node是kubernetes集群的工作节点，可以是物理机也可以是虚拟机

Node的状态

Node包括如下状态信息：

* Address

HostName：可以被kubelet中的--hostname-override参数替代。

ExternalIP：可以被集群外部路由到的IP地址。

InternalIP：集群内部使用的IP，集群外部无法访问。

* Condition

OutOfDisk：磁盘空间不足时为True

Ready：Node controller 40秒内没有收到node的状态报告为Unknown，健康为True，否则为False。

MemoryPressure：当node有内存压力时为True，否则为False。

DiskPressure：当node有磁盘压力时为True，否则为False。

* Capacity

CPU

内存

可运行的最大Pod个数

* Info：节点的一些版本信息，如OS、kubernetes、docker等

#### Namespace

集群可用ns创建多个“虚拟集群”，ns之间可完全隔离。用户应用默认在default下，与集群管理相关的应用一般部署在kube-system下。

kubectl get ns



#### Pod

Pod封装应用容器，存储、独立的网络IP，Pod代表着部署的一个单位：k8s中应用的一个实例，可能由一个或者多个容器组合在一起共享资源。

集群Pod有两种使用方式：

* 一个Pod中运行一个容器(常用)。
* 在一个Pod中同时运行多个容器。
* get：kubectl get pods [--all-namespaces/-A]
* delete：kubectl delete pods [pod-name]
* describe：kubectl describe pod <pod-name> [--namespace=kube-system]

创建pod

kubectl run <pod-nane> <--image=image> [--port=port] [--replicas=NUM]

* --replicas 副本数量

示例：

kubectl run nan-my1 --image=ndong211/my:v2 --port=8080

创建过程

[Reference](https://www.kubernetes.org.cn/5759.html)

* 缺省创建资源deployment,缺省参数–restart-policy=Always
* 创建过程：kubectl http--> kube-apiserver --> deployment 持久化到etcd 创建ReplicaSet和Pod，每个 Pod 都被调度到合适的节点  Kubelet按PodSpec定义，创建并启动容器

#### Deployment

* get：kubectl get deploy/deployments [name]
* create：kubectl create deployment <name> --image=image [--dry-run]
* delete：kubectl delete deployments <deploy-name>
* 扩容：kubectl scale deployment nan-my1 --replicas 2
* 更新镜像：kubectl set image deployment/nginx-deployment nginx=nginx:1.9.1
* 回滚：kubectl rollout undo deployment/nginx-deployment

ReplicaSet

kubectl get replicasets/rs

kubectl describe replicasets

Pause容器

又叫Infra容器。主要为每个业务容器提供以下功能：

* 在pod中担任Linux命名空间共享的基础；
* 启用pid命名空间，开启init进程。
* 解析：pause容器将内部的80端口映射到宿主机的8880端口

### 2.2.2 K8s架构

k8s架构



Master架构



Node架构



每个Node节点上至少需要运行container runtime（Docker）、kubelet和kube-proxy服务。

Kubelet

处理 Scheduler下发到本节点任务/管理Pod生命周期。

这意味着它将处理 Pod 与 Container Runtime之间所有的转换逻辑，包括挂载卷、容器日志、垃圾回收以及其他重要事件。

可以把 Kubelet当成一种特殊的 Controller，它每隔20秒向kube-apiserver查询 Pod，过滤 NodeName 与自身所在节点匹配的 Pod 列表。

Kubelet启动容器过程

拉取容器的镜像

通过 CRI 创建容器

启动容器

### 2.1.3 控制器

Kubernetes中内建了很多controller（控制器），用来控制Pod的具体状态和行为。

#### Deployment

Deployment 为 Pod 和 ReplicaSet 提供了一个声明式定义(declarative)方法，用来替代以前的ReplicationController 来方便的管理应用。典型的应用场景包括：

定义Deployment来创建Pod和ReplicaSet

* 滚动升级和回滚应用
* 扩容和缩容
* 暂停和继续Deployment

ReplicaSet

#### Horizontal Pod Autoscaling

[Reference](https://jimmysong.io/kubernetes-handbook/concepts/horizontal-pod-autoscaling.html)

应用的资源使用率通常都有高峰和低谷的时候，如何削峰填谷，提高集群的整体资源利用率，让service中的Pod个数自动调整。

HPA最能体现kubernetes之于传统运维价值的地方，不再需要手动扩容了，实现自动化。

kubectl autoscale (-f FILENAME | TYPE NAME | TYPE/NAME) [--min=MINPODS] --max=MAXPODS

[--cpu-percent=CPU] [flags] [options]

示例

kubectl autoscale deployment foo --min=2 --max=5 --cpu-percent=80

工作原理

kubernetes 能够根据监测到的 CPU 利用率（alpha版中支持应用提供的metric）自动的扩容 replication controller，deployment 和 replica set。

自定义指标

todo

### 2.1.4 服务发现

k8s负载均衡主要用到两种机制：

Service：集群内部的负载均衡，Kube-proxy负责将service请求负载均衡到后端的Pod中

Ingress：使用Ingress提供集群外部的负载均衡

#### Service

服务发现和负载均衡。

将资源暴露为新的Kubernetes Service，资源包括：pod，service，replication controller，deployment，replica set

* get：kubectl get svc/services
* delete：kubectl delete services <name>
* describe：kubectl describe services [label]

expose

kubectl expose <-f fileName | type name> [--port=port] [--target-port=port] [--type=type]

* port:绑定server端口
* target-port：容器端口，number-or-name
* type：NodePort/ClusterIP/LoadBalancer

示例：

1. 将pod暴露给外网

kubectl expose deployment nan-my1 --type=NodePort

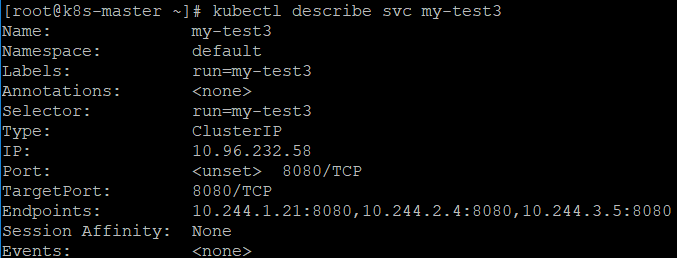
2.为rc创建service，并通过Service的80端口转发至容器的8000端口上。

kubectl expose rc nginx --port=80 --target-port=8000

3.由“nginx-controller.yaml”中指定的type和name标识的RC创建Service

kubectl expose -f nginx-controller.yaml --port=80 --target-port=8000

Describe Service

负

ServiceType

* ClusterIP：为service分配一个cluster内部可访问的虚拟IP，通过ip访问各节点（默认）。
* NodePort：在ClusterIP基础上，为Service绑定一个端口，通过NodeIP：NodePort来访问该服务
* LoadBalancer：在NodePort的基础上，通过外部的负载均衡器来访问，将请求转发到NodeIP:NodePort

负载分发策略

* RoundRobin：轮询模式，即轮询将请求转发到后端的各个pod上（默认模式）；
* SessionAffinity：基于客户端IP地址进行会话保持的模式，第一次客户端访问后端某个pod，之后的请求都转发到这个pod上。

#### Ingress&&Ingress Controller

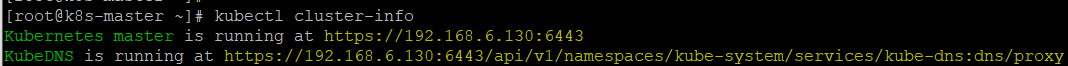
[Reference](https://imroc.io/posts/kubernetes/understand-service-ingress-and-ingress-controller/)

### 2.1.5 监控

[微服务架构之监控系统](https://mp.weixin.qq.com/s?__biz=MzU0MzQ5MDA0Mw==&mid=2247485699&idx=2&sn=cbe73bb979a003101dd540d37cc091f5&chksm=fb0be397cc7c6a81df6b1e025e9fd9dd1c835e279f2900abf551060f6d0fa0729adfe63137e6&mpshare=1&scene=1&srcid=&key=bd7df0327aba25f5be26f128207b10403780898f1ca928920fe9d29840abe0fd639e17167213106666aeec8961215cf6b5f0bcfdf7fc9a0c01befb13513fc9b2dfa8d5d6b7dcd298c4996cbb51e134fe&ascene=1&uin=Mjc3ODQ1MTk0MA%3D%3D&devicetype=Windows+10&version=62070152&lang=zh_CN&pass_ticket=MCT3fCw7MCE676VxtSEKLj06vJaYj34U2Mb9jdEdZh8vRmoBGD%2FJdxemhK0hUxv2)

### 查看集群

kubectl cluster-info



查看组件状态

kubectl get cs

kubectl get ep/endpoints

查看未授权的 CSR 请求

kubectl get csr (Certificate Signing Request)

### 2.1.5 存储

### 2.1.6 集群安全

用户

k8s 中有2种用户，一般用户和serviceAccount。

* 一般用户: 给集群外部用户使用，例如集群管理员通过 kubectl这个客户端来操作集群，使用的就是一般用户，这个用户通过 k8s 的 RBAC 权限系统获得相应权限。
* serviceAccount: 给集群内的资源使用，例如pod访问apiserver。每个pod都有一个 default serviceAccount。

Kublet的认证授权

* 匿名访问，调用 kubelet API 的能力应受到限制
* 启用 X509 客户端证书身份验证：
* 启用 API bearer token（包括 service account token）。

kubelet bootstrap

kubelet 使用低权限的 bootstrap token 跟 api server 建立连接后，要能够自动向 api server 申请自己的证书，并且 api server 要能够自动审批证书。

kubectl create clusterrolebinding kubelet-bootstrap --user=kubelet-bootstrap --clusterrole=system:node-bootstrapper

给bootstrap token代表的用户 system:bootstrap:abcdef 赋予 clusterole certificatesigningrequests.certificates.k8s.io/nodeclient 和 system:node-bootstrapper，让该用户可以访问 csr API 以及自动审批其创建的 csr

给新的 work node 代表的用户 system:node:test-node 赋予 clusterrole system:certificates.k8s.io:certificatesigningrequests:selfnodeclient，让它发送的证书 renew 的请求能被自动审批

kubectl 审批

# 通过 CSR 请求

kubectl certificate approve <name>

kubectl certificate deny <name>

命令扩展：

kubectl delete csr 节点名称 #删除单个节点的请求

kubectl delete csr --all #删除所有节点请求

kubectl delete nodes node名称 #删除加入的节点

kubectl delete nodes --all #删除所有节点

RoleBinding

如果想限制user用户行为，需要使用 RBAC创建角色绑定以将该用户的行为限制在某个或某几个 namespace 空间范围内，例如

赋予用户kubelet-bootstrap角色

kubectl create clusterrolebinding kubelet-bootstrap --clusterrole=system:node-bootstrapper --user=kubelet-bootstrap

kubectl get clusterrolebindings

否则报异常node exception：

server.go:271] failed to run Kubelet: cannot create certificate signing request: User "kubelet-bootstrap" cannot create certificatesigningrequests.certificates.k8s.io at the cluster scope. (post certificatesigningrequests.certificates.k8s.io)



roles

kubectl get clusterroles



RBAC-基于角色的访问权限控制

-authorization-mode=RBAC启动API Server

# 第三章 Linux

## 3.1 Base

### 环境变量

方法一 暂时生效,只会对当前回话生

export PATH=$PATH:/usr/local/apache-flume-1.6.0-bin

方法二对当前登陆用户生效，永久生效

vi ~/.bash\_profile

source ~/.bash\_profile

方法三 对所有系统用户生效，永久生效

vi /etc/profile

PATH=$PATH:/usr/local/apache-flume-1.6.0-bin

（export PATH）

最后执行命令 source /etc/profile

### 服务

服务：开机不用登录就可以执行的命令

开机启动

1.在路径/usr/lib/systemd/system添加服务文件updateDNS.service

2.开机启动systemctl enable updateDNS.service

3.修改后重启 systemctl daemon-reload

**原理**

systemctl enable/ disable updateDNS

原理：systemctl enable updateDNS.service时，updateDNS.service的链接文件/etc/systemd/system/ multi-user.target.wants/路径下。

**启动模式**

修改文件xx.service

[Install]

WantedBy=服务组 Target

multi-user.target 这个组开机都启动。缺省模式。

graphical.target：图形用户状态。

服务配置

路径1：/usr/lib/systemd/system 开机不用登录

路径2：/usr/lib/systemd/user/

路径3：/etc/systemd/system

路径下有四种类型文件

xxx.mount 挂载点。

xxx.service 定义服务

xxx.target 基础组件，供.service调用

xxx.wants 服务组

xx.service

三部分： Unit、Service、Install

[Unit]

Description=update dns server at home machine

[Service]

ExecStart=/usr/local/myservice/updateDNS/start.sh

[Install]

运行模式。

Service

执行/etc/init.d/目录下脚本

增加服务方法

1.脚本，普通shell，但要实现service的start等， 参考/etc/init.d/updateDNS

2.updateDNS脚本放/etc/init.d/目录

开机启动

chkconfig [--level 3] httpd on/off

或者chkconfig --add/del updateDNS。(原理：执行后/etc/rc2.d,/etc/rc3.d等添加K10/S10链接)

chkconfig --list

service updateDNS start 相对于 updateDNS start，不同在于脚本在init.d目录。

很多服务以d结尾，d代表 deamon 守护进程

init.d

/etc/init.d 是 /etc/rc.d/init.d 的软链接(soft link)。放init.d保证不同linux版本兼容性。

chkconfig

配置某服务开关机自动启动。默认启动2345四个级别。

执行级别

等级0表示：表示关机

等级1表示：单用户模式

等级2表示：无网络连接的多用户命令行模式

等级3表示：有网络连接的多用户命令行模式

等级4表示：不可用

等级5表示：带图形界面的多用户模式

等级6表示：重新启动

修改 chkconfig --level 35 updateDNS on

### Log

系统日志一般都存在/var/log下

常用的系统日志如下:

核心启动日志:/var/log/dmesg

系统报错日志:/var/log/messages

邮件系统日志:/var/log/maillog

FTP系统日志:/var/log/xferlog

/var/log/messages

## 3.2 文件管理

### 系统目录

/bin 二进制可执行命令。一般用户可用。ls,cp

/sbin 系统管理命令。shutdown,reboot。

/etc 系统管理和配置文件

|--/etc/rc.d 启动的配置文件和脚本

|--/etc/init.d

/home 用户主目录的基点，比如我的主目录就是/home/nadong，可用~表示

/usr 最庞大的目录，要用到的应用程序和文件几乎都在这个目录。其中包含：

|--/usr/local 本地安装的程序，配置和其他。系统升级无需重新安装。

|--/usr/local/bin 本地增加的命令

|--/usr/local/lib 本地增加的库 比如java

|--/usr/local/etc配置文件

|--/usr/local/share其它的资源文件放在。

|--/usr/bin 预装，随系统升级改变。 和/usr/local/bin优先级看path配置。

|--/usr/sbin 非必须的系统管理命令。

|--/usr/doc linux文档

|--/usr/include linux下开发和编译应用程序所需要的头文件

|--/usr/lib 常用的动态链接库和软件包的配置文件

|--/usr/man 帮助文档

|--/usr/src 源代码，linux内核的源代码就放在/usr/src/linux里

|--/usr/x11r6 存放x window的目录

/root 系统管理员的主目录（特权阶级）

/lib 标准程序设计库，又叫动态链接共享库，作用类似windows里的.dll文件

/dev 设备特殊文件

/tmp 公用的临时文件存储点

/var 某些大文件的溢出区，比方说各种服务的日志文件

/mnt 系统提供这个目录是让用户临时挂载其他的文件系统。

/proc 虚拟的目录，是系统内存的映射。可直接访问这个目录来获取系统信息。

/lost+found 平时空的，系统非正常关机而留下“无家可归”的文件。

Warning

usr 指 Unix System Resource，而不是User

path配置，前面覆盖后面的。

### 文件属性

文件颜色

绿色文件---------- 可执行文件，可执行的程序

红色文件-----------压缩文件或者包文件

蓝色文件----------目录

白色文件----------一般性文件，如文本文件，配置文件，[源码](https://www.2cto.com/ym/)文件等

浅蓝色文件----------链接文件，主要是使用ln命令建立的文件



链接文件

分配不同权限

缩写路径

软链接 VS 硬链接

软链接和Windows的快捷方式类似

文件权限

chmod

chmod u+w test

chmod ugo+r test;

chown

修改文件拥有者。一般只有系统管理员(root)拥有此操作权限。

### 文件搜索

which       查看可执行文件的位置   
whereis    查看文件的位置   
locate       配合数据库查看文件位置   
find          查询文件名称

find

find [dir] -name filename

1.在当前及子文件夹查找

find –name common.jar

2.在etc及子文件夹查找

find /etc -name httpd.conf

3.模糊查找

find –name ‘httpd\*'

Grep

-n ：输出行号

-r ：在某一目录下递归[所有子目录]查找某一字串:

grep -r ‘str\*' . (注意此处有点)

grep -r a /scratch

Whereis

whereis useradd

### 文件查看及编辑

more

Enter:一行

Space:翻页

less

less 的用法比起 more 更加的有弹性。在 more 的时候，我们并没有办法向前面翻， 只能往后面看

Space:翻页

G:倒序查看

vim

**全部删除：**按esc后，然后dG

lsof

list open files

查看端口连接情况

lsof -i:80



## 3.3 系统资源管理

### CUP线程

#### top

–p pid 监控某个进程

-H show thread, top –Hp pid

4 users：当前登录user数

Tasks:进程数。-

Cpus(s)

us/sy — 用户/内核 占用CPU的百分比。

ni — 改变过优先级的进程占用CPU的百分比

id — 空闲CPU百分比

wa — IO等待占用CPU的百分比

hi — 硬中断（Hardware IRQ）占用CPU的百分比

si — 软中断（Software Interrupts）占用CPU的百分比

Title

PR 优先级

VIRT 进程使用的虚拟内存总量，单位kb。VIRT=SWAP+RES

RES 进程使用的、未被换出的物理内存大小，单位kb。RES=CODE+DATA

SHR 共享内存大小，单位kb

%CPU  CPU时间占用百分比

%MEM  进程使用的物理内存百分比

TIME+  进程使用的CPU时间总计，单位1/100秒  
CMD 进程名称（命令名/命令行）

**快捷键**

P:CPU排序。默认按cpu占用量排序。

M:内存排序

H: show thread

1:show cups 列出单个cpu的负荷情况

#### ps

查看线程。Process Status

a 显示现行终端机下的所有程序，包括其他用户的程序

f 用ASCII字符显示树状结构，表达程序间的相互关系

e 列出程序时，显示程序所使用的环境变量

-A 显示所有进程

-u 以用户为主的格式来显示程序状况

-x 显示所有程序，不以终端机来区分

常用

ef



USER 用户名

UID User ID

PID Process ID（进程），当然jps命令也可以查到进行ID

PPID parent pid

TIME 进程使用的总cpu时间

COMMAND 正在执行的命令行命令

aux



SID 会话ID（Session id）

%CPU **进程的cpu占用率，单个CPU，双核的最大值为200%**

%MEM 进程的内存占用率

VSZ 进程所使用的虚存的大小（Virtual Size）

RSS 进程使用的驻留集大小或者是实际内存的大小，Kbytes字节。

TTY 与进程关联的终端（tty）

STAT 进程的状态：进程状态使用字符表示的（STAT的状态码）

START 进程启动时间和日期

### 内存

#### free

[Reference](http://www.cnblogs.com/coldplayerest/archive/2010/02/20/1669949.html)



-/+ buffers/cache: 应用程序认为系统被用掉多少内存

设置缓存大小：

### 硬盘

#### df

Disk File

-i 查看inodes

df –hl

df -h /home/nadong

#### du

Disk usage of each File

du –sh

分区

### 4.3.4 网络

net-tools VS IPRoute2

[Reference](https://blog.csdn.net/zhaihaifei/article/details/54617516)

|  |  |
| --- | --- |
| Net-tools | IPRoute2 |
| ifconfig | ip link/addr |
| Ifconfig –a | ip addr show |
| Ifconfig eth0 up | ip link set eth0 up |
| netstat | ss |
| route | Ip route |
| arp –na | ip neigh |

link：device信息，包括 MTU，MAC 地址等。

addr/address：link信息+ IP信息。

#### Net-tools

##### netstat

系统所有的网络套接字连接情况，包括 tcp, udp 以及 unix 套接字。

Active Internet connections，称为有源TCP连接

Active UNIX domain sockets，称为有源Unix域套接口(和网络套接字一样，但是只能用于本机通信，性能可以提高一倍)

Param

-a (all)显示所有选项，默认不显示LISTEN相关

-t (tcp)仅显示tcp相关选项

-n 拒绝显示别名(域名)，能显示数字的全部转化成数字(ip)。

-u (udp)仅显示udp相关选项  
-p 显示建立相关链接的程序名

-l 仅列出有在 Listen (监听) 的服務状态

-x只列出所有监听 UNIX 端口

提示：LISTEN和LISTENING的状态只有用-a或者-l才能看到

-p 显示建立相关链接的程序名  
-r 显示路由信息，路由表  
-e 显示扩展信息，例如uid等  
-s 按各个协议进行统计  
-c 每隔一个固定时间，执行该netstat命令

**常用**

1)netstat -at|grep slc01boa

2)统计80端口连接数

netstat -nat|grep -i "8080"|wc –l

3)Linux下查看tomcat连接数

netstat -na | grep ESTABLISHED | grep 8080 | wc -l

1. 统计httpd协议连接数  
   ps -ef|grep httpd|wc –l
2. 统计已连接上的，状态为“established

netstat -na|grep ESTABLISHED|wc -l

3)找出程序运行的端口

netstat -ap | grep java

**4)**[根据端口查找进程](http://www.cnblogs.com/paul8339/p/6638370.html)

netstat -apn | grep 8989

netstat -n | awk '/^tcp/ {++S[$NF]} END {for(a in S) print a, S[a]}'

windows

netstat -ano -p tcp -n 1|find "8080"

-n 1间隔1s自动刷新



PID是进程

统计

netstat -an|find "ESTABLISHED" /c

title

local address本机的地址

foreign address 外部电脑电脑有联系的ip地址

##### ifconfig

[Reference](https://zhuanlan.zhihu.com/p/27551846)



ens3s0: <UP,BROADCAST,RUNNING,MULTICAST>

* UP 代表网卡是开启状态
* RUNNING 代表这个网卡是处于网络连接状态
* MULTICAST 代表这个网卡是支持组播的

Ether 网卡mac地址

最后四行

* RX packets 网卡接收数据包信息
* RX errors 网卡接收数据时错误的信息
* TX packets 网卡发送数据包的信息
* TX errors 网卡发送数据是错误的信息

网卡名

CentOS7采用dmidecode采集命名方案，en表示ethernet以太网

enX（X常见有下面3中类型）：

o:主板半载网卡，集成设备的设备索引号

p:独立网卡，PCI网卡

s:热插拔网卡，USB之类

nnn（数字）表示：MAC地址+主板信息计算得出的唯一序列

##### route

-n 不解析名字

-v 显示详细的处理信息



Gateway为0.0.0.0的网络，通常是直连到网络设备上的，不需要网关的。

Flags标志说明：

U Up表示此路由当前为启动状态

H Host，表示此网关为一主机

G Gateway，表示此网关为一路由器

#### IPRoute2

#### 网卡管理

启动：ip link set <eth0> up/down

添加ip：ip addr add/del <192.168.0.100/24> dev <dev-name>

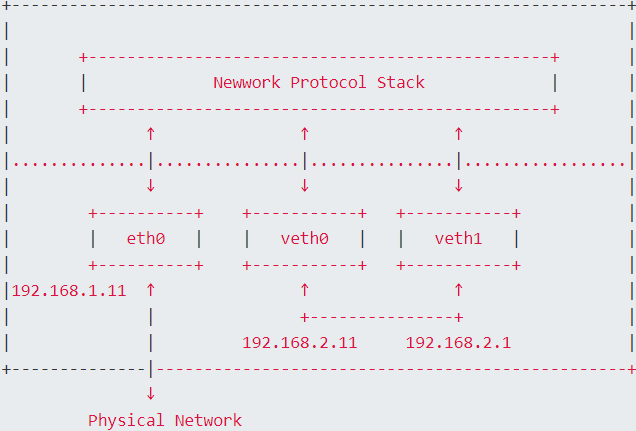
##### veth虚拟网卡

Virtual Ethernet 。一种成对的设备，专为container所建。把从一个network namespace 发出的数据包转发到另一个 namespace。

仅有veth-pair设备，容器是无法访问网络的。因为容器发出的数据包，实质上直接进入了veth1设备的协议栈里。如果容器需要访问网络，需要使用bridge等技术，将veth1接收到的数据包通过某种方式转发出去。

创建

ip link add <veth0> type veth peer name <veth1>



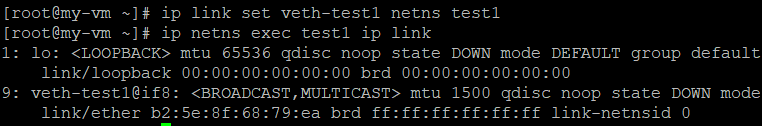
##### network namespace

查看：ip netns list

添加ns：ip netns add/delete <namespace-name>

查看ns下地址：ip netns exec <namespace-name> ip addr

添加到namespace：ip link set <dev-name> netns <namspace>



添加地址：ip netns exec test1 ip addr add 192.168.1.1/24 dev veth-test1

up：ip netns exec test1 ip link set dev veth-test1 up

##### Bridge

虚拟网络设备，所以具有网络设备的特征，可以配置IP、MAC地址等。其次bridge是一个虚拟交换机，软件实现物理交换机类似的功能。

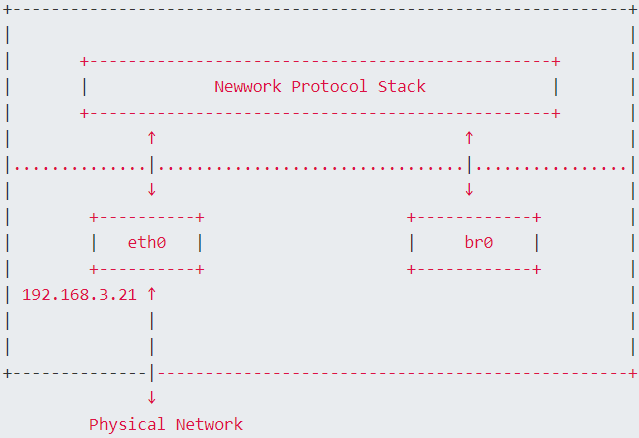
普通网络设备只有两端，一端进数据另一端出，如物理网卡从外面网络中收到的数据会转发给内核协议栈，而从协议栈过来的数据会转发到外面的物理网络中。

而bridge不同，bridge有多个端口，数据可从任何端口进来，进来后从哪个口出去和物理交换机的原理差不多，要看mac地址。

创建bridge

ip link add name br0 type bridge

ip link set br0 up

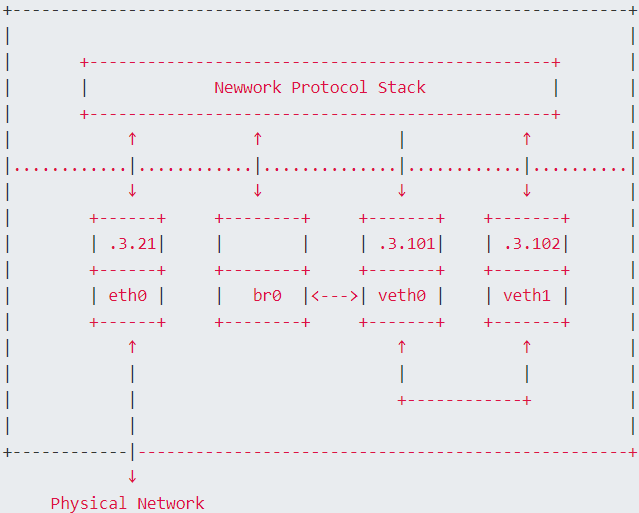


创建的bridge是一个独立的网络设备，只有一个端口连着协议栈，其它的端口啥都没连。

veth连网桥

ip link set dev veth0 master br0

连上后网络



连上后查看：bridge link

删除

若删除应先移除关联的所有端口，同时关闭端口的混杂模式并关闭端口以恢复至原始状态。

ip link set eth0 promisc off

ip link set veth0 down

ip link set dev eth0 nomaster

当网桥的配置清空后就可以将其删除：

ip link delete br0 type bridge

p1：rtnetlink answers file exists

ip addr flush dev veth0

网桥原理



#### ping

[Reference](https://www.runoob.com/linux/linux-comm-ping.html)

-c 要求回应的次数

-I 指定网卡送出数据包

通过网卡enp3s0发送数据

ping 192.168.0.1 -c 1 -I enp3s0



ARP 将IP转换成MAC地址的协议

#### arp

-n addrss使用数字方式显示

-s设置指定主机的mac地址映射

-i显示指定设备的arp缓冲区

添加静态映射

arp -i eth0 -s 192.168.1.6 ff:ee:ee:ee:ee:ee //将目标ip地址映射固定mac

#### tcpdump

-n 表示不要解析域名，直接显示 ip。

-i 监听所有的网卡

tcpdump -n -i veth0



## 3.4 权限

[Reference](https://www.cnblogs.com/cisum/p/8005641.html)

权限管理实际上都是对有关的系统文件进行修改，与用户和用户组相关的信息都存放在一些系统文件中，这些文件包括/etc/passwd, /etc/shadow, /etc/group等。

### 3.3.3 User

#### 查看用户

cat /etc/passwd



用户名:口令:UID:GID:注释性描述:主目录:Shell

* 口令：密码标识。真正的密码存在/etc/shadow，只有root有权限读写。
* UID：用户标识号。0管理员。1~499系统用户，用于系统用户，不能登陆。
* GID：组标识号。
* 主目录：登录后所处的目录。
* Shell：登陆后的Shell。

#### adduser

useradd userName



默认创建用户组nan，用户目录为/home/nan1。

* -d 指定用户主目录；
* -g 指定用户组，删除初始组；-G 指定附加组，不删除原来组)。
* -c comment
* -s 指定用户的登录Shell。

nologin shell

使用nologin账号来运行程序，提升安全性

* 创建：useradd -s /sbin/nologin nan
* 运行程序：su -s /bin/bash -c "ls" nginx或者sudo -u nginx command

#### userdel



默认删除用户组love，

* -r 删除用户主目录，默认不删

usermod

#### usermod

修改用户

更改登陆目录usermod -d /home/hnlinux root

### 3.3.2 group

每个用户都属于某个用户组，一个组中可以有多个用户，一个用户也可以属于不同的组。

**用户组在adduser同时创建。userdel时同时删除。**

#### 查看用户组

cat /etc/group



格式：groupName:口令:GID:UserList

#### groupadd

groupadd group1

**groups** 查看当前登录用户的组内成员

**groupmod:** groupmod -n newName groupName (-n改组名 –g 改gid)

#### groupdel

groupdel groupName先删除group的用户，才能删除群组

删除组成员

vi /etc/group

### 3.3.4 passwd

-l禁用账号。

-u 口令解锁。

-d 使账号无口令。

-f 强迫下次登录时修改口令。

超级用户修改/su不需要输入用户密码



普通用户直接输passwd

2./etc/shadow

moon:$6$007qnwIe$kS:17795:0:99999:7:::

用户名:口令:last\_update:最小时间间隔:最大时间间隔:警告时间:不活动时间:失效时间:标志

passwd文件所有用户可读，如果密码放passwd文件，即便加密也很容易破解。真正的加密后的用户口令字存放到shadow文件中，只有超级用户可读。

其他

**chown**将指定文件的拥有者改为指定的用户或组

chown -R mysql: mysql ./

su:Su switch use /sudo输入当前用户的密码。

sudo1.需要root赋予sudo权限

## 3.5 安全

### 5.5.1 网络安全

#### 开放端口

firewall-cmd

添加端口：firewall-cmd --zone=public --add-port=80/tcp --permanent

Reload：firewall-cmd --reload

停止：systemctl stop firewalld

查看端口状态：firewall-cmd --query-port=9200/tcp

iptables添加端口

iptables -I INPUT -p tcp --dport 7001 -j ACCEPT

iptables VS Firewalld

Cent7以后，iptables服务的启动脚本已被忽略。firewalld和iptables一样需要通过内核的netfilter来实现防火墙功能。

firewalld使用XML进行配置。除非非常特殊的配置，而应该使用firewalld-cmd。

配置文件：

/usr/lib/firewalld    #保存默认配置，避免修改他们；

/etc/firewalld    #保存系统配置文件，这些文佳你将覆盖默认配置

Pass: 6d%SW5e)[.xvzt@Z](mailto:.xvzt@Z)?

#### SSL证书

[Reference](https://blog.csdn.net/liuchunming033/article/details/48470575)

* .key：私钥
* .csr：Certificate Signing Request，即证书签名请求，含公钥信息，生成证书时要把这个提交给权威的证书颁发机构。
* .crt： 即 certificate的缩写，即证书文件。
* .pem：用于导出，导入证书时候的证书的格式。

证书文件

一般以.crt结尾，根据文件内容编码格式，分为二种格式：

* pem：Privacy Enhanced Mail。
* der-Distinguished Encoding Rules,打开看是二进制格式,不可读.

X.509

一种证书格式，认证者总是CA或由CA指定的人。一份X.509证书是一些标准字段的集合，这些字段包含有关用户或设备及其相应公钥的信息。

CA根证书生成步骤

自签名得到根证书ca.crt，自签指是CA给自已颁发的证书。

1.Generate CA private key

openssl genrsa 2048 -out **ca.key**

2.Generate CSR

openssl req -new -key ca.key -out **ca.csr**

3.Generate Self Signed certificate（CA 根证书）

openssl x509 -req -days 365 -in ca.csr -signkey ca.key -out **ca.crt**

颁发证书

服务器端用户证书：

1.private key

$openssl genrsa -des3 1024 -out server.key

2.generate csr

$openssl req -new -key server.key -out server.csr

3.generate certificate

$openssl ca **-cert ca.crt** -in server.csr -keyfile ca.key -out **server.crt**

server.crt 就是我们需要的证书

客户端用户证书：

$openssl genrsa -des3 1024 -out client.key

$openssl req -new -key client.key -out client.csr

$openssl ca **-cert ca.crt** -in client.csr -keyfile ca.key -out client.crt

生成pem格式证书

合并证书（crt）和私钥（key）来生成, 可以用以下方式

$cat client.crt client.key> client.pem

$cat server.crt server.key > server.pem

结果：

服务端证书：ca.crt, server.key, server.crt, server.pem

客户端证书：ca.crt, client.key, client.crt, client.pem

## 3.6 Shell

解释器，看成一个线程

登录shell：是需要用户名、密码登录后才能进入的shell

非登录shell： 不需要输入用户名和密码即可打开的Shell，例如：直接命令“bash”就是打开一个新的非登录shell，在Gnome或KDE中打开一个“终端”（terminal）窗口程序也是一个非登录shell。

/bin/nologin l，

交互式：在终端上执行，shell等待你的输入

非交互式：以shell script方式执行。

交互Shell:

Shell是用户与Linux系统之间的接口。常用的有

Bourne Shell（/usr/bin/sh或/bin/sh）

Bourne Again Shell（/bin/bash）

C Shell（/usr/bin/csh）

K Shell（/usr/bin/ksh）

Shell for Root（/sbin/sh）

系统管理员可以根据系统情况和用户习惯为用户指定某个Shell。

用户的登录Shell也可以指定为某个特定的程序（此程序不是一个命令解释器）。

利用这一特点，我们可以限制用户只能运行指定的应用程序，在该应用程序运行结束后，用户就自动退出了系统。

登陆shell：户登陆时，输入用户名和密码后启动的shell；

### Shell Script

扩展名为 sh（sh代表shell）

1. #!/bin/bash
2. echo "Hello World !"

#! 是一个约定的标记，告诉系统这个脚本需要什么解释器来执行，即使用哪一种 Shell。

#### 运行

1./test.sh 脚本加#!/bin/bash

2.sh test.sh

#### 变量

$# 是传给脚本的参数个数

$0 是脚本本身的名字

$1 是传递给该shell脚本的第一个参数

$2 是传递给该shell脚本的第二个参数

$@ 是传给脚本的所有参数的列表

$\* 是以一个单字符串显示所有向脚本传递的参数，与位置变量不同，参数可超过9个

$$ 是脚本运行的当前进程ID号

$? 是显示最后命令的退出状态，0 = success, 其他 = error

${var}等价于$var（简写）

if/else

[Reference](http://www.cnblogs.com/kangyoung/p/3556173.html)

#### 基本运算符

[Reference](https://www.runoob.com/linux/linux-shell-basic-operators.html)

算术运算符

* -n 检测字符串长度是否为0，不为0返回 true。
* $ 检测字符串是否为空，不为空返回 true。
* -z 检测字符串长度是否为0，为0返回 true。
* = 检测两个字符串是否相等，相等返回 true。

#### 比较

if ([ condition ] || [condition])

then

cmd

else

cmd

fi

if method()

method return 0=true ,1=false

#### 文件表达式

-e filename 如果 filename存在，则为真  
-d filename 如果 filename为目录，则为真   
-f filename 如果 filename为常规文件，则为真  
-L filename 如果 filename为符号链接，则为真  
-r filename 如果 filename可读，则为真   
-w filename 如果 filename可写，则为真   
-x filename 如果 filename可执行，则为真  
-s filename 如果文件长度不为0，则为真  
-h filename 如果文件是软链接，则为真

整数变量

eq:=

ne:!=

gt:>

ge:>=

lt: <

le <=

函数

LaunchJDeveloper(){

DisplayBanner

}

#### expr

手工命令行计数器



模式匹配

# 第四章 Install&Config

## 4.1 Docker

安装 [Reference](https://www.runoob.com/docker/centos-docker-install.html)

启动：systemctl start docker

#### 安装

配置文件：/usr/lib/systemd/system/docker.service

设置仓库

yum-config-manager \

--add-repo \

https://download.docker.com/linux/centos/docker-ce.repo

Install

yum install docker-ce-18.09.9 -y

systemctl enable docker

systemctl start docker

查看版本 yum list docker-ce --showduplicates | sort -r

#### 重新安装

1.remove docker

yum remove -y docker docker-latest docker-ce docker-common \

docker-selinux container-selinux \

docker-engine \

docker-ce-cli \

docker-client docker-client-latest \

docker-logrotate docker-latest-logrotate

containerd.io

2.删除本地文件

docker的本地文件，包括images/containers/存储卷(volumes)等，都需要手工删除。

rm -rf /var/lib/docker (默认路径)

配置文件

rm –rf /etc/docker

rm –rf /root/.docker

## 4.2 k8s

安装

[Reference](https://kubernetes.io/docs/tasks/tools/install-kubectl/)

配置对多集群的访问:[Reference](https://kubernetes.io/zh/docs/tasks/access-application-cluster/configure-access-multiple-clusters/)

### 4.2.1 kubeadm

[Reference](https://blog.csdn.net/weixin_41806245/article/details/89381752)

[kubeadm搭建kubernetes集群之二](https://cloud.tencent.com/developer/article/1012924)

#### 准备环境

关闭swap分区 、防火墙 、selinux等

1.禁用swap:

vi /etc/fstab 注释 /dev/mapper/centos-swap

重新加载：sysctl --system

2.修改内核模块:

vi /etc/sysctl.d/k8s.conf

net.bridge.bridge-nf-call-ip6tables = 1

net.bridge.bridge-nf-call-iptables = 1

vm.swappiness=0

3.防火墙：

systemctl stop firewalld

systemctl disable firewalld

4.永久关闭selinux

vi /etc/sysconfig/selinux

SELINUX=enforcing替换为SELINUX=disabled

5.修改docker

1.vi /etc/docker/daemon.json，加入下面的内容：

{

"exec-opts": ["native.cgroupdriver=systemd"]

}

systemctl enable docker

systemctl start docker

下载镜像

#!/bin/bash

images=(kube-proxy:v1.14.0 kube-scheduler:v1.14.0 kube-controller-manager:v1.14.0 kube-apiserver:v1.14.0 etcd:3.3.10 coredns:1.3.1 pause:3.1 )

for imageName in ${images[@]} ; do

docker pull xiliangma/$imageName

docker tag xiliangma/$imageName k8s.gcr.io/$imageName

docker rmi xiliangma/$imageName

done

2Exception

[ERROR DirAvailable--etc-kubernetes-manifests]: /etc/kubernetes/manifests is not empty

[ERROR FileAvailable--etc-kubernetes-kubelet.conf]: /etc/kubernetes/kubelet.conf already exists

安装kubeadm

准备k8s.repo

vi /etc/yum.repos.d/k8s.repo

[kubernetes]

name=Kubernetes

baseurl=https://mirrors.aliyun.com/kubernetes/yum/repos/kubernetes-el7-x86\_64/

enabled=1

gpgcheck=1

repo\_gpgcheck=1

gpgkey=https://mirrors.aliyun.com/kubernetes/yum/doc/yum-key.gpg https://mirrors.aliyun.com/kubernetes/yum/doc/rpm-package-key.gpg

yum -y install kubelet-1.14.0 kubeadm-1.14.0 kubectl-1.14.0

#### K8s-Master

1) kubeadm init

--apiserver-advertise-address 192.168.0.110 指定监听地址

--pod-network-cidr ：Specify range of IP addresses for the pod network.

--kubernetes-version=v1.14.0

示例：

kubeadm init --apiserver-advertise-address 192.168.6.130 --pod-network-cidr=10.244.0.0/16

2) kubelet config：

mkdir -p $HOME/.kube

cp -i /etc/kubernetes/admin.conf $HOME/.kube/admin.conf

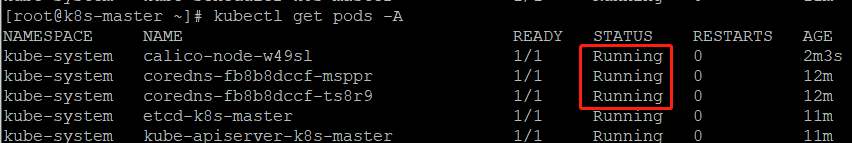
chown $(id -u):$(id -g) $HOME/.kube/admin.conf

export KUBECONFIG=$HOME/.kube/admin.conf

3) 添加网络组件

kubectl apply -f \

https://docs.projectcalico.org/v3.5/getting-started/kubernetes/installation/hosted/kubernetes-datastore/calico-networking/1.7/calico.yaml



#### k8s-Node

[安装镜像：kube-proxy,pause]

kubeadm join 192.168.6.133:6443 --token balro4.c1svv2f8rcz3bveb \

--discovery-token-ca-cert-hash sha256:26d141131d93adadcb535…

docker pull calico/cni:v3.5.8

docker pull calico/node:v3.5.8

查看Token：kubeadm token list

create(永久)：kubeadm token create --ttl 0

create(临时)： kubeadm token create

查看ca证书

[Reference](https://blog.csdn.net/qq_19734597/article/details/97672148)

方法一

openssl x509 -pubkey -in /etc/kubernetes/pki/ca.crt | openssl rsa -pubin -outform der 2>/dev/null | openssl dgst -sha256 -hex | sed 's/^.\* //'

方法2

kubectl -n kube-system describe $(kubectl -n kube-system get secret -n kube-system -o name| grep namespace) | grep token

#### 重新安装

kubeadm reset

注意：Changes made to this host by 'kubeadm init' or 'kubeadm join' will be reverted.

rm -rf $HOME/.kube ?

#### Kube Dashboad

kubectl delete -f dashboard.yaml

[mini Dashboad](http://192.168.0.8:8001/api/v1/namespaces/kubernetes-dashboard/services/http:kubernetes-dashboard:/proxy/#/overview?namespace=default)

vm driver

~/.minikube/machines/minikube/config.json

### 其他安装方式

#### 手动安装

[Reference](https://o-my-chenjian.com/2017/04/26/Deploy-Master-Of-K8s/)

##### flannel

master和node上都要安装Flannel网络

配置

vi /usr/lib/systemd/system/flanneld.service

vi /etc/sysconfig/flanneld

etcdctl --endpoints=${ETCD\_ENDPOINTS} put /kubernetes/network/config '{"Network":"'${CLUSTER\_CIDR}'", "SubnetLen": 24, "Backend": {"Type": "vxlan"}}'

etcdctl --endpoints=${ETCD\_ENDPOINTS} get /kubernetes/network/config

--cacert=/etc/kubernetes/ssl/ca.pem --key=/etc/etcd/ssl/etcd-key.pem --cert=/etc/etcd/ssl/etcd.pem

通过http向etcd增加key

curl -X PUT http://192.168.0.110:2379/v2/keys/kubernetes/network/config -d value='{"Network":"172.30.0.0/16", "SubnetLen": 24, "Backend": {"Type": "vxlan"}}'

##### master节点

安装kubernetes： yum -y install kubernetes

kube-apiserver

vi /usr/lib/systemd/system/kube-apiserver.service

vi /etc/kubernetes/apiserver

systemctl daemon-reload

systemctl restart kube-apiserver.service

kube-controller-manager

vi /usr/lib/systemd/system/kube-controller-manager.service

systemctl daemon-reload

systemctl restart kube-controller-manager.service

查看节点

curl http://192.168.0.110:8080/api/v1/nodes

kube-scheduler

vi /usr/lib/systemd/system/kube-scheduler.service

systemctl enable kube-scheduler

##### Node节点

1.安装kubernetes

yum -y install kubernetes&vi /etc/kubernetes/config

* KUBE\_LOGTOSTDERR="--logtostderr=true"
* KUBE\_LOG\_LEVEL="--v=0"
* KUBE\_ALLOW\_PRIV="--allow-privileged=false"
* KUBE\_MASTER="--master=http://k8s-master:8080"

kubelet

vi /usr/lib/systemd/system/kubelet.service

vi /etc/kubernetes/kubelet

* KUBELET\_ADDRESS="--address=0.0.0.0"
* KUBELET\_HOSTNAME="--hostname-override=k8s-node-1"
* KUBELET\_API\_SERVER="--api-servers=http://k8s-master:8080"
* KUBELET\_POD\_INFRA\_CONTAINER="--pod-infra-container-image=registry.access.redhat.com/rhel7/pod-infrastructure:latest" #确认此镜像存在，否则pod无法启动
* KUBELET\_ARGS=""

/etc/kubernetes/kubelet.conf

–client-ca-file

3.启动服务

systemctl daemon-reload

systemctl restart kubelet

kube-proxy

/usr/lib/systemd/system/kube-proxy.service

Vi

##### etcd

etcdctl version

ETCDCTL\_API=3

配置

vi /usr/lib/systemd/system/etcd.service

vi /etc/etcd/etcd.conf

#[Member]

ETCD\_DATA\_DIR：etcd数据保存目录

ETCD\_LISTEN\_CLIENT\_URLS：供外部**客户端**使用的url

ETCD\_ADVERTISE\_CLIENT\_URLS：暴露给外部**客户端**使用的url

ETCD\_NAME：etcd实例名称

#[Clustering]

ETCD\_LISTEN\_PEER\_URLS：集群内部通信使用的URL

ETCD\_INITIAL\_ADVERTISE\_PEER\_URLS：广播给集群内其他成员访问的URL

ETCD\_INITIAL\_CLUSTER：初始集群成员列表

ETCD\_INITIAL\_CLUSTER\_TOKEN：集群的名称

ETCD\_INITIAL\_CLUSTER\_STATE：初始集群状态，new为新建集群

rm -rf /var/lib/etcd/default.etcd

systemctl daemon-reload

systemctl restart etcd

etcdctl cluster-health

配置TSL

etcdctl member list --endpoints=https://192.168.0.111:2379 --cacert=/etc/kubernetes/ssl/ca.pem --key=/etc/etcd/ssl/etcd-key.pem --cert=/etc/etcd/ssl/etcd.pem

CMD

* -w, --write-out="simple" 输出格式(fields, json, protobuf, simple, table)
* --endpoints=[127.0.0.1:2379] ：gRPC endpoints

Restful api

curl http://192.168.0.110:2379/v2/keys/

查看集群

etcdctl endpoint status

etcdctl [--write-out=table] endpoint health

etcdctl member list

etcdctl endpoint health --endpoints=https://127.0.0.1:2379 --cacert=/etc/kubernetes/ssl/ca.pem --key=/etc/kubernetes/ssl/etcd-key.pem --cert=/etc/kubernetes/ssl/etcd.pem

示例：

etcdctl --endpoints=http://192.168.0.110:2379 member list -w table

集群成员

跟集群成员相关的命令如下：

member add Adds a member into the cluster

member remove Removes a member from the cluster

member update Updates a member in the cluster

member list Lists all members in the cluster

示例

etcdctl member list -w table



CRUD

* 增：etcdctl --endpoints=$ENDPOINTS put foo "Hello World!"
* 查：etcdctl [--endpoints=$ENDPOINTS] [--write-out="json"] get foo
* 删：etcdctl --endpoints=$ENDPOINTS del key

Failed to connect to apiserver: the server has asked for the client to provide credentials

Failed to connect to apiserver: Forbidden: "/healthz?timeout=1s

#### kubectl

kubectl version：Print the client and server version information

Client Version: version.Info{Major:"1", Minor:"6", GitVersion:"v1.6.0", GoVersion:"go1.7.5", Platform:"linux/amd64"}

Server Version: version.Info{Major:"1", Minor:"5", GitVersion:"v1.5.2", GitTreeState:"clean", GoVersion:"go1.7.4", Compiler:"gc", }

##### kubeconfig

kubeconfig文件记录k8s集群的各种信息，对集群构建非常重要。

* kubectl从~/.kube/config，即kubectl的kubeconfig文件中获取访问kube-apiserver的地址，证书和用户名等信息
* kubelet/kube-proxy等在Node上的程序进程同样通过bootstrap.kubeconfig和kube-proxy.kubeconfig上提供的认证与授权信息与Master进行通讯

查询集群地址和凭证

kubectl config view

1. apiVersion: v1
2. clusters:
3. - cluster:
4. certificate-authority-data: LS0tLS1CRUdJTiBDRVJUSU....
5. server: https://192.168.0.110:6443
6. name: kubernetes
7. contexts:
8. - context:
9. cluster: kubernetes
10. user: kubelet-bootstrap
11. name: default
12. current-context: default
13. kind: Config
14. preferences: {}
15. users:
16. - name: kubelet-bootstrap
17. user:
18. token: 5123398c0116eb8617398e4c38c3be

certificate-authority-data 是 ca.pem 的base64加密的内容

client-key-data:

kubectl config get-contexts

##### Volume存储数据卷

exception

Orphaned pod "cdb27b68-9e8d-4708-8179-b5579dab861f" found, but volume paths are still present on disk.

rm -rf /var/lib/kubelet/pods/\*

#### 集群安全设置

[Reference](https://o-my-chenjian.com/2017/04/25/Security-Settings-Of-K8s/)

#### Minikube

[Reference](https://yq.aliyun.com/articles/221687)

安装

**1.下载minikube**

curl -Lo minikube http://kubernetes.oss-cn-hangzhou.aliyuncs.com/minikube/releases/v1.4.0/minikube-linux-amd64 && chmod +x minikube && sudo mv minikube /usr/local/bin/

启动

minikube start --vm-driver=none

查看状态：minikube status

minikube dashboard

kubectl proxy --port=8001 --address='192.168.0.5' --accept-hosts='^.\*' &

更新

如需更新minikube，需要更新 minikube 安装包

minikube delete 删除现有虚机，删除 ~/.minikube 目录缓存的文件

minikube查看service

minikube service kube-nginx999 [–url] [-n NAMESPACE]



minikube查看日志：minikube logs

### grafana

[监控k8s](https://blog.csdn.net/lindao99/article/details/79977794)

## 4.3 服务器

### Nginx

查看配置文件位置 nginx –t

配置

1. upstream cluster1 {
2. server localhost:8080;
3. server localhost:8081;
4. }
6. server {
7. listen       81;
8. server\_name  localhost;
9. client\_max\_body\_size 1024M;
11. location /home {
12. proxy\_pass http://cluster1;
13. proxy\_set\_header Host $host:$server\_port;
14. }
15. }

上面upstream默认负载均衡策略为RR

http服务器

1. # 所有静态请求都由nginx处理，存放目录为html
2. location ~ \.(gif|jpg|jpeg|png|bmp|swf|css|js)${
3. root    e:\wwwroot;
4. }
6. # 所有动态请求都转发给tomcat处理
7. location ~\.(jsp|do)$ {
8. proxy\_pass  http://test;
9. }
10. error\_page  500 502 503 504 /50x.html;
11. location =/50x.html {
12. root   e:\wwwroot;
13. }

Configuration summary

+ using system PCRE library

+ OpenSSL library is not used

+ using system zlib library

nginx path prefix: "/usr/local/nginx"

nginx binary file: "/usr/local/nginx/sbin/nginx"

nginx modules path: "/usr/local/nginx/modules"

nginx configuration prefix: "/usr/local/nginx/conf"

nginx configuration file: "/usr/local/nginx/conf/nginx.conf"

nginx pid file: "/usr/local/nginx/logs/nginx.pid"

nginx error log file: "/usr/local/nginx/logs/error.log"

nginx http access log file: "/usr/local/nginx/logs/access.log"

nginx http client request body temporary files: "client\_body\_temp"

nginx http proxy temporary files: "proxy\_temp"

nginx http fastcgi temporary files: "fastcgi\_temp"

nginx http uwsgi temporary files: "uwsgi\_temp"

nginx http scgi temporary files: "scgi\_temp"

1)启动操作

nginx -c /usr/local/nginx/conf/nginx.conf

2)停止操作

从容停止Nginx

快速停止Nginx：

kill -TERM 主进程号

强制停止Nginx：

kill -9 主进程号

3)

nginx -s reload

reload

./nginx -s reload

reload会重新加载配置文件，服务不中断，出错会rollback用上一次正确配置文件。

Request\_body

[Reference](http://www.madblog.cn/posts/e4dd5bbae50fa621.html)

### Mysql

Mgr

mysql [-h slc11fsp.us.oracle.com] -uroot -p

启动服务

service mysqld start

版本：mysql –V；或者select version();

Tables：show databases/tables;

查看表结构

desc table\_name;

Show DDl: show create table ld\_cart;

indexs

Show: show index from fund\_history\G;

Create: create index idx\_fund\_name on fund(name);

GRANT ALL PRIVILEGES ON \*.\* TO 'root'@'%' IDENTIFIED BY '123456' WITH GRANT OPTION;

## 4.4 大数据处理

### Flume

启动

bin/flume-ng agent -c ./conf -f ./conf/2.conf -n agent1 -Dflume.root.logger=INFO,console

* -c conf 指定flume自身的配置文件所在目录
* -f conf/netcat-logger.con 指定我们所描述的采集方案
* -n a1 指定agent的名字

组合

[Reference](http://www.voidcn.com/article/p-wgqahebc-ev.html)

### ElasticSearch

安装 [Reference](https://www.cnblogs.com/yufeng218/p/8331438.html)

启动es

su elastic

/usr/local/elasticsearch-5.6.0/bin/elasticsearch –d

chown elastic /usr/local/elasticsearch-6.8.0/ -R

外网访问

vi elasticsearch.yml

network.bind\_host: 0.0.0.0 (network.host: 0.0.0.0 5.6?)

安装问题汇总

<https://blog.51cto.com/10950710/2124131>

vm

sysctl -w vm.max\_map\_count=262144

修改jvm ops

vi config/jvm.options

3.修改es文件拥有着 chown elastic:elastic -R ./\*

#### kibana

1.kibana.yml server.host: "0.0.0.0"

Test url

http://localhost:5601/

### Kafka

[Reference](https://juejin.im/entry/5cbfe36b6fb9a032036187aa)

[Reference](https://juejin.im/post/5c1f9b3bf265da61590ba9b1)

1.Docker安装集群

新建网络

docker network create --driver bridge --subnet 172.23.0.0/25 --gateway 172.23.0.1 \ zookeeper\_network

2.安装

ver: kafka\_2.11-2.1.0

启动/停止

* 启动 ./bin/kafka-server-start.sh ./config/server.properties &
* 停止 ./bin/kafka-server-stop.sh ./config/server.properties &

配置

vi ./config/server.properties

1.Zookeeper

* zookeeper.connect=localhost:2181
* zookeeper.connection.timeout.ms=6000

2. socket settings

* listeners：the socket server listens on
* advertised.listeners：broker的地址。配置外网访问.(0.9.x后新增，废弃了 advertised.host.name 和 advertised.host.port)

3.broker

broker.id：broker id

## 4.5安装centos7

[Reference](https://www.linuxidc.com/Linux/2014-10/108014.htm)

1.修改bios启动顺序

2.进入安装界面，按e，修改参数

linuxefi /images/pxeboot/vmlinuz inst.stage2=hd:/dev/sdb4 nomodeset quiet

initrdefi /images/pxeboot/initrd.img

然后使用Ctrl+X启动即可进入正常安装流程

### 静态IP

1.静态ip

[Reference](https://blog.csdn.net/johnnycode/article/details/40624403)

vi /etc/sysconfig/network-scripts/ifcfg-enp3s0

1. BOOTPROTO=static #dhcp改为static
2. ONBOOT="yes" #开机启用本配置
3. IPADDR=192.168.7.106 #静态IP
4. GATEWAY=192.168.7.1 #默认网关
5. NETMASK=255.255.255.0 #子网掩码
6. DNS1=192.168.7.1 #DNS 配置

2.重启网络

service network restart

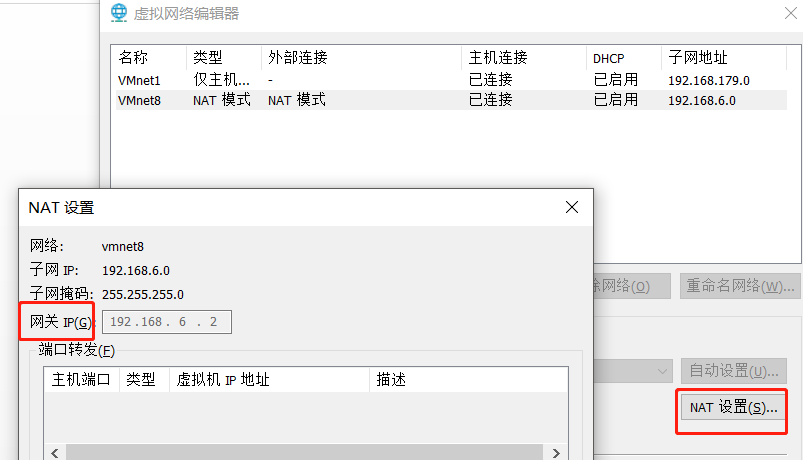
### 修改hostname

/etc/hostname

### 盒盖息屏

[Reference](https://www.jianshu.com/p/cfa90c8a900f)

### vmvare



### Shadowsocks

系统版本：Centos7

安装

wget --no-check-certificate https://raw.githubusercontent.com/teddysun/shadowsocks\_install/master/shadowsocks.sh

chmod +x shadowsocks.sh

./shadowsocks.sh 2>&1 | tee shadowsocks.log

1.输入密码/端口

2.加密 aes-256-cfb

Congratulations, Shadowsocks-python server install completed!

Your Server IP : 45.32.35.9

Your Server Port : 18548

Your Password : ccx3801=

Your Encryption Method: aes-256-cfb

修改配置

vi /etc/shadowsocks.json

/bin/python /usr/bin/ssserver -c /etc/shadowsocks.json -d restart

异常

1.configure: error: no acceptable C compiler found in $PATH

 yum install gcc

## 4.6 安装工具

### yum

yum list 所有已安装和可安裝的软件清单

yum search java 查找软件包

yum list installed

yum install xxx

设置代理

/etc/yum.conf

proxy=http://cn-proxy.jp.oracle.com:80/

### make

configure

生成Makefile:GUN规范

./configure –prefix=/usr 将该软件安装在 /usr 下面，执行文件就会安装在 /usr/bin 资源文件就会安装在 /usr/share

make 编译

默认安装

使用make编译安装软件时，若./configure不指定prefix，则：

可执行文件默认放在/usr/local/bin（/usr/local/libexec中也可能有一些），

库文件默认放在/usr/local/lib，

配置文件默认放在/usr/local/etc，

其它的资源文件放在/usr/local/share。

卸载

1.在原来的make目录下用一次make uninstall（前提是make文件指定过uninstall）

2.去上述目录里面把相关的文件一个个手工删掉。

-

配置文件

通常情况下，用户软件的配置文件放在自已的 home 目录下，比如 /home/riku/ 下，一般以 . 作为目录开始字符，比如 .gnome 。

而系统软件的配置文件一般放在 /etc/ 目录下。

### Systemd

#### Service配置

Unit

Systemd 可以管理所有系统资源。不同的资源统称为 Unit（单位）。共分12种。

* Service unit：系统服务
* Target unit：多个 Unit 构成的一个组
* Device Unit：硬件设备
* Mount Unit：文件系统的挂载点
* Automount Unit：自动挂载点
* Path Unit：文件或路径
* Scope Unit：不是由 Systemd 启动的外部进程
* Slice Unit：进程组
* Snapshot Unit：Systemd 快照，可以切回某个快照
* Socket Unit：进程间通信的 socket
* Swap Unit：swap 文件
* Timer Unit：定时器

Unit配置文件

路径：/usr/lib/systemd/system/xxx.service

[Unit]定义 Unit 的元数据，以及配置与其他 Unit 的关系

* Description 简短描述
* After
* Wants与当前 Unit 配合的其他 Unit，如果它们没有运行，当前 Unit 不会启动。

[Service]区块用来 Service 的配置，只有 Service 类型的 Unit 才有这个区块。

* Type 定义启动时的进程行为。simple/notify/idle
* WorkingDirectory
* EnvironmentFile环境参数文件
* User
* ExecStart启动当前服务的命令
* Restart定义何种情况 Systemd 会自动重启当前服务，可能的值包括always（总是重启）、on-success、on-failure、on-abnormal、on-abort、on-watchdog
* LimitNOFILE

[Install]用来定义开机启动行为

* WantedBy字段：表示该服务所在的 Target。

比如Target 是multi-user.target。在这个组里的所有服务，都将开机启动。

修改配置文件后重启

修改配置文件以后，需要重新加载配置文件，然后重新启动相关服务。

# 重新加载配置文件

$ systemctl daemon-reload

# 重启相关服务

$ sudo systemctl restart etcd

开机启动

systemctl enable etcd.service

开机时，Systemd执行/etc/systemd/system目录里面的配置文件， enable用于建立符号链接关系。

# enable等同于

$ ln -s '/usr/lib/systemd/system/etcd.service' '/etc/systemd/system/multi-user.target.wants/etcd.service'

如果配置文件里面设置了开机启动，systemctl enable命令相当于激活开机启动。

#### systemctl

systemctl是 Systemd 的主命令，用于管理系统。

#重载所有修改过的配置文件

systemctl daemon-reload

#### journalctl

-k 内核日志

-u 按服务过滤

-f 同tail -f

journalctl -f -u kubelet

# 附件1

## Linux命令

Base

uname –a

locate cmd

whereis

### tail

-f 等同于--follow=descriptor，根据文件描述符进行追踪，当文件改名或被删除，追踪停止

-F 等同于--follow=name --retry，根据文件名进行追踪，并保持重试，即该文件被删除或改名后，如果再次创建相同的文件名，会继续追踪

### cat

cat > /root/test1.txt <<EOF

>

EOF

### hostnamectl

# 显示当前主机的信息

$ hostnamectl

# 设置主机名。

$ sudo hostnamectl set-hostname k8s-master

### ssh

[Reference](https://blog.csdn.net/u011054333/article/details/52443061)

-v debug

-p port

ssh -v -p 8080 nadong@slc11fsp.us.oracle.com

TODO1:如何防止中间人攻击？

# 附件2开发工具

### Git

Clone git clone https://github.com/lovemooner/moon.git

Branch

本地分支：git branch。【-a】 切换分支：git checkout branchName

Commit

git commit -a

git push origin master

git push origin HEAD:refs/for/reserved (or other branch you wish to push, start with refs/for/)（push内容在其他库并非立即可见）

撤销本地修改 git checkout filename

删除文件 git rm filename (-r 文件夹)

git log

--stat 加stat，仅显示修改的文件)

查看单个文件

git log <filename> 查看所有commit历史(包括本地未push到库里的)

--pretty=oneline pretty

-n 显示最后的几次提交日志

-p 详细包括diff内容

Config

查看当前配置 git config –list

配置：git config [--global] user.email "[email address]"

设置proxy：git config --global https.proxy http://127.0.0.1:1080

system级别 /global（用户级别）/local（当前仓库）三个 设置先从system-》global-》local

删除 git config --global --unset https.proxy

remote

git remote set-url origin [url]



全红：删除 全绿：增加

#### 分支管理

##### 分支查看

git branch -a 查看所有分支，包括local/remote

git branch 查看本地分支

##### 分支创建

* git branch mybranch 创建分支
* git checkout mybranch切换分支
* git checkout -b mybranch：相当于git branch+git checkout

分支合并

Git merge dev：合并dev到当前分支

##### 提交

撤销本地提交

git reset HEAD~

#### log

**-p 详细包括diff内容**

git log -p <filename>查看某个文件的修改历史

**-n**

显示最后的几次提交日志

git log -2

git log –p -2 查看最近2次更新内容

**show**

git show <commit-hash-id>查看某次commit的修改内容  
git show [--stat]

查看本地修改了还未提交

git diff filename

权限

查看key

cd ~/.ssh

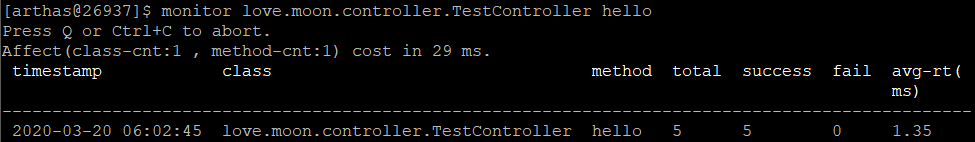
Grap

Git图谱

### Arthas

[Reference](https://alibaba.github.io/arthas/dashboard.html)

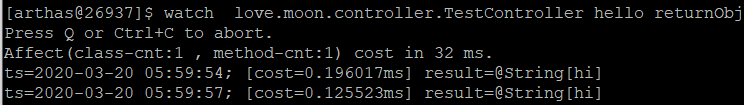
monitor



Watch

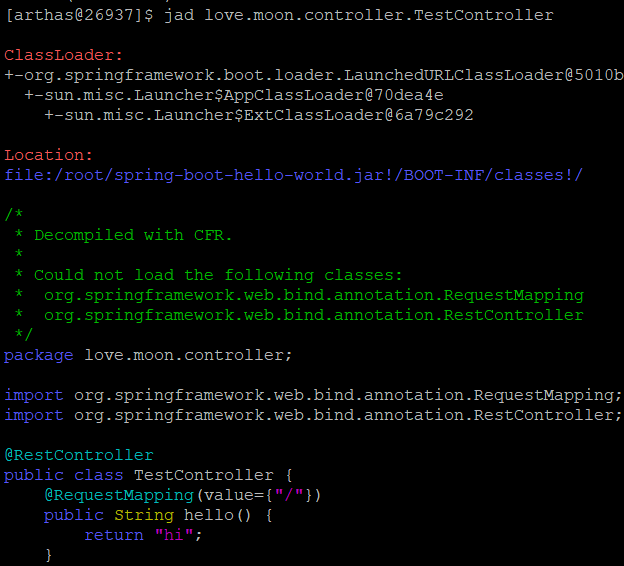
方法执行数据观测

watch -f love.moon.controller.TestController hello returnObj

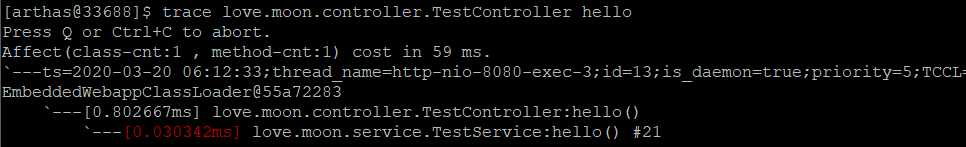


Jad

反编译 jad love.moon.controller.TestController



trace



### Jar

java-jar

-c 创建一个jar包

-t 显示jar中的内容列表

-x 解压jar包

-u 添加文件到jar包中

-f 指定jar包的文件名

-v生成详细的报造，并输出至标准设备

1）解压

Jar -xvf test.ear

2）创建

Jar -cvf test.jar ./\*

3）更新

jar -uvf RESTWebService.ear APP-INF/lib/AdfHcmUsersRestModel.jar

4）查看

jar tvf test.jar

5）搜索：

jar tvf ZhcLib.jar | grep 'pattern'

搜索整个目录下的所有jar文件：

find . -name "\*.jar"|awk '{print "jar -tvf "$1}' | sh | grep ".\*.xml"

## other

### Zookeeper

启动

./bin/zkServer.sh start &

Kafka自带启动

./bin/zookeeper-server-start.sh ./config/zookeeper.properties

### redis

redis

1)启动： redis-server.exe redis.windows.conf

开启memcache，并连接测试：

以守护进程模式启动memcache

memcached-1.5.2/bin/memcached -d -l slc11fsp.us.oracle.com -p 11211 -m 2048 -u root

### svn

svn ls svn://slc02vlf.us.oracle.com:3690/repos

svn checkout svn://slc02vlf.us.oracle.com:3690/repos/parking

# log

svn log url

### vnc

• Kill VNC: vncserver -kill :1

• Create VNC: vncserver -geometry 1910x1050

• VNC set copy: vncconfig -nowin &

• List: vncserver -