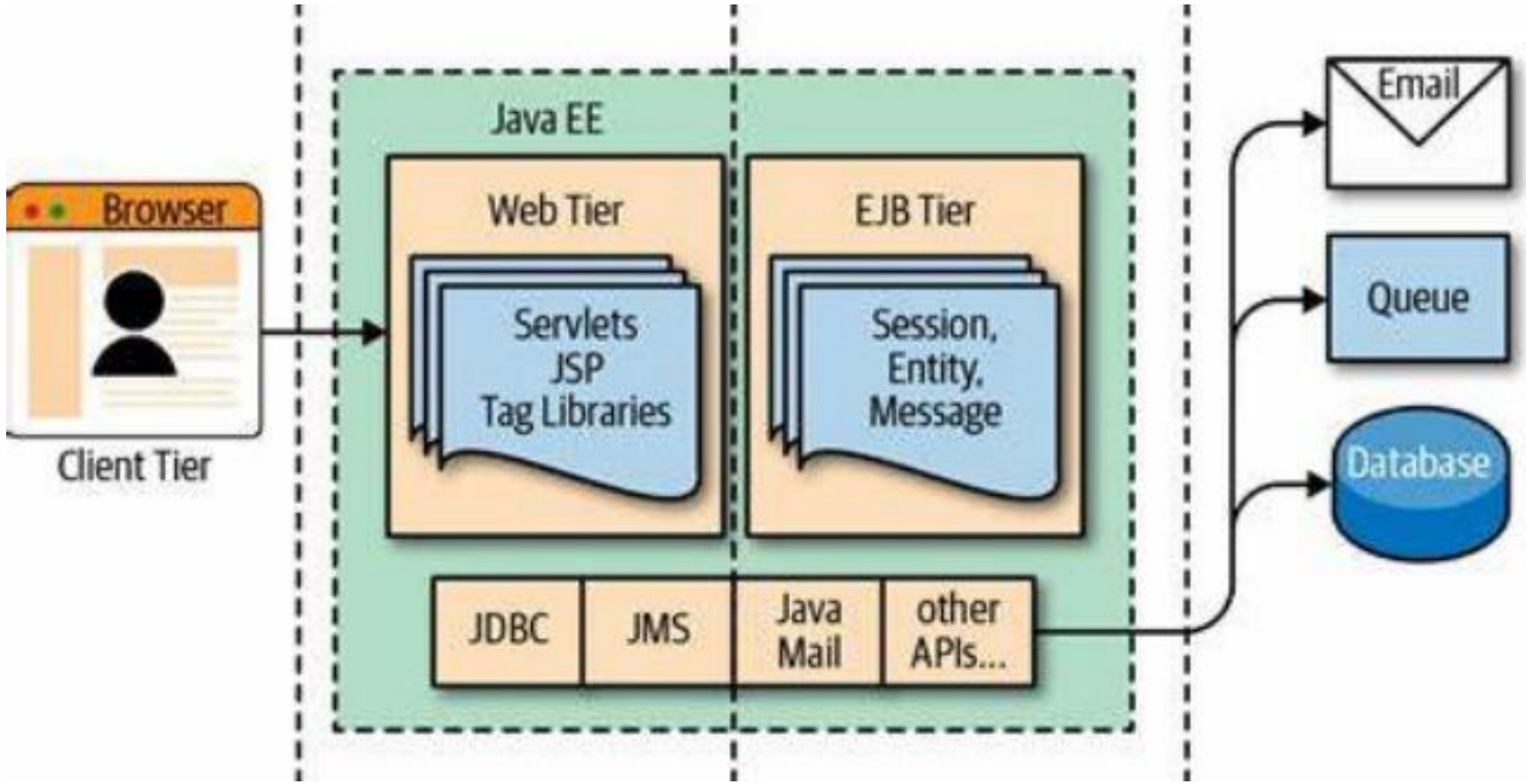
从单机应用说起


Java EE
Browser
Client Tier
Web Tier
Servlets
JSP
Tag Libraries
EJB Tier
Session,
Entity,
Message
JDBC
JMS
Java
Mail
other
APIs...
Email
Queue
Database

单机的问题

1、性能瓶颈：

1) 单机节点资源不足

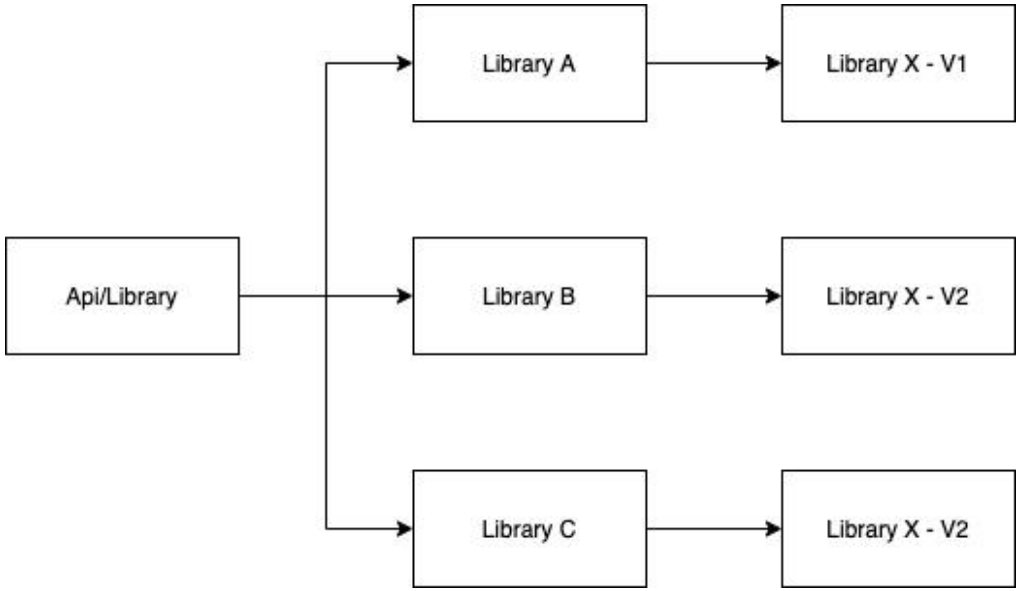
解决方案：集群部署

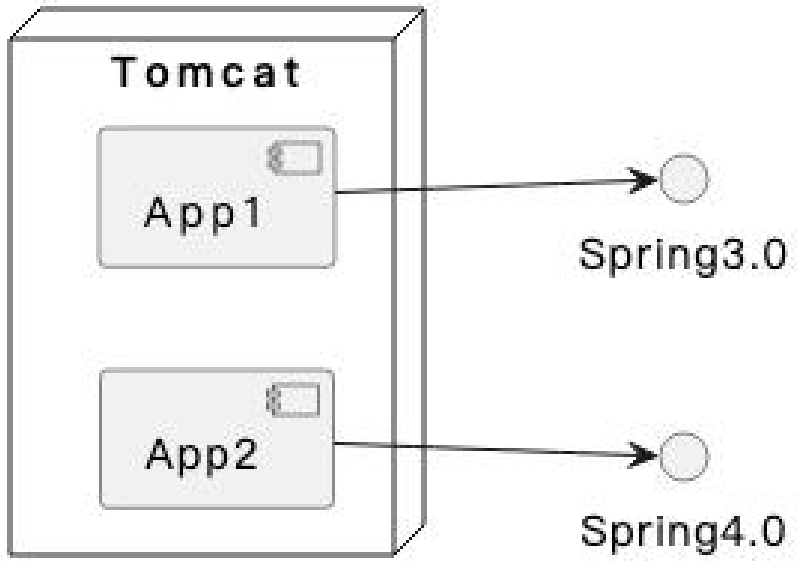
2) 不同业务性能负载不均衡

解决方案：业务拆分成微服务
针对不同业务进行集群部署。

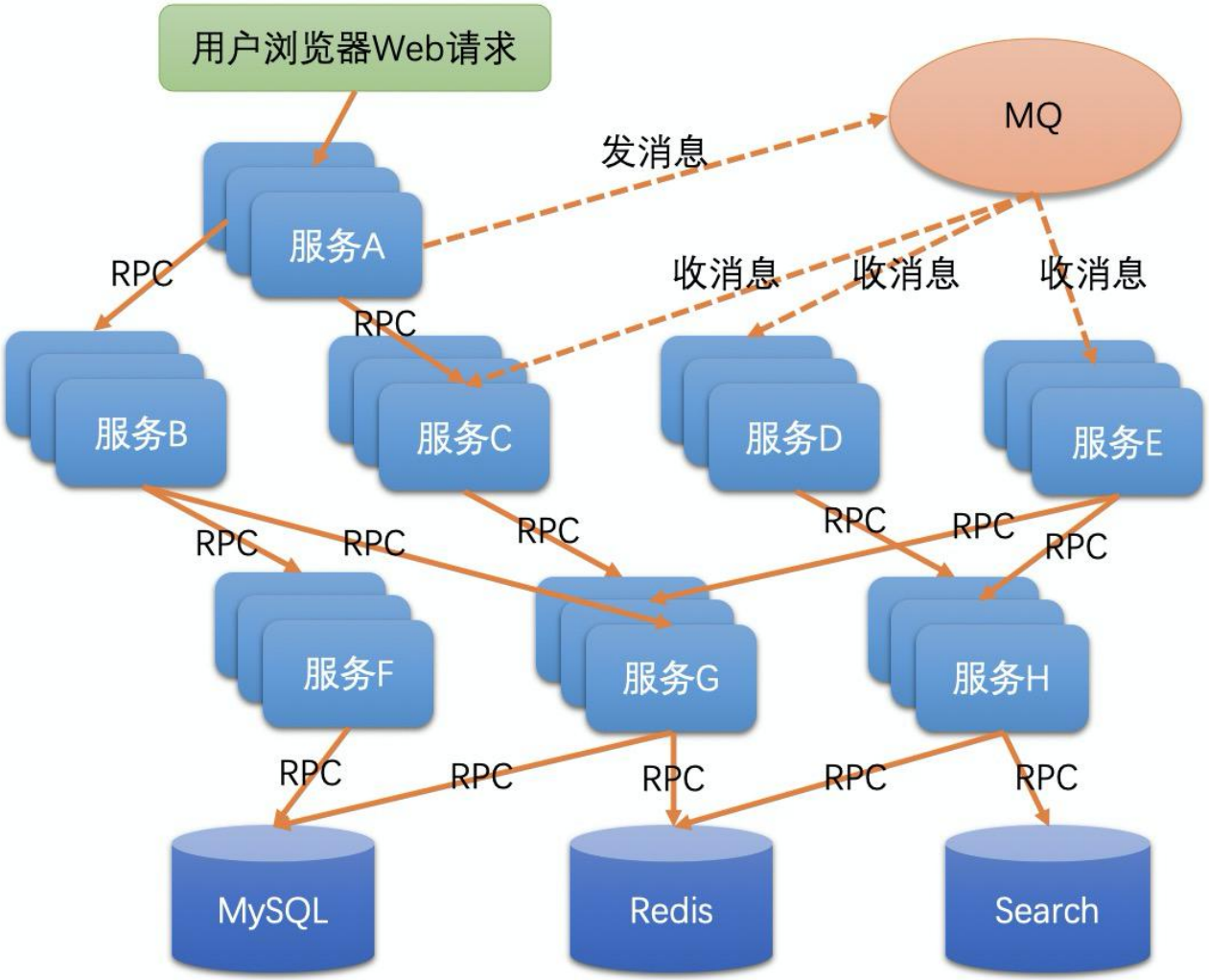
单机的问题

2、依赖过多，容易出现冲突；代码多编译时间长且升级重构困难



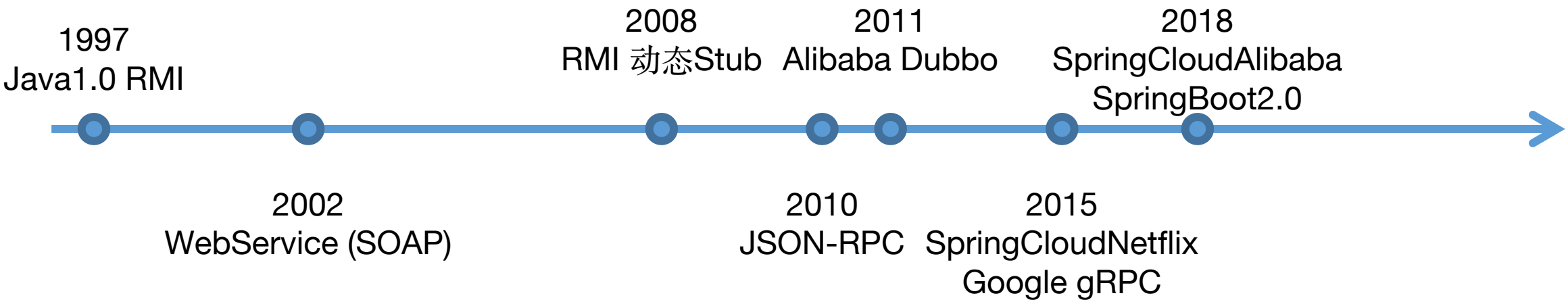


微服务架构


用户浏览器Web请求
MQ
发消息
服务A
RPC
收消息
收消息
收消息
RPC
服务B
服务C
服务D
服务E
RPC
RPC
RPC
RPC
RPC
RPC
服务F
服务G
服务H
RPC
RPC
RPC
RPC
RPC
MySQL
Redis
Search

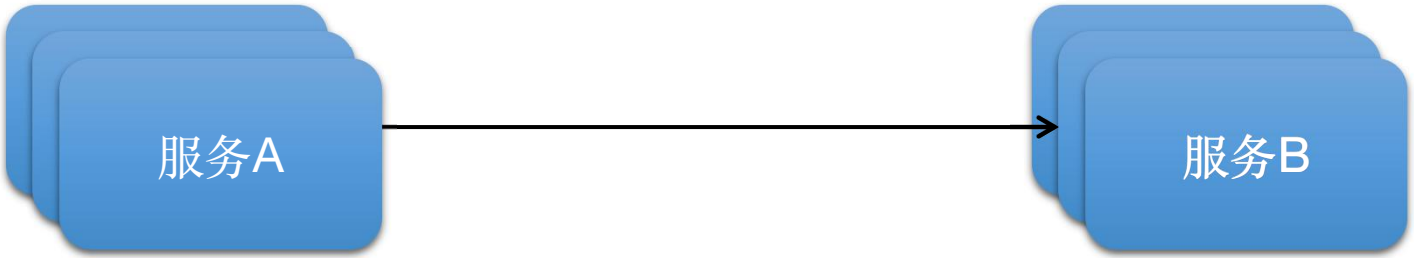
微服务的问题与治理

1、怎么通信



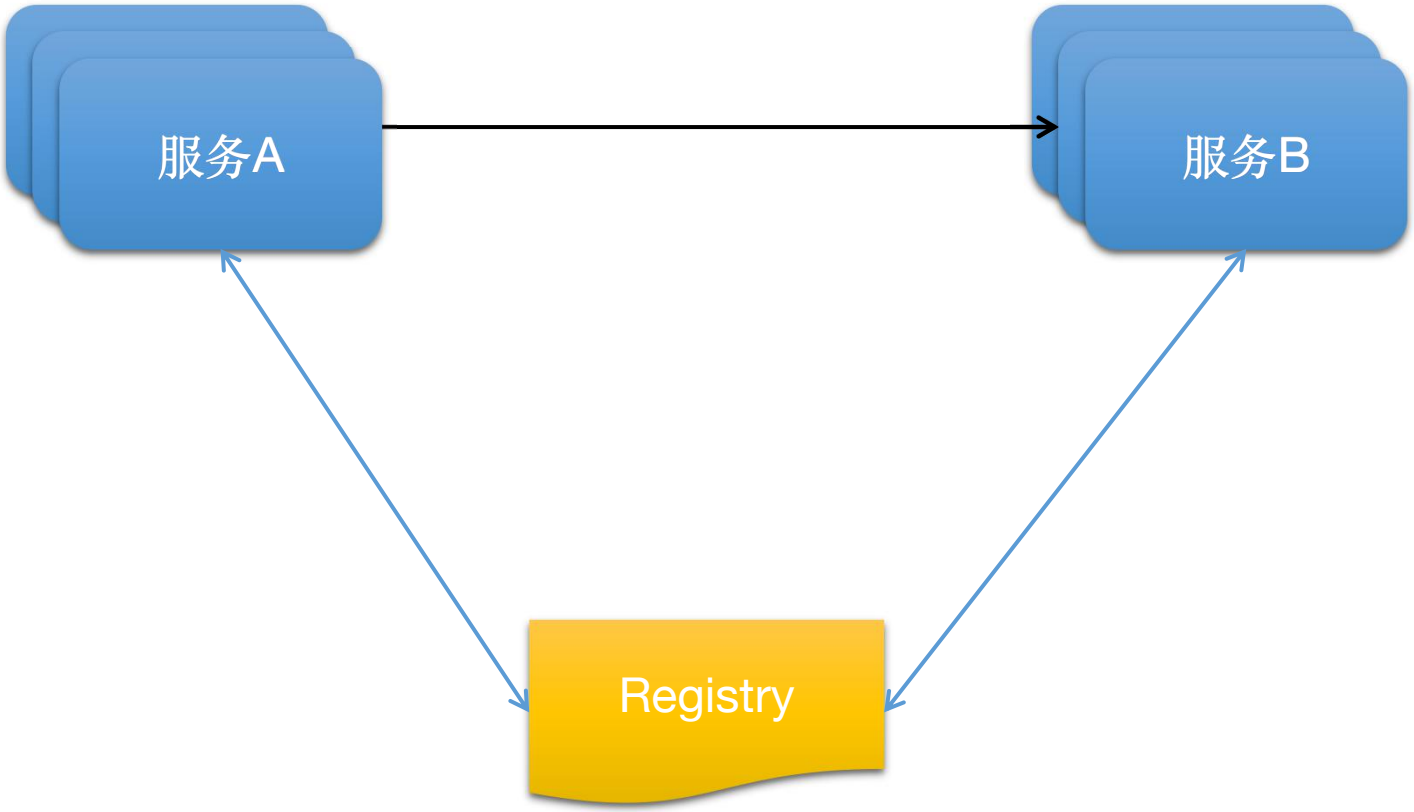
服务发现

在ip经常变更的场景下，怎么让服务之间能正常调用呢?



服务发现

java中最早使用服务发现机制的是RMI



参考DNS

注册中心的实现

要解决的问题：

1、 数据的持久化

2、 服务下线剔除：超时续约

3、 客户端容灾：本地缓存

4、 高可用：数据复制&CAP(一致性、可用性、分区容错性)

开源实现：

Apache Zookeeper(CAP)

Netflix Eureka(CAP)

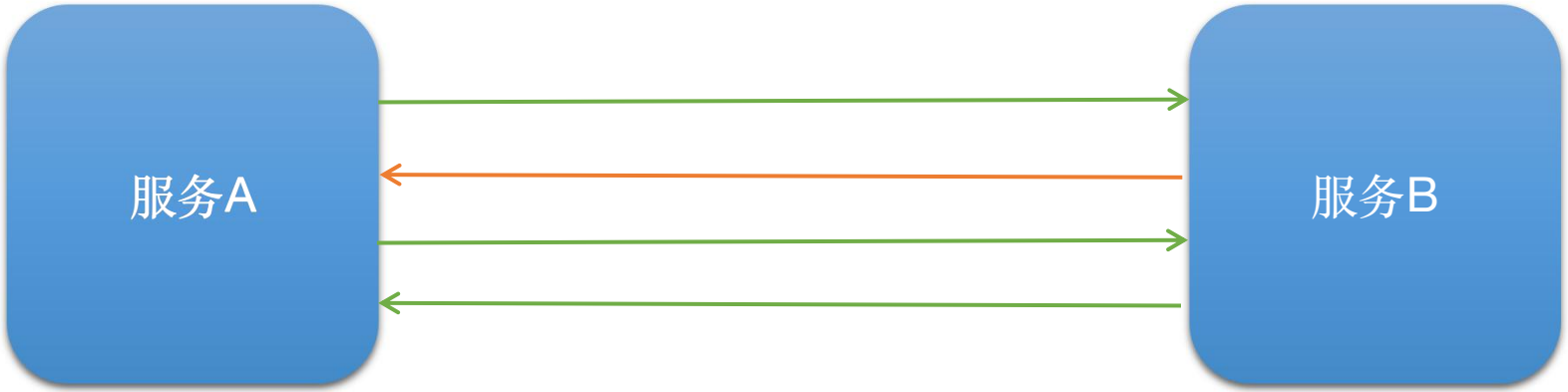
Alibaba Nacos(Raft:CAP & distro:CAP)

服务调用接口重试问题

1、接口重复调用：

解决方案：接口的幂等性

调用方提供outerId，服务方通过数据库唯一性约束



服务调用接口重试问题

2、服务雪崩问题：重试加大了流量压力

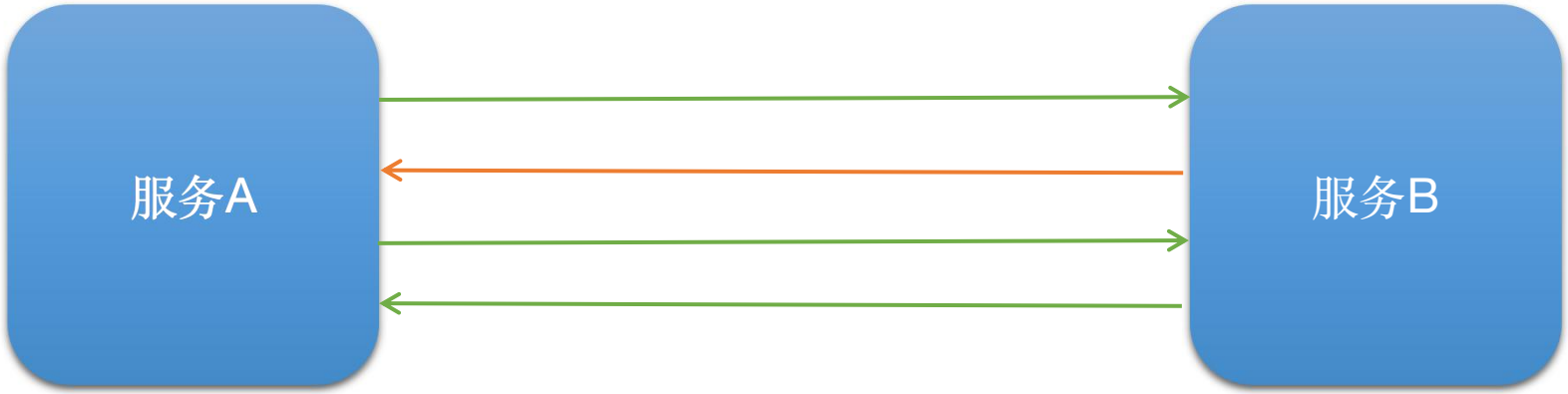
解决方案：

1) 服务端限流，超过承受范围直接拒绝服务，保证服务可用性。

guava RateLimiter

2) 客户端熔断降级，客户端在调用失败后，使用其他逻辑

Feign fallback



配置中心化

1、是否需要集中配置?

dev、test、prod环境的配置是不一样的。

不同环境的配置可以通过Spring profiles实现

application.yaml application.test.yaml application-prod.yaml

2、应用集群化后，改配置不方便。

改配置需要重启应用

一个好的配置中心应该具备的功能?

除了注册中心具备的基本功能外还应该有:

1) 配置更新推送 2) 权限管理 3) 版本管理

参考阅读：https://juejin.cn/post/6844903846041387016

配置中心的开源实现

spring：https://github.com/spring-cloud/spring-cloud-config

HashiCorp：https://github.com/hashicorp/consul

携程：https://github.com/apolloconfig/apollo

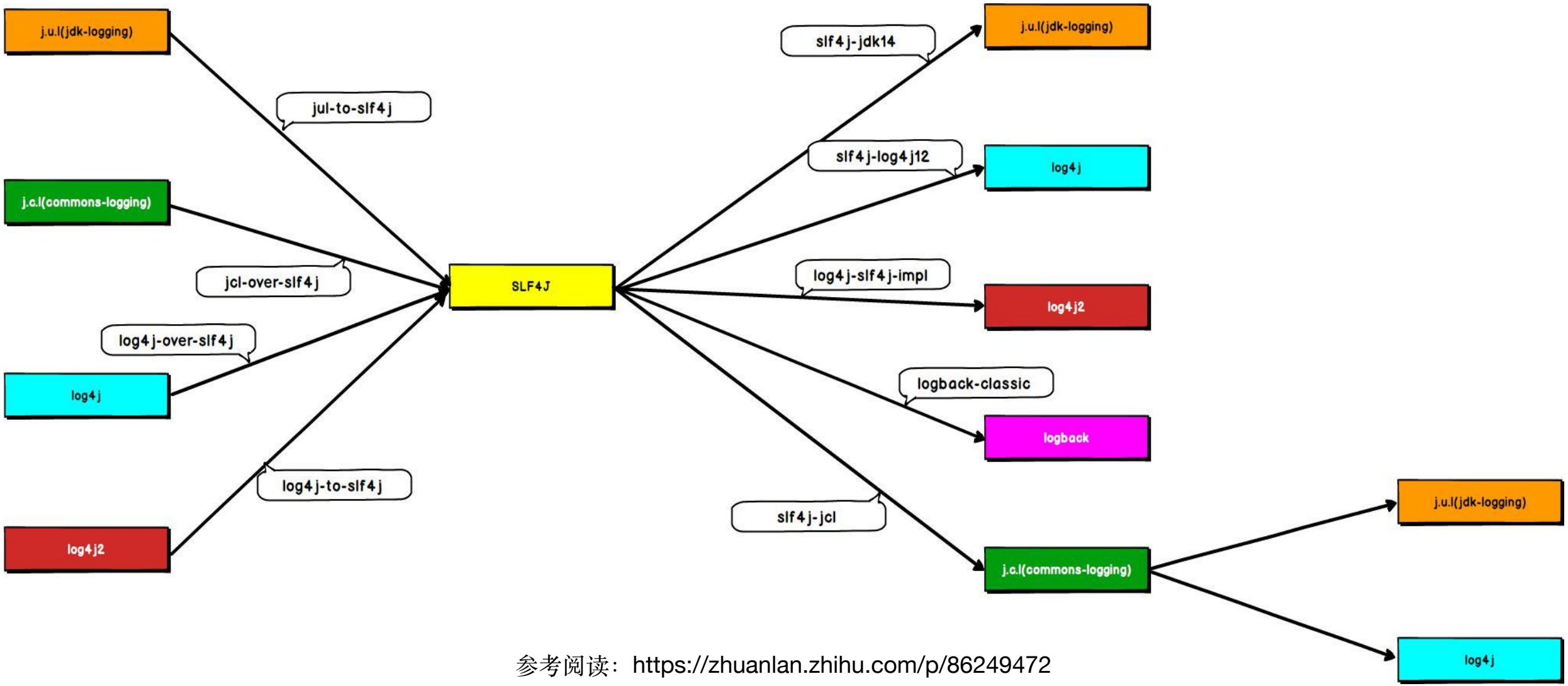
阿里：https://github.com/alibaba/nacos

https://github.com/topics/service-discovery

参考阅读：https://blog.hufeifei.cn/2021/09/Java/spring-cloud-config-refresh/

日志框架
Logging

日志框架
Logging



参考阅读：https://zhuanlan.zhihu.com/p/86249472

单机日志
Logging

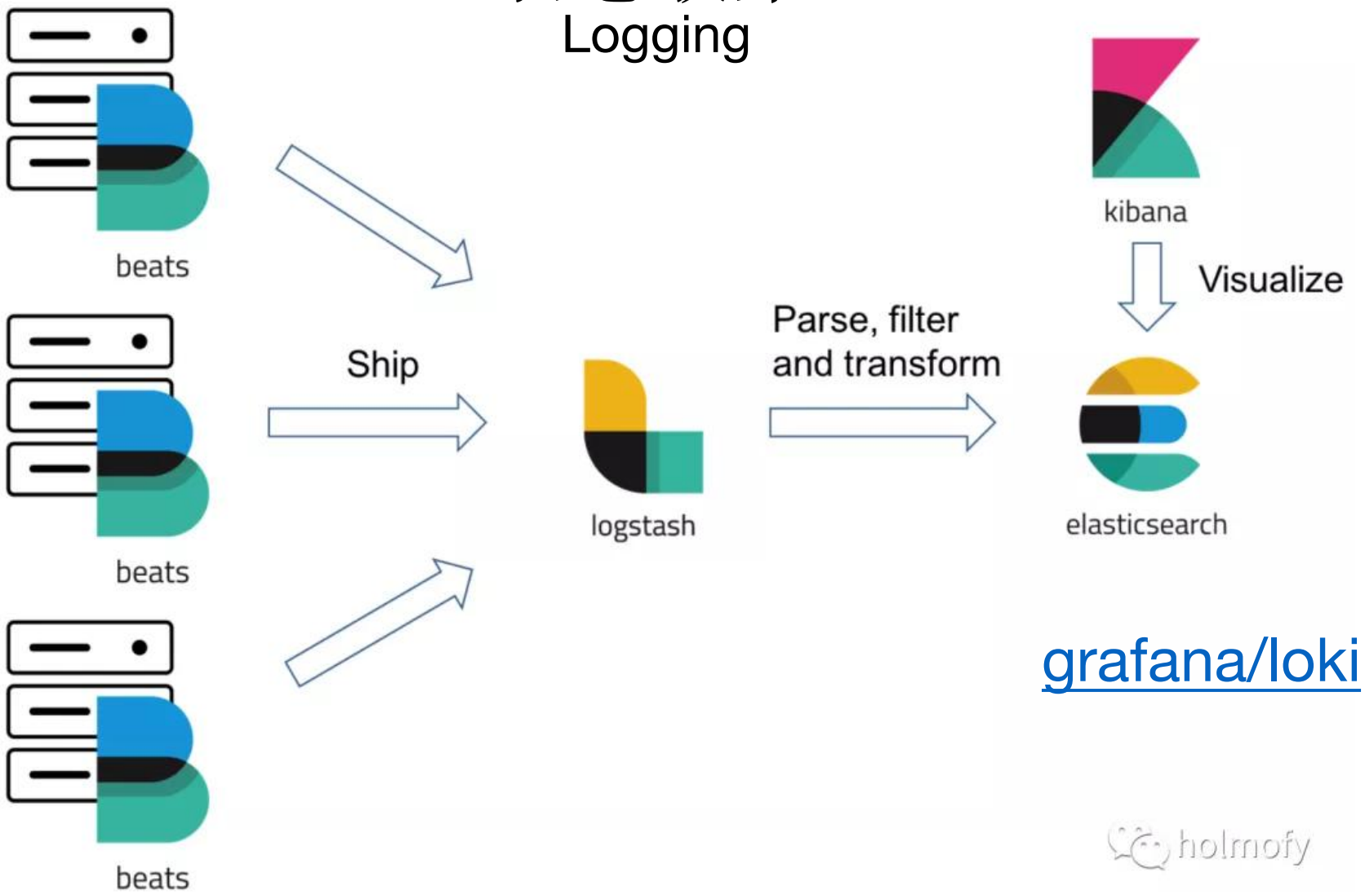
单机日志查看：

less、more、

head、tail

```
                    3.1.0
{com.limin.business.preshiftmeeting.mapper.PreShiftMeetingMapper.deleteById} Has been loaded by XML or SqlProvider, ignoring the injection of the SQL.
{com.limin.business.preshiftmeeting.mapper.PreShiftMeetingMapper.selectById} Has been loaded by XML or SqlProvider, ignoring the injection of the SQL.
2022-10-11 17:59:18.951 [main] WARN  c.b.mybatisplus.core.toolkit.TableInfoHelper - Warn: Could not find @TableId in Class:
com.limin.business.actualvideo.entity.UnitCameraArea.
2022-10-11 17:59:23.159 [main] INFO  com.limin.schedule.util.executor.XxlExecutorConfig - >>>>>>>>>>> XxlJobSpringExecutor register config init.
2022-10-11 17:59:23.166 [main] INFO  com.limin.schedule.util.executor.XxlExecutorConfig - >>>>>>>>>>> XxlJobSpringExecutor activeSuffix : prepro
2022-10-11 17:59:23.167 [main] INFO  com.limin.schedule.util.executor.XxlExecutorConfig - >>>>>>>>>>> XxlJobSpringExecutor suffix : prepro
2022-10-11 17:59:23.432 [main] INFO  com.limin.schedule.util.executor.XxlExecutorConfig - >>>>>>>>>>> XxlJobSpringExecutor {appName: [gzw-business-service-
executor-prepro], address: [172.254.141.216:17516], } auto register success. config complete.
2022-10-11 17:59:23.673 [main] WARN  com.netflix.config.sources.URLConfigurationSource - No URLs will be polled as dynamic configuration sources.
2022-10-11 17:59:23.683 [main] WARN  com.netflix.config.sources.URLConfigurationSource - No URLs will be polled as dynamic configuration sources.
2022-10-11 17:59:27.576 [main] INFO  com.limin.business.GzwBusinessApplication - Started GzwBusinessApplication in 21.903 seconds (JVM running for 22.906)
2022-10-11 18:08:43.611 [http-nio-8758-exec-7] DEBUG c.l.business.app.controller.XmMeetingController - isCompanyAdmin-begin-operateBy-1000000001>>>param:
XmMeetingQueryVo(id=null, meetingType=null, meetingSource=null, groupId=null, trainType=null, trainLevel=null, unitId=1560203510666199042, userId=1000000001,
address=null, title=null, startTime=null)
2022-10-11 18:08:43.664 [http-nio-8758-exec-7] DEBUG c.l.b.app.mapper.XmMeetingMapper.isCompanyAdmin - ==>  Preparing: select count(id) from
gzw_user_center.uc_user_post where unit_id = ? and user_id = ? and post_sequence = 20001111
2022-10-11 18:08:43.699 [http-nio-8758-exec-7] DEBUG c.l.b.app.mapper.XmMeetingMapper.isCompanyAdmin - ==> Parameters: 1560203510666199042(Long),
1000000001(Long)
2022-10-11 18:08:43.714 [http-nio-8758-exec-7] DEBUG c.l.b.app.mapper.XmMeetingMapper.isCompanyAdmin - <==      Total: 1
2022-10-11 18:08:43.718 [http-nio-8758-exec-7] DEBUG c.l.business.app.controller.XmMeetingController - isCompanyAdmin-end<<
2022-10-11 18:17:42.935 [http-nio-8758-exec-5] DEBUG c.l.business.app.controller.XmMeetingController - getLastMeetingAddress-begin-operateBy-
1565271082533502978>>>param: XmMeetingQueryVo(id=null, meetingType=20009253, meetingSource=20009302, groupId=null, trainType=null, trainLevel=null,
unitId=null, userId=1565271082533502978, address=null, title=null, startTime=null)
2022-10-11 18:17:42.998 [http-nio-8758-exec-5] DEBUG c.l.b.s.m.S.getLastMeetingAddress - ==>  Preparing: SELECT addr FROM safety_training WHERE is_deleted = 0
AND meeting_source = ? AND create_by = ? order by create_at desc limit 1
2022-10-11 18:17:42.998 [http-nio-8758-exec-5] DEBUG c.l.b.s.m.S.getLastMeetingAddress - ==> Parameters: 20009302(Long), 1565271082533502978(Long)
2022-10-11 18:17:43.000 [http-nio-8758-exec-5] DEBUG c.l.b.s.m.S.getLastMeetingAddress - <==      Total: 0
2022-10-11 18:17:43.001 [http-nio-8758-exec-5] DEBUG c.l.business.app.controller.XmMeetingController - getLastMeetingAddress-end<<
2022-10-11 18:17:43.114 [http-nio-8758-exec-6] DEBUG c.l.business.app.controller.XmMeetingController - getLastMeetingAddress-begin-operateBy-
1565271082533502978>>>param: XmMeetingQueryVo(id=null, meetingType=20009253, meetingSource=20009302, groupId=null, trainType=null, trainLevel=null,
unitId=null, userId=1565271082533502978, address=null, title=null, startTime=null)
2022-10-11 18:17:43.115 [http-nio-8758-exec-6] DEBUG c.l.b.s.m.S.getLastMeetingAddress - ==>  Preparing: SELECT addr FROM safety_training WHERE is_deleted = 0
AND meeting_source = ? AND create_by = ? order by create_at desc limit 1
2022-10-11 18:17:43.116 [http-nio-8758-exec-6] DEBUG c.l.b.s.m.S.getLastMeetingAddress - ==> Parameters: 20009302(Long), 1565271082533502978(Long)
```

日志收集
Logging



grafana/loki

Fluentd
fluent-bit

ElasticSearch做日志存储的问题

1、日志量庞大，用ES对日志做全文索引会消耗非常多的资源。

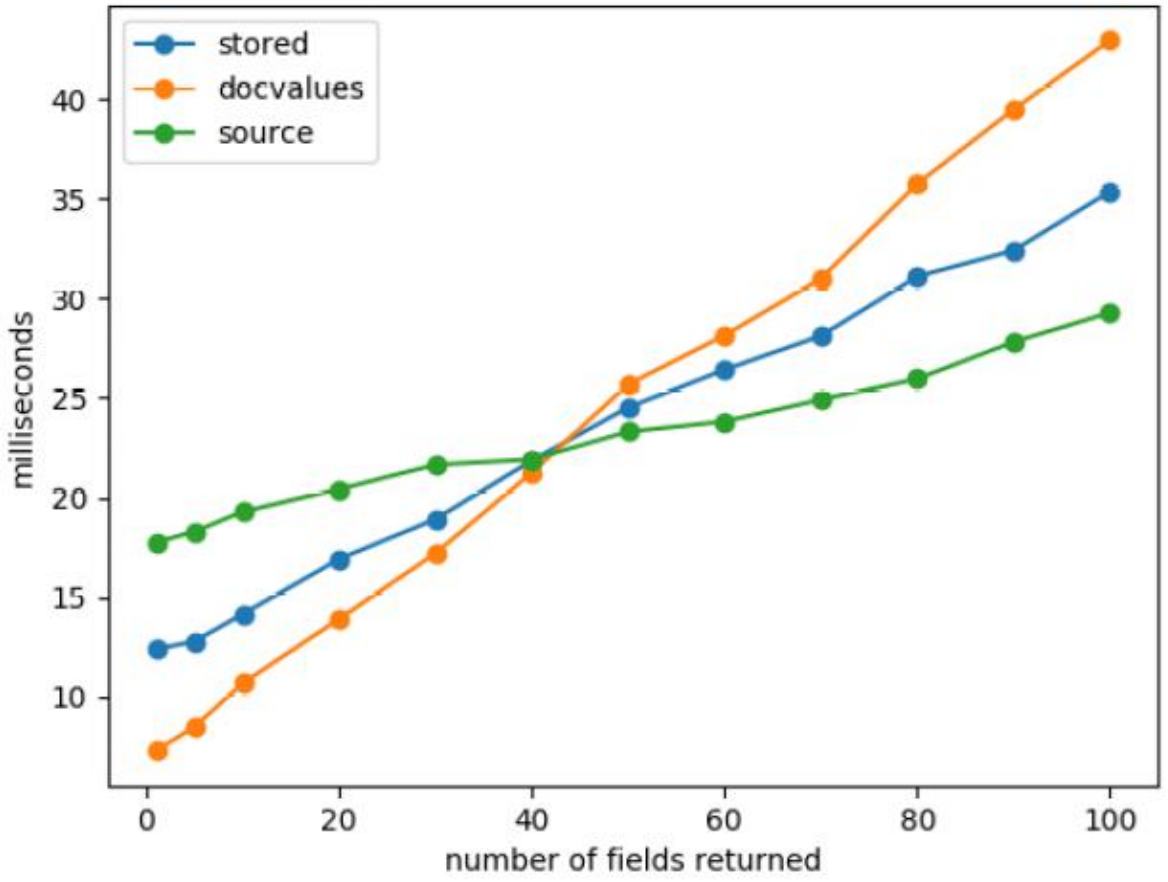
ES对数据做了三种格式的存储：以json文档存储的source、以行式存储的store_fields、以列式存储的doc_values，不精通ES的人很可能创建出低效的索引。ES对内存的要求比较高，ES用到大量内存映射来缓存索引以提高搜索速度，通常建议Java虚拟机设置成机器内存的1/2，预留一半的内存给索引的native缓存使用。

2、全文索引对日志检索来说，用处并不大。

有时候因为分词不准确反而会影响日志的搜索。

3、当并发量过大的时候，ES会reject掉处理不了的请求，如果中间没有kafka做日志转储，很容易丢日志。而往往出问题需要排查的时候就是并发量大的时候。

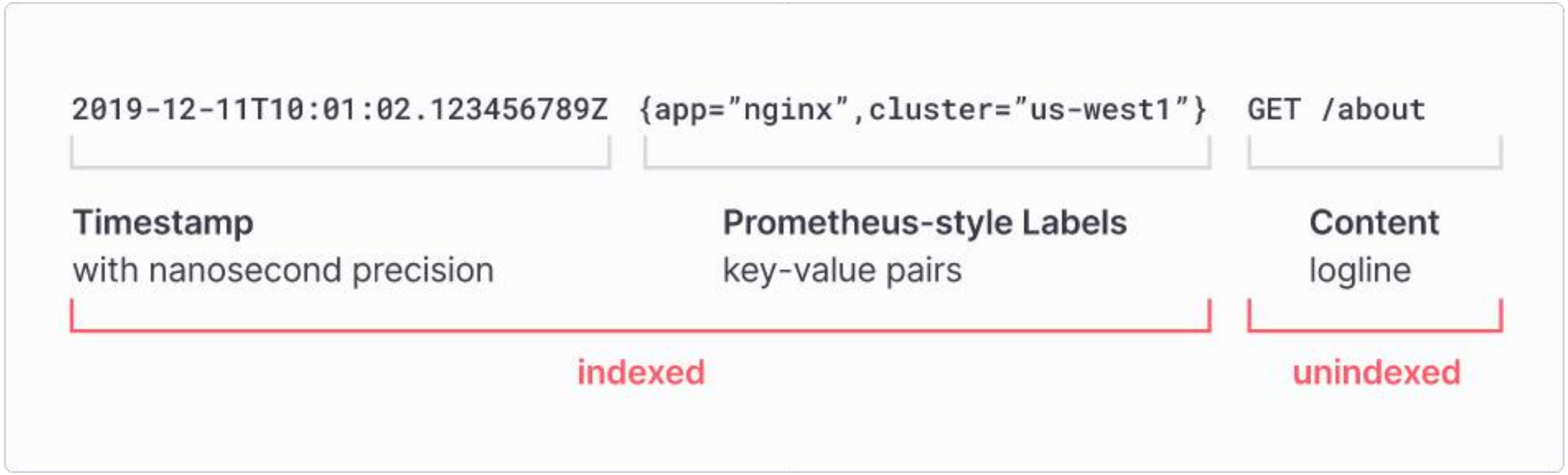
ElasticSearch三种存储方式的读取性能对比



Loki的口号是：Loki: like Prometheus, but for logs.

它是Grafana专门为日志设计的一款存储。它并不索引日志内容，只是对每个日志记录打上label，在grafana中使用label进行搜索。

只索引元数据

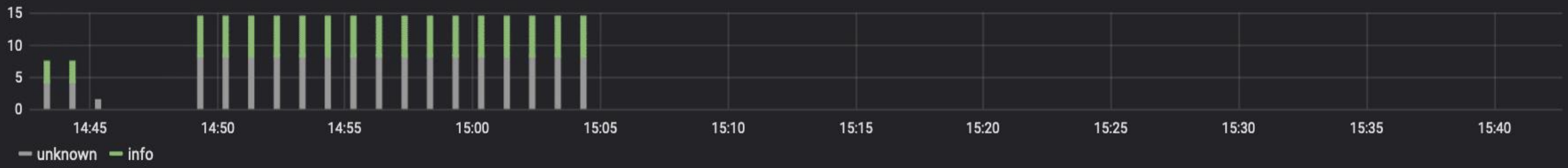

2019-12-11T10:01:02.123456789Z
{app="nginx",cluster="us-west1"}
GET /about
Timestamp
with nanosecond precision
Prometheus-style Labels
key-value pairs
Content
logline
indexed
unindexed

Explore | Loki | Split | Last 1 hour | Clear All | Run Query

Log labels {driver="Python",location="eu-central-1",name=~"fun.*"} Conf

0.1s

Logs



Time Labels Dedup none exact numbers signature

Common labels: Python eu-central-1 Limit: 1000 (239 returned)

```
END RequestId: fcee1b99-d40d-498c-9afe-5a2192d731ac
REPORT RequestId: fcee1b99-d40d-498c-9afe-5a2192d731ac Duration: 23.89 ms Billed Duration: 100 ms Memory Size: 128 MB Max Memory Used: 57 MB XRAY TraceId: 1-5d7107d7-9f71b1293f6c041367809b69 SegmentId: 6a39626530cdf50f Sampled: false
[INFO] 2019-09-05T13:04:23.678Z fcee1b99-d40d-498c-9afe-5a2192d731ac func2 started...
[INFO] 2019-09-05T13:04:23.679Z fcee1b99-d40d-498c-9afe-5a2192d731ac Hey there! This should go to Loki please!
[INFO] 2019-09-05T13:04:23.679Z fcee1b99-d40d-498c-9afe-5a2192d731ac Config: {'name': 'func2', 'info': 'Hey there Loki. I am: func2', 'driver': 'Python', 'location': 'eu-central-1'}
START RequestId: fcee1b99-d40d-498c-9afe-5a2192d731ac Version: $LATEST
END RequestId: 5e6c9458-47b7-4c01-a1f9-40a9b1d2a409
REPORT RequestId: 5e6c9458-47b7-4c01-a1f9-40a9b1d2a409 Duration: 4.02 ms Billed Duration: 100 ms Memory Size: 128 MB Max Memory Used: 57 MB XRAY TraceId: 1-5d7107d1-3b2bbd404e6444e857bc3480 SegmentId: 2e3152764ca3db24 Sampled: false
[INFO] 2019-09-05T13:04:17.374Z 5e6c9458-47b7-4c01-a1f9-40a9b1d2a409 func1 started...
[INFO] 2019-09-05T13:04:17.374Z 5e6c9458-47b7-4c01-a1f9-40a9b1d2a409 Hey there! This should go to Loki please!
[INFO] 2019-09-05T13:04:17.374Z 5e6c9458-47b7-4c01-a1f9-40a9b1d2a409 Config: {'name': 'func1', 'info': 'Hey there Loki. I am: func1', 'driver': 'Python', 'location': 'eu-ce
```

链路追踪
Tracing

单机链路追踪：

1、线程堆栈：

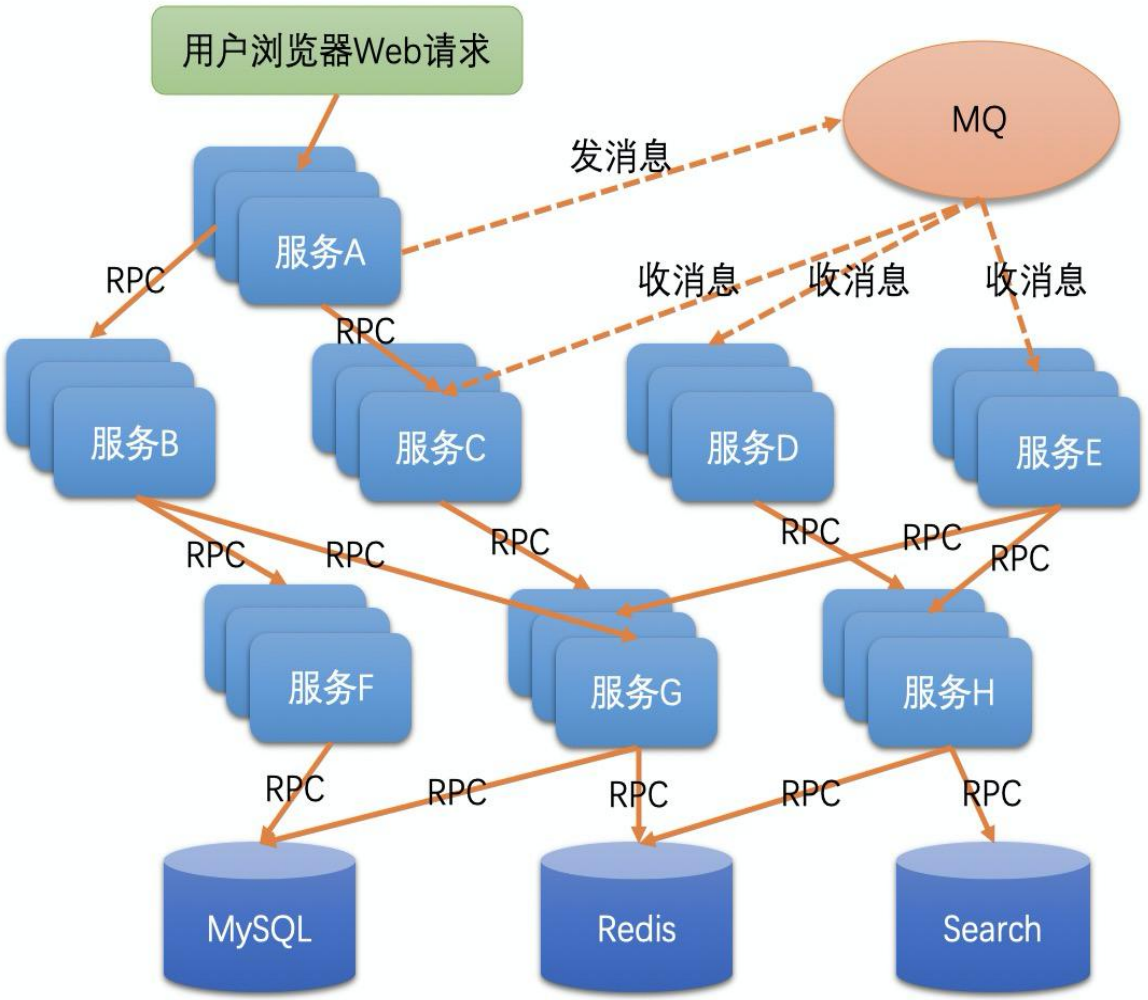
jstack

2、链路性能分析：

arthas

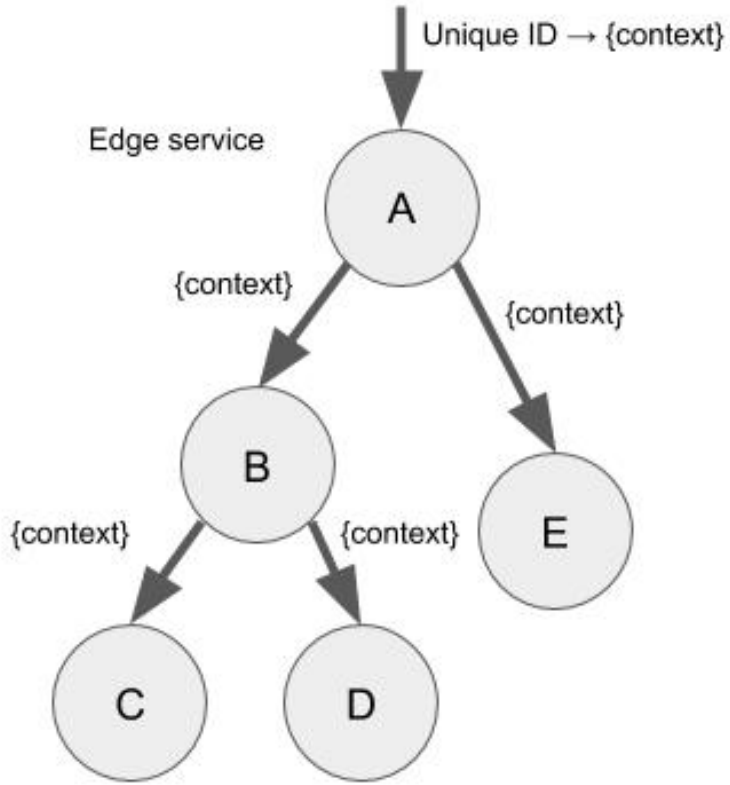
example：[github]

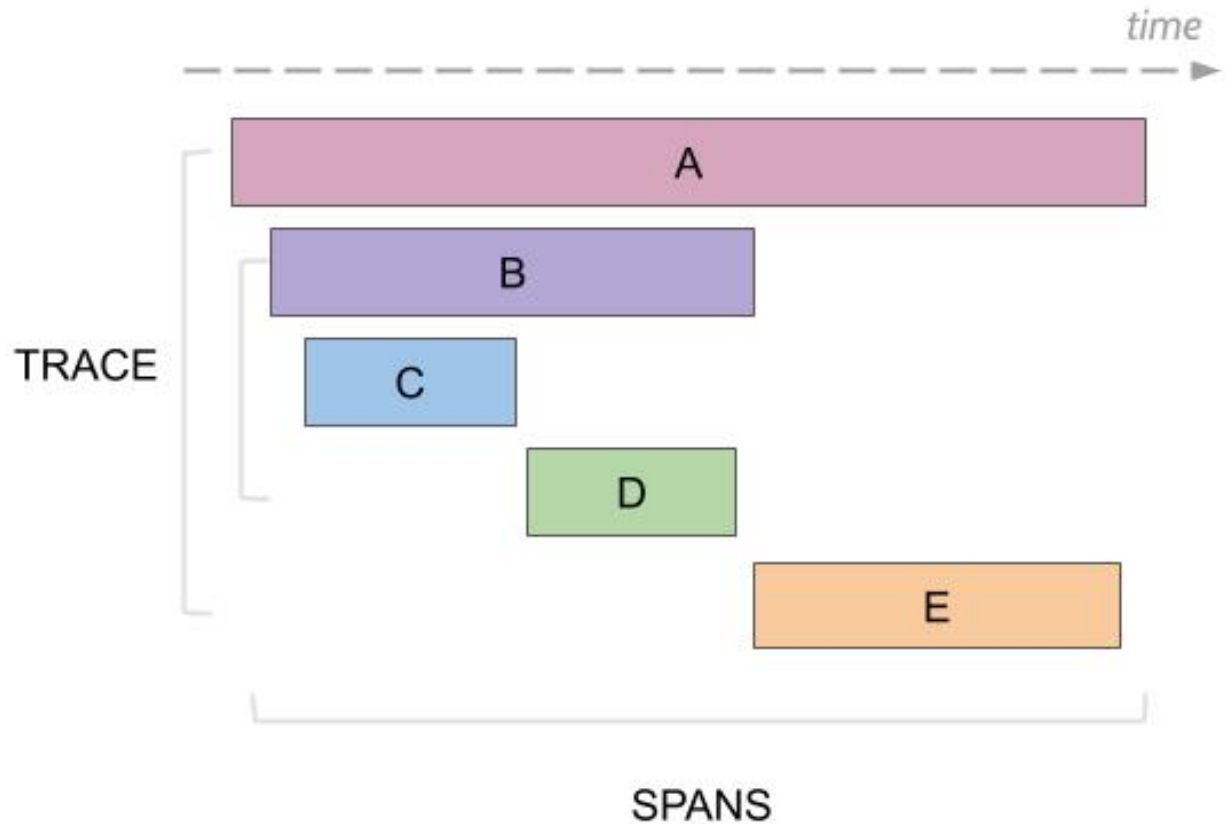
微服务集群中的链路追踪



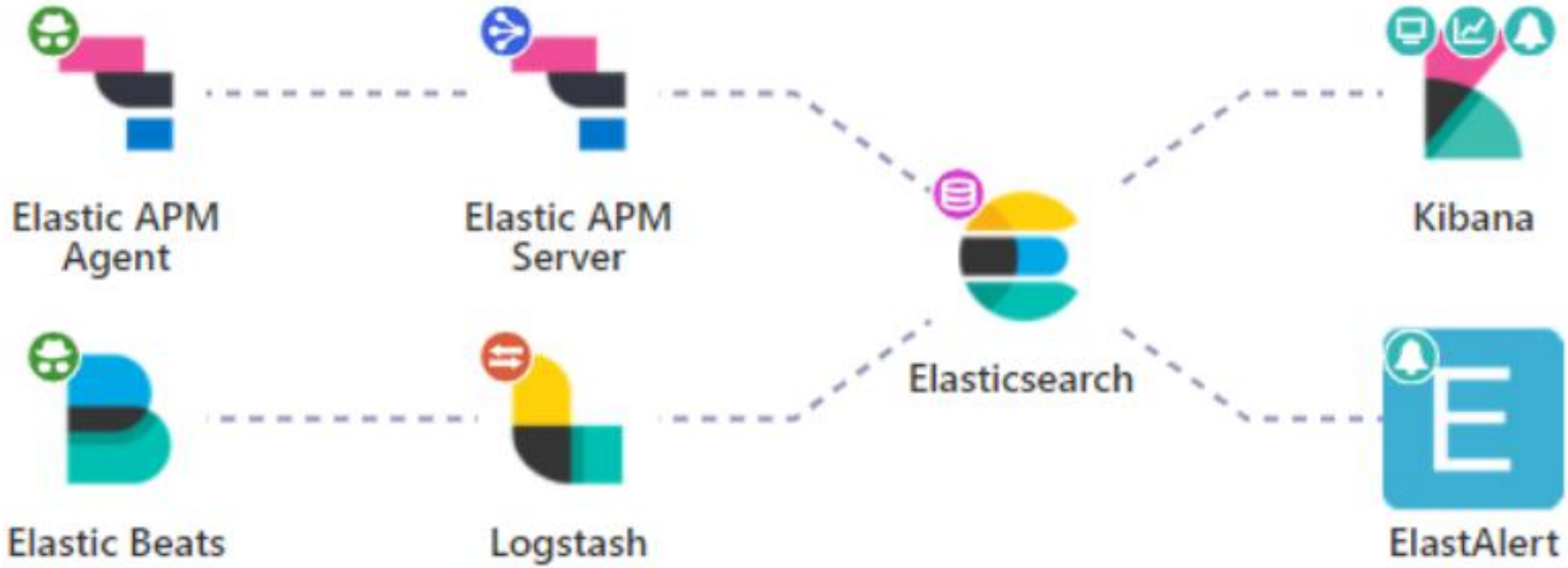
- 分布式系统服务非常多，很复杂
- 每个服务可能由不同项目组开发，没有一个人能详细地了解所有的系统。
- 每个服务都可能集群部署，有很多台机器，整个系统可能有成千上万台机器。
- 服务可能由不同语言开发的。
- 当需要了解系统的整体表现或系统瓶颈时，需要知道整个调用链路的每个部分的耗时情况。
- 当一次链路过程调用出错了，需要知道具体是哪个服务的哪一台机器出错，而不是到每一台机器上去看日志。

参考阅读：https://blog.hufeifei.cn/2020/07/Alibaba/distributed-tracing/

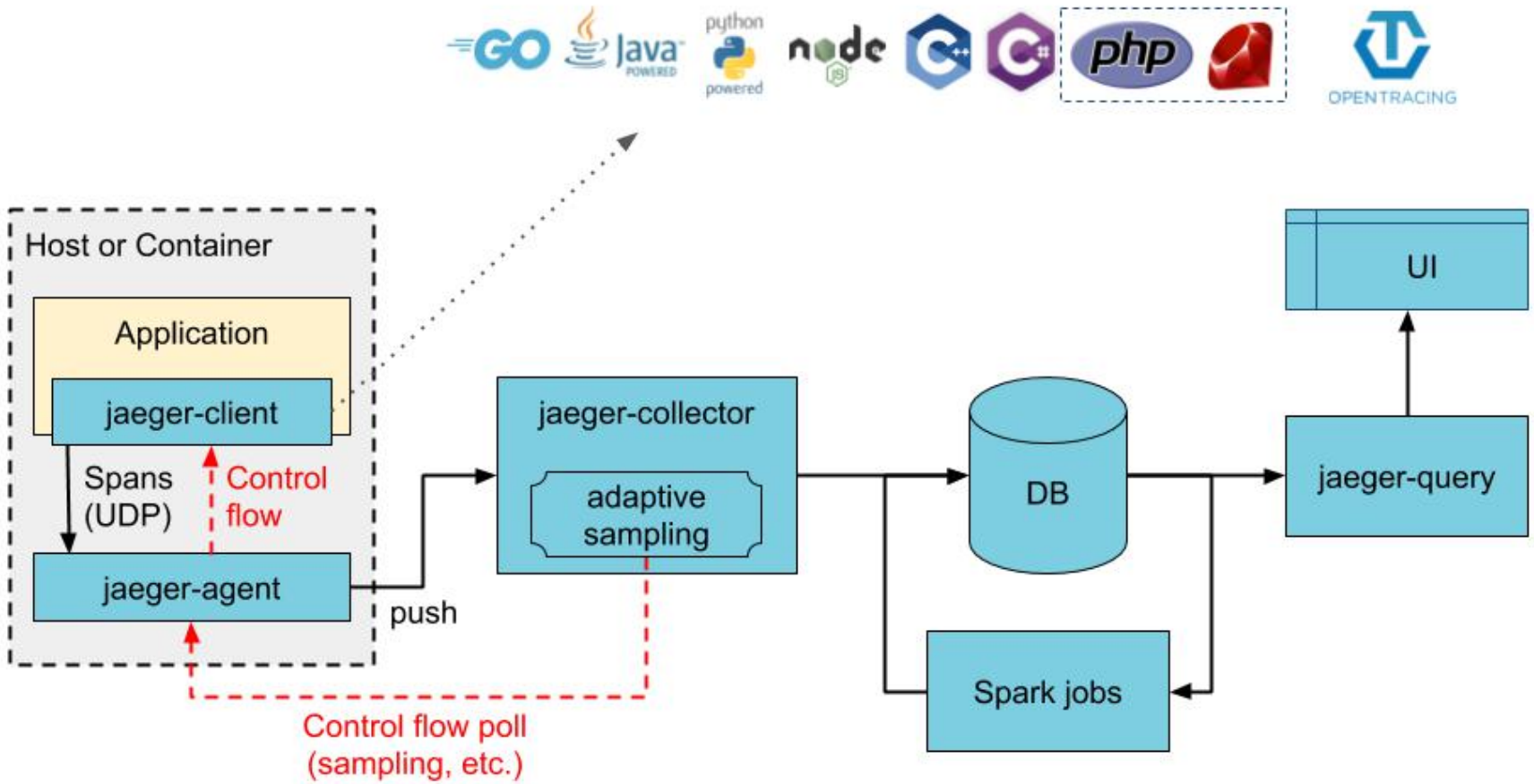

Unique ID → {context}
Edge service
A
{context}
{context}
B
E
{context}
{context}
C
D


time
A
B
C
D
E
TRACE
SPANS

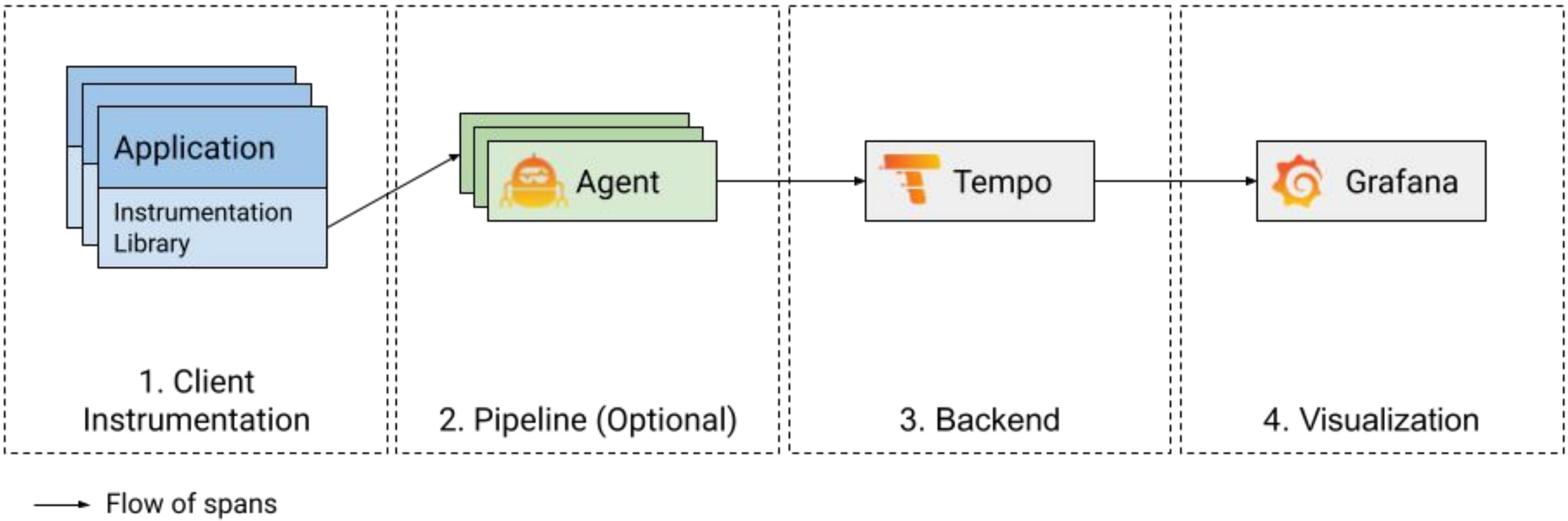
Skywalking链路追踪


Elastic APM
Agent
Elastic APM
Server
Kibana
Elasticsearch
Elastic Beats
Logstash
ElastAlert

Uber Jaeger


Host or Container
Application
jaeger-client
Spans
(UDP)
Control
flow
jaeger-agent
push
jaeger-collector
adaptive
sampling
DB
UI
jaeger-query
Spark jobs
Control flow poll
(sampling, etc.)
OPENTRACING

Tempo


Application
Instrumentation Library
Agent
Tempo
Grafana
1. Client Instrumentation
2. Pipeline (Optional)
3. Backend
4. Visualization
Flow of spans

Query type | Search | TraceID

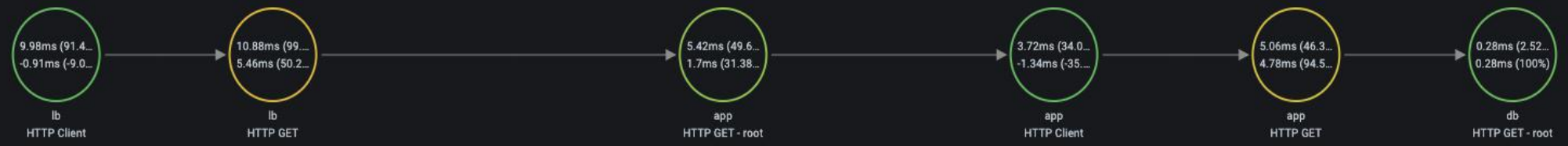
0.1s

Trace ID 2907d63401acff4

Help >

+ Add query | Query history | Inspector

Node graph Beta



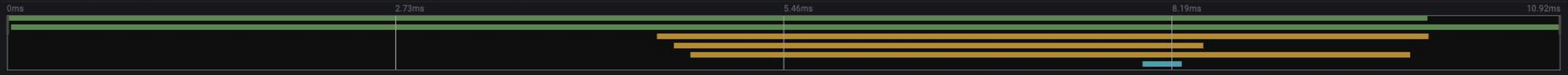
Total time (% of trace) | Self time (% of total) | Self time / Trace duration

Trace View

lb: HTTP Client 0184785449952923636

Find...

Trace Start **May 20 2021, 14:59:21.634** Duration **10.92ms** Services **3** Depth **6** Total Spans **6**



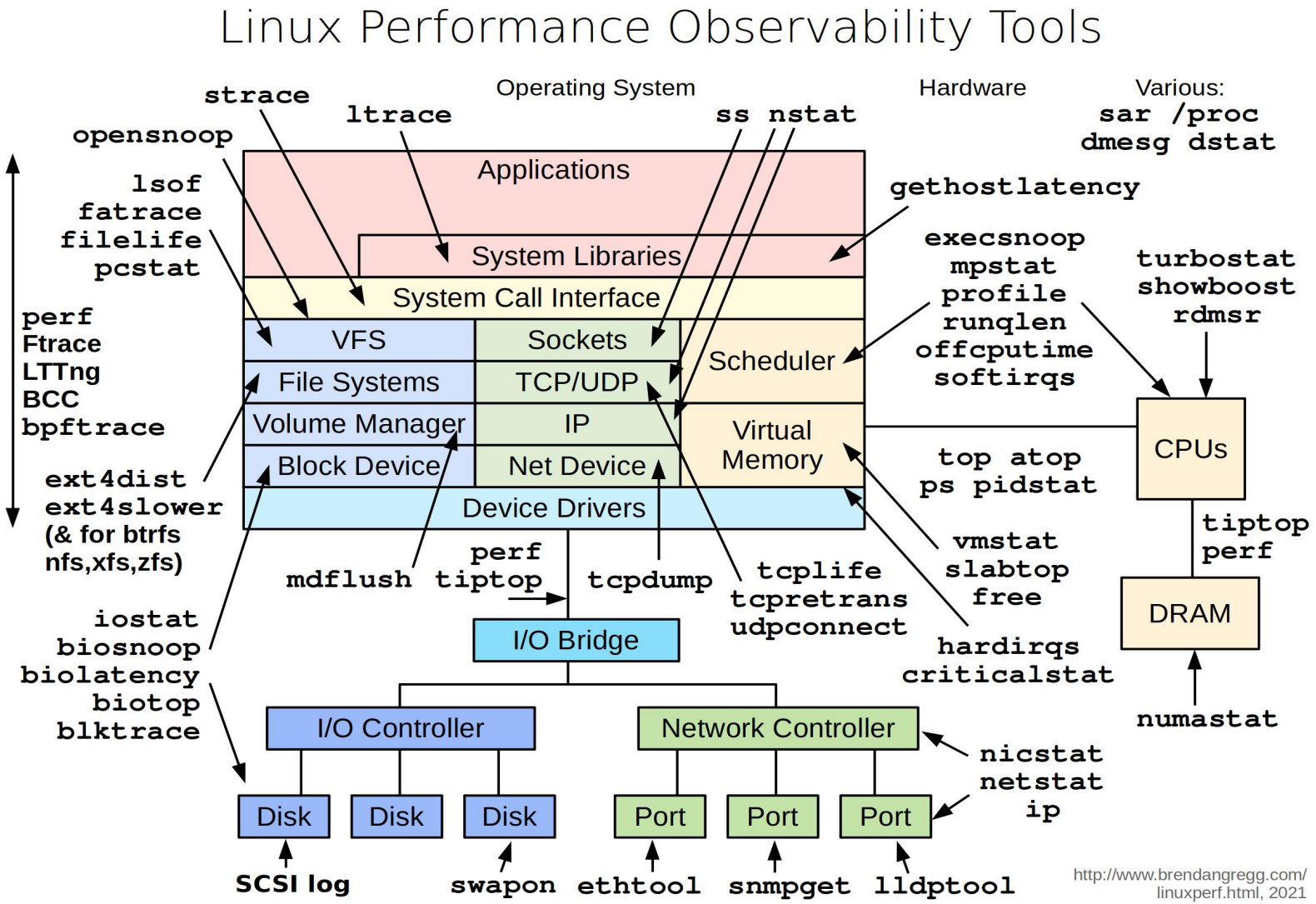
Service & Operation	0ms	2.73ms	5.46ms	8.19ms	10.92ms
lb HTTP Client					9.98ms
lb HTTP GET					
app HTTP GET - root		5.42ms			
app HTTP Client		3.72ms			
app HTTP GET		5.06ms			
db HTTP GET - root		0.28ms			

指标监控
Metrics

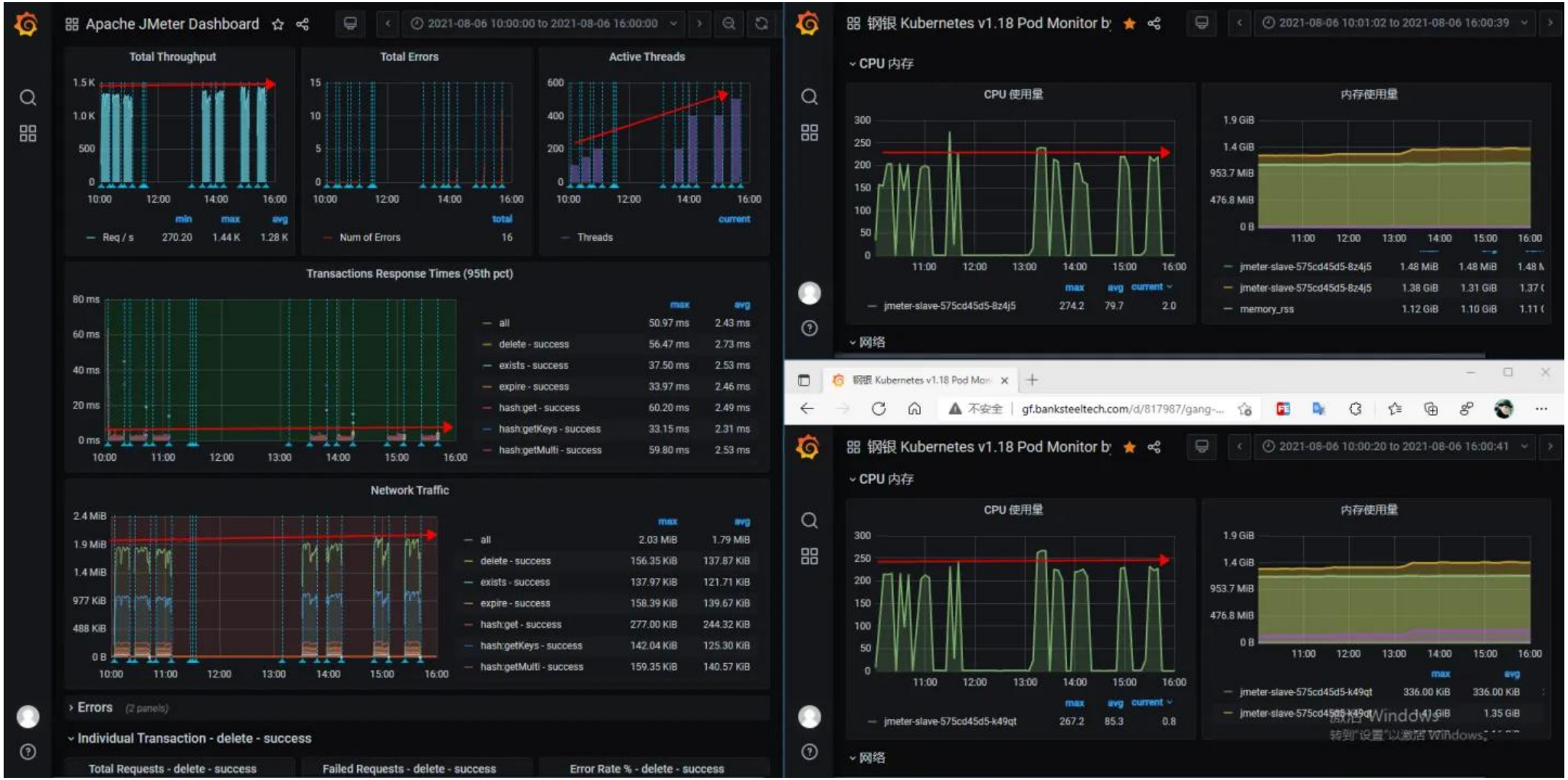
单机监控：

top、jconsole、

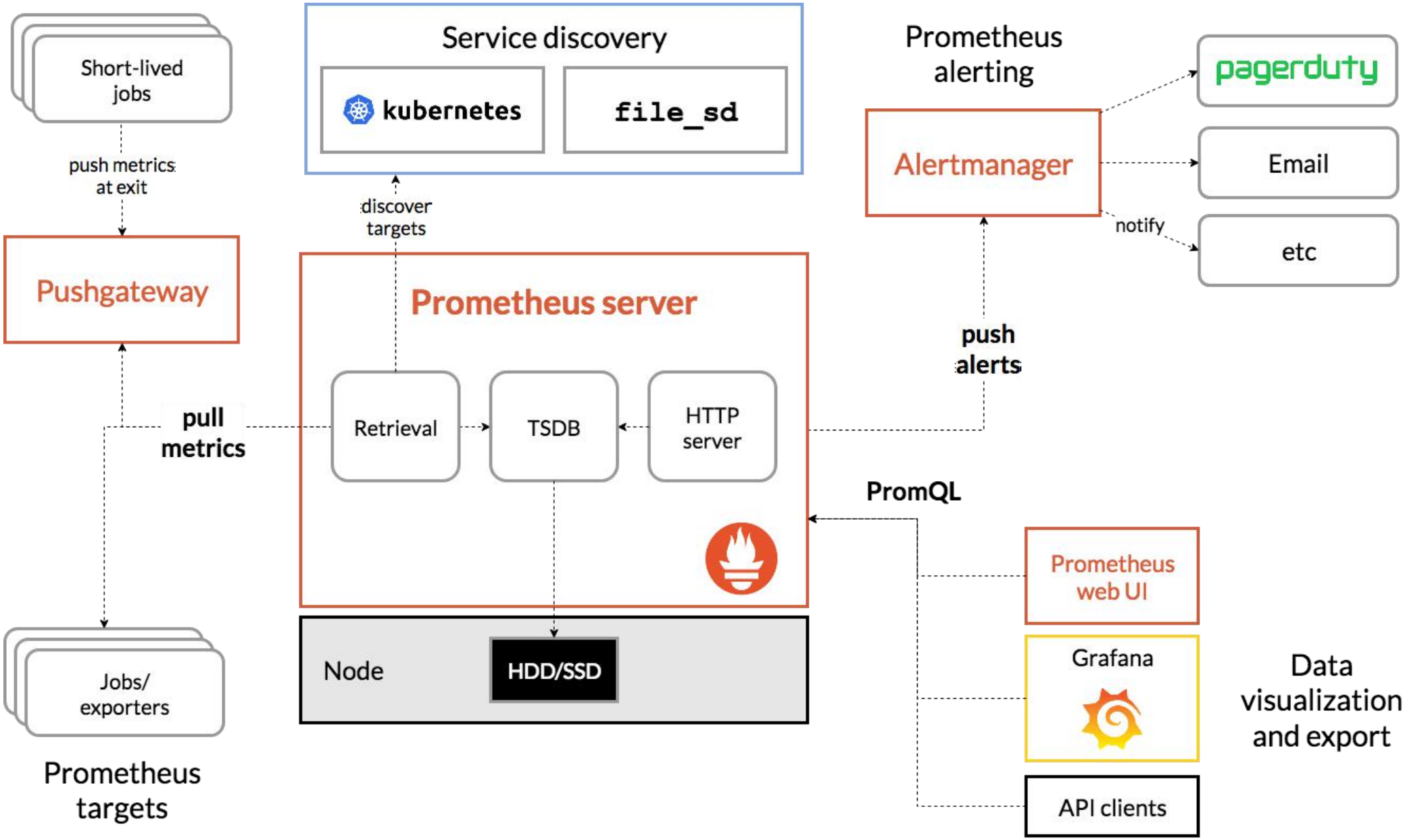
java MXBean



分布式系统的性能监控

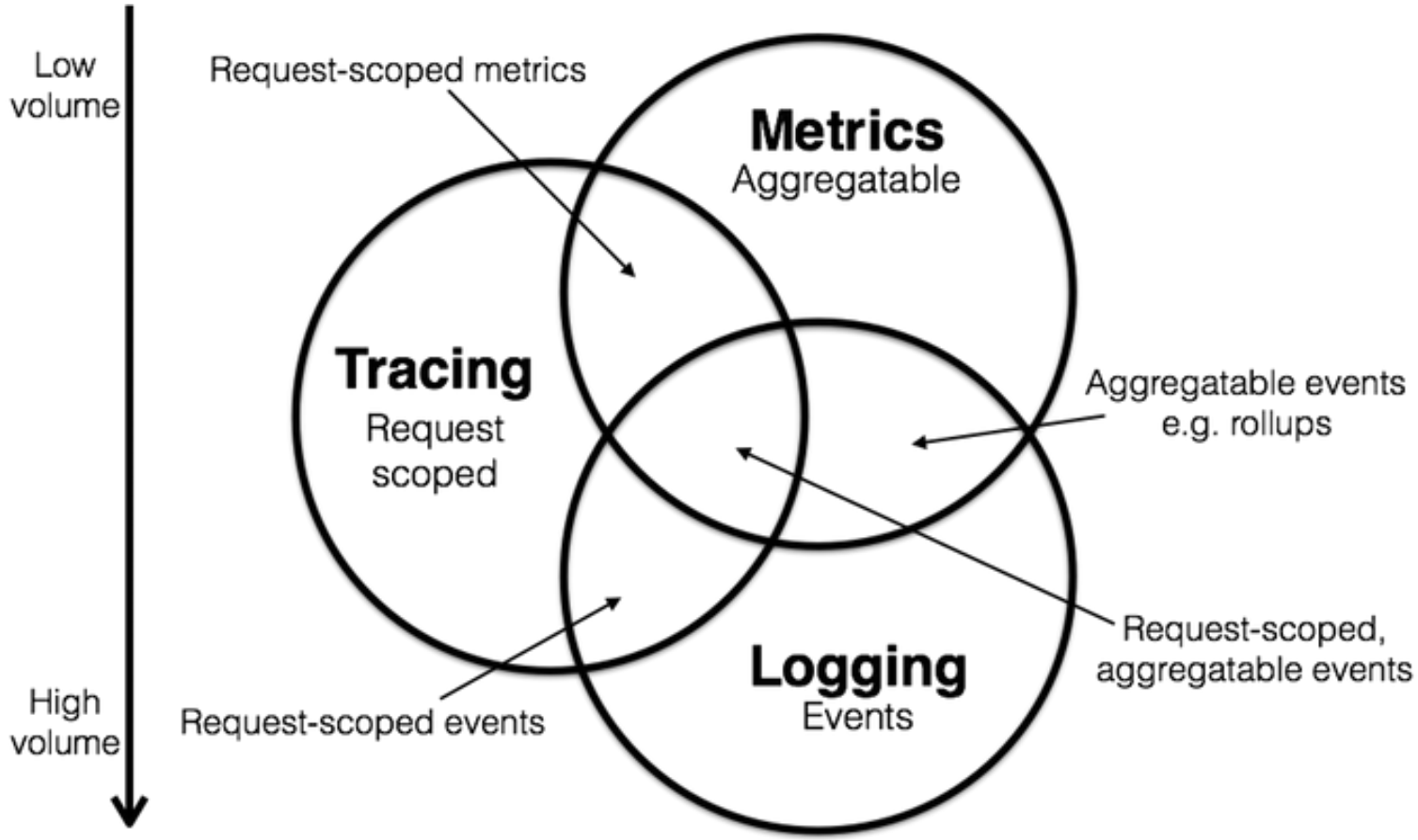


参考阅读：https://zhuanlan.zhihu.com/p/397762671


Short-lived jobs
push metrics at exit
Pushgateway
Service discovery
kubernetes
file_sd
discover targets
Prometheus server
Retrieval
TSDB
HTTP server
pull metrics
Node
HDD/SSD
Jobs/ exporters
Prometheus targets
Prometheus alerting
Alertmanager
pagerduty
Email
etc
notify
push alerts
PromQL
Prometheus web UI
Grafana
API clients
Data visualization and export

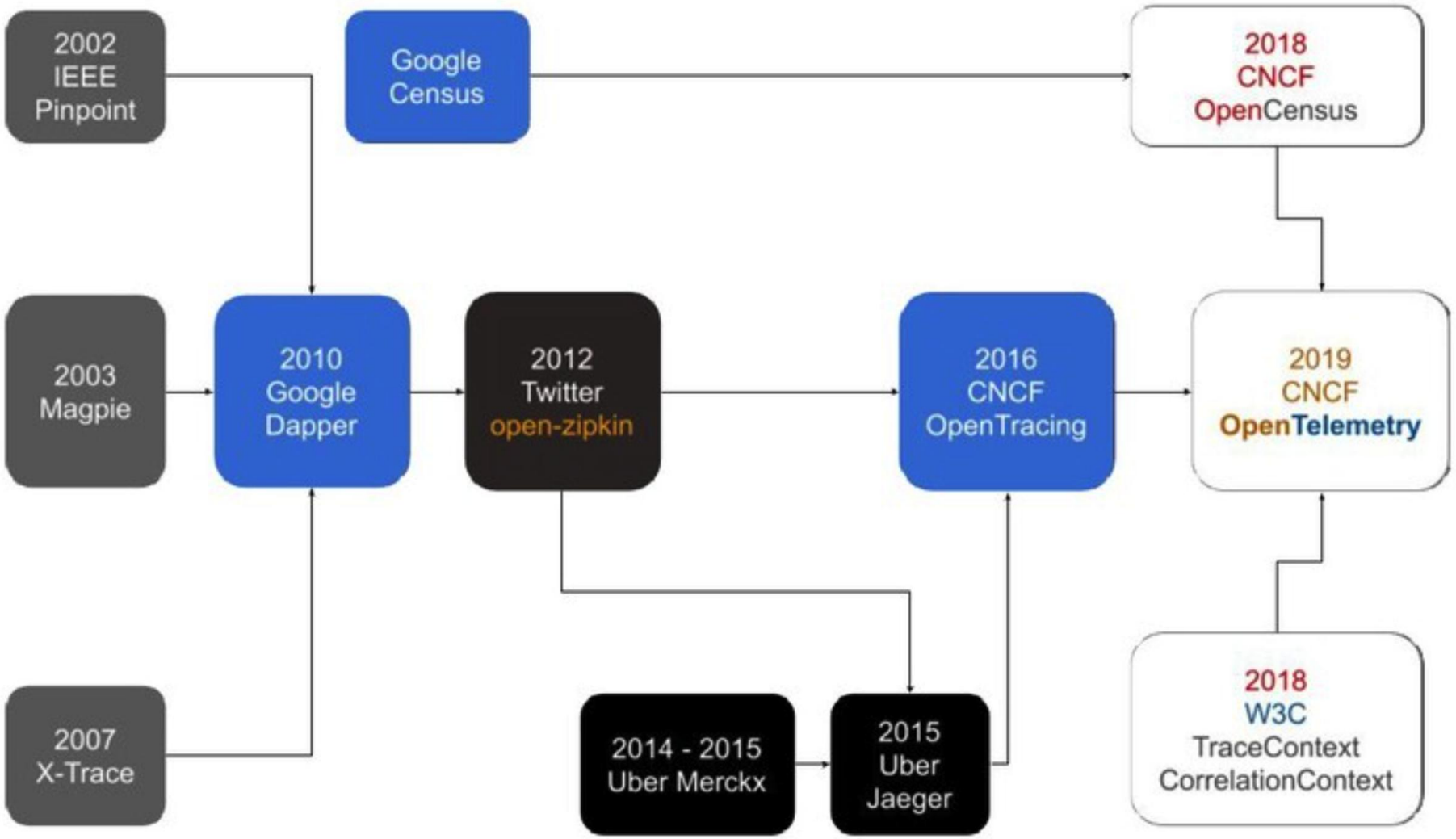
分布式系统的可观测性

Logging+Tracing+Metric



参考阅读：https://blog.hufeifei.cn/2021/09/Distribution/grafana/

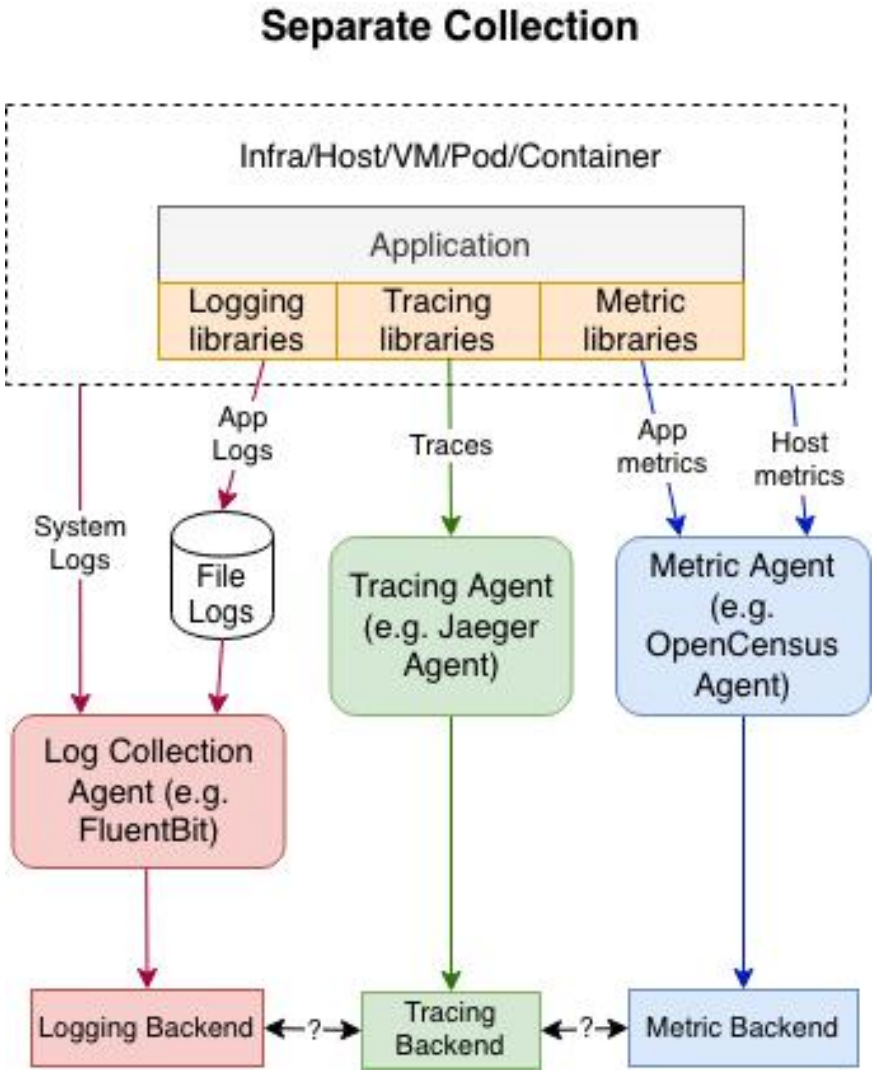
业内标准

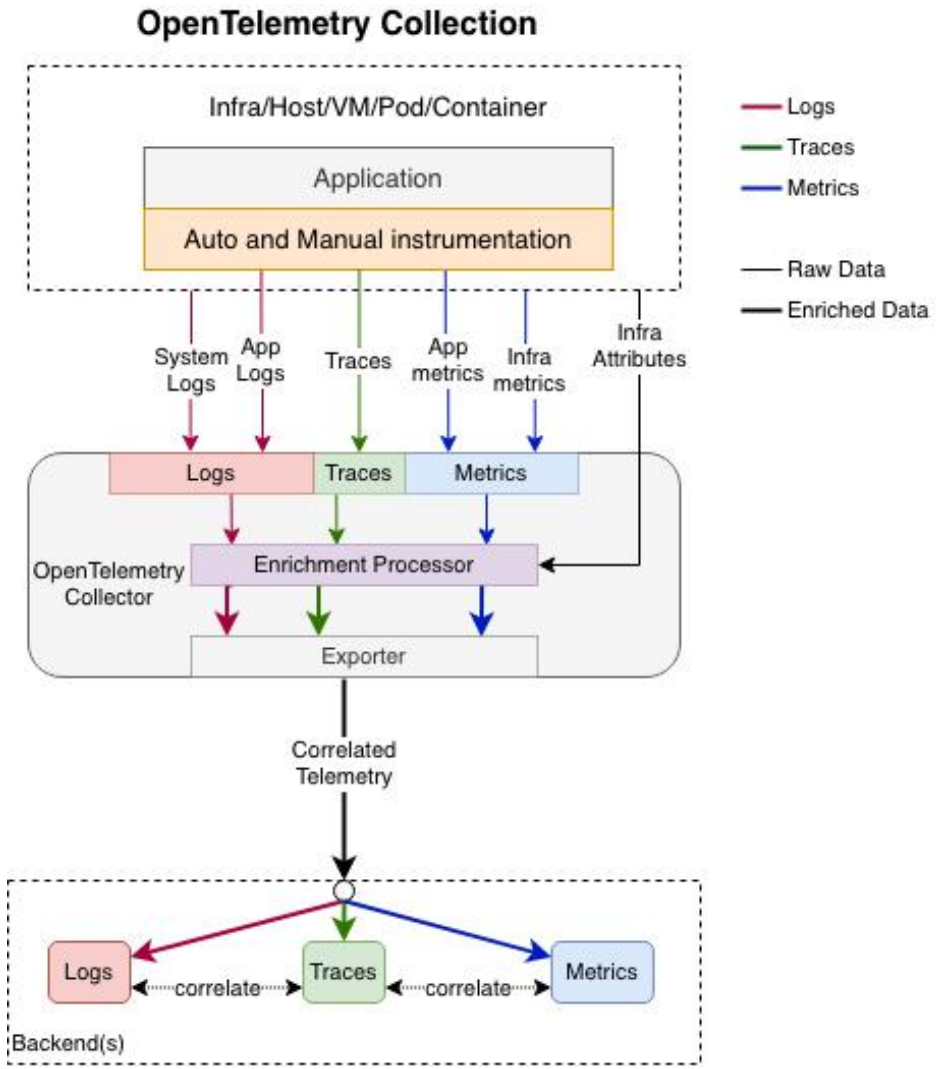

2002
IEEE
Pinpoint
Google
Census
2018
CNCF
OpenCensus
2003
Magpie
2010
Google
Dapper
2012
Twitter
open-zipkin
2016
CNCF
OpenTracing
2019
CNCF
OpenTelemetry
2007
X-Trace
2014 - 2015
Uber Merckx
2015
Uber
Jaeger
2018
W3C
TraceContext
CorrelationContext

业内标准

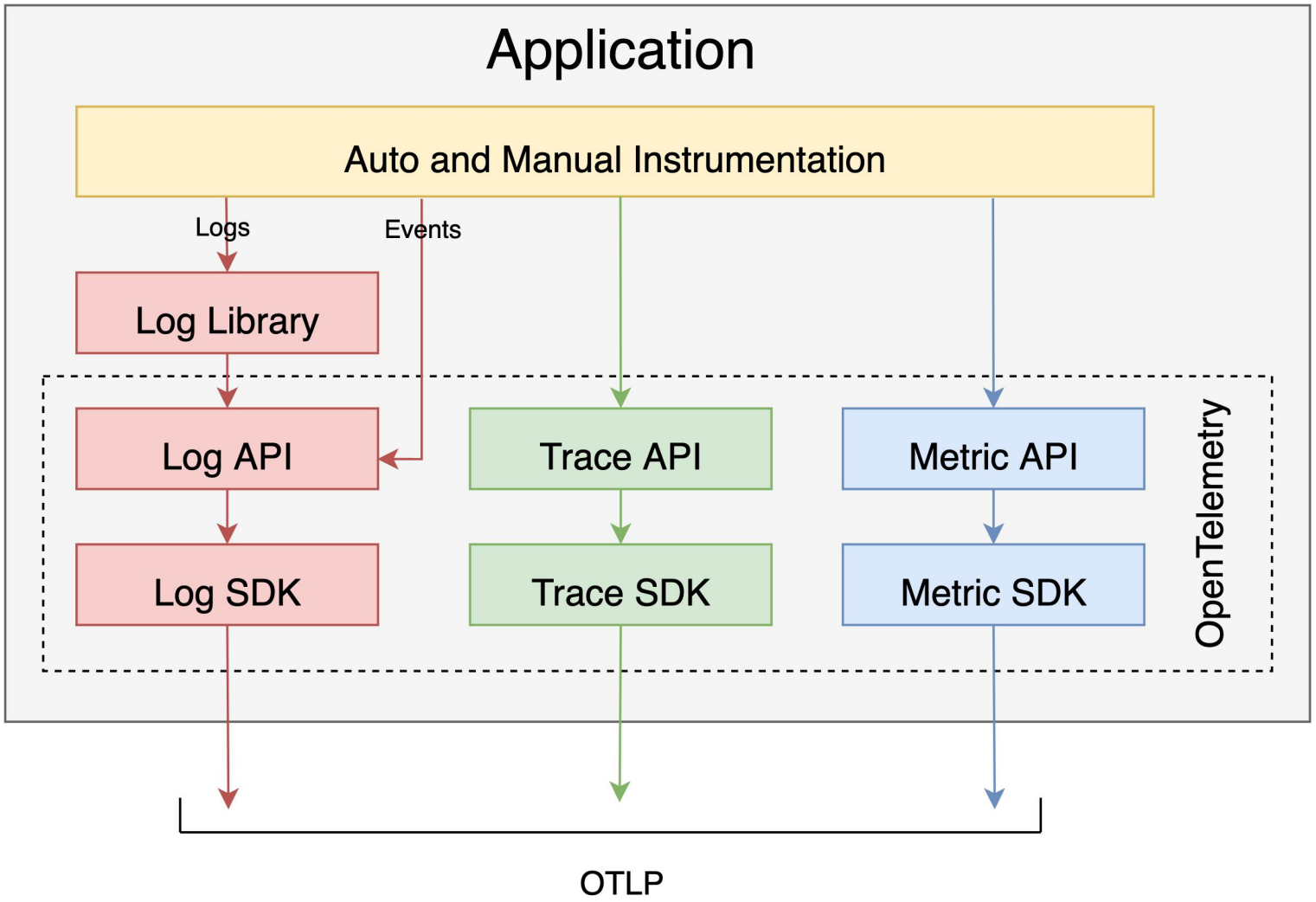
- https://github.com/openzipkin
- https://spring.io/projects/spring-cloud-sleuth
- https://github.com/jaegertracing
- https://github.com/opentracing
- https://openmetrics.io/
- https://github.com/census-instrumentation
- https://github.com/open-telemetry
- https://github.com/topics/distributed-tracing
- https://github.com/topics/metrics
- https://github.com/topics/logging

分布式系统观测性的未来：OpenTelemetry





OpenTelemetry架构设计



参考阅读：https://github.com/open-telemetry/opentelemetry-specification/blob/main/specification/logs/README.md

OpenTelemetry当前的进展

Status and Releases

The current status of the major functional components for OpenTelemetry Java is as follows:

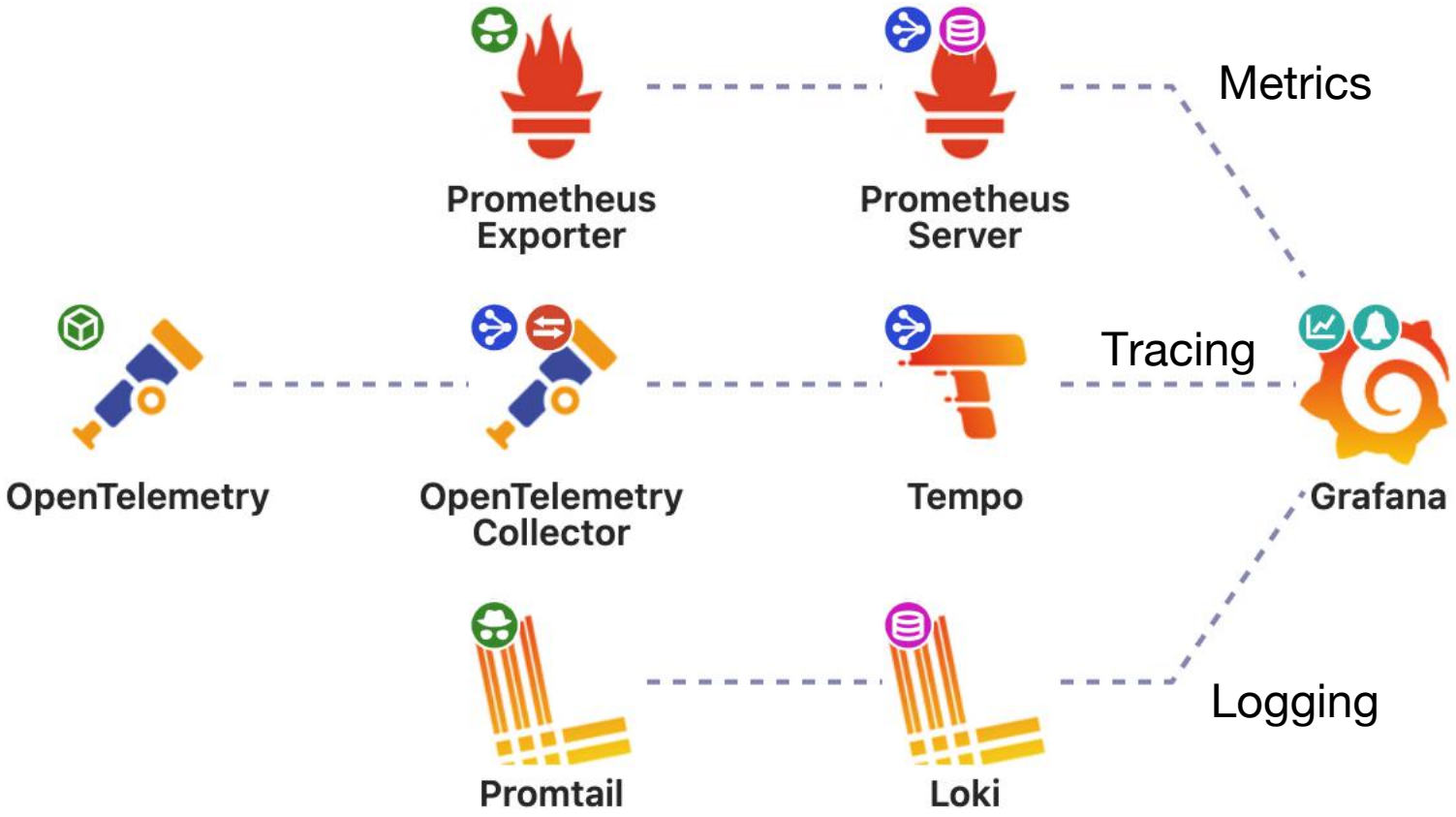
Traces	Metrics	Logs
Stable	Stable	Experimental

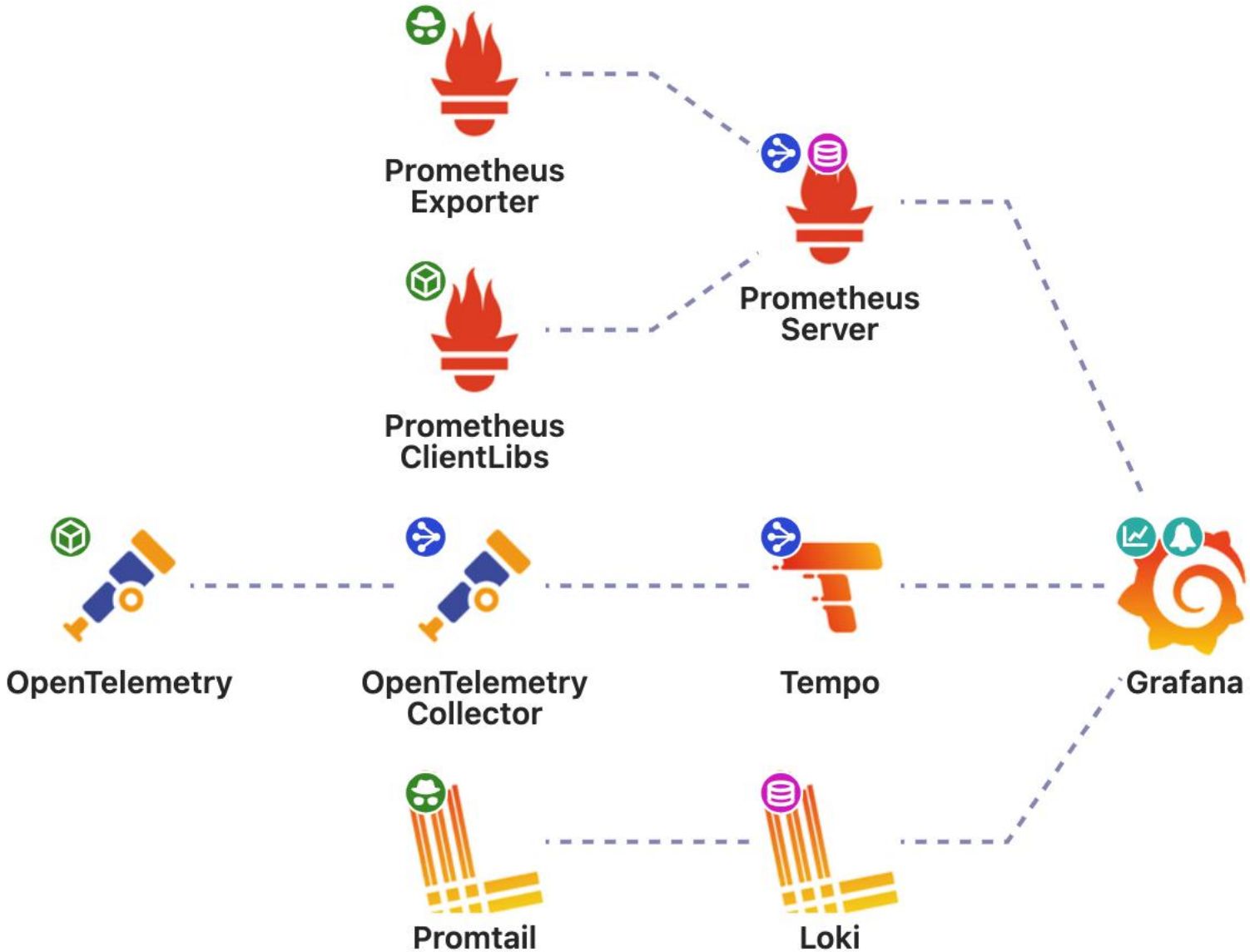
For releases, including the latest release, see Releases.

OpenTelemetry-metrics的生态还没有Promethus成熟
https://github.com/prometheus/client_java
https://github.com/open-telemetry/opentelemetry-java-contrib

参考阅读：https://opentelemetry.io/status/

分布式系统观测性技术选型

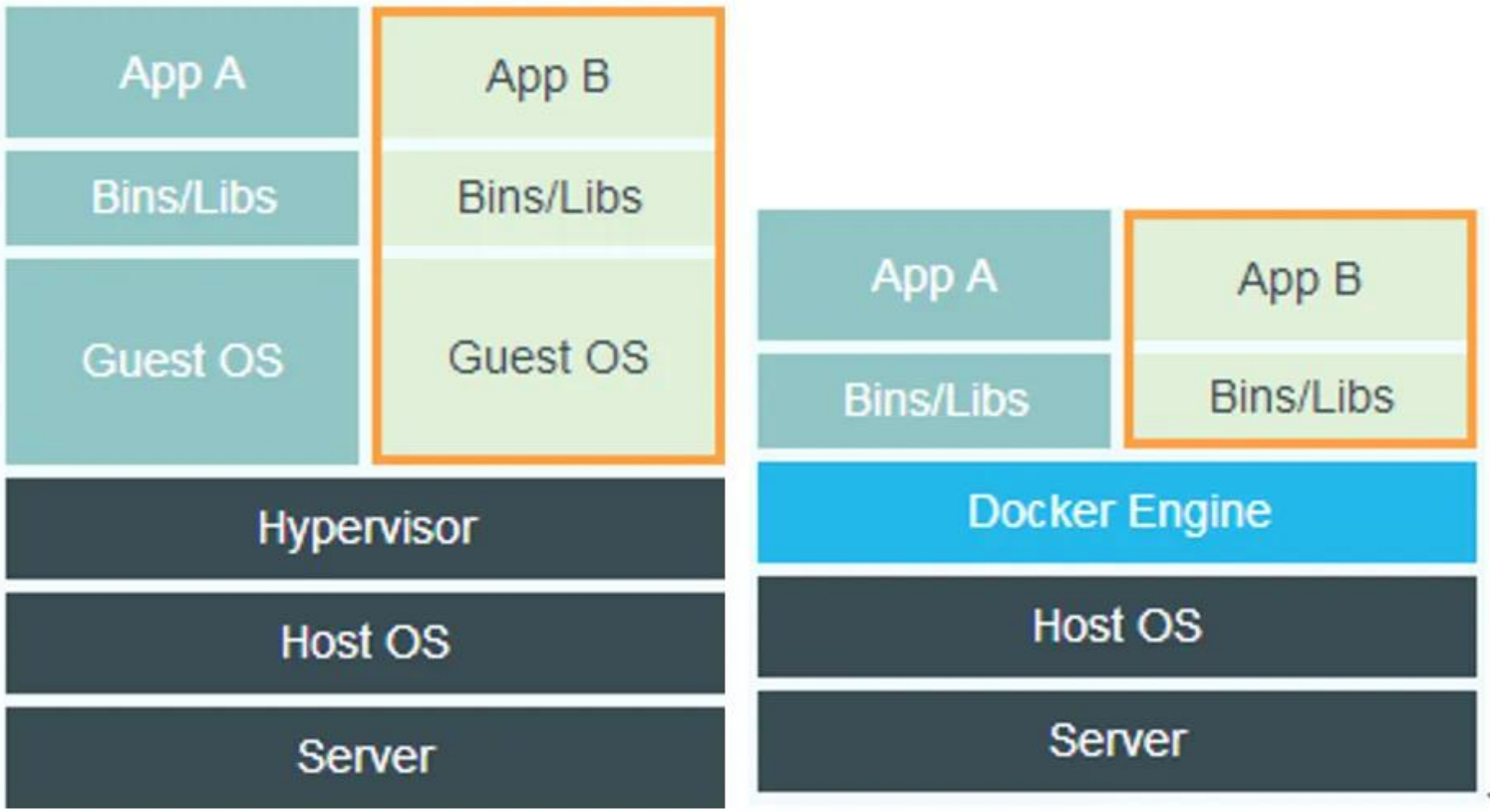



Prometheus Exporter
Prometheus ClientLibs
Prometheus Server
OpenTelemetry
OpenTelemetry Collector
Tempo
Grafana
Promtail
Loki

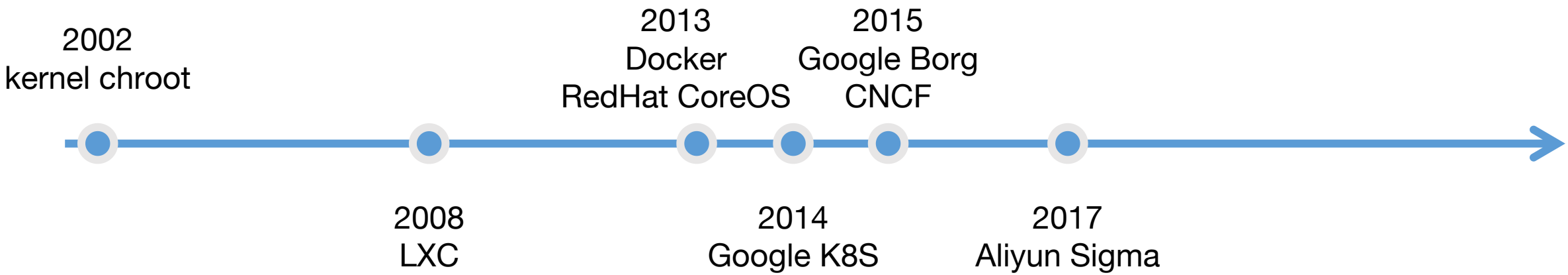
微服务与容器化

容器化出现的背景：部署难、运维复杂

从虚拟机到容器


App A
App B
Bins/Libs
Bins/Libs
Guest OS
Guest OS
Hypervisor
Host OS
Server
App A
App B
Bins/Libs
Bins/Libs
Docker Engine
Host OS
Server

容器化发展历程


2002
kernel chroot
2008
LXC
2013
Docker
RedHat CoreOS
2014
Google K8S
2015
Google Borg
CNCF
2017
Aliyun Sigma

Docker容器

Docker的目标：

Build Once, Run Anywhere

Docker实现原理：

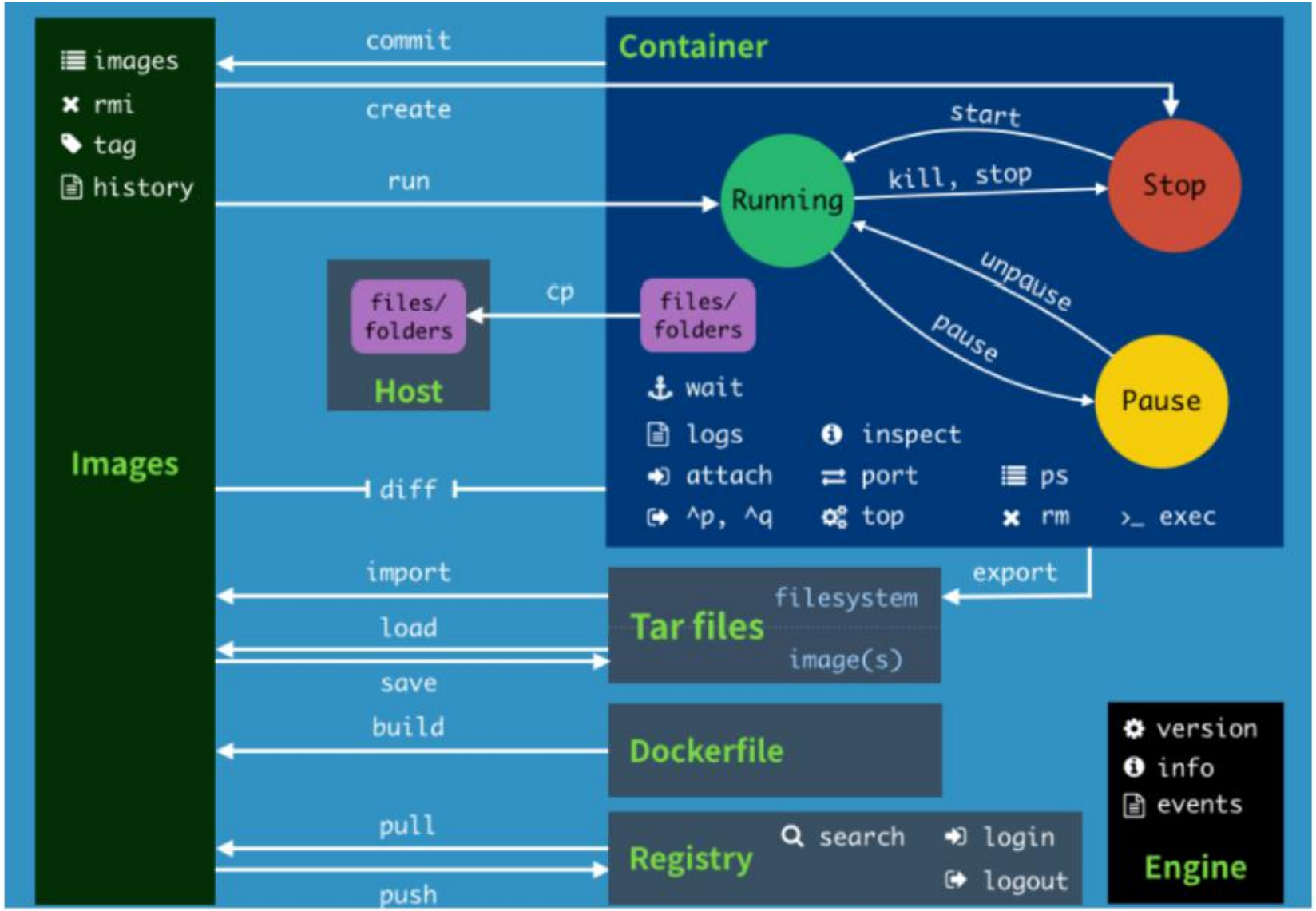
https://man7.org/linux/man-pages/man7/namespaces.7.html

https://man7.org/linux/man-pages/man7/cgroups.7.html

https://learn.microsoft.com/en-us/virtualization/windowscontainers/

参考阅读：https://posts.hufeifei.cn/backend/docker-core-technologies/

docker相关命令概览



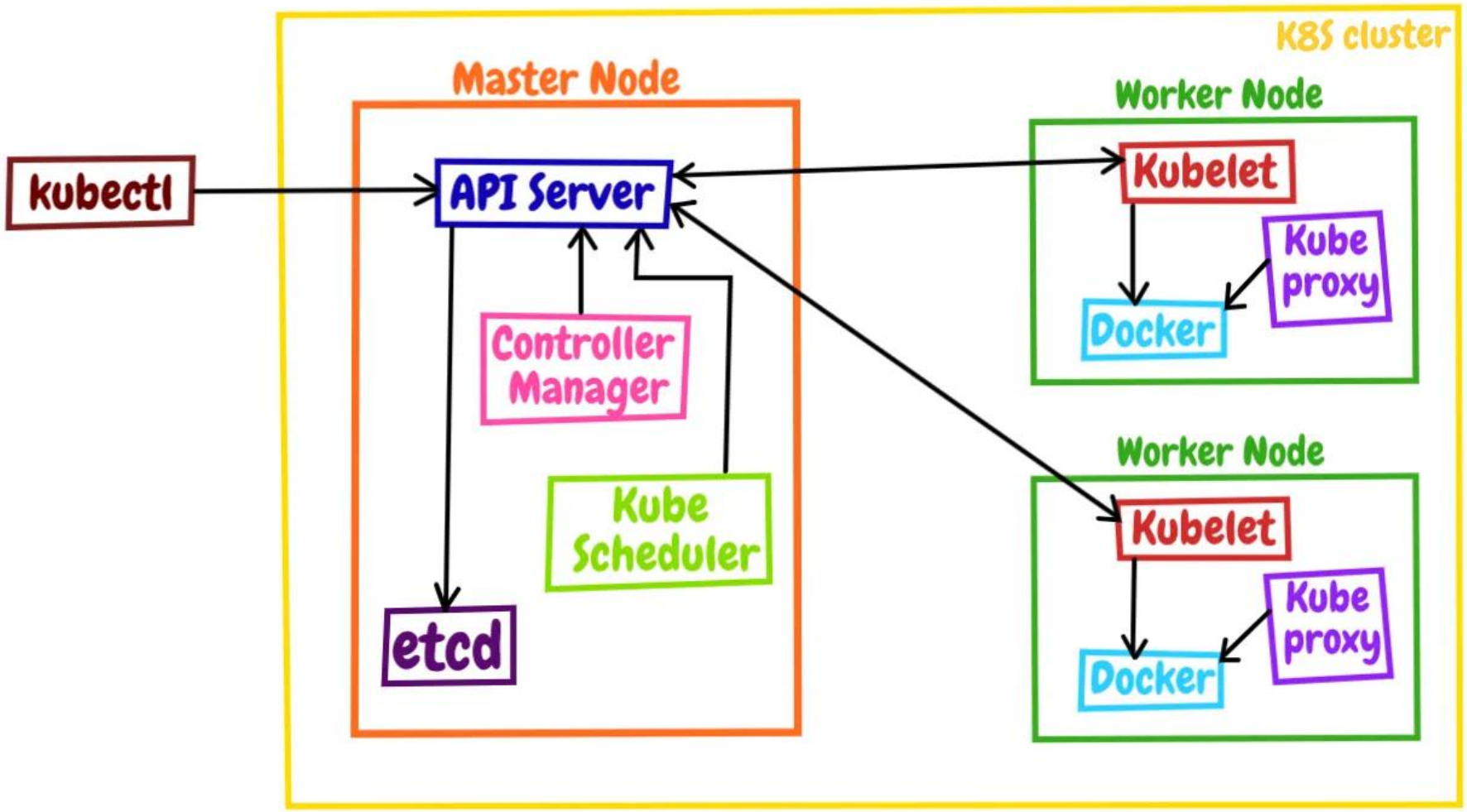
容器的集群化管理：K8S (google 2014)

容器集群化后面临的问题：

1、怎么调度

2、怎么通信

3、怎么存储

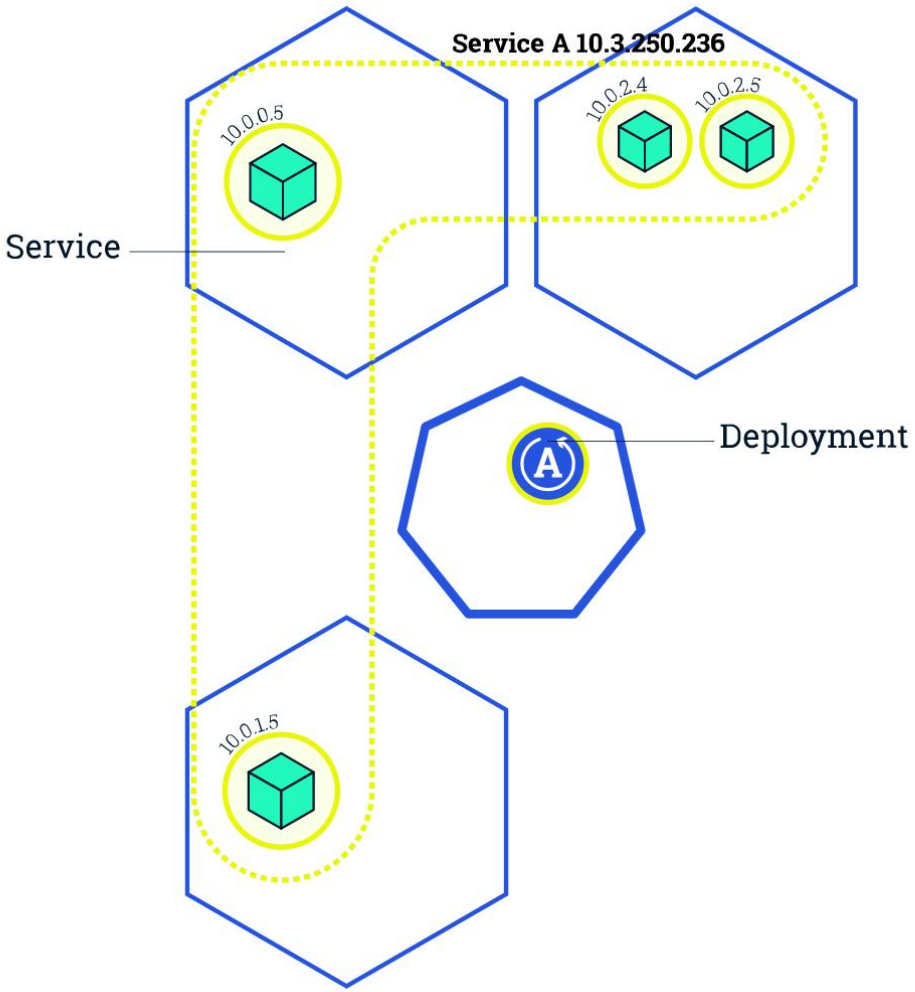


官方文档：https://kubernetes.io/docs/concepts/overview/components/

最小的调度单元：Pod

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: nginx
spec:
  containers:
  - name: nginx
    image: nginx:1.14.2
    ports:
    - containerPort: 80
```

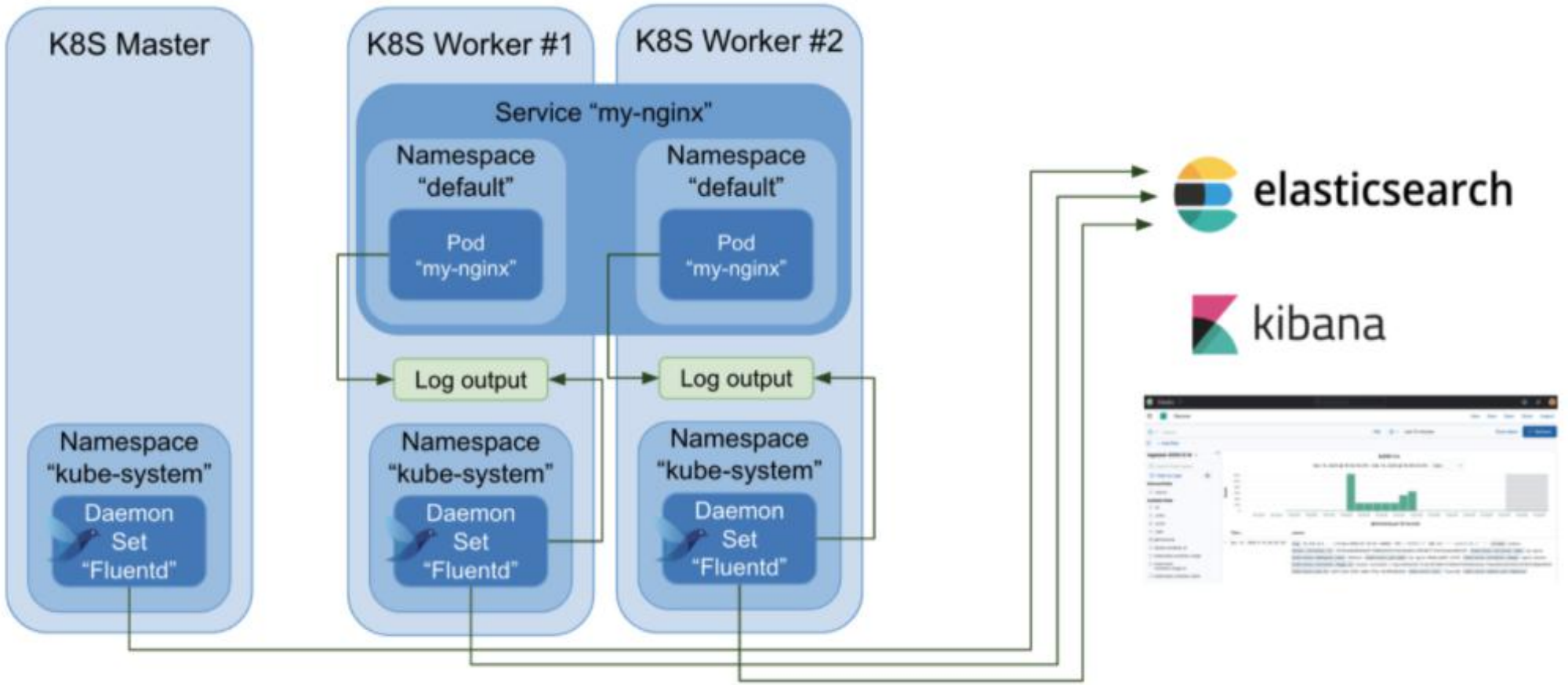
应用副本：ReplicaSet & 应用部署管理：Deployments




```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  labels:
    app: nginx
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.14.2
        ports:
        - containerPort: 80
```

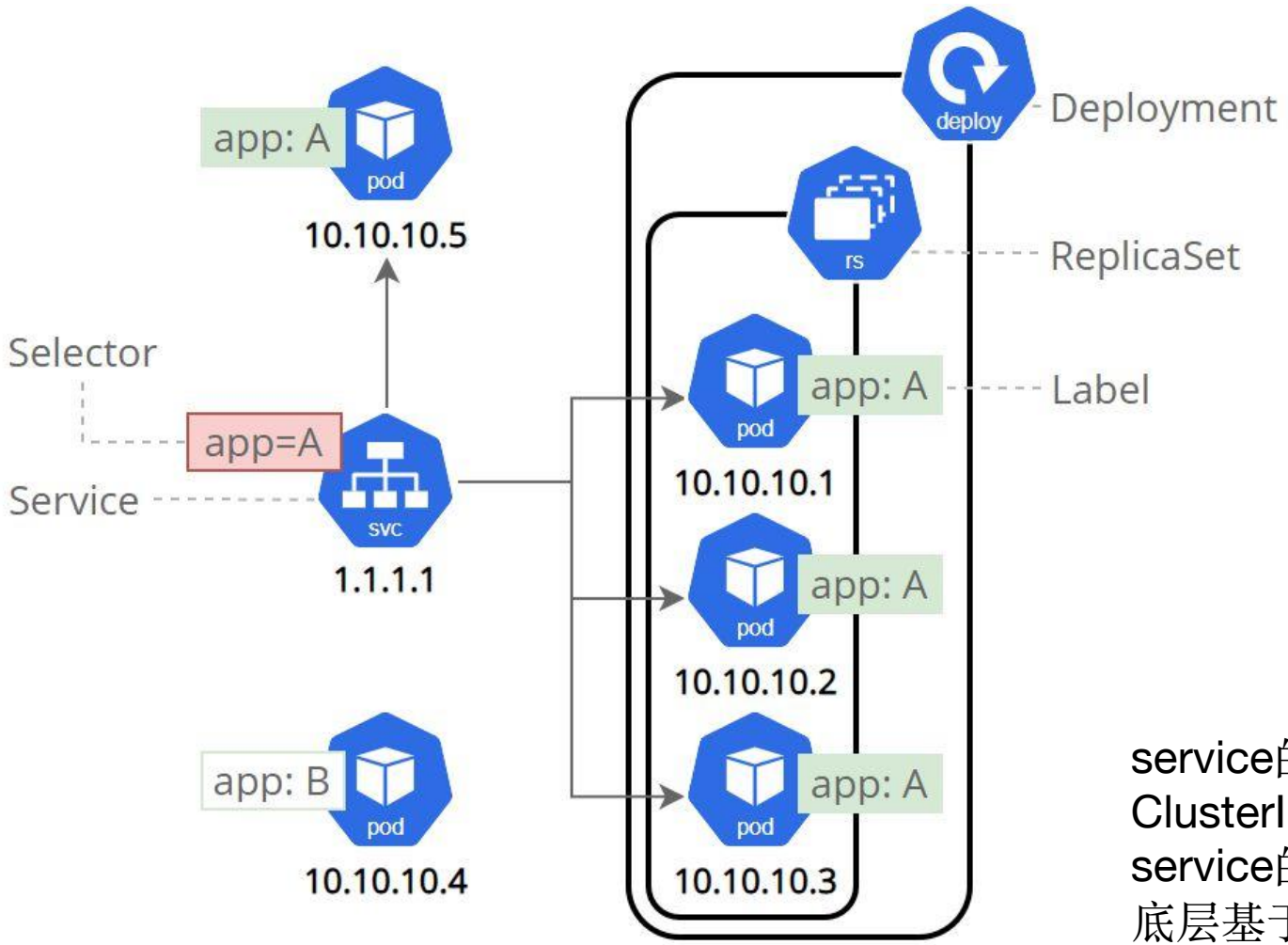
官方文档：https://kubernetes.io/docs/concepts/workloads/controllers/deployment/

每个节点部署一个Pod：DaemonSet



参考阅读：https://fluentbit.io/blog/2021/01/25/logging-fluentd-with-kubernetes/
官方文档：https://kubernetes.io/docs/concepts/workloads/controllers/daemonset/

怎么暴露服务：Service

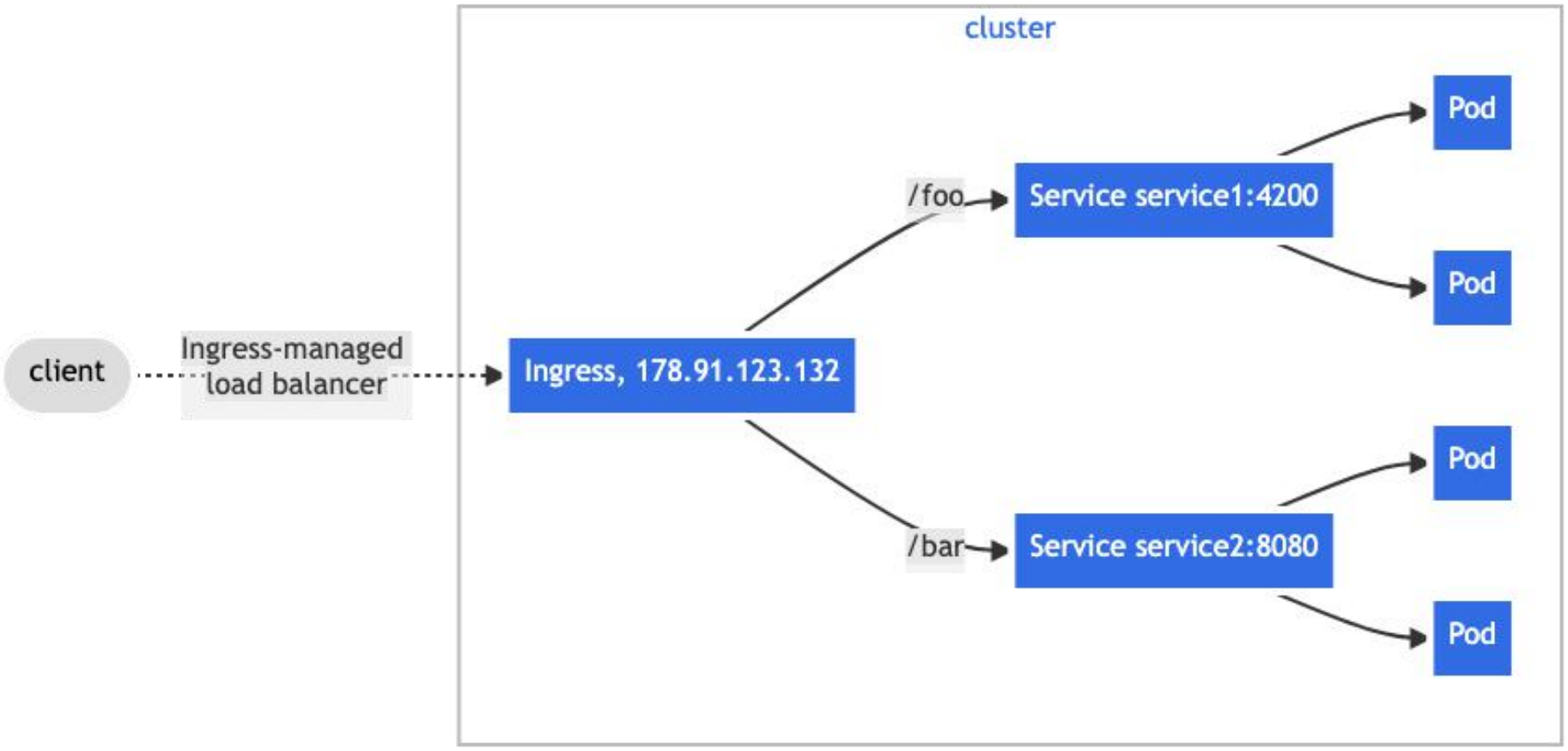



```yaml
apiVersion: v1
kind: Service
metadata:
  name: my-service
spec:
  selector:
    app.kubernetes.io/name: MyApp
  ports:
    - protocol: TCP
      port: 80
      targetPort: 9376
```

service的四种类型：
ClusterIP、NodePort、ExternalName、LoadBalancer
service的局限性：
底层基于iptables或ipvs实现，工作在tcp/ip层，只能实现四层代理。无法实现应用层的负载均衡。

官方文档：https://kubernetes.io/docs/concepts/services-networking/service/

暴露到外网：Ingress



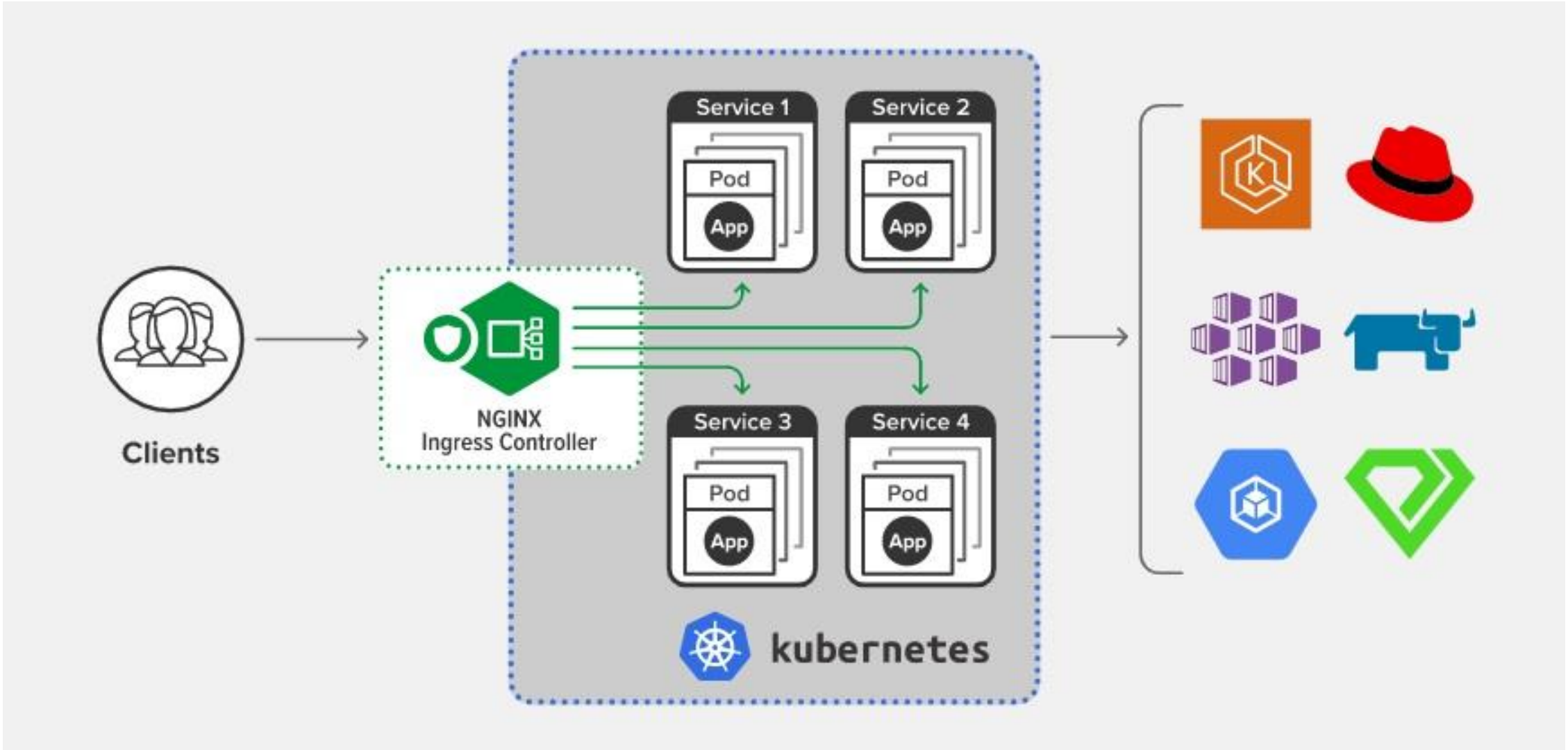

```yaml
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: simple-fanout-example
spec:
  rules:
  - host: foo.bar.com
    http:
      paths:
      - path: /foo
        pathType: Prefix
        backend:
          service:
            name: service1
            port:
              number: 4200
      - path: /bar
        pathType: Prefix
        backend:
          service:
            name: service2
            port:
              number: 8080
```

官方文档：https://kubernetes.io/docs/concepts/services-networking/ingress/

管理Ingress： IngressController

K8S没有提供IngressController实现

- AKS Application Gateway Ingress Controller is an ingress controller that configures the Azure Application Gateway.
- Ambassador API Gateway is an Envoy-based ingress controller.
- Apache APISIX ingress controller is an Apache APISIX-based ingress controller.
- Avi Kubernetes Operator provides L4-L7 load-balancing using VMware NSX Advanced Load Balancer.
- BFE Ingress Controller is a BFE-based ingress controller.
- The Citrix ingress controller works with Citrix Application Delivery Controller.
- Contour is an Envoy based ingress controller.
- EnRoute is an Envoy based API gateway that can run as an ingress controller.
- Easegress IngressController is an Easegress based API gateway that can run as an ingress controller.
- F5 BIG-IP Container Ingress Services for Kubernetes lets you use an Ingress to configure F5 BIG-IP virtual servers.
- Gloo is an open-source ingress controller based on Envoy, which offers API gateway functionality.
- HAProxy Ingress is an ingress controller for HAProxy.
- The HAProxy Ingress Controller for Kubernetes is also an ingress controller for HAProxy.
- Istio Ingress is an Istio based ingress controller.
- The Kong Ingress Controller for Kubernetes is an ingress controller driving Kong Gateway.
- Kusk Gateway is an OpenAPI-driven ingress controller based on Envoy.
- The NGINX Ingress Controller for Kubernetes works with the NGINX webserver (as a proxy).
- The Pomerium Ingress Controller is based on Pomerium, which offers context-aware access policy.
- Skipper HTTP router and reverse proxy for service composition, including use cases like Kubernetes Ingress, designed as a library to build your custom proxy.
- The Traefik Kubernetes Ingress provider is an ingress controller for the Traefik proxy.
- Tyk Operator extends Ingress with Custom Resources to bring API Management capabilities to Ingress. Tyk Operator works with the Open Source Tyk Gateway & Tyk Cloud control plane.
- Voyager is an ingress controller for HAProxy.



官方文档：https://kubernetes.github.io/ingress-nginx/

挂载配置文件：ConfigMap

定义配置

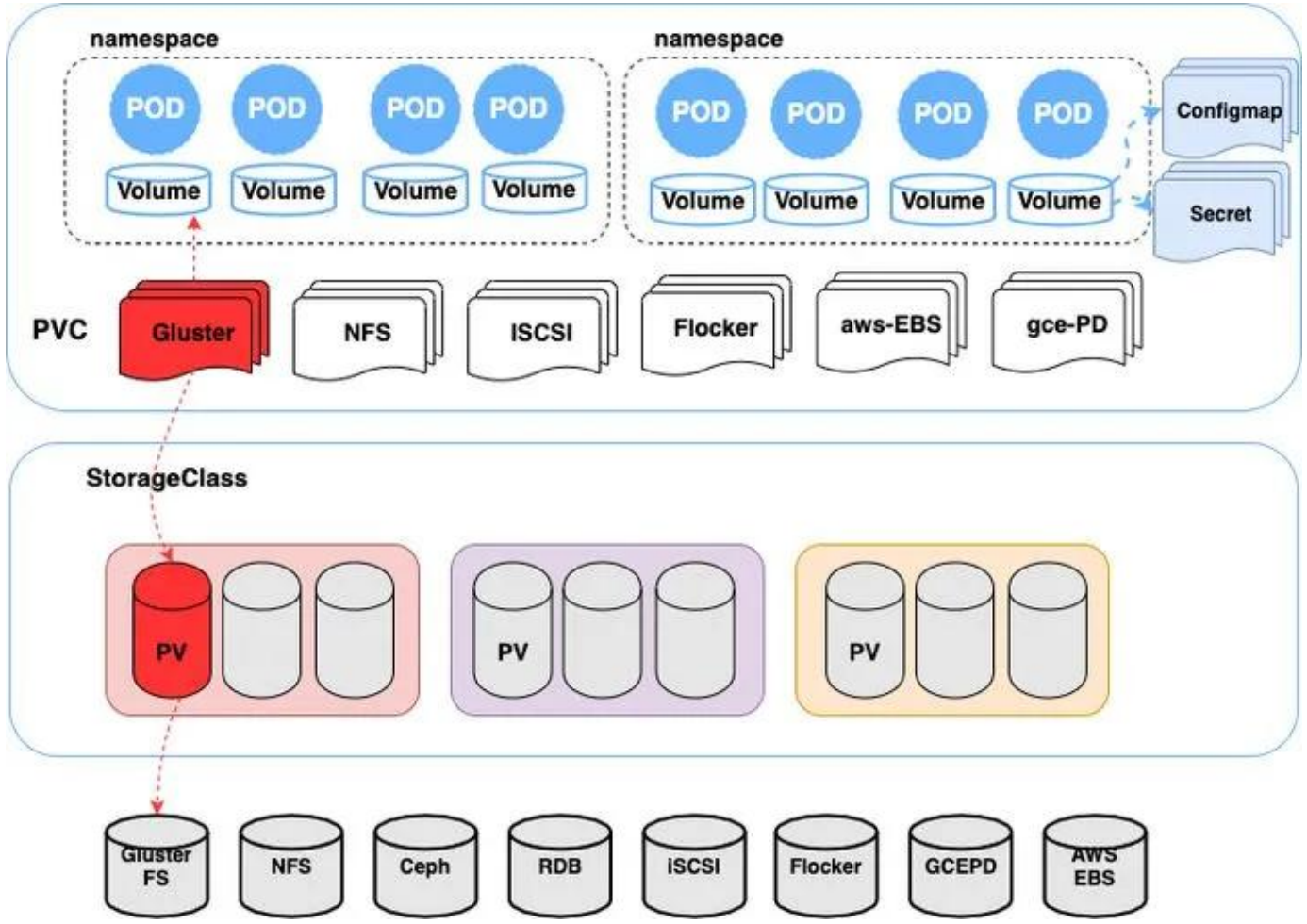
```yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: game-demo
data:
  # property-like keys; each key maps to a simple value
  player_initial_lives: "3"
  ui_properties_file_name: "user-interface.properties"

  # file-like keys
  game.properties: |
    enemy.types=aliens,monsters
    player.maximum-lives=5
  user-interface.properties: |
    color.good=purple
    color.bad=yellow
    allow.textmode=true
```

使用配置

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: configmap-demo-pod
spec:
  containers:
    - name: demo
      image: alpine
      command: ["sleep", "3600"]
      env:
        # Define the environment variable
        - name: PLAYER_INITIAL_LIVES # Notice that the case is different here
                                     # from the key name in the ConfigMap.
          valueFrom:
            configMapKeyRef:
              name: game-demo           # The ConfigMap this value comes from.
              key: player_initial_lives # The key to fetch.
        - name: UI_PROPERTIES_FILE_NAME
          valueFrom:
            configMapKeyRef:
              name: game-demo
              key: ui_properties_file_name
      volumeMounts:
      - name: config
        mountPath: "/config"
        readOnly: true
  volumes:
    # You set volumes at the Pod level, then mount them into containers inside that Pod
    - name: config
      configMap:
        # Provide the name of the ConfigMap you want to mount.
        name: game-demo
        # An array of keys from the ConfigMap to create as files
        items:
        - key: "game.properties"
          path: "game.properties"
        - key: "user-interface.properties"
          path: "user-interface.properties"
```

怎么存储：PVC & StorageClass




```yaml
apiVersion: v1
kind: Pod
metadata:
  name: task-pv-pod
spec:
  nodeSelector:
    kubernetes.io/hostname: kube-01
  volumes:
    - name: task-pv-storage
      persistentVolumeClaim:
        claimName: task-pv-claim
  containers:
    - name: task-pv-container
      image: nginx
      ports:
        - containerPort: 80
          name: "http-server"
      volumeMounts:
        - mountPath: "/usr/share/nginx/html"
          name: task-pv-storage
```

K8S的问题与局限性

1、水平扩展性：kubernates适合10～5000个节点的集群

与几万个节点的Apache Mesos、50,000 节点的微软 YARN 集群相比，K8S集群规模差了一个数量级

2、批处理的调度

批处理任务和流式任务等工作负载的运行从 Kubernetes 诞生第一天起到今天都不是它的强项，大多数的公司都会使用 Kubernetes 运行在线服务处理用户请求，用 Yarn 管理的集群运行批处理的负载。

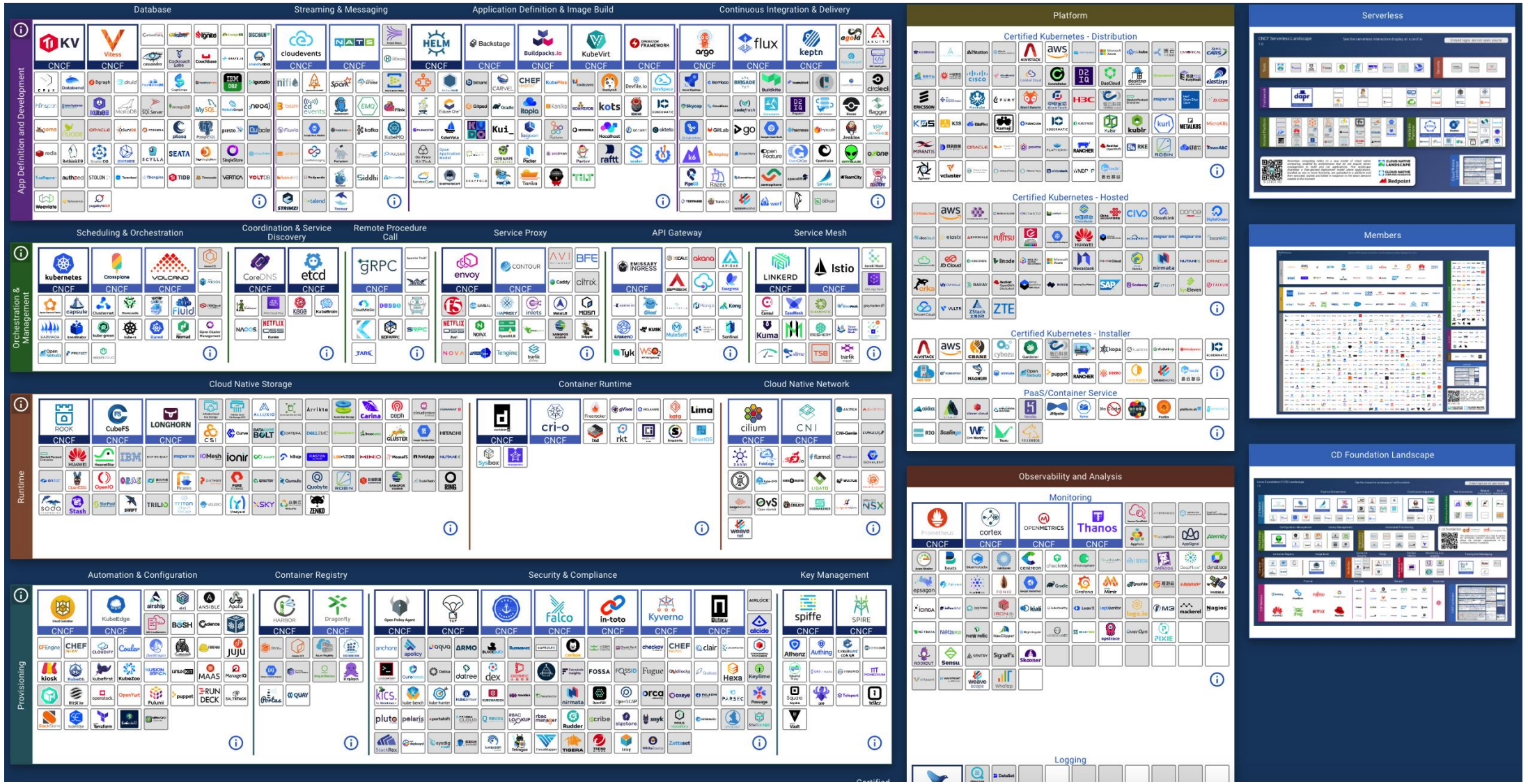
在 Kubernetes 调度器引入调度框架之前，所有的 Pod 在调度器看来是没有任何关联的，也就是说对于大集群的分布式计算的调度，k8s无法做到。这需要另外实现，比如：

https://github.com/kubernetes-sigs/kube-batch

https://github.com/GoogleCloudPlatform/spark-on-k8s-operator

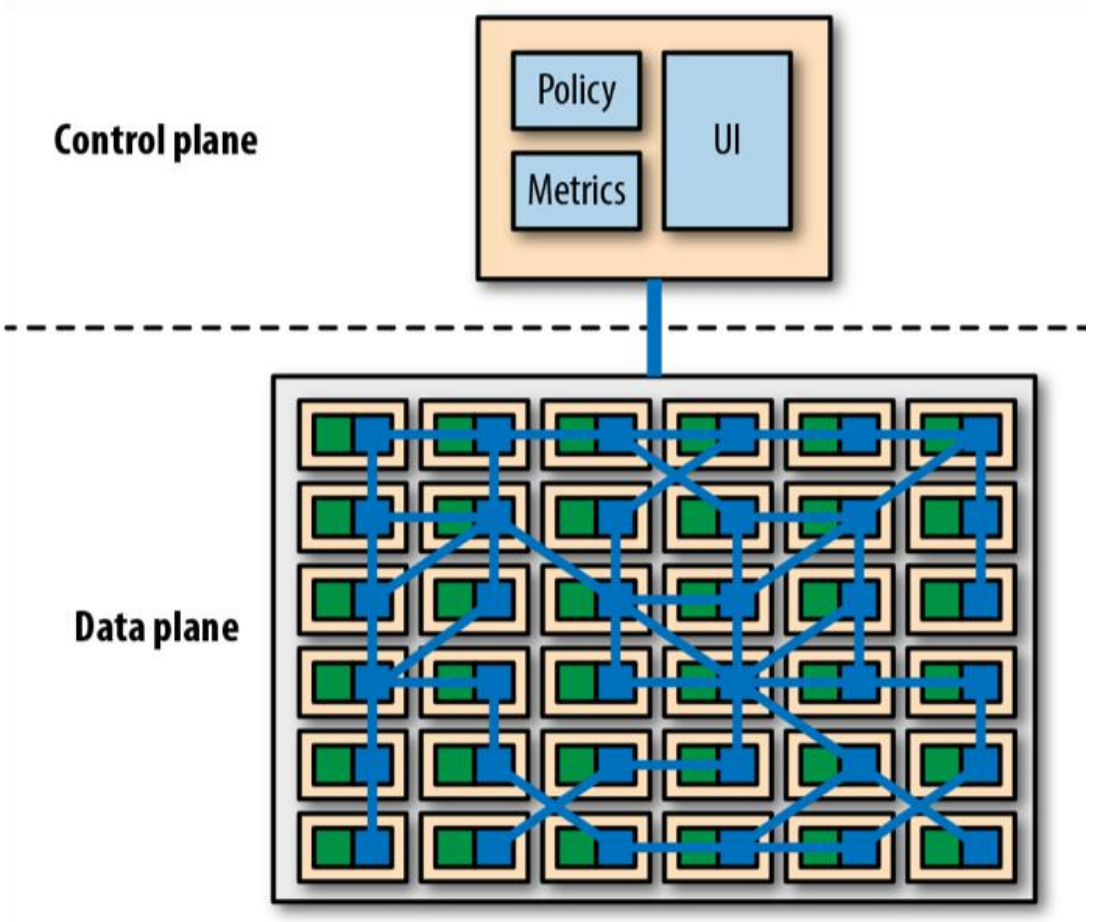
参考：https://kubernetes.io/zh-cn/docs/setup/best-practices/cluster-large/
https://posts.hufeifei.cn/backend/kuberentes-limitations/

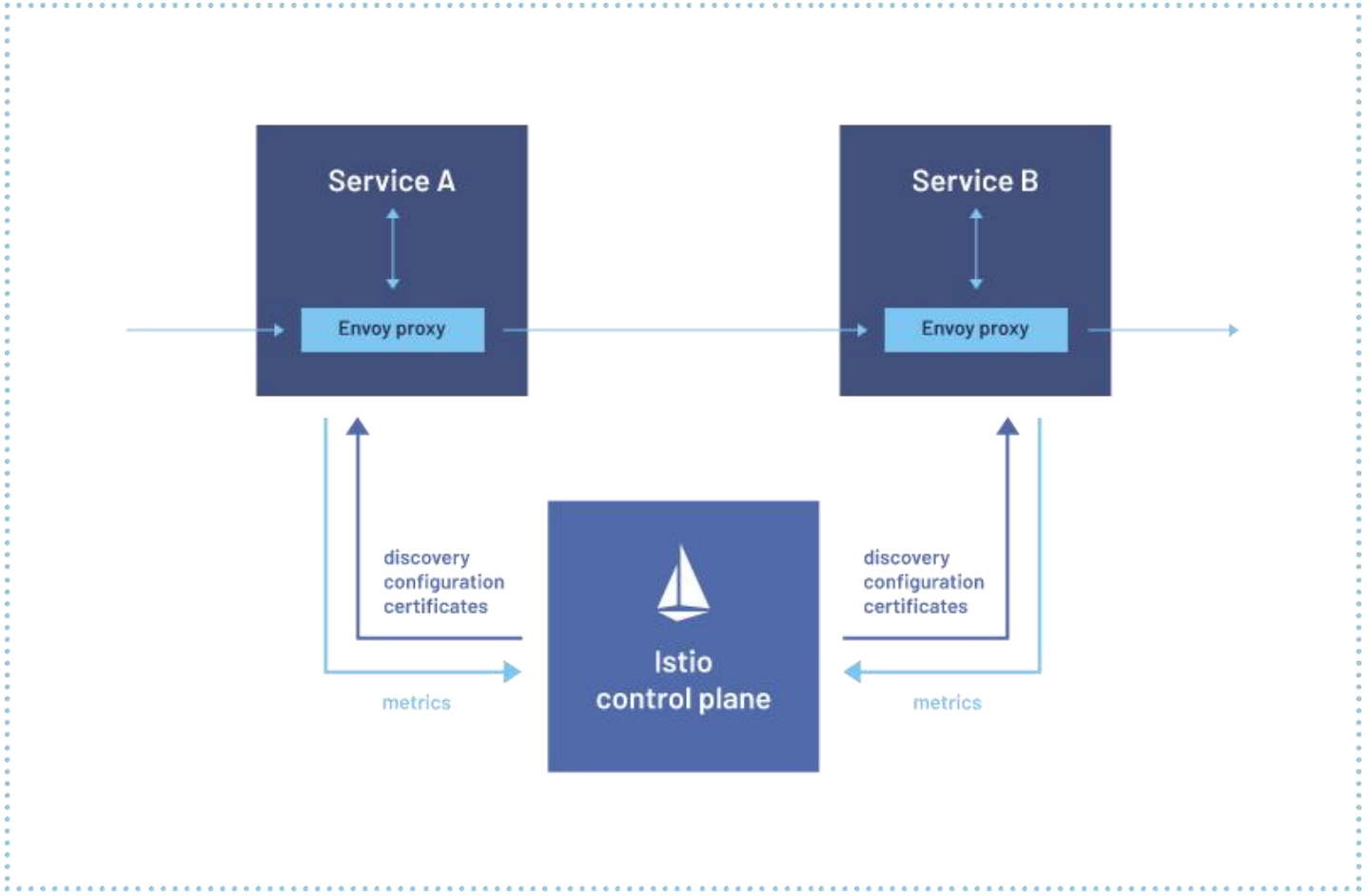
云原生的未来


Database
Streaming & Messaging
Application Definition & Image Build
Continuous Integration & Delivery
App Definition and Development
Scheduling & Orchestration
Coordination & Service Discovery
Remote Procedure Call
Service Proxy
API Gateway
Service Mesh
Orchestration & Management
Cloud Native Storage
Container Runtime
Cloud Native Network
Runtime
Automation & Configuration
Container Registry
Security & Compliance
Key Management
Provisioning
Platform
Certified Kubernetes - Distribution
Certified Kubernetes - Hosted
Certified Kubernetes - Installer
PaaS/Container Service
Observability and Analysis
Monitoring
Logging
Serverless
Members
CD Foundation Landscape

云原生的未来-ServiceMesh

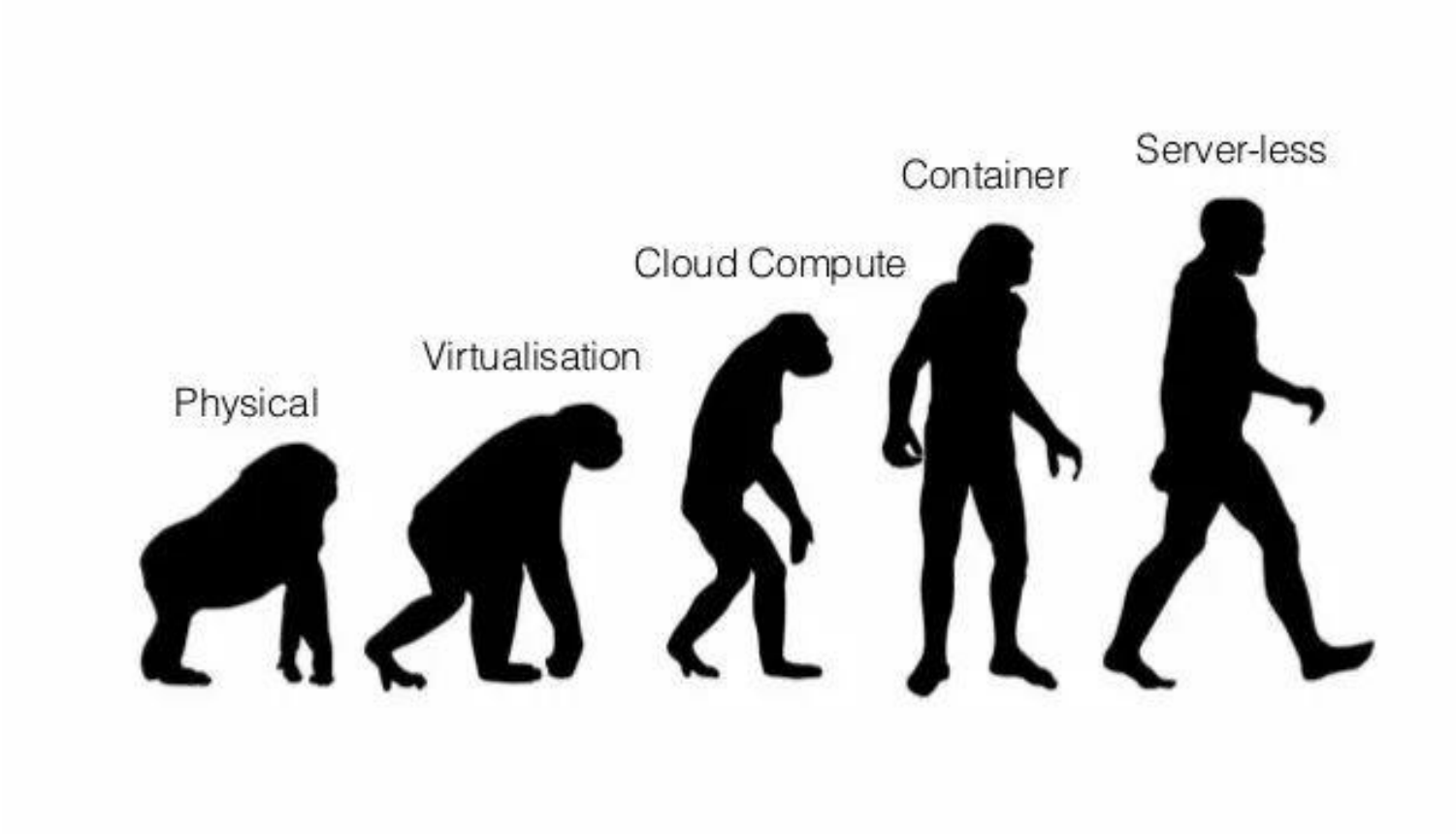
https://github.com/topics/service-mesh





参考阅读：https://www.baeldung.com/ops/istio-service-mesh

云原生的未来-Serverless云函数



https://spring.io/projects/spring-cloud-function

云原生的未来-Serverless云函数

Java在云原生场景下面临的挑战

1、SpringBoot启动时间长，在Serverless的场景下劣势明显

2、为了防止伸缩扩容导致FullGC，通常会把Java堆内存设成固定值，这也导致了JVM占用内存较大。

3、由于JIT即时编译，应用需要预热，才能达到最高性能

解决方案：Spring Native/quarkus & GraalVM / Golang