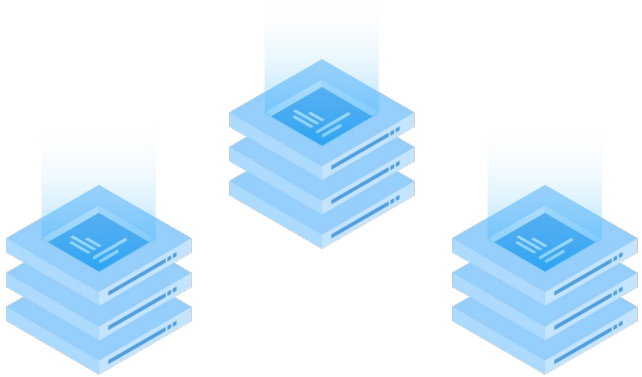
TiDB-Operator 设计与实现

Presented by Liang Yin

PingCAP TiDB

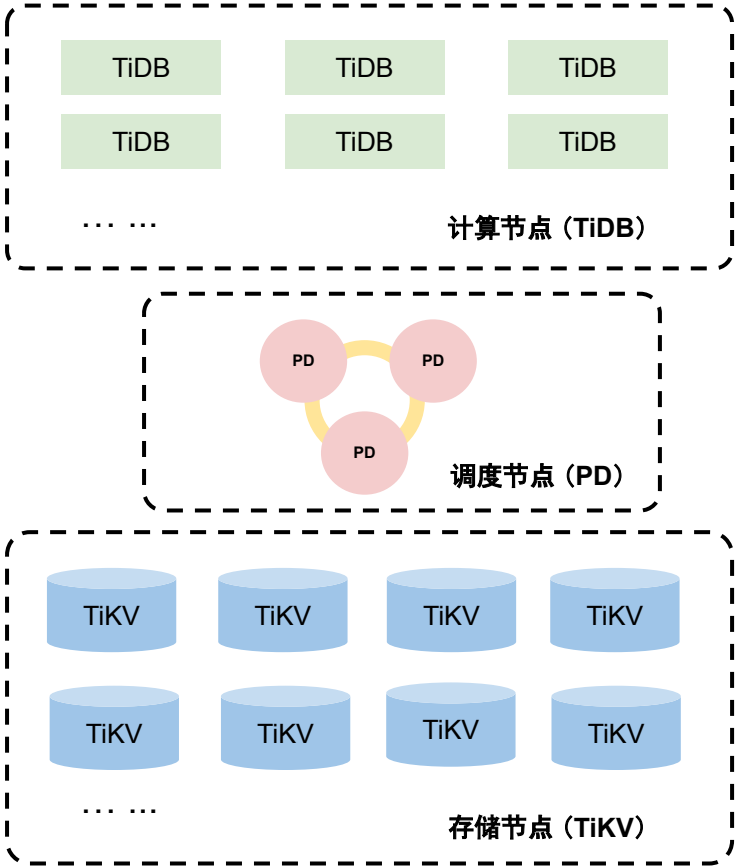


PingCAP

Part I - 设计目的

TiDB 核心组件

- TiDB
  - 计算密集型
  - 无状态
  - 横向扩展
- TiKV
  - 计算和 I/O 密集型
  - 有状态(本地盘存储)
  - 横向扩展
- PD
  - 对 CPU/MEM 要求低
  - 有状态



TiDB 传统运维

PD1

pd-server --initial=pd1 --data-dir=...

TiDB 传统运维

PD1 pd-server --initial=pd1 --data-dir=...

PD2 pd-server **--join**=pd1 --data-dir=...

PD3 pd-server **--join**=pd1,p2 --data-dir=...

TiDB 传统运维

PD1 pd-server --initial=pd1 --data-dir=...

PD2 pd-server --join=pd1 --data-dir=...

PD3 pd-server --join=pd1,p2 --data-dir=...

TiKV1

TiKV2

TiKV3

tikv-server --pd=pd1,pd2,pd3 --data-dir=...

PingCAP TiDB

TiDB 传统运维

PD1 pd-server --initial=pd1 --data-dir=...

PD2 pd-server --join=pd1 --data-dir=...

PD3 pd-server --join=pd1,p2 --data-dir=...

TiKV1
TiKV2
TiKV3
tikv-server --pd=pd1,pd2,pd3 --data-dir=...

TiDB1
TiDB2
tidb-server --pd=pd1,pd2,pd3

TiDB 传统运维

PD1 pd-server --initial=pd1 --data-dir=...

PD2 pd-server --join=pd1 --data-dir=...

PD3 pd-server --join=pd1,p2 --data-dir=...

PD4 pd-server **--join**=pd1,pd2,pd3 --data-dir=...

PD5 pd-server **--join**=pd1,pd2,pd3,pd4 --data-dir=...

TiKV1
TiKV2
TiKV3
TiKV4
TiKV5

tikv-server --pd=pd1,pd2,pd3 --data-dir=...

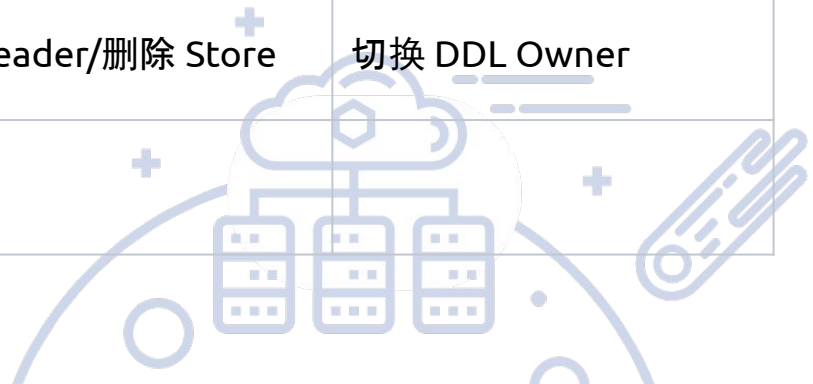
TiDB1
TiDB2
TiDB3

tidb-server --pd=pd1,pd2,pd3

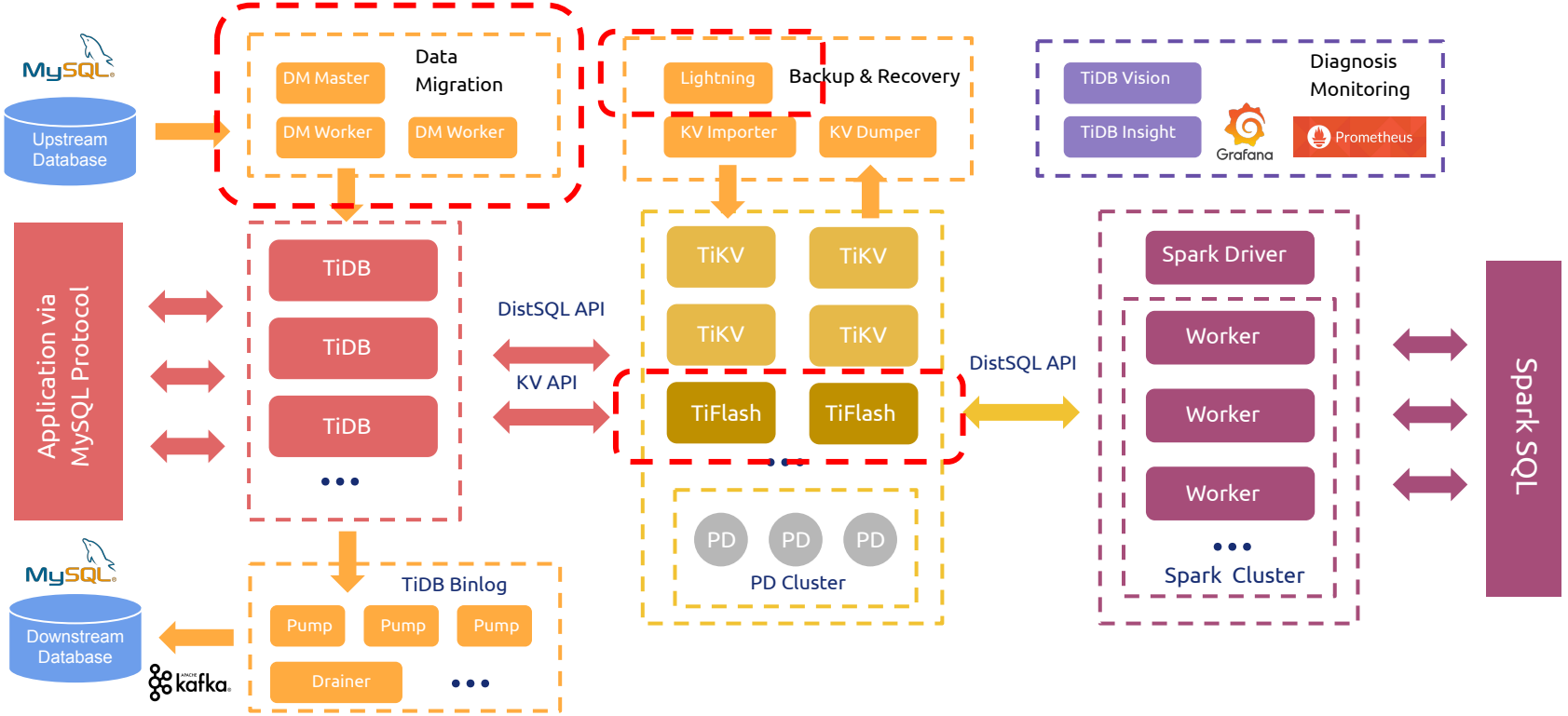
PingCAP TiDB

TiDB 传统运维

操作	PD	TiKV	TiDB
持久化存储	是	是	否
滚动更新有顺序	1	2	3
每个节点启动参数	不同	相同	相同
节点重启/下线	切换 Leader/删除 Member	驱逐 Leader/删除 Store	切换 DDL Owner
自动故障转移			



TiDB 周边工具


MySQL
Upstream Database
DM Master
DM Worker
DM Worker
Data Migration
Lightning
KV Importer
KV Dumper
Backup & Recovery
TiDB Vision
TiDB Insight
Grafana
Prometheus
Diagnosis Monitoring
Application via MySQL Protocol
TiDB
TiDB
TiDB
DistSQL API
KV API
TiKV
TiKV
TiKV
TiKV
TiFlash
TiFlash
PD
PD
PD
PD Cluster
DistSQL API
Spark Driver
Worker
Worker
Worker
Spark Cluster
Spark SQL
MySQL
Downstream Database
kafka
TiDB Binlog
Pump
Pump
Pump
Drainer

TiDB-Operator 的核心目标

- 核心目标是解决如何在大规模集群中更有效地管理多套 TiDB 数据库实例，以及提高集群资源使用率的问题。核心功能包括：
  - 一键建库
  - 自动化运维
  - 资源隔离
  - 故障自治愈
  - 跨地域高可用

PingCAP

Part II - 设计与实现

Operator 模式

- API 对象是 “对意图的记录”
- Controller 通过控制循环 “实现意图”

自定义 API 对象　　自定义控制器　　Operator 模式

声明式 API

- **spec**: 期望状态
- **status**: 当前状态
- 控制器(Controller)不断与 ApiServer 进行交互(**控制循环**), 对比期望状态与实际状态, 并进行 “调谐” 操作: 驱动实际状态向期望状态转移

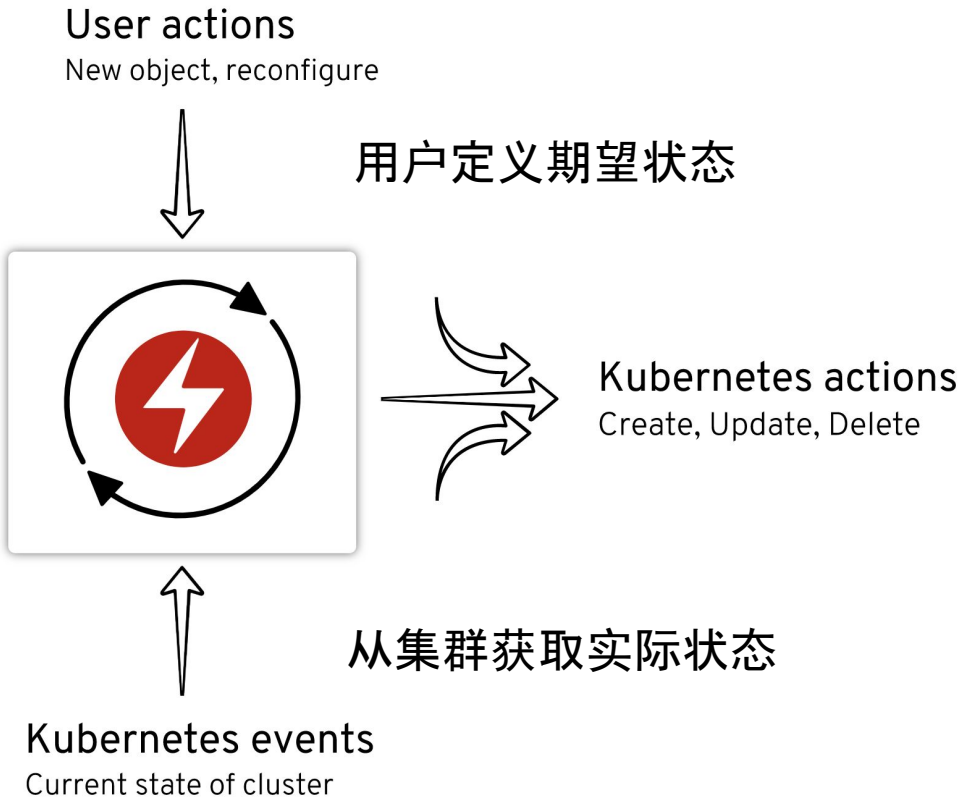
```
apiVersion: apps/v1
kind: ReplicaSet
spec:
  replicas: 3
  selector:
    matchLabels:
      app: web
  template:
    spec:
      containers:
      - image: nginx
        name: front-end
status:
  availableReplicas: 2
  replicas: 2
```

期望副本数: 3

```
loop {
 if actual != desired {
  reconcile()
 }
}
```

当前副本数: 2

自定义控制器



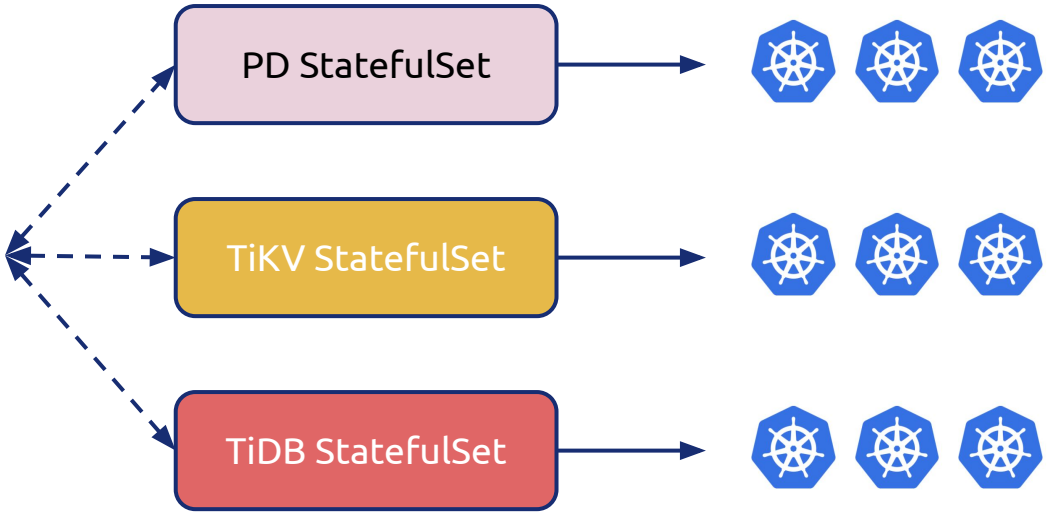
相关框架:

Operator-SDK

Kubebuilder

CRD

```
apiVersion: pingcap.com/v1alpha1
kind: TidbCluster
metadata:
  name: demo
spec:
  pd:
    image: pingcap/pd:v2.1.3
    replicas: 3
    ...
  tikv:
    image: pingcap/tikv:v2.1.3
    replicas: 5
    ...
  tidb:
    image: pingcap/tidb:v2.1.3
    replicas: 2
    ...
```



复用 Kubernetes 内置对象

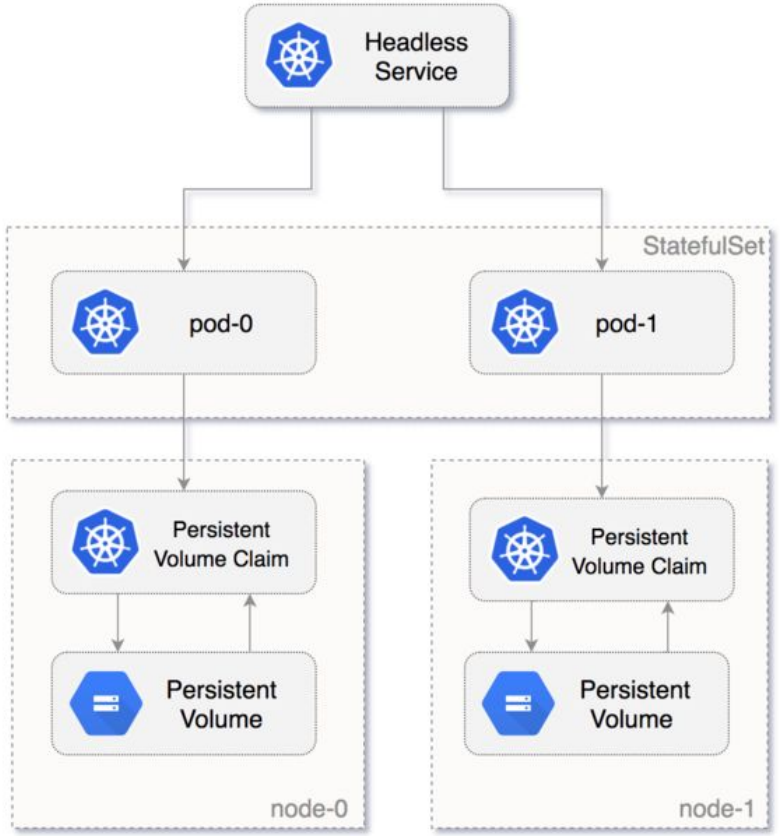
- 基于裸 Pod，我们可以做所有的事情。但我们不想重新创造一个轮子
- ReplicaSet，Deployment 和其他对象用于管理无状 态应用程序

最终我们选择使用 StatefulSet 原生对象对 集群进行管理

StatefulSet

StatefulSet是用于管理有状态应用程序的工作负载API对象。

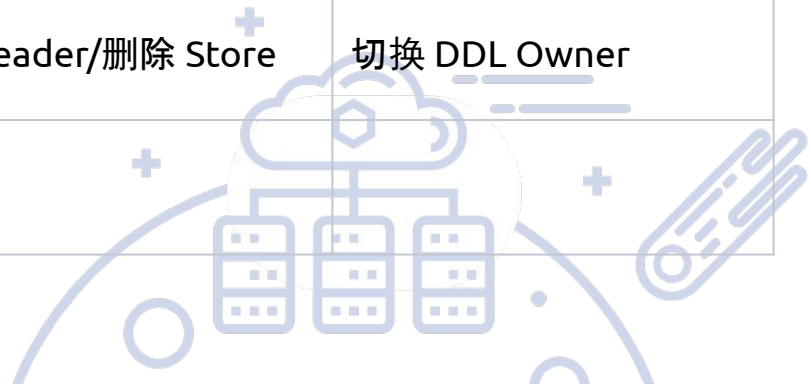
- 稳定，唯一的网络标识符。
- 稳定，持久的存储。
- 有序，优雅的部署和扩缩容。
- 有序的自动滚动更新。



PingCAP TiDB

vs 传统运维

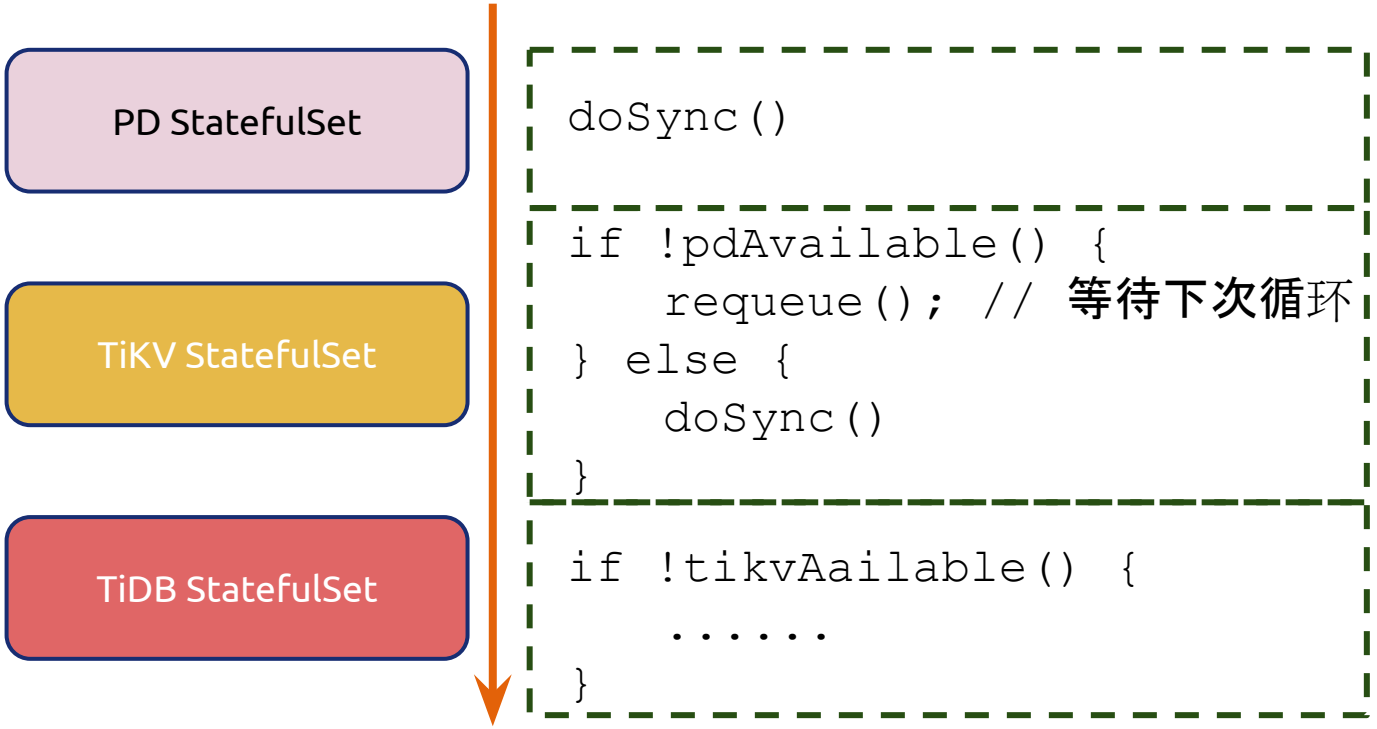
操作	PD	TiKV	TiDB
持久化存储	是	是	否
滚动更新有顺序	1	2	3
每个节点启动参数	不同	相同	相同
节点重启/下线	切换 Leader/删除 Member	驱逐 Leader/删除 Store	切换 DDL Owner
自动故障转移			



滚动更新有顺序

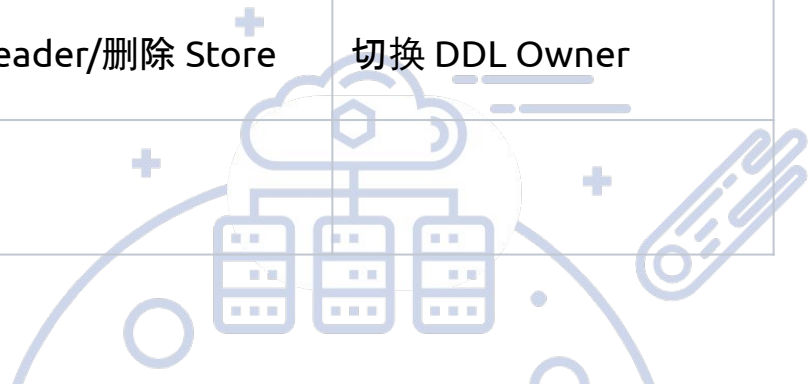
创建和管理三个 StatefulSet 对象：PD StatefulSet，TiKV StatefulSet，TiDB StatefulSet

Controller: 依赖顺序

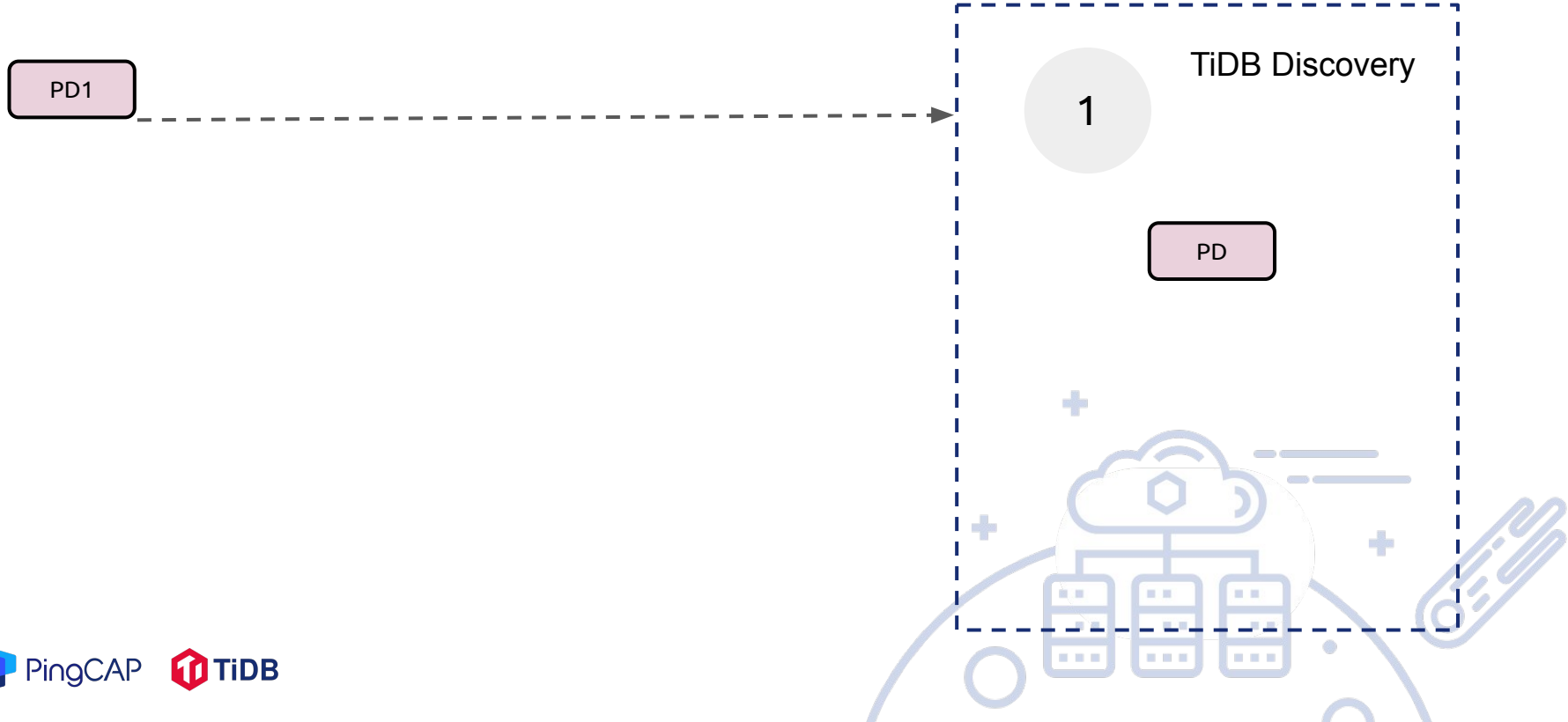


vs 传统运维

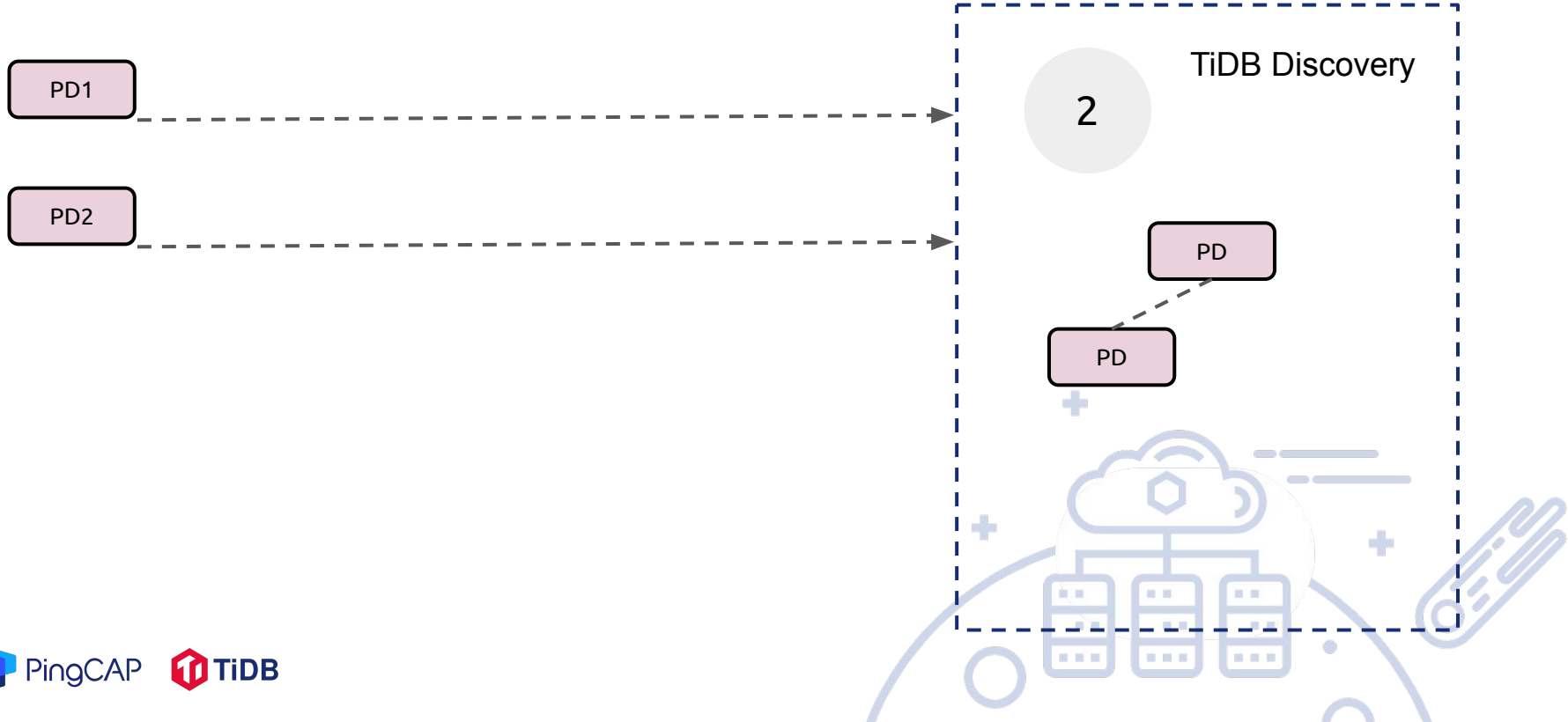
操作	PD	TiKV	TiDB
持久化存储	是	是	否
滚动更新有顺序	1	2	3
每个节点启动参数	不同	相同	相同
节点重启/下线	切换 Leader/删除 Member	驱逐 Leader/删除 Store	切换 DDL Owner
自动故障转移			



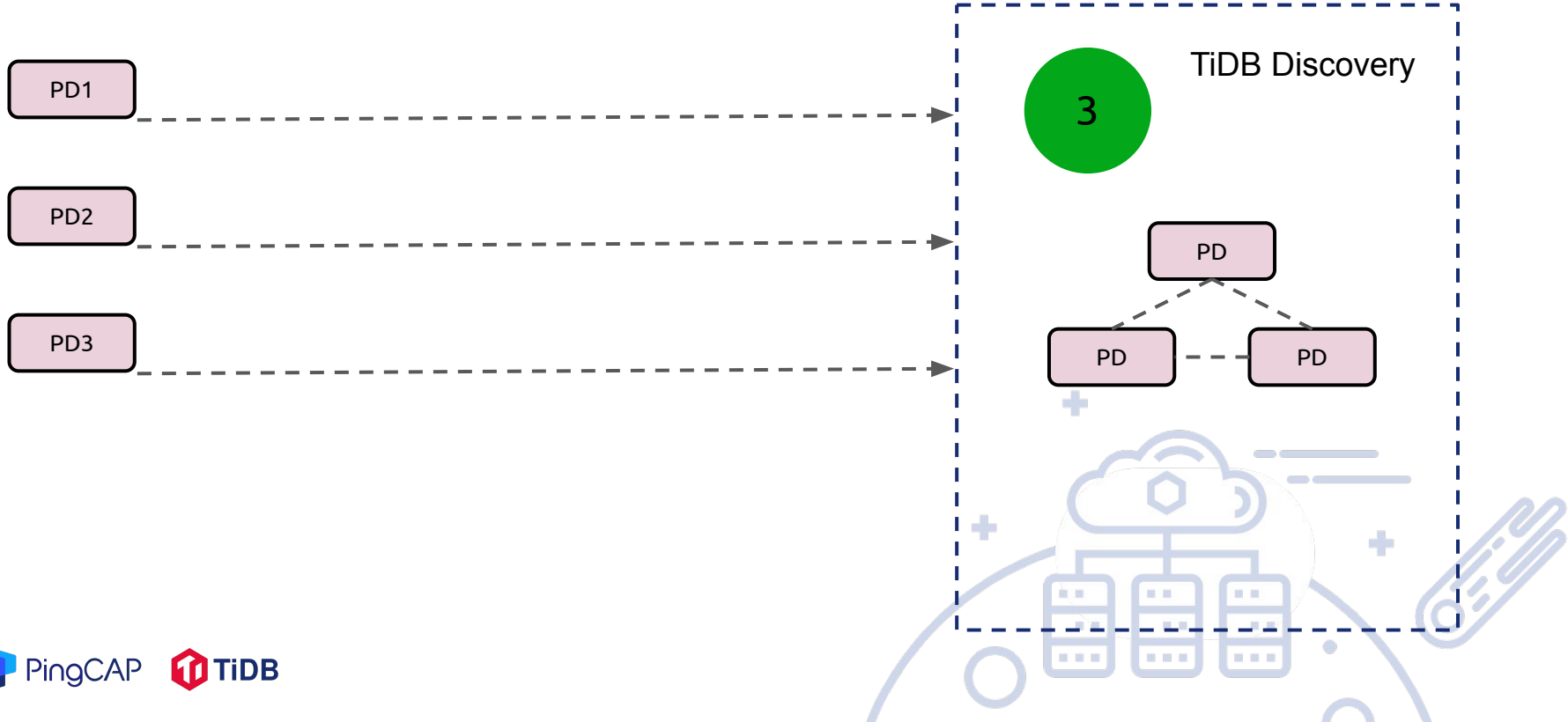
PD 启动参数



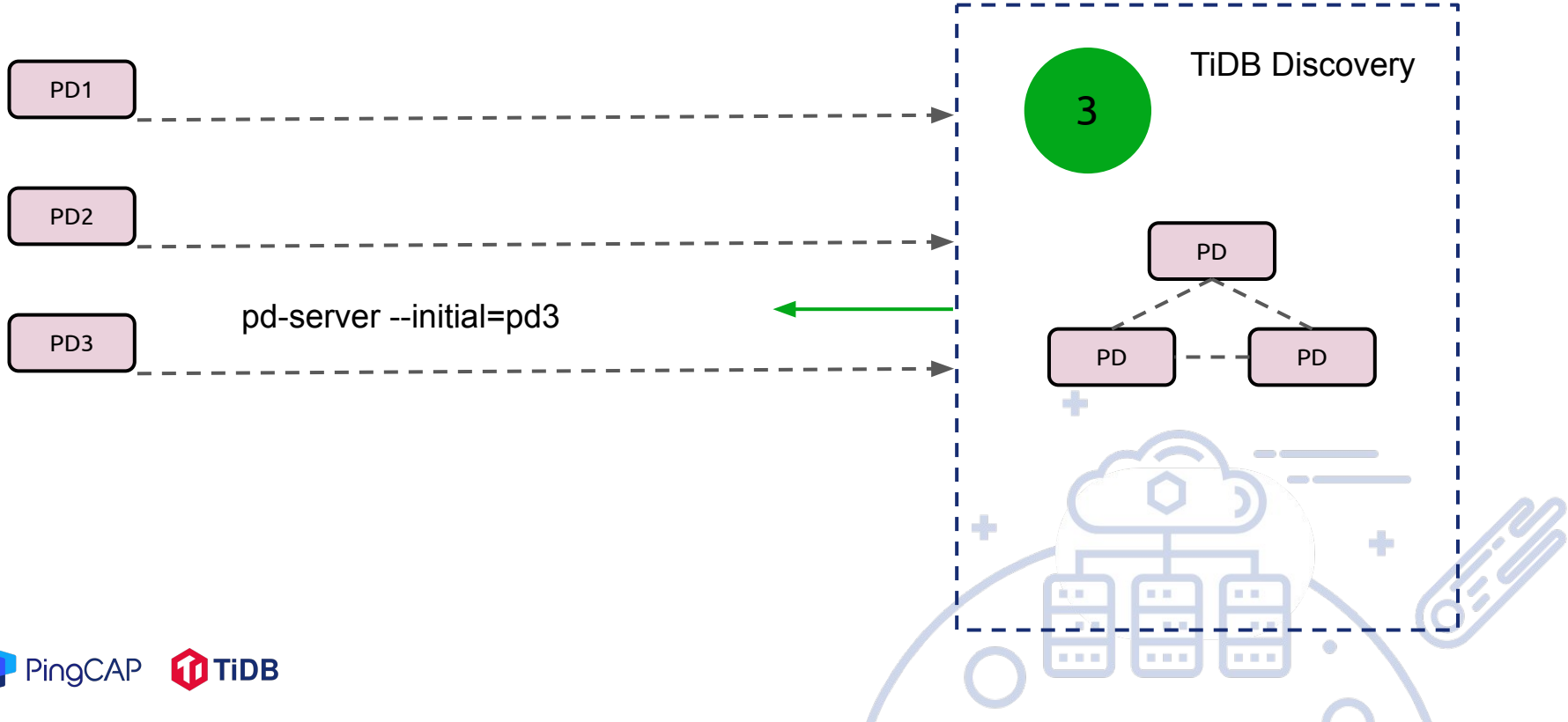
PD 启动参数



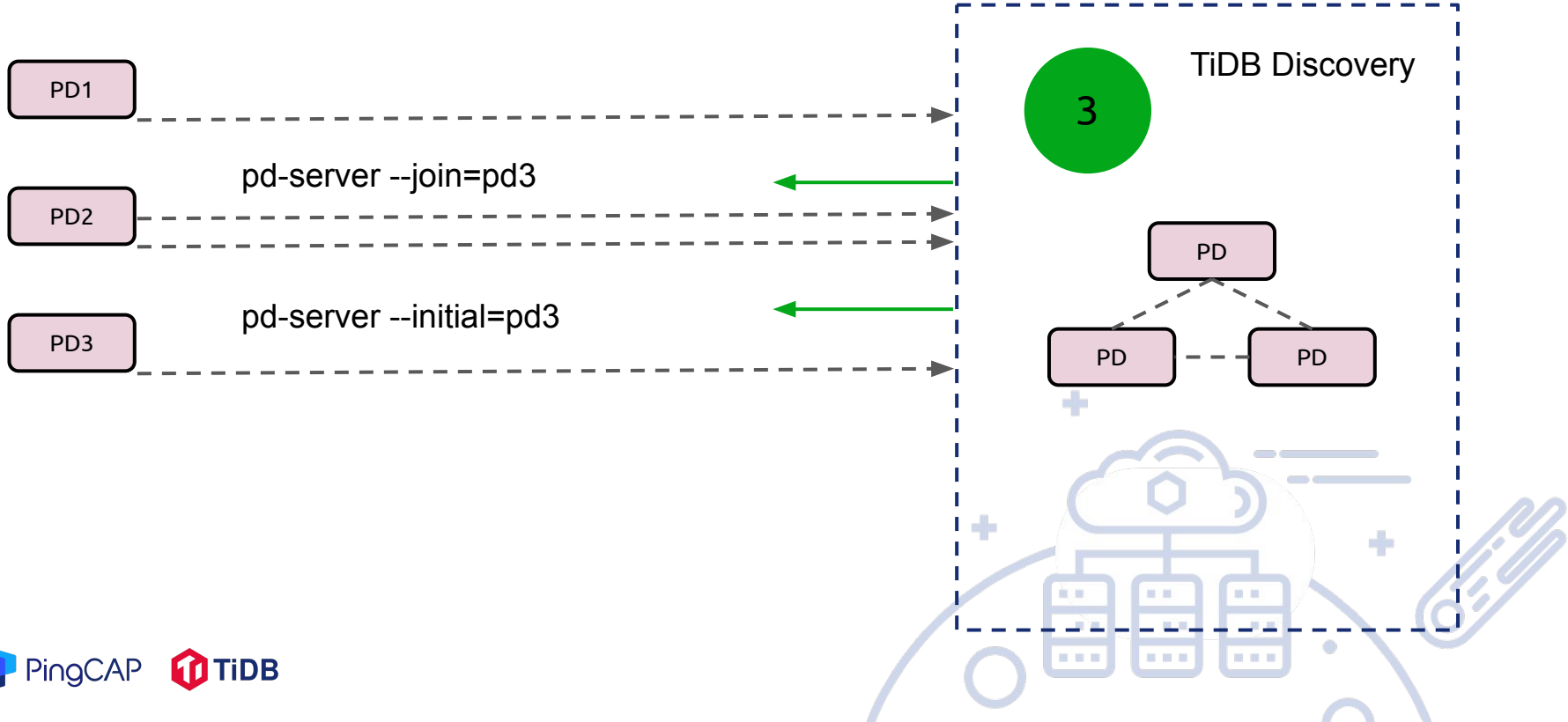
PD 启动参数



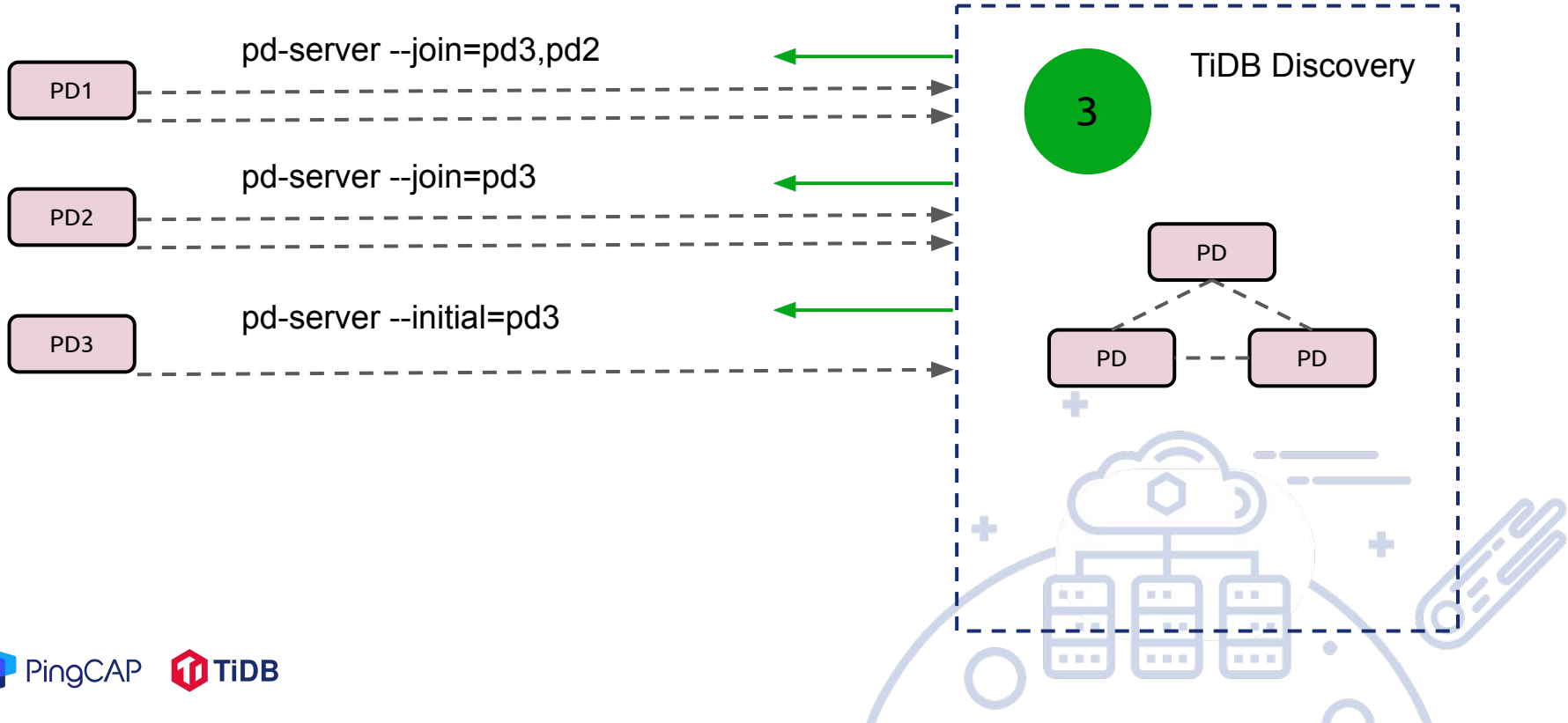
PD 启动参数



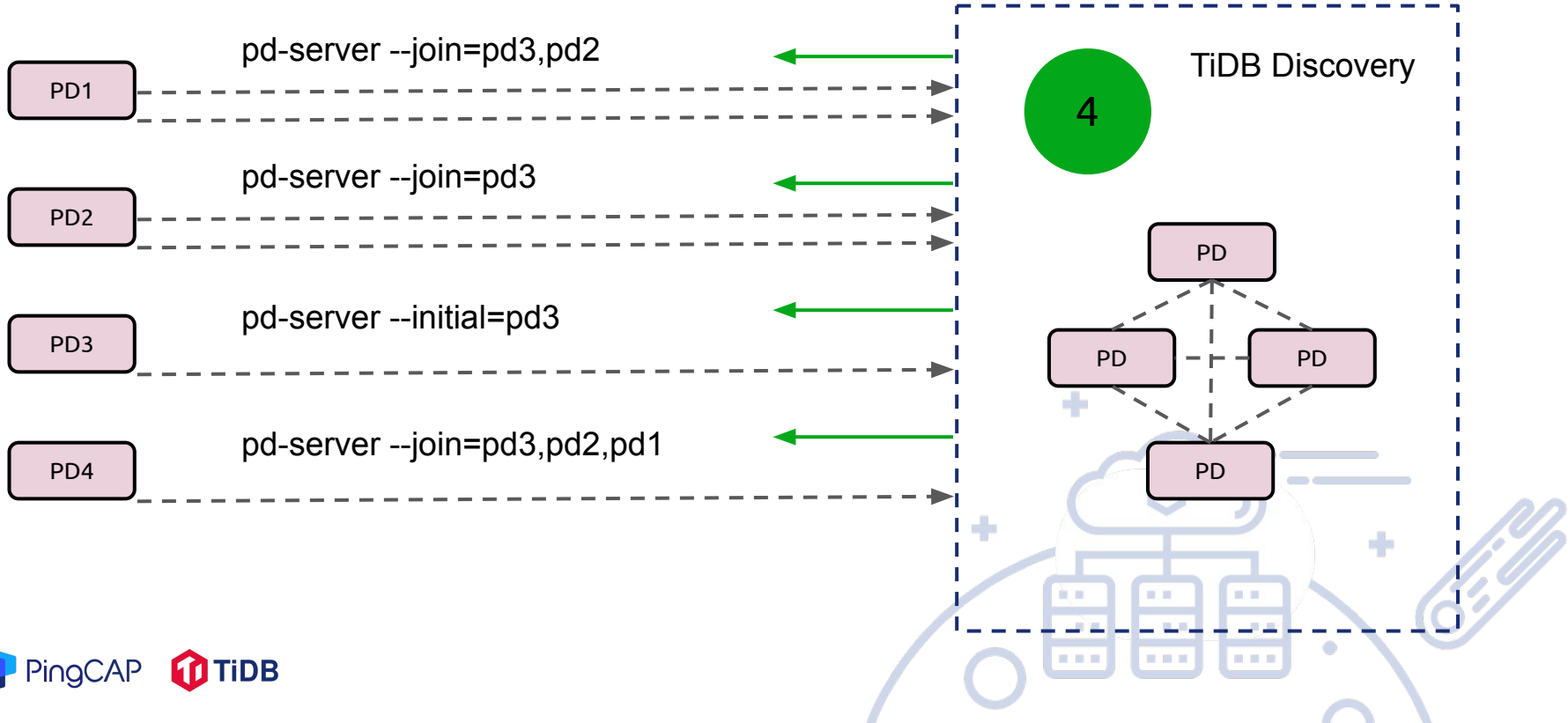
PD 启动参数


PD1
PD2
PD3
pd-server --join=pd3
pd-server --initial=pd3
TiDB Discovery
3
PD
PD
PD

PD 启动参数

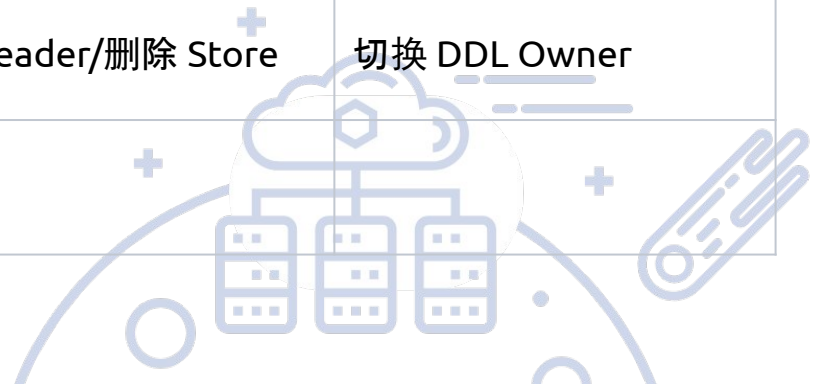

pd-server --join=pd3,pd2
PD1
pd-server --join=pd3
PD2
pd-server --initial=pd3
PD3
TiDB Discovery
3
PD
PD
PD

PD 启动参数


pd-server --join=pd3,pd2
PD1
pd-server --join=pd3
PD2
pd-server --initial=pd3
PD3
pd-server --join=pd3,pd2,pd1
PD4
TiDB Discovery
4
PD
PD
PD
PD

vs 传统运维

操作	PD	TiKV	TiDB
持久化存储	是	是	否
滚动更新有顺序	1	2	3
每个节点启动参数	不同	相同	相同
节点重启/下线	切换 Leader/删除 Member	驱逐 Leader/删除 Store	切换 DDL Owner
自动故障转移			

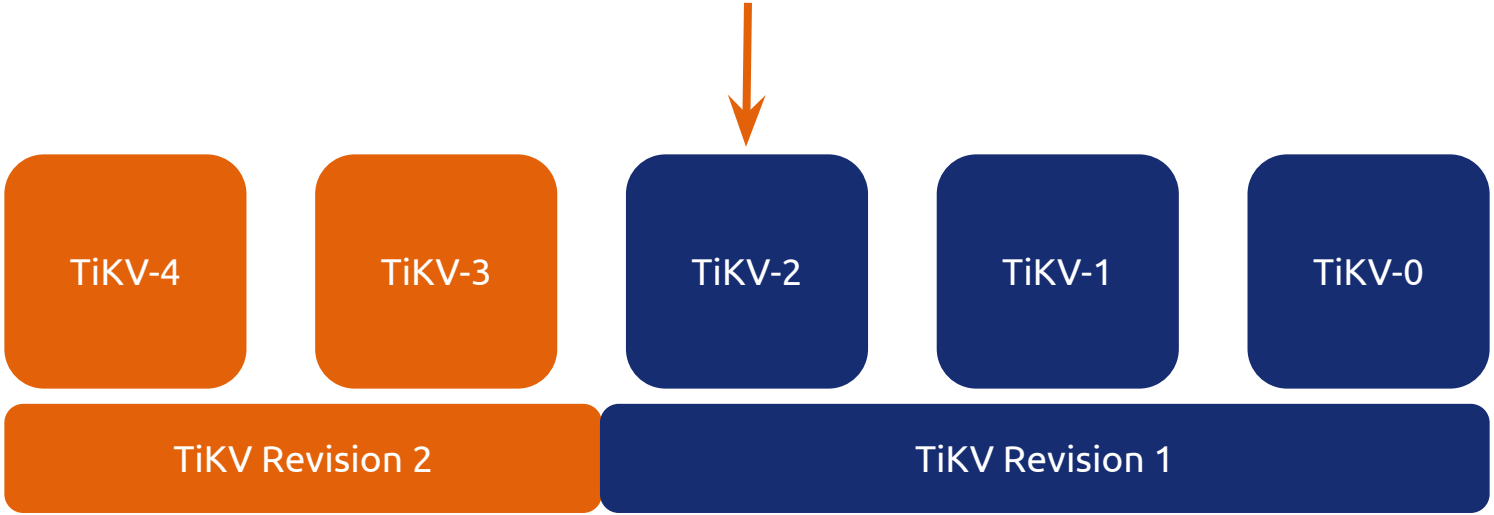


节点重启/下线

通过控制 StatefulSet 对象节奏（控制 partition 字段），在 Pod 删除之前进行清理操作：

- PD：驱逐 Leader，删除 Member
- TiKV：驱逐 Leader 删除 Store
- TiDB：Resign DDL Owner
- etc.

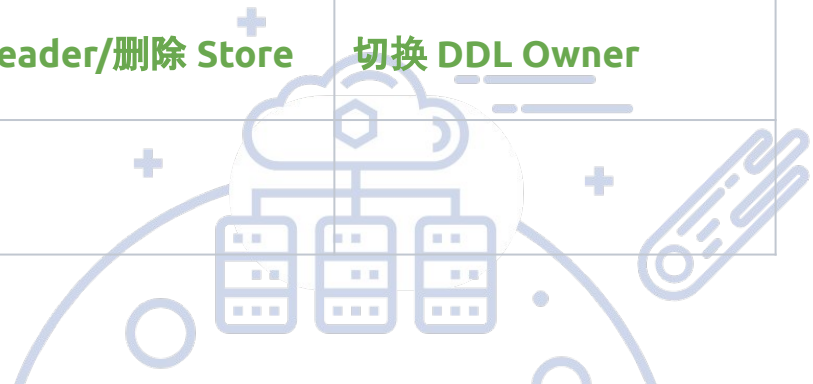
Controller: 优雅升级



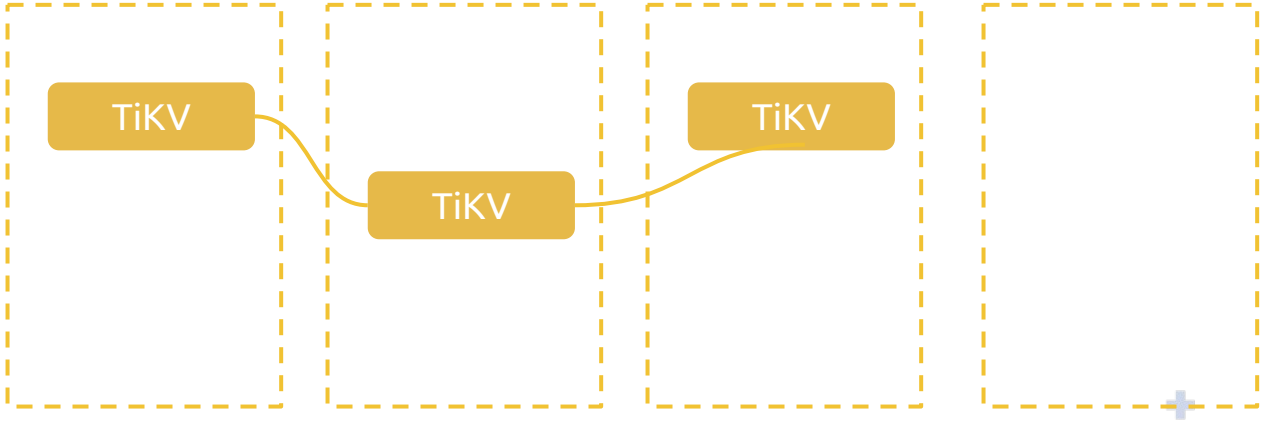

```
if store.LeaderCount != 0 {
    evictLeader(store.ID); return
}
setUpgradePartition()   // 写 ApiServer，将升级进度调
                        // 整到 TiKV-2
```

vs 传统运维

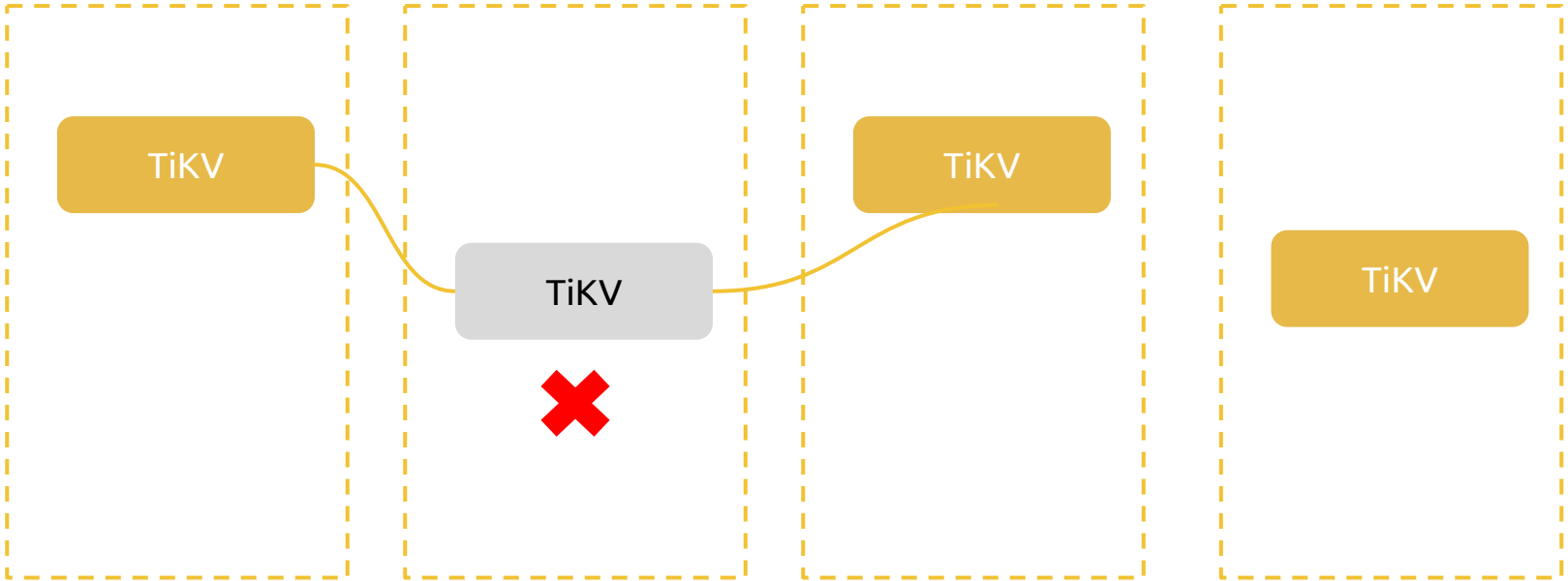
操作	PD	TiKV	TiDB
持久化存储	是	是	否
滚动更新有顺序	1	2	3
每个节点启动参数	不同	相同	相同
节点重启/下线	切换 Leader/删除 Member	驱逐 Leader/删除 Store	切换 DDL Owner
自动故障转移			



自动故障转移

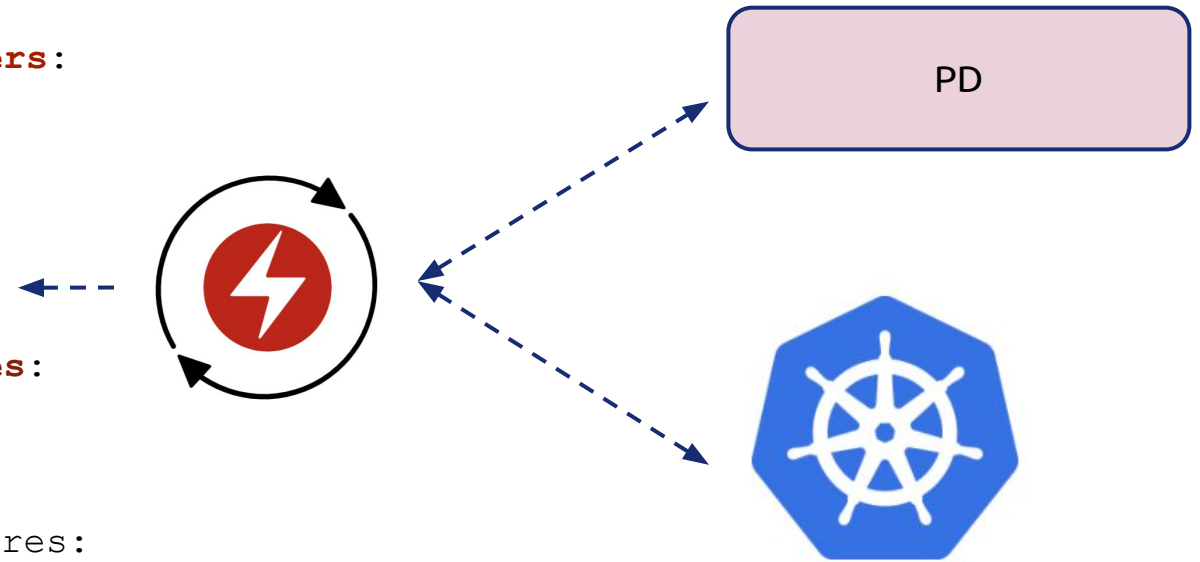


Controller: 故障转移



Controller: 故障转移

```
status:
   pd:
      failureMembers:
        ...
      leader:
        ...
      members:
        ...
   tikv:
      failureStores:
        ...
      stores:
        ...
      tombstoneStores:
        ...
```

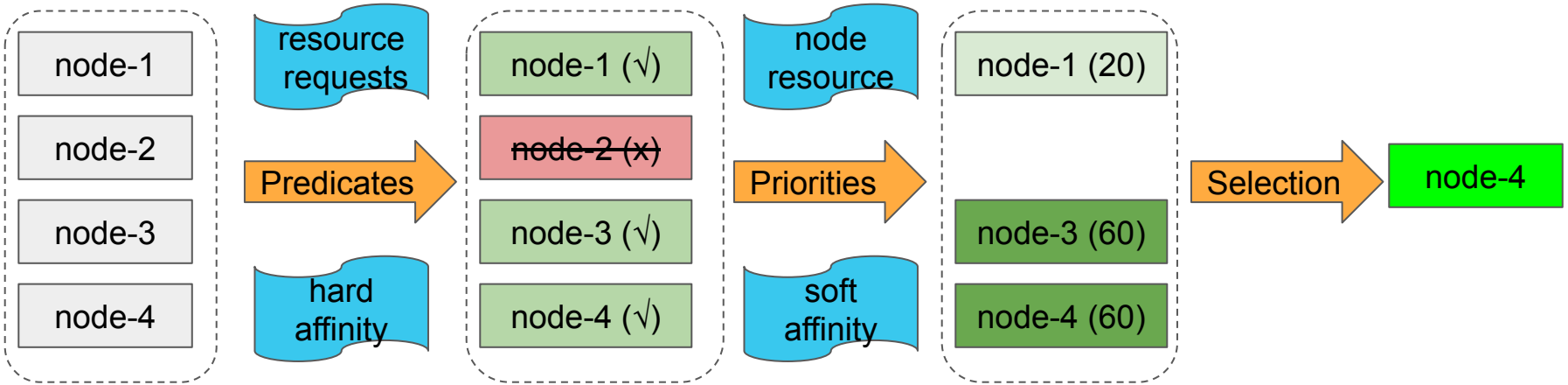


不仅从 k8s, 还要从 **PD** 不断收集**实际状态**

K8s 调度器 Scheduler

资源条件：CPU，内存，存储等
约束条件（pod 与 node，pod 与 pod 间）：亲和，反亲和
调度过程：

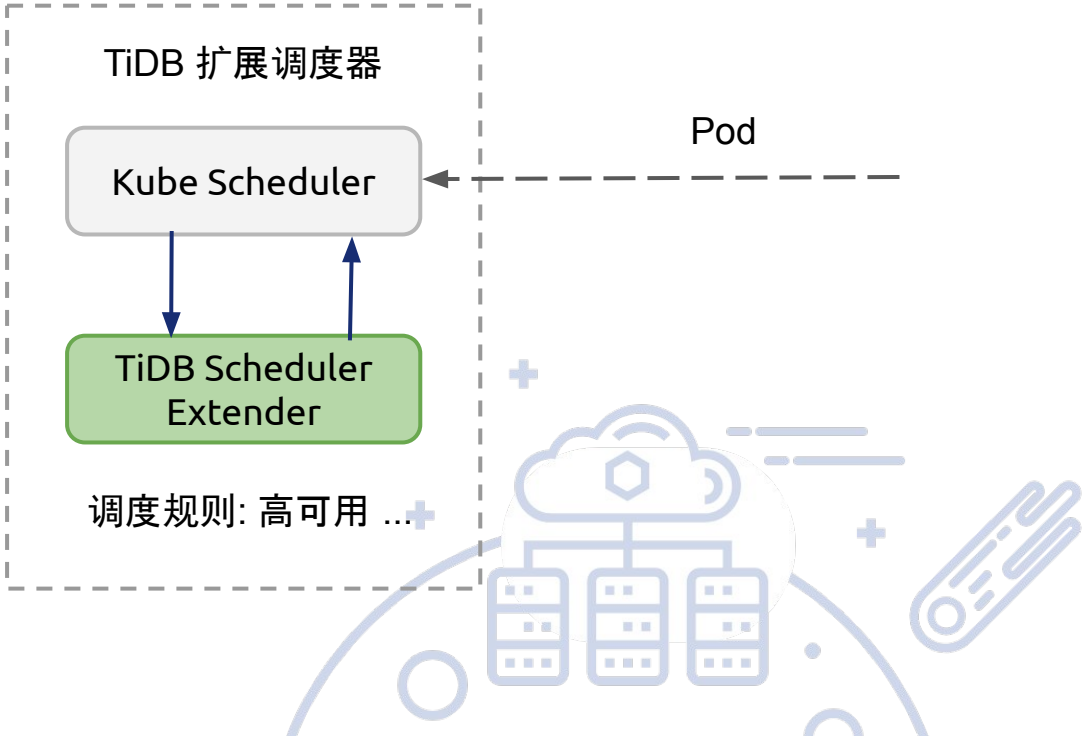
1. 断言（Predicates）：根据资源条件以及硬性约束条件断言 Pod 是否能够调度到该节点
2. 优先级（Priorities）：根据节点资源情况以及软性约束条件给节点打分
3. 选定（Selection）：选出打分最高的节点进行调度，多个并列第一则随机选择



扩展了 K8s 调度器

为了数据安全，我们不允许将两个TiKV实例安排到同一节点

```
apiVersion: apps/v1
kind: StatefulSet
…
spec:
  template:
    spec:
      schedulerName: tidb-scheduler
      containers:
...
```



扩展了 K8s 调度器

TiDB Operator 实现了 K8s 扩展调度器，在 K8s 原生调度策略的基础之上，进一步增强了 PD 和 TiKV 节点的高可用。

Raft 算法：多副本存活（> n/2 + 1）
PD 基于物理部署拓扑的数据 调度策略：region, zone, rack, host
TiDB Scheduler 组件基于物理部署拓扑的 节点调度策略：region, zone, rack, host

PingCAP

Part III - 操作示例

K8s 包管理工具：Helm/Charts

Helm 安装管理 K8s 应用：

- helm ls
- helm install stable/mysql --name=test-mysql
- helm upgrade test-mysql --set xxx=yyy
- helm delete

TiDB Operator 基本操作

集群部署：

```
$ helm install ~/tidb-cluster --name=tidb-cluster --namespace=tidb -f values.yaml
$ watch kubectl get pods --namespace tidb -l app.kubernetes.io/instance=demo -o wide
```

集群验证：

```
$ mysql -h <cluster-ip> -P 4000 -u root
```

PingCAP TiDB

TiDB Operator 基本操作

集群扩缩容：

```
$ vim values.yaml # 将 PD 和 TiKV 节点数修改为 5，TiDB 节点数修改为 3
$ helm upgrade tidb-cluster ~/tidb-cluster --namespace=tidb -f values.yaml
$ watch kubectl get pods --namespace tidb -l app.kubernetes.io/instance=demo -o wide
```

集群升级：

```
$ vim values.yaml # 将 TiDB 镜像版本修改为 v2.1.1
$ helm upgrade tidb-cluster ~/tidb-cluster --namespace=tidb -f values.yaml
$ watch kubectl get pods --namespace tidb -l app.kubernetes.io/instance=demo -o wide
```

See also

操作文档：

https://pingcap.com/docs-cn/v3.0/tidb-in-kubernetes/tidb-operator-overview/

repo：

https://github.com/pingcap/tidb-operator

PingCAP

Part IV - 更多工作

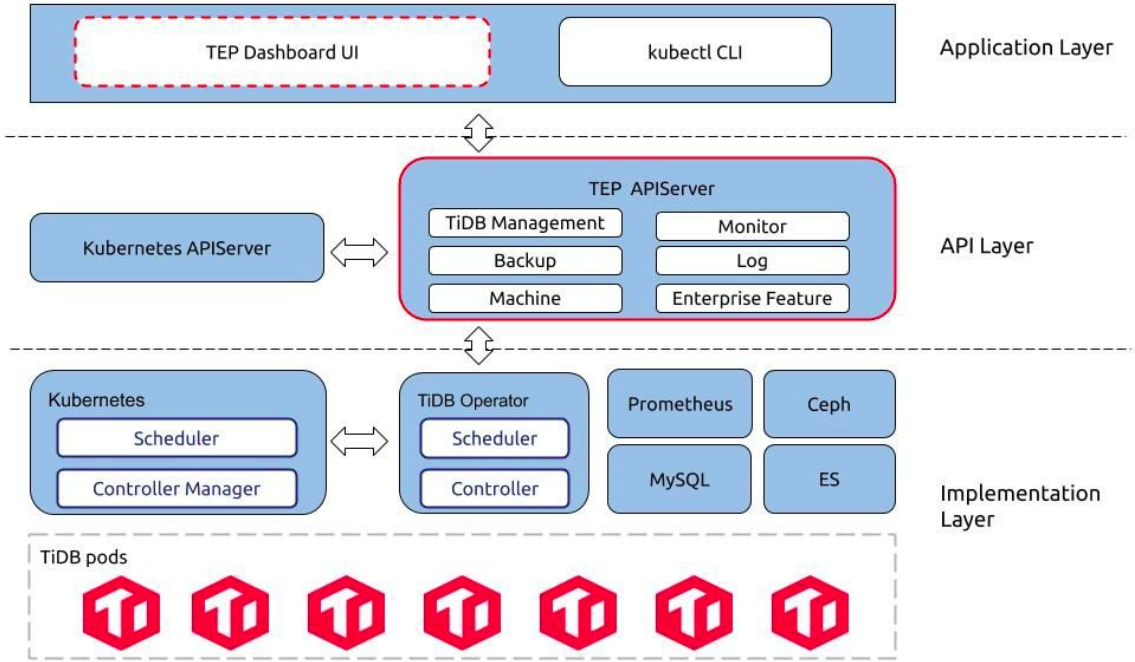
TiDB-Operator 的问题

- 使用起来需要一定的运维基础
- 不容易接入本地系统
- 工具本身也不太容易上手

想以更熟悉（简单）的方式来管理 TiDB 集群 ？

TEP（aggregated apiserver 架构）

- UI，OpenAPI 接入方式
- 基于 kube-apiserver 架构进行开发，完美支持 Kubectl 工具
- 更加友好的 api 模式
- 如果需要，同样能实现自定义 API 对象以及 controller 的需求
- 更多实用功能



Operator: CRD vs AA

CRD	AA
Simple, mature framework support	Complex, apiserver-builder is alpha
Validting: OpenAPI v3 and webhook	As you like
Defaulting: mutating webhook	As you like
Same storage with apiserver (etcd)	As you like
authn/authz: kube-apiserver	As you like
Common, reuse most of kubernetes api featrures: Multiple version, CRUD, ListAndWatch, Built-in Authz & Authn, UI/CLI intergration, Admission Webhooks, Finalizers, Labels & Annotations	

PingCAP

Welcome!

We are hiring 👉

hire@pingcap.com