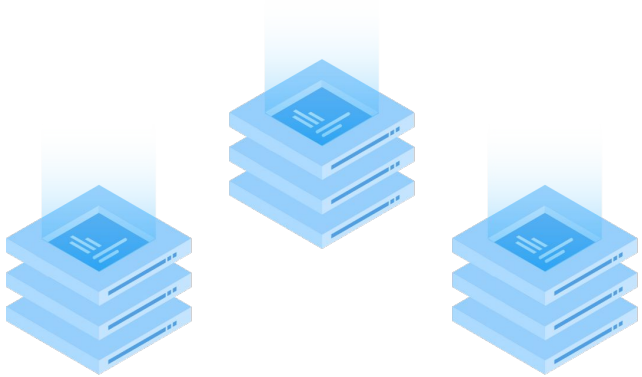
TiDB ❤ K8s

在 Kubernetes 上部署你的 TiDB 集群

Yin Liu

PingCAP TiDB



写在开始之前

- 内容：介绍如何在 K8s 上面跑 TiDB
- 目标：了解 TiDB Operator 的设计原理
- 目标听众：
  - 角色：架构师，技术人员，开发人员
  - Level：对容器和 K8s 有一定基础的了解
- 预计时间：40 分钟
- 议程：
  - 为什么要在 K8s 上跑 TiDB？
  - K8s 的基本原理
  - 实现自动化：TiDB Operator
  - 最佳实践

Part I: 为什么要在 K8s 上跑 TiDB？

为什么要在 K8s 上跑 TiDB？

通常你公司的基础设施和技术栈已经作出了选择

为什么要在 K8s 上跑 TiDB？

通常你公司的基础设施和技术栈已经作出了选择

如果你的业务都跑在 K8s 上，那么数据库自然也不应该例外

为什么要在 K8s 上跑 TiDB？

通常你公司的基础设施和技术栈已经作出了选择

如果你的业务都跑在 K8s 上，那么数据库自然也不应该例外

反之亦然

为什么要在 K8s 上跑 TiDB？

通常你公司的基础设施和技术栈已经作出了选择

假如还没有 all in K8s，是不是可以考虑一下了？

TiDB 是否适合 K8s？

- 容器有自己的生命周期，从创建到销毁
- 容器是不可变的，他们只能被替换而不能原地更新

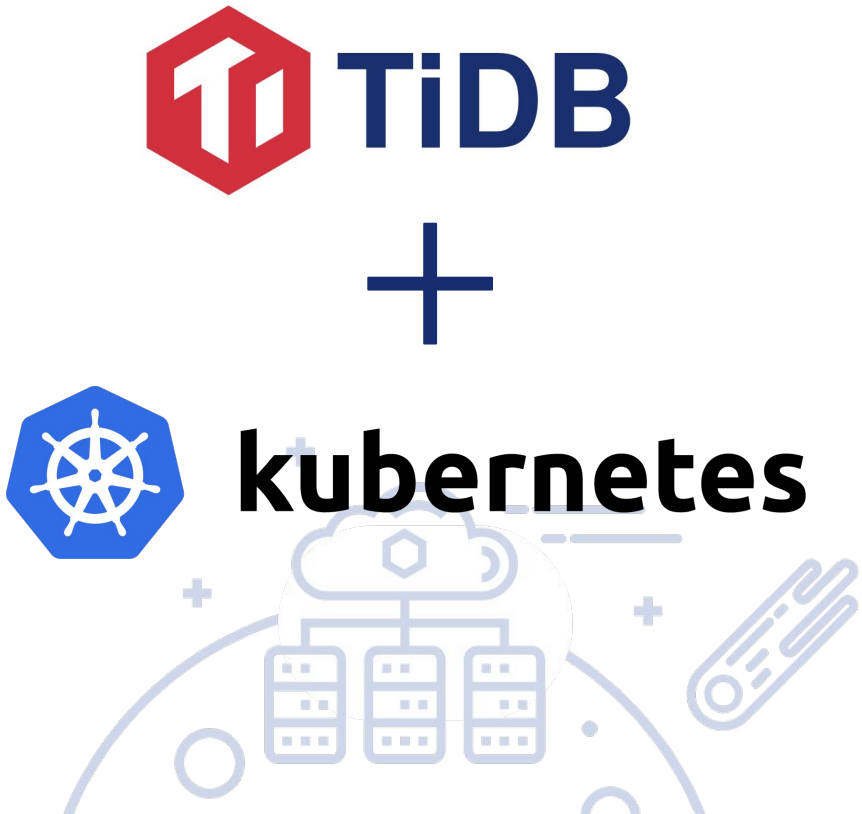
对于传统的单机数据库来说，这似乎不是一个理想环境，不过...

TiDB 是否适合 K8s？

而 TiDB 是一个云原生的数据库

- 水平伸缩很容易
- 容忍单点故障
- 内置的可观测性

跟云原生的 K8s 是天生的好基友



K8s 能给 TiDB 带来的好处

- 可以跟其他的无状态应用类似
  - 自动部署，滚动升级和自动故障恢复
  - 资源隔离和弹性调度，提升硬件利用率
  - 减少劳动，降低人为造成的犯错风险

K8s 能给 TiDB 带来的好处

- 可以跟其他的无状态应用类似
  - 自动部署，滚动升级和自动故障恢复
  - 资源隔离和弹性分配，提升硬件利用率
  - 减少劳动，降低人为造成的犯错风险

- 另外值得一提，K8s 具备非常良好的扩展性
  - 可以很容易注入个性化的东西，而且不破坏统一性

Part II: Get started

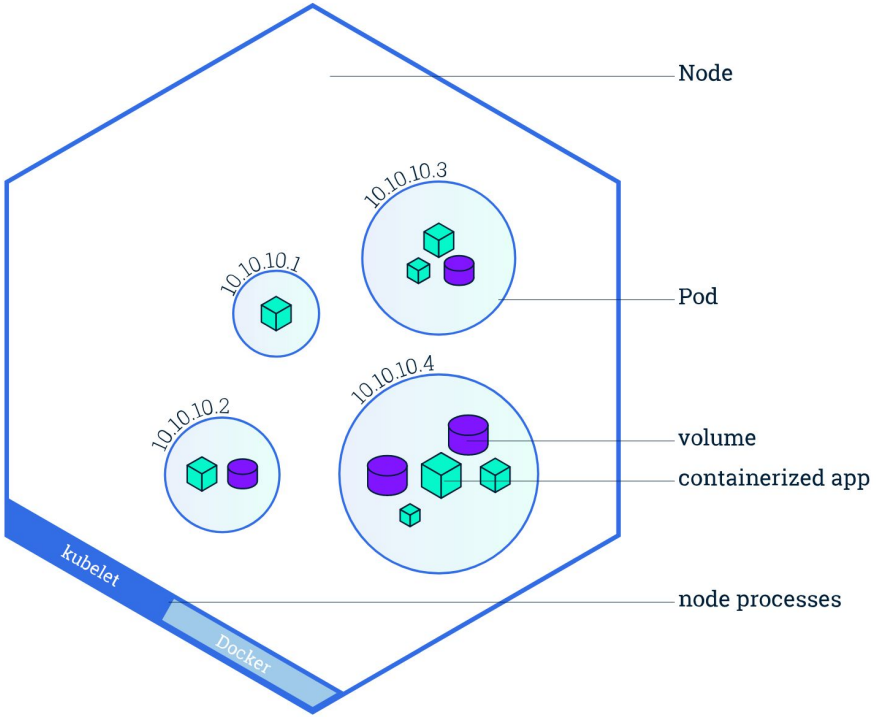
在 K8s 上跑 TiDB
Round 1

先跑起来

PingCAP TiDB

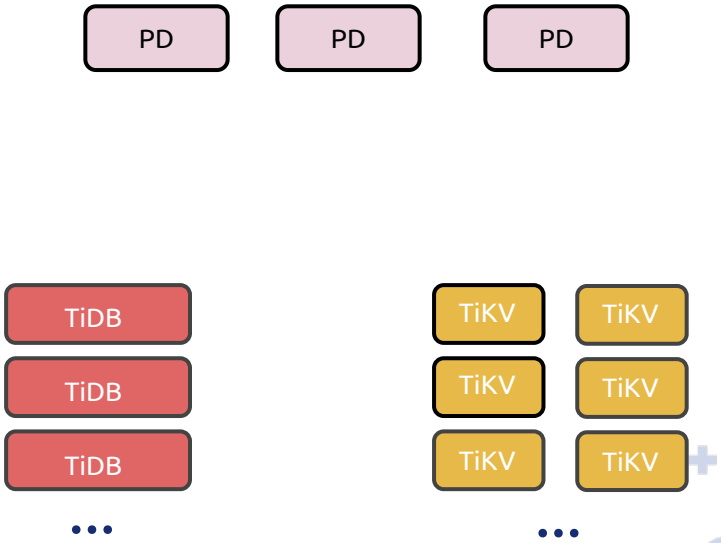
Pods

- 由一个或多个容器打包组成
- 有独立的 IP (Pod IP)
- 一般情况，一个 Pod 只放一个 TiDB 的进程 (例如：tidb-server, tikv-server 或者 pd-server)

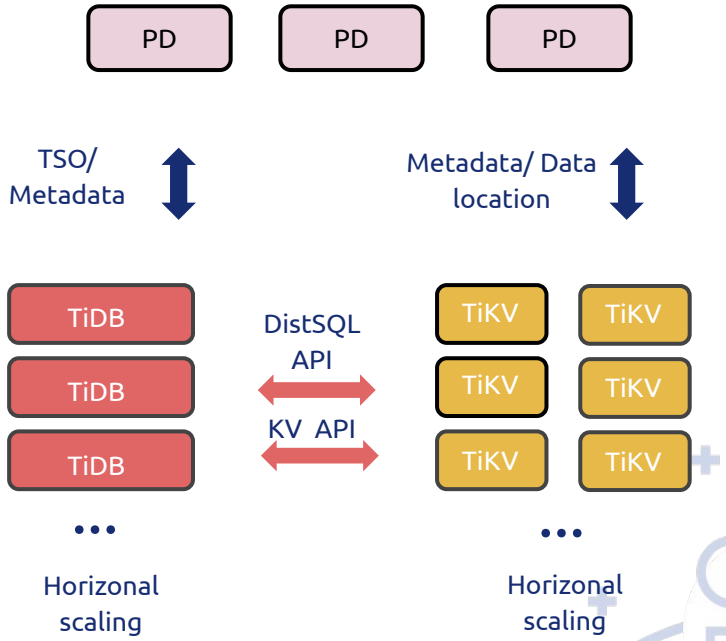


旅程开始

1. 创建各种 Pods

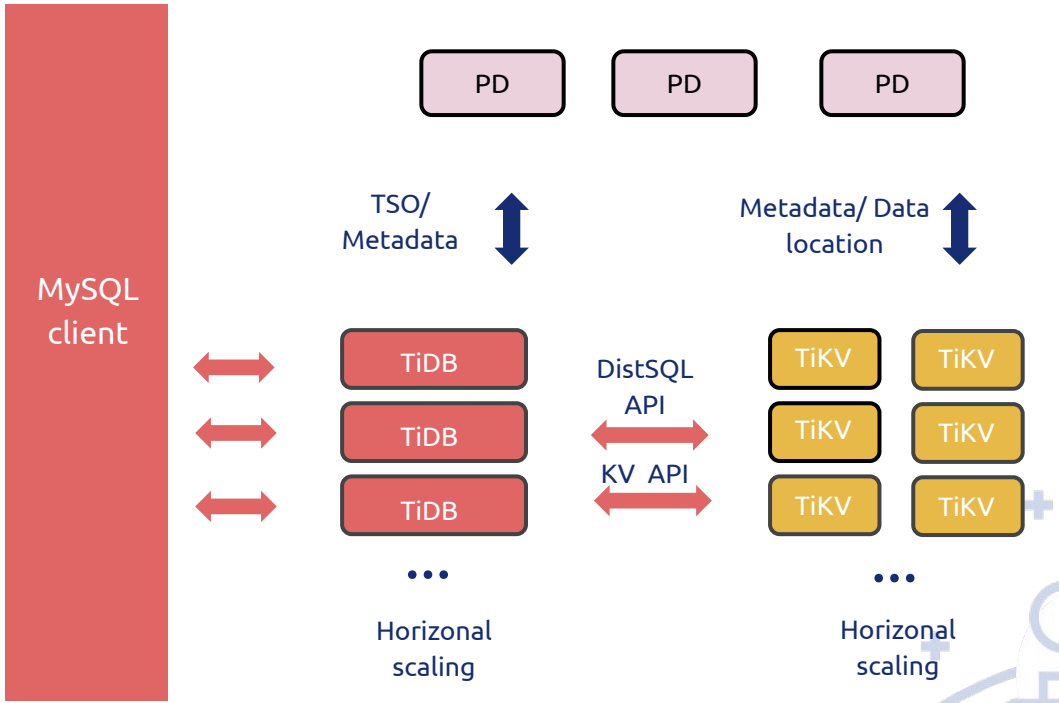


旅程开始



1. 创建各种 Pods
2. 配置组件之间通过 Pod IP 进行通讯

旅程开始



1. 创建各种 Pods
2. 配置组件之间通过 Pod IP 进行通讯
3. 暴露 Pod IP 向外部提供访问

一堆头疼的 YAML 文件

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: pd-0
spec:
  containers:
    - image: pingcap/pd
      name: pd
      command:
        - /pd-server
```

x 3

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: tikv-0
spec:
  containers:
  - image: pingcap/tikv
    name: tikv
    command:
      - /tikv-server
      - --pd=${pd_ip}:2379
```

x 6

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: tidb-0
spec:
  containers:
    - image: pingcap/tidb
      name: tidb
      command:
        - /tidb-server
        - --path=${pd_ip}:2379
```

x 3

PingCAP TiDB

裸 Pods 有什么问题？

- Pods are mortal for the following two reasons
  - Pods 与 Node(机器) 是绑定的
  - Kuberetes 不会自动故障转移 Pods 到其他的 Node 上
- 人肉操作太多了，所有 Pods 都要靠手工维护

Controller 出场了

- Controller 可以帮助你来管 Pods
  - 也就是常说的：自动部署，扩容缩容，滚动升级，故障转移
- 让 Pods 按照你的 “**意图**” 行事

意图如何变成现实？

- K8s 内置了很多 **API 对象**，并允许对其进行 CRUD 操作
- **API 对象**也是对用户意图的记录
- **spec**: 希望的样子
- **status**: 当前的样子
- “**Controller**” 的作用就是不断把当前的样子变成所希望的样子

```yaml
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
```

需要 3 个副本

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

实际只有 2 个副本

PingCAP TiDB

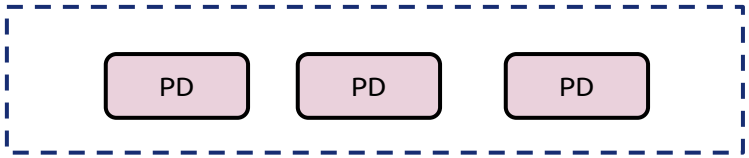
使用 Controllers 来跑 TiDB

选哪个 Controller 更合适？

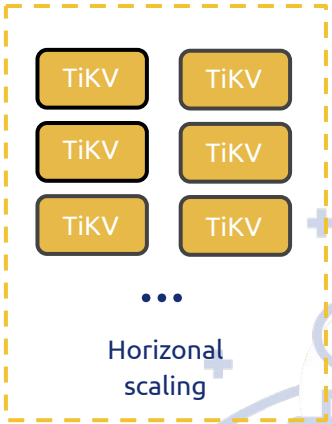
- *Deployment:* 通常针对部署无状态应用
  - 可以按声明的方式指定副本数量

- *StatefulSet:* 通常针对有状态的应用（啥叫有状态？）
  - 每个副本都有固定名称和地址，基于对等发现的 DNS 解析
  - 每个副本都要存持久化的数据，数据内容各不相同

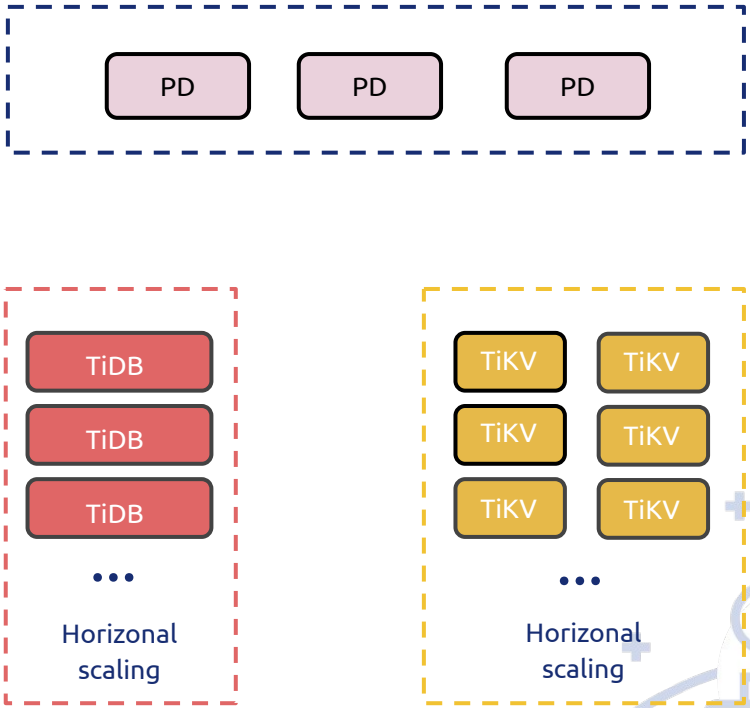
使用 Controllers 来跑 TiDB



1. 为 PD 和 TiKV 创建 StatefulSet

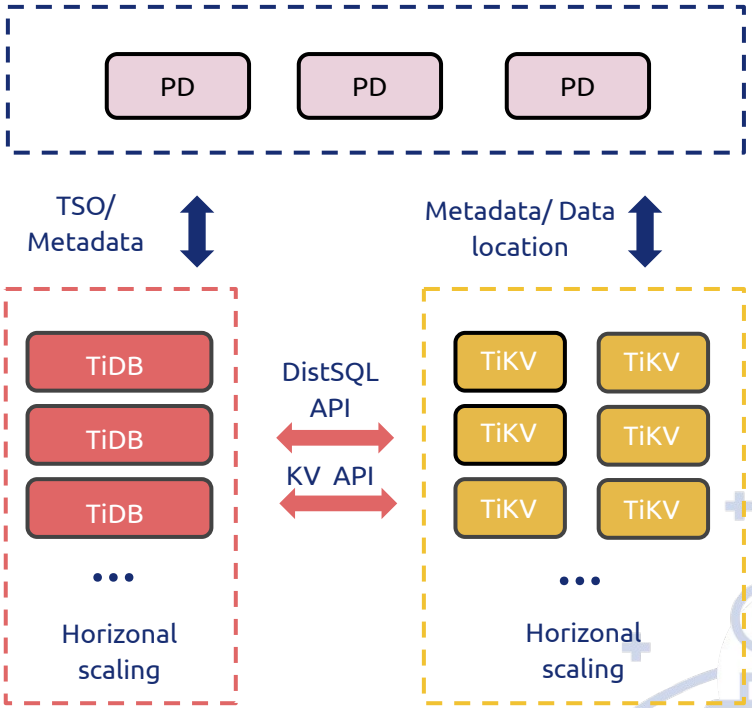


使用 Controllers 来跑 TiDB



1. 为 PD 和 TiKV 创建 StatefulSet
2. 为 TiDB 创建 Deployment

使用 Controllers 来跑 TiDB

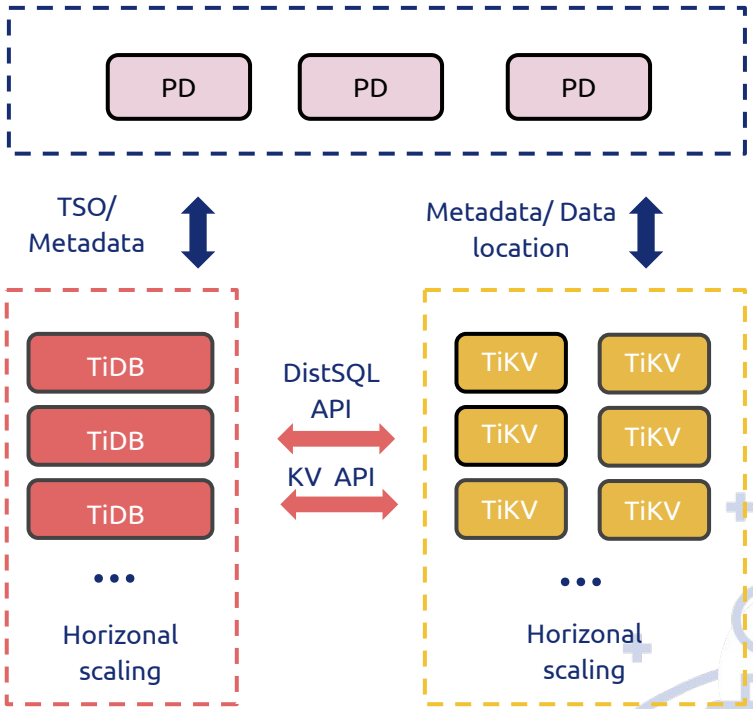


1. 为 PD 和 TiKV 创建 StatefulSet
2. 为 TiDB 创建 Deployment
3. 配置组件之间通过 Pod IP 进行通讯

使用 Controllers 来跑 TiDB

等等！

如果 Pods 被转移到其他节点，Pod IP 也会变

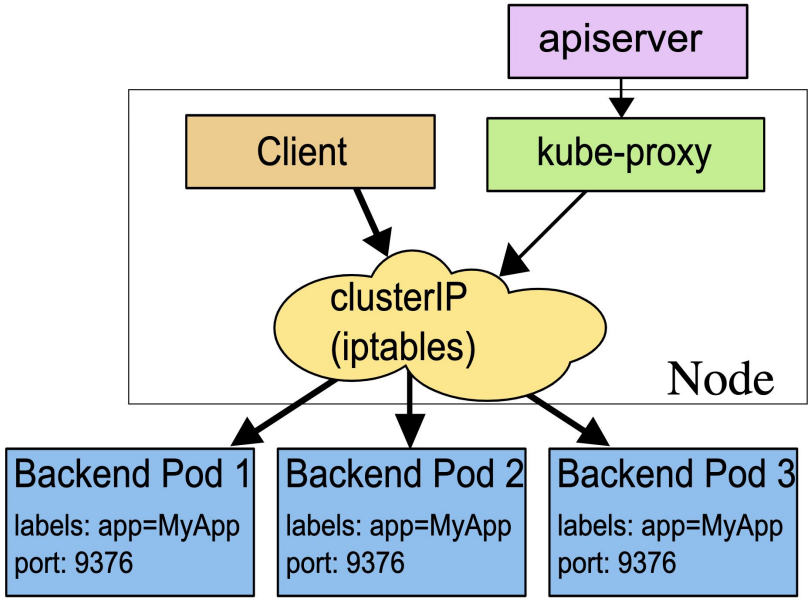


1. 为 PD 和 TiKV 创建 StatefulSet
2. 为 TiDB 创建 Deployment
3. 配置组件之间通过 Pod IP 进行通讯

通过 Service 来访问 Pods

来看看什么是 Service

- Service 的工作原理跟负载均衡器差不多，他接受并分派流量到 Pods
- 同样，负载均衡的策略也是使用 API 对象来描述用户的意图



通过 DNS 来访问 Pods

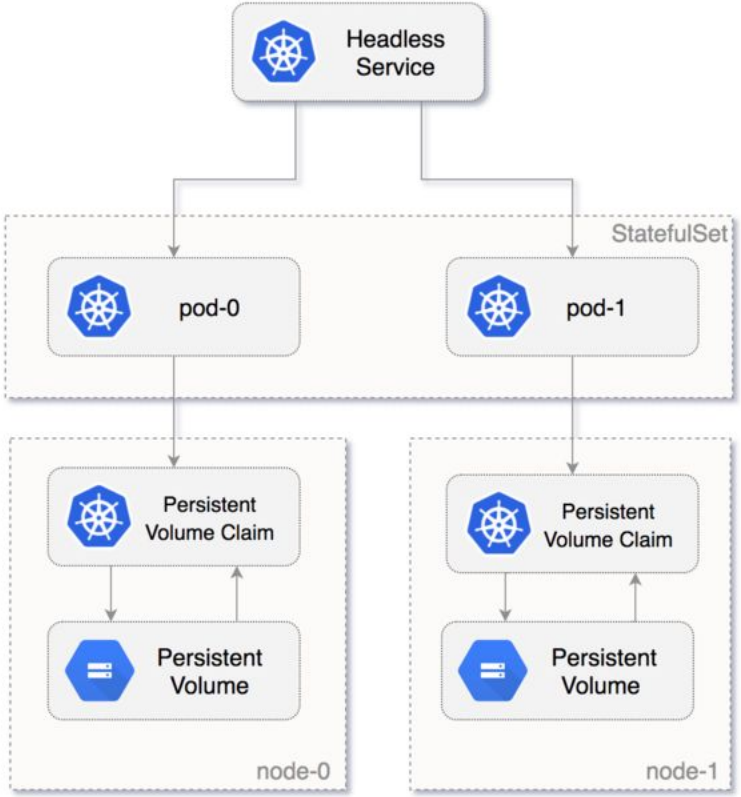
假如你只想访问某个指定的 Pod 怎么办？

- 一个比较 Hack 的方法是创建一个 Service 仅指向到这个 Pod
- 但实际上，是可以通过域名来访问指定的 Pod
  - Kubernetes 具有可选的 DNS 附加组件
  - 由 StatefulSet 来管理的 Pods 是可以由一个稳定的域名，即时发生了重建

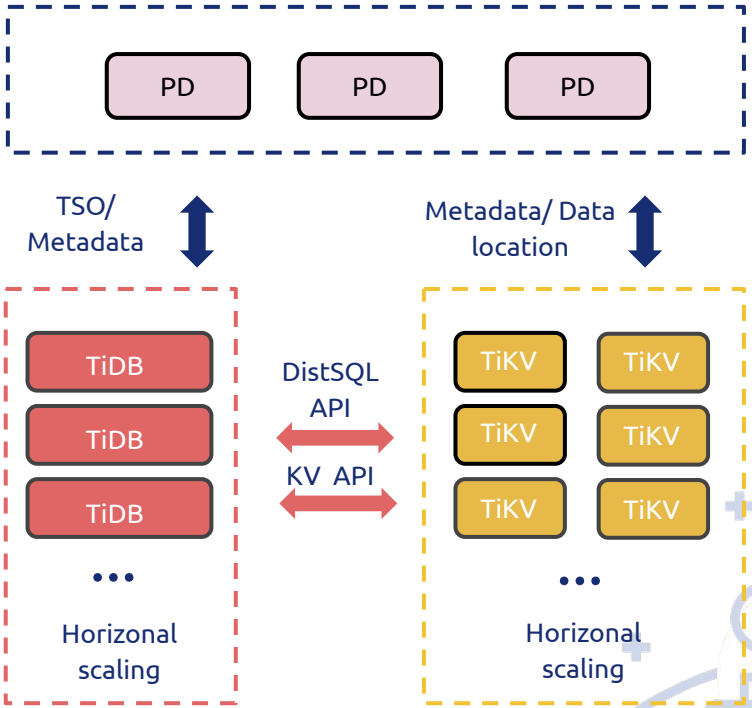
通过 DNS 来访问 Pods

在 *StatefulSet* 中:

- Pods 按顺序命名, 比如: pod-0, pod-1. 结合这样的头部组合成一个固定的 Pod 域名
- 每个域名同样和多个稳定存储相关联

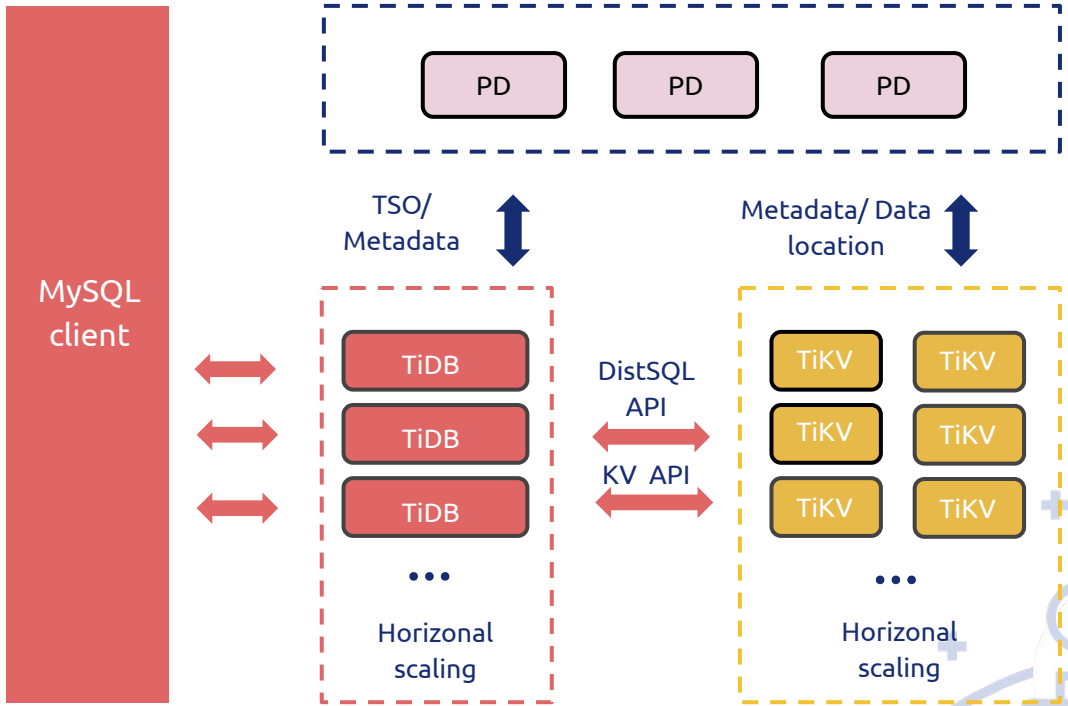


使用 Controllers 来跑 TiDB



1. 为 PD 和 TiKV 创建 StatefulSet
2. 为 TiDB 创建 Deployment
3. 配置组件之间通过域名进行通讯

使用 Controllers 来跑 TiDB



1. 为 PD 和 TiKV 创建 StatefulSet
2. 为 TiDB 创建 Deployment
3. 配置组件之间通过域名进行通讯
4. 向外暴露 TiDB Service 给用户

只需要维护 2 个 StatefulSet 和 1 个 Deployment

```yaml
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: pd
  labels:
    app: pd
spec:
  selector:
    matchLabels:
      app: pd
  replicas: 3
  template:
    spec:
      containers:
      - image: pingcap/pd
        name: pd
        command:
        - /pd-server
  serviceName: pd-headless
```

```yaml
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: tikv
  labels:
    app: tikv
spec:
  selector:
    matchLabels:
      app: tikv
  replicas: 6
  template:
    spec:
      containers:
      - image: pingcap/tikv
        name: tikv
        command:
        - /tikv-server
        - --pd=pd:2379
  serviceName: tikv-headless
```

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: tidb-0
  labels:
    app: tidb
spec:
  selector:
    matchLabels:
      app: tidb
  replicas: 3
  template:
    spec:
      containers:
      - image: pingcap/tidb
        name: tidb
        command:
        - /tidb-server
        - --path=pd:2379
```

是不是漏掉了什么？

目前这样一套方案已经可以 work 了，而也就是用来体验下功能还差不多。如果考虑上生产环境，那可能是另外一个故事了。

是不是漏掉了什么？

太多细节要考虑:

- 配置开启 TLS
- 优化 K8s Service 性能，比如：externalTrafficPolicy
- 调度把 Pod 尽量打散，降低单点故障的影响
- 探测应用程序的健康状态
- …

是不是漏掉了什么？

仍然有很多问题难以解决:

- Local SSD 是和 Node 物理绑定，这会导致 K8s 的自动故障转移失效
- K8s 不会自动帮助我们处理 PD 和 TiKV 之间的关系
- …

是不是漏掉了什么？

有太多的 TiDB 领域知识需要我们人肉处理

是不是漏掉了什么？

有太多的 TiDB 领域知识需要我们人肉处理

So, 编程来实现它

Part III: TiDB Operator

在 K8s 上跑 TiDB
Round 2

面向生产环境

PingCAP TiDB

回忆一下:
如何告诉 K8s 我们需要一些 TiKV 的 Pods ？

通过提交一个 StatefulSet 对象来描述它

Q: 那么如果我们想要一个完整的 TiDB 集群呢?

Q: 那么如果我们想要一个完整的 TiDB 集群呢?

A: 如果有一个叫 “TiDB cluster” 的 API 对象就好了

自定义资源对象

- 打个比方： 如同面向对象编程的类定义和对象实例之间的关系

```yaml
apiVersion: apiextensions.k8s.io/v1beta1
kind: CustomResourceDefinition
metadata:
  name: tidbclusters.pingcap.com
spec:
  group: pingcap.com
  scope: Namespaced
  names:
    plural: tidbclusters
    singular: tidbcluster
    kind: TidbCluster
    shortNames:
    - tc
  validation:
    openAPIV3Schema:
```

资源类型定义

```yaml
kind: TidbCluster
metadata:
  name: aylei-tidb
spec:
  schedulerName: tidb-scheduler
  pd:
    image: pingcap/pd:v2.1.0
    replicas: 3
  tidb:
    image: pingcap/tidb:v2.1.0
    maxFailoverCount: 3
    replicas: 4
  tikv:
    image: pingcap/tikv:v2.1.0
    replicas: 5
```

资源实例

PingCAP TiDB

...然而, K8s 还不能理解 TiDB Cluster 究竟是个啥

...然而, K8s 还不能理解 TiDB Cluster 究竟是个啥

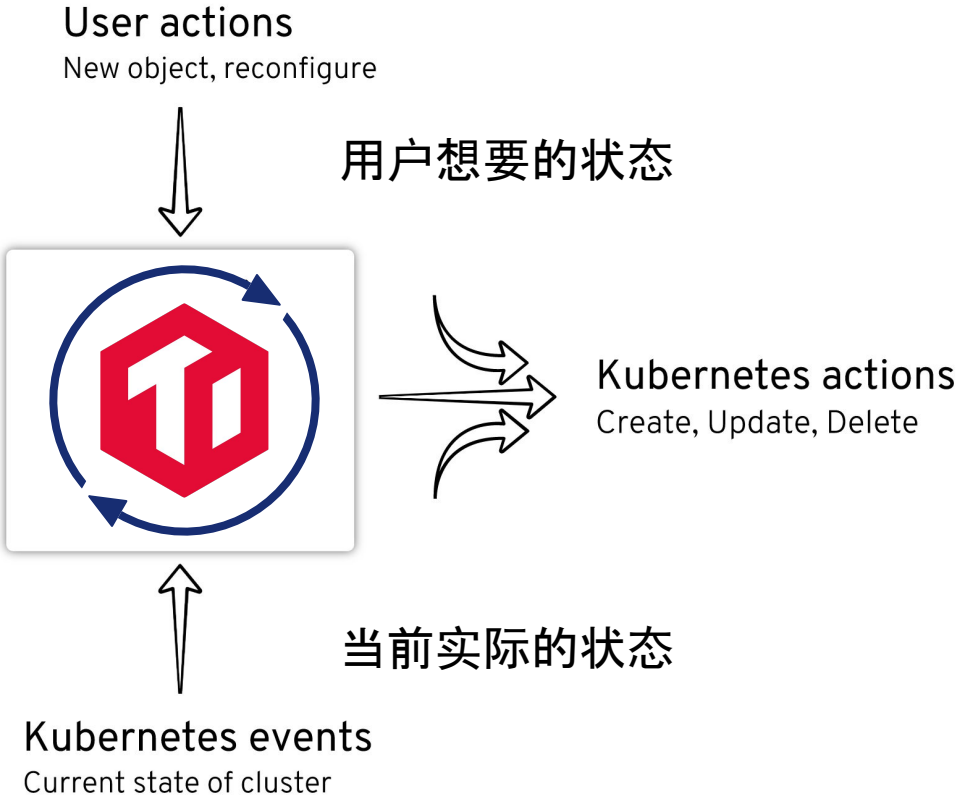
好吧，那我们自己写一个 Controller 吧

TiDB Operator

这样就是一个 “Operator” 了：

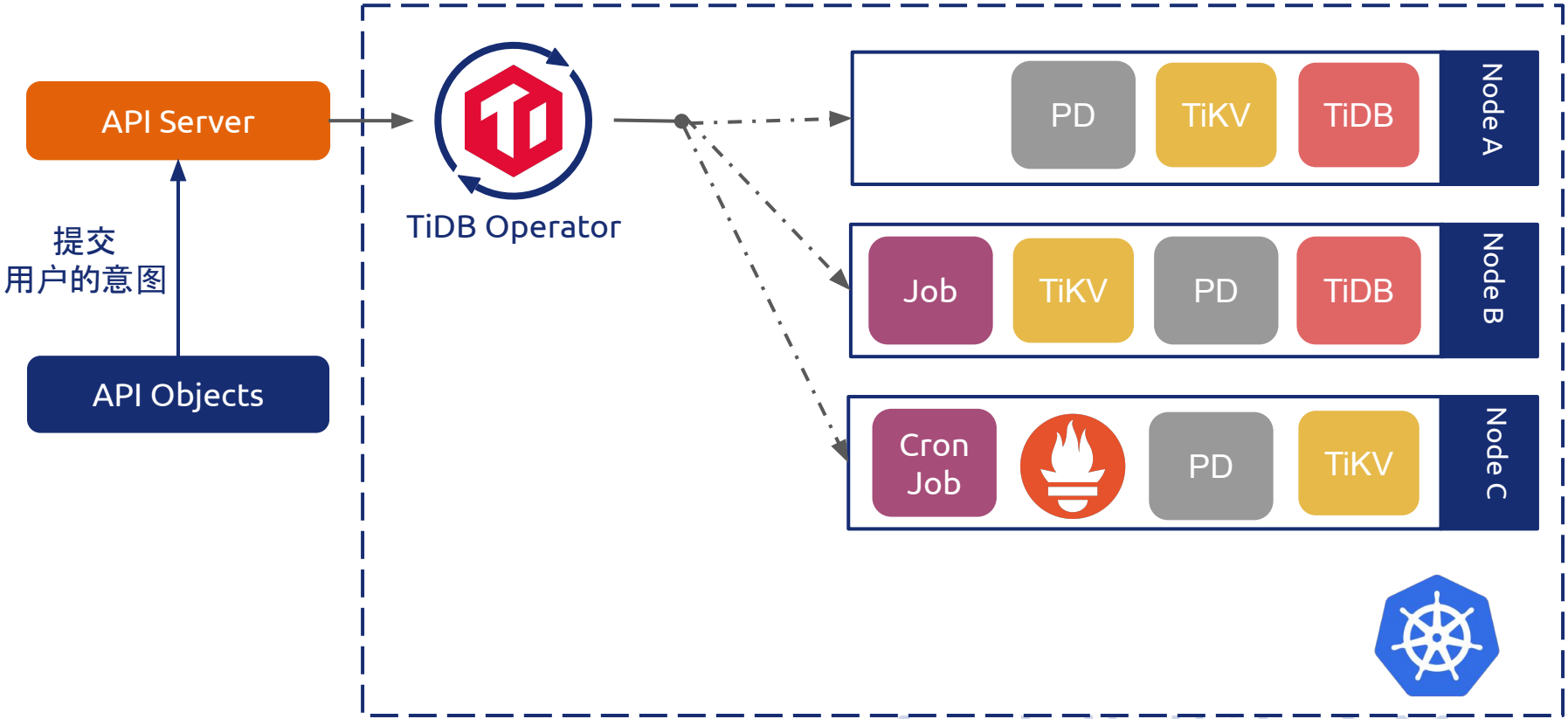
- 定义你的资源对象, 写一个 *CustomResourceDefintion*.
- 写一个 Controller 来完成用户期望资源对象干的事
  - TiDB Operator 里面有好几个 controller，都跑在一个进程里面, 所以称作 “*tidb-controller-manager*”.

TiDB Operator: 控制循环



PingCAP TiDB

TiDB Operator: 控制循环



定义了很多个 CRDs

- TidbCluster
  - 我想要一个 “像这样的” TiDB 集群
- TidbMonitor
  - 我想要监控 “A” 和 “B” 两个集群
- TidbClusterAutoScaler
  - 当满足某些条件时我希望让我的集群自动扩容
- TidbInitializer
  - 我想让我的集群初始化成这个样子

Examples

```yaml
apiVersion: pingcap.com/v1alpha1
kind: TidbCluster
metadata:
  name: awesome-cluster
spec:
  version: v3.0.8
  pd:
    replicas: 3
    requests:
      storage: "10Gi"
  tikv:
    replicas: 3
    requests:
      storage: "500Gi"
  tidb:
    replicas: 2
    service:
      type: ClusterIP
```

```yaml
apiVersion: pingcap.com/v1alpha1
kind: TidbMonitor
metadata:
  name: awesome-monitor
spec:
  clusters:
  - name: awesome-cluster
```

PingCAP TiDB

Examples

```
➜  kubectl get po
NAME                                       READY   STATUS    RESTARTS   AGE
awesome-cluster-discovery-577775b856-5f54z   1/1     Running   0          18h
awesome-cluster-monitor-554698b76c-77d64     3/3     Running   0          18h
awesome-cluster-pd-0                         1/1     Running   0          18h
awesome-cluster-pd-1                         1/1     Running   0          18h
awesome-cluster-pd-2                         1/1     Running   1          18h
awesome-cluster-tidb-0                       2/2     Running   0          18h
awesome-cluster-tikv-0                       1/1     Running   0          18h
awesome-cluster-tikv-1                       1/1     Running   0          18h
awesome-cluster-tikv-2                       1/1     Running   0          18h

➜  kubectl get statefulset,deployment
NAME                                        READY   AGE
statefulset.apps/awesome-cluster-pd         3/3     18h
statefulset.apps/awesome-cluster-tidb       1/1     18h
statefulset.apps/awesome-cluster-tikv       3/3     18h

NAME                                                  READY   UP-TO-DATE   AVAILABLE   AGE
deployment.extensions/awesome-cluster-discovery       1/1     1            1           18h
deployment.extensions/awesome-cluster-monitor         1/1     1            1           18h
```

PingCAP TiDB

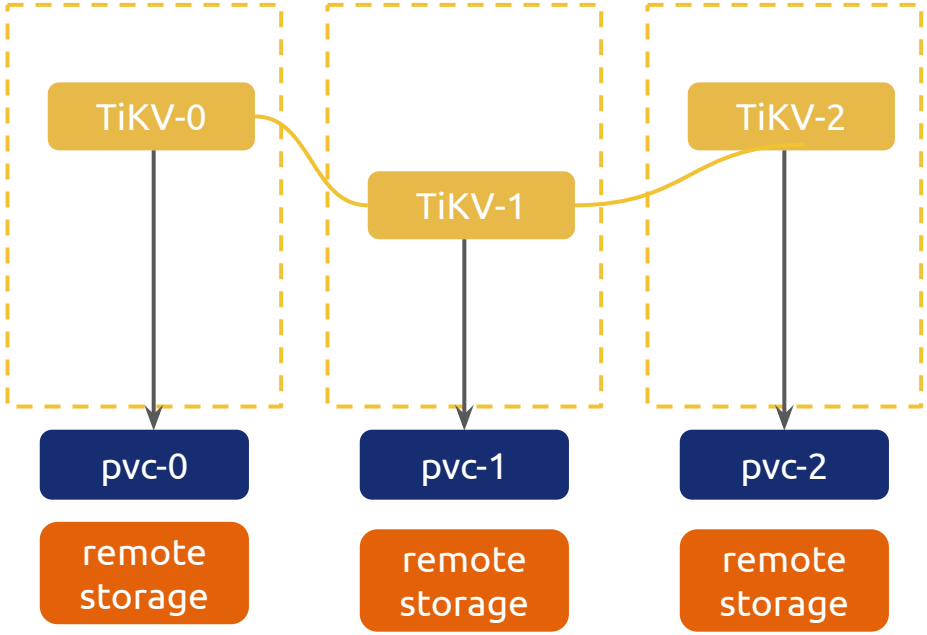
再深入一些

- Controller 到底做了哪些事？
- 其实跟前一章介绍的通过人肉做的事非常类似
  - 创建并更新 *Deployment/StatefulSet*
  - 生产环境需要关心的细节，比如：配置 TLS，保证 Pods 分布足够分散
  - 监控集群状态并处理自动故障转移、自动扩缩容等

再深入一些

For example.

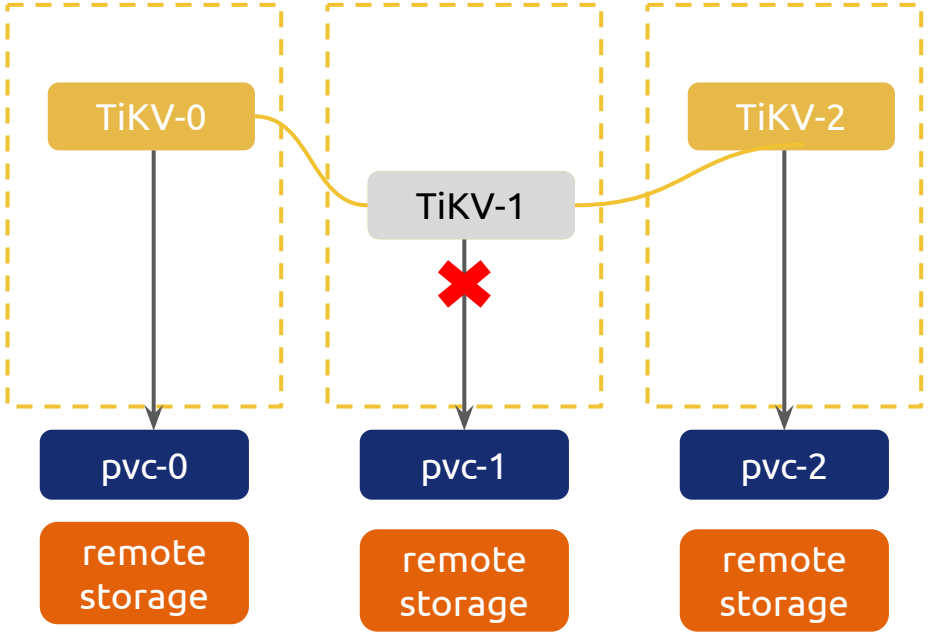
自动故障恢复



3 个 TiKV Pods 使用网络盘

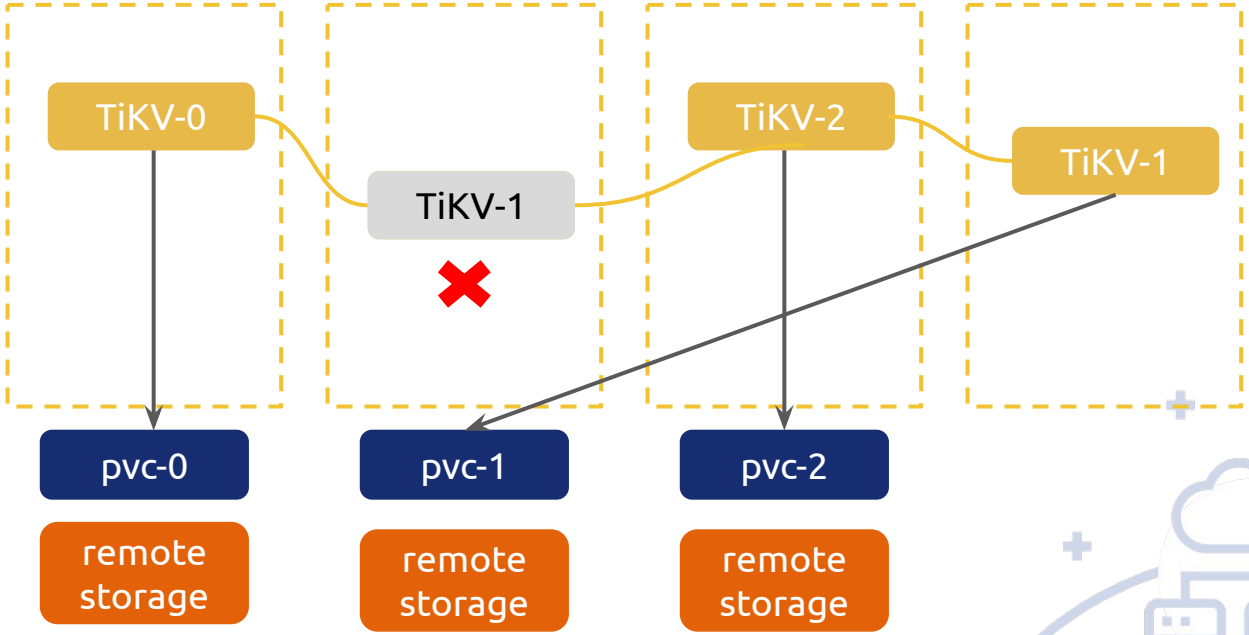
自动故障恢复

一个 Node 挂了



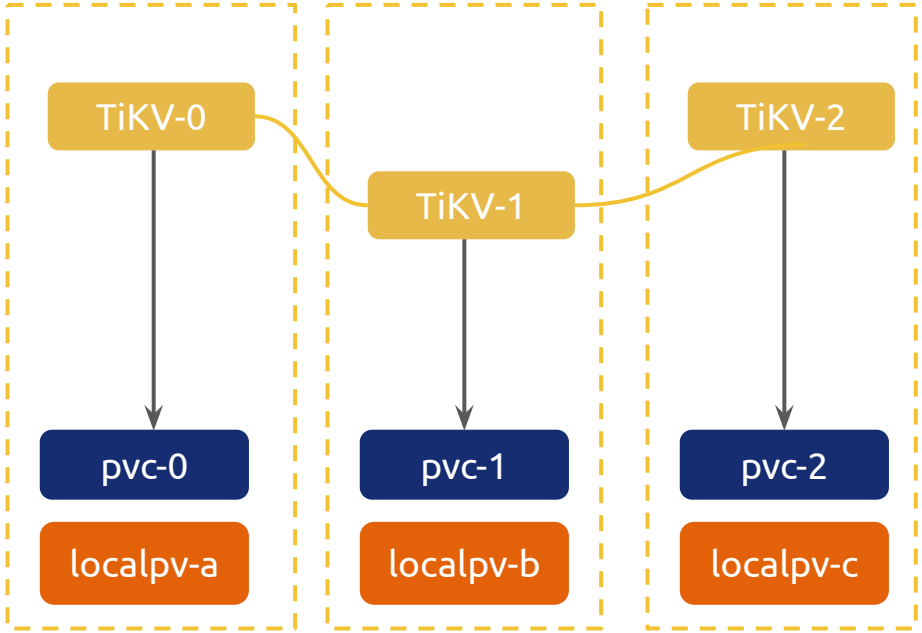
PingCAP TiDB

自动故障恢复



很容易将 Pod 转移到其他 Node，但仍然关联原来的存储（因为是网络盘）

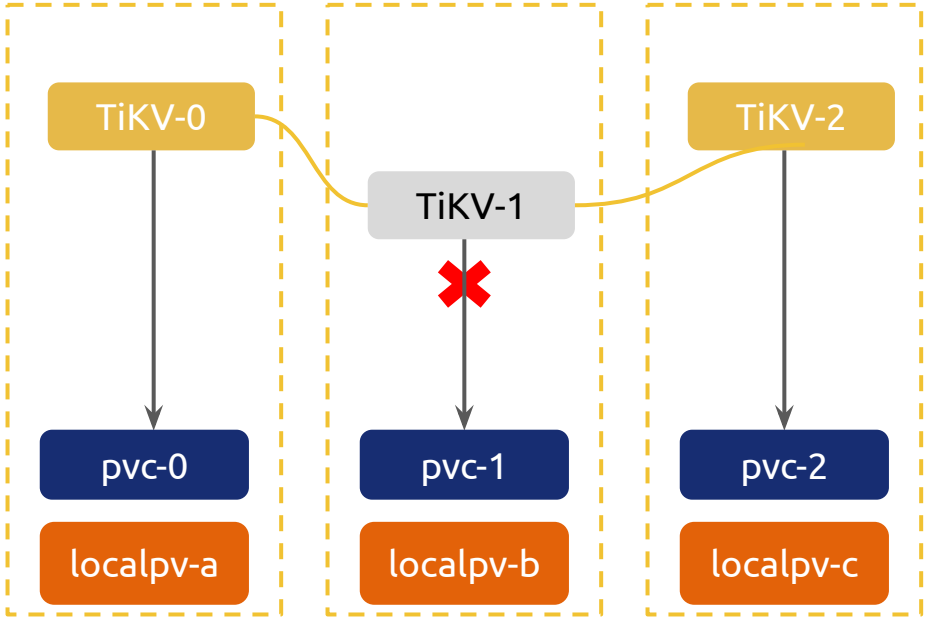
自动故障恢复



Local SSD 带来高 IO 性能, 不过:

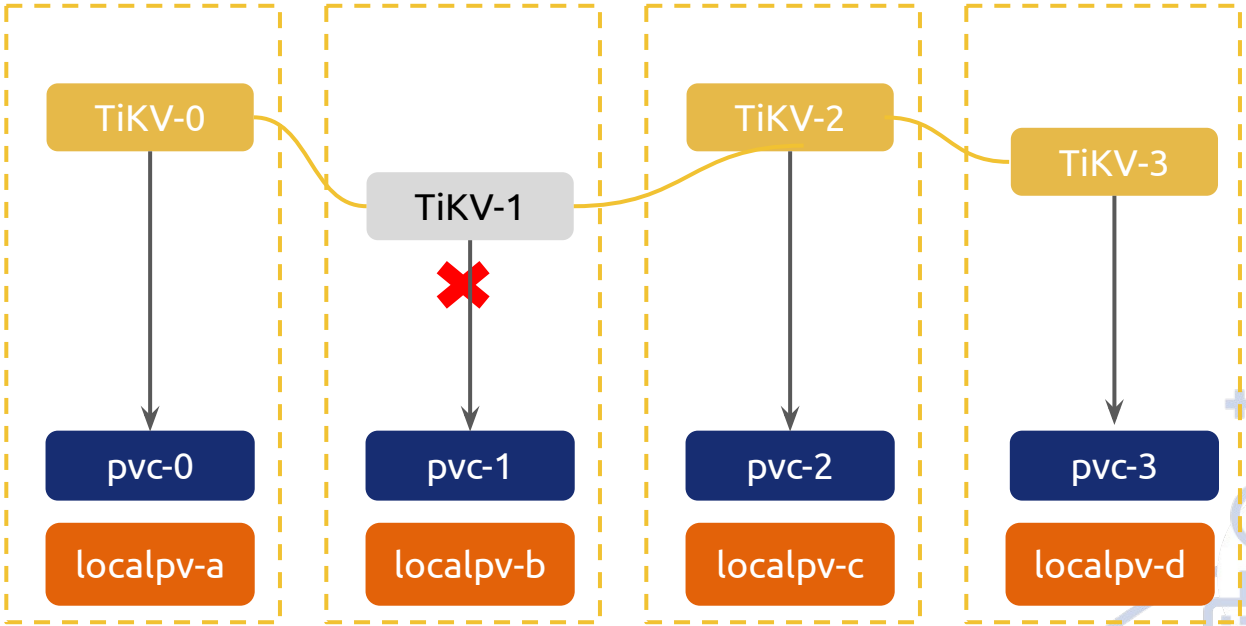
- 本地盘的生命周期跟 Node 绑定在一起
- 跟网络盘相比，本地盘存在易失性

自动故障恢复



当 TiKV-1 所在 Node 挂了，直接转移 Pod 到其他 Node 行不通，因为本地盘只有在挂掉的 Node 上才能用（盘不能飘走）

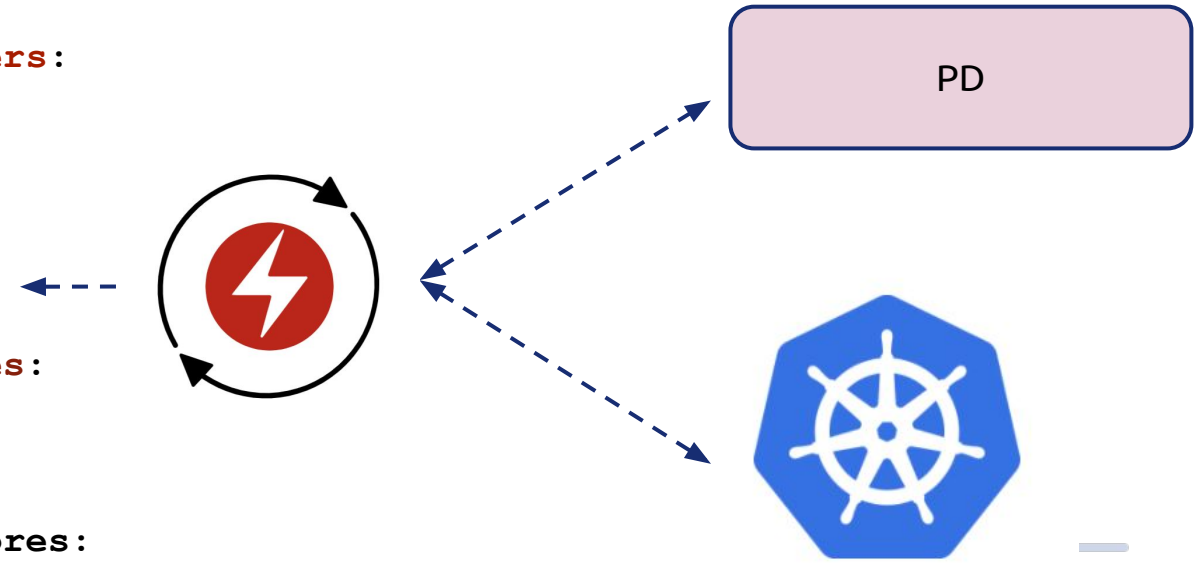
自动故障恢复



这种情况下，TiDB Operator 会增加一个新的 TiKV store，并且由 TiKV 内部的复制来迁移 Region 副本到新增加的 store（补副本操作）

自动故障恢复

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



Controller 同时监控 K8s 和 PD 的状态，从而自动发现故障并采取行动

要点回顾

- 自定义资源对象
- 自定义 Controllers 和 Control Loop 的逻辑
- 现在我们可以通过一个单独的 YAML 文件来管理一个 TiDB 集群
- 你可以定义自己的资源对象，并写入你的 Controller 逻辑，这样就实现了一个 “Operator”

  feel free to hack!

Part IV: TiDB on Cloud

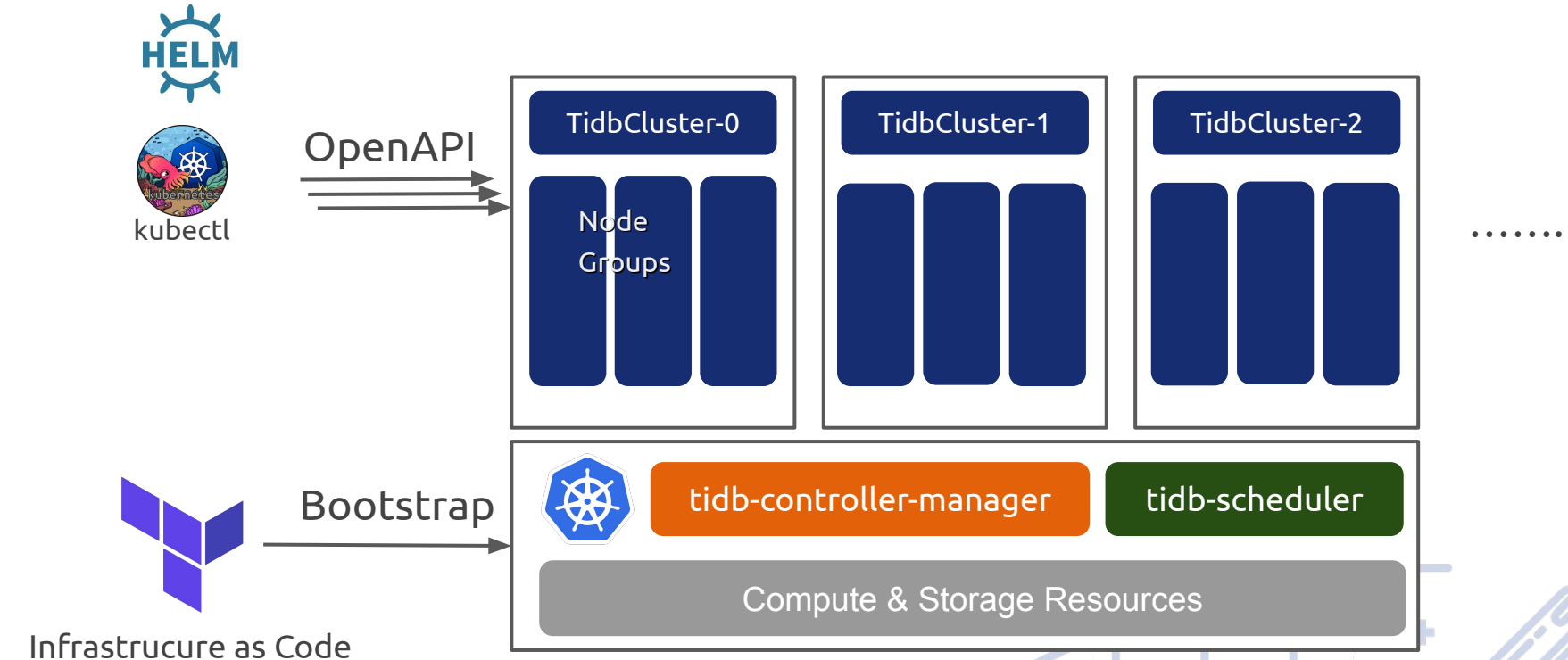
在公有云上部署 TiDB

PingCAP TiDB

公有云部署

- Bootstrap:
  - 环境置备
    - VPC, VMs, LoadBalancer…
    - Kubernetes
  - 安装 TiDB Operator
- TiDB 集群管理:
  - 通过 K8s API
  - 常用的命令行客户端:
    - kubectl
    - K8s 包管理器, 比如：helm

公有云部署



PingCAP TiDB

公有云部署

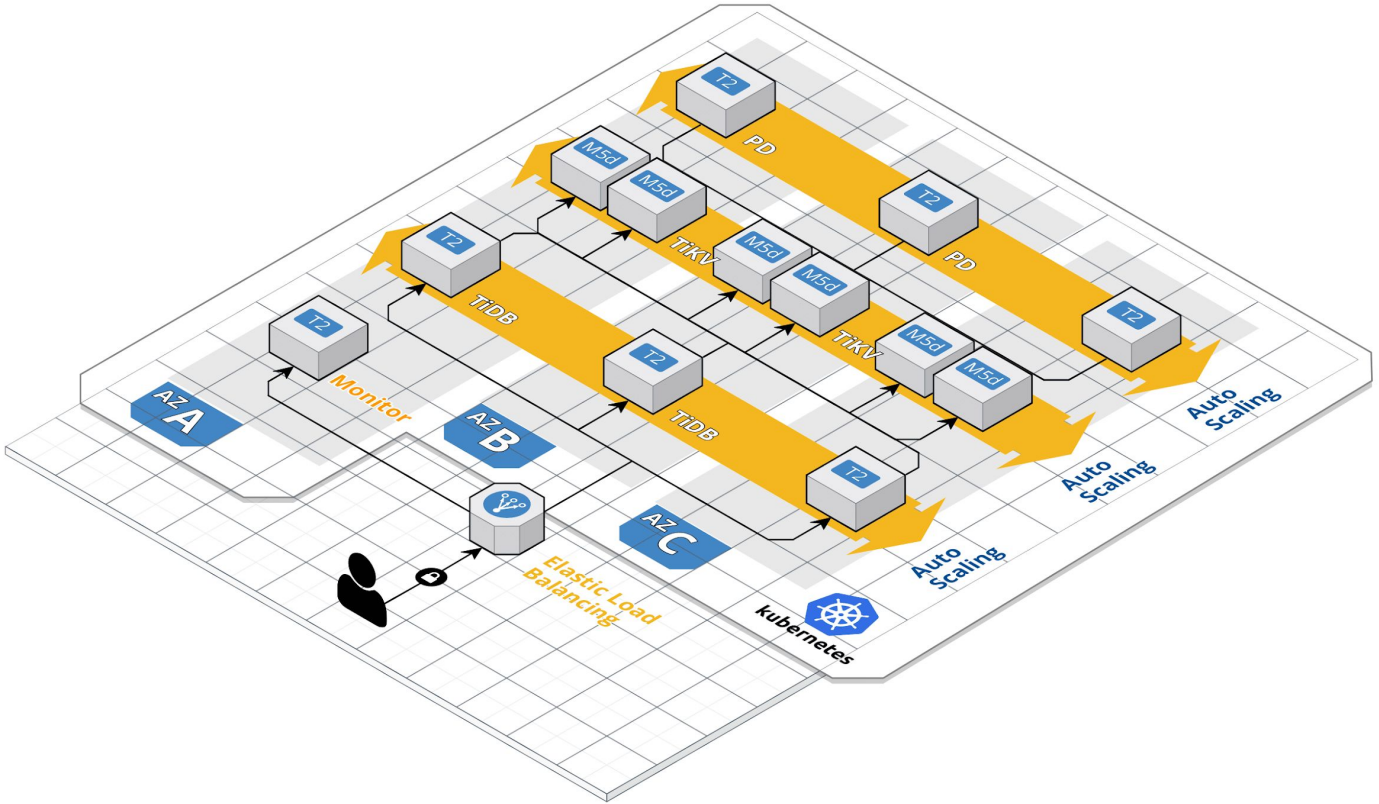
- 跨可用区部署对于生产环境来说至关重要
- 我们提供 terraform 脚本用于 AWS, GCP 和阿里云环境的一键部署，以及最佳实践，敬请参考
  - https://github.com/pingcap/tidb-operator/tree/master/deploy

aws

Google Cloud

Alibaba Cloud

公有云容灾示意图

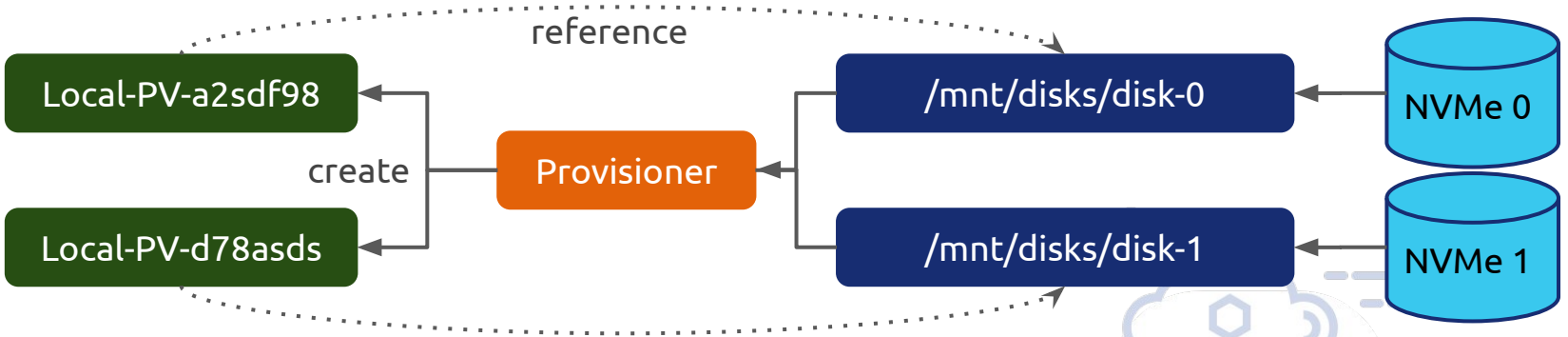


PingCAP TiDB

性能:挂载本地盘

- 本地盘对于发挥 TiDB 性能尤为重要
- 安装 local-volume-provisioner 组件实现对本地盘的管理:

```
kubectl apply -f manifests/local-dind/local-volume-provisioner.yaml
```



性能：使用 HostNetwork

- 如果一个 Node 仅部署一个数据库的 Pod，考虑使用 HostNetwork 来消除虚拟网络的额外损耗
  - 在公有云上资源已经被切分，因此可以做到 Node 的规格等于 Pod 的规格

```
apiVersion: pingcap.com/v1alpha1
kind: TidbCluster
metadata:
  name: awesome-cluster
spec:
  version: v3.0.8
  hostNetwork: true
  pd:
```

Conclusion

Conclusion

- TiDB 跟 K8s 很般配，因为他们有近似的 cloud-native 原则
- 回顾 *Pod*, *StatefulSet*, *Service*，他们提供的原语更适合 TiDB 的自动化
- 回顾 TiDB Operator 的工作原理: 它完全是由 Custom Resource Definitions 和自定义的 Controllers 组成
- 快速在云上部署一个生产可用的 TiDB 集群

Q & A

PingCAP

Thank You !