90Kubernetes 系列（八十四）什么是 Operator 模式？

自动化任务总是特别的。当我们想要执行某些任务时，我们需要能够对某些特定事件做出反应或被触发。但是很多事件不能轻易监听，尤其是在 Kubernetes 集群中。所以今天，我们将看看如何尝试使用 Operator 模式来解决它。

> Operator 是 Kubernetes 的软件扩展，它利用自定义资源来管理应用程序及其组件。 Operator 遵循 Kubernetes 原则，尤其是控制循环。

什么是 Operator 模式？

这种模式允许 Kubernetes 用户创建自己的资源控制器，以便自动管理其应用程序/产品堆栈。

Operator 模式使用 CRD（自定义资源定义）来促进资源/任务配置。

这是一个来自 Strimzi 的 CRD 示例，它让我们创建了一个完整的 Kafka 集群。

```
apiVersion: kafka.strimzi.io/v1beta2
kind: Kafka
metadata:
  name: my-cluster
spec:
  kafka:
    version: 3.4.0
    replicas: 3
    listeners:
      - name: plain
        port: 9092
        type: internal
        tls: false
      - name: tls
        port: 9093
        type: internal
        tls: true
    config:
      offsets.topic.replication.factor: 3
      transaction.state.log.replication.factor: 3
      transaction.state.log.min.isr: 2
      default.replication.factor: 3
      min.insync.replicas: 2
      inter.broker.protocol.version: "3.4"
    storage:
      type: jbod
      volumes:
      - id: 0
        type: persistent-claim
```

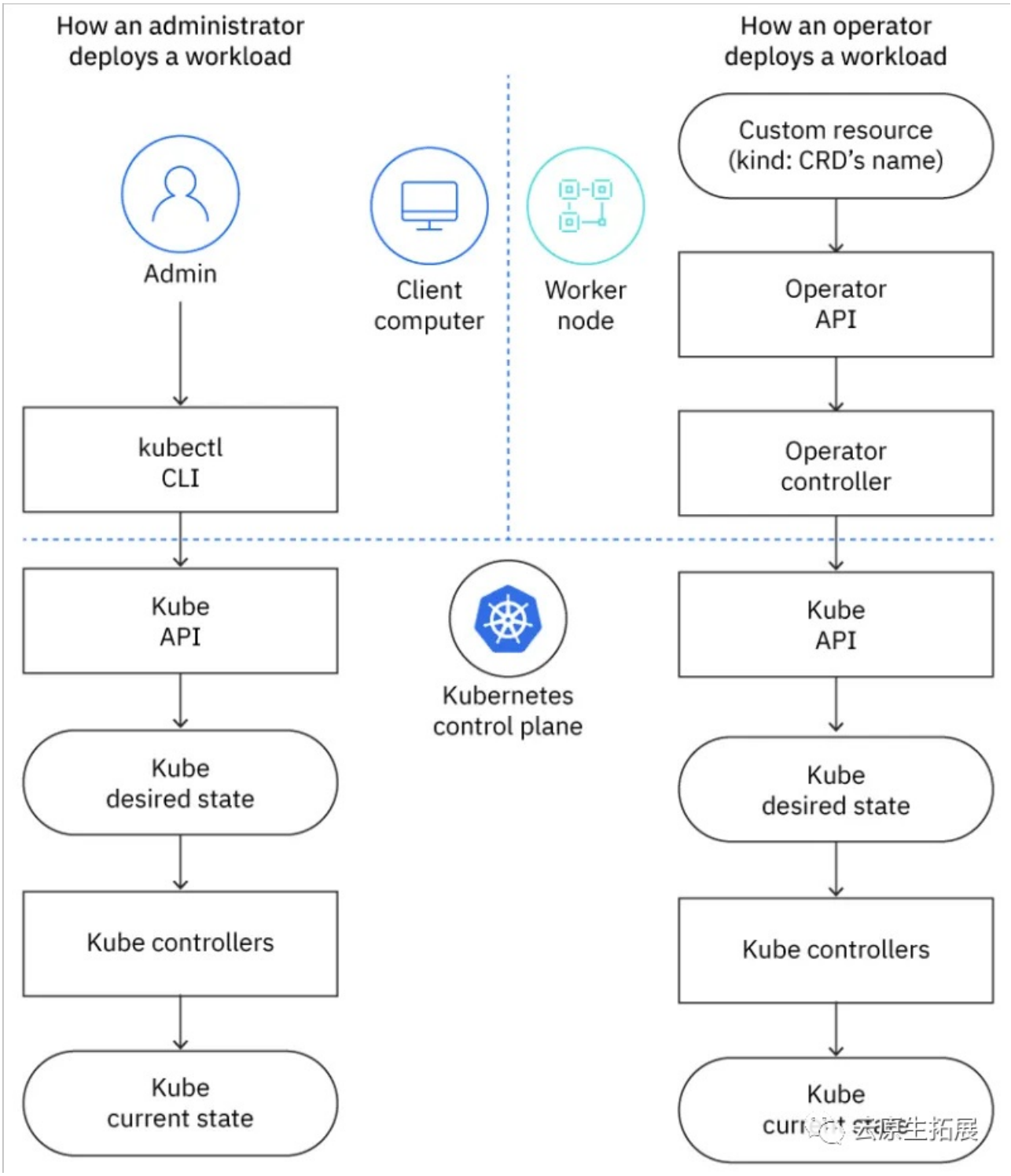
```
        size: 100Gi
        deleteClaim: false
  zookeeper:
    replicas: 3
    storage:
      type: persistent-claim
      size: 100Gi
      deleteClaim: false
  entityOperator:
    topicOperator: {}
    userOperator: {}
```*Exemple d'un CRD de Strimzi permettant de créer un cluster Kafka en quelques lignes.*

```yaml
apiVersion: kafka.strimzi.io/v1beta2
kind: Kafka
metadata:
  name: my-cluster
spec:
  kafka:
    version: 3.4.0
    replicas: 3
    listeners:
      - name: plain
        port: 9092
        type: internal
        tls: false
      - name: tls
        port: 9093
        type: internal
        tls: true
    config:
      offsets.topic.replication.factor: 3
      transaction.state.log.replication.factor: 3
      transaction.state.log.min.isr: 2
      default.replication.factor: 3
      min.insync.replicas: 2
      inter.broker.protocol.version: "3.4"
    storage:
      type: jbod
      volumes:
      - id: 0
        type: persistent-claim
        size: 100Gi
        deleteClaim: false
  zookeeper:
    replicas: 3
    storage:
      type: persistent-claim
      size: 100Gi
      deleteClaim: false
```

```
  entityOperator:
    topicOperator: {}
    userOperator: {}
```

由于它遵循许多 Kubernetes 原则作为控制循环，因此您的 Operator 将（取决于您的配置和您在代码中编写的内容）能够监视它已创建的资源和/或集群上的其他资源。

这是来自 RedHat 博客的方案，说明了上下文。



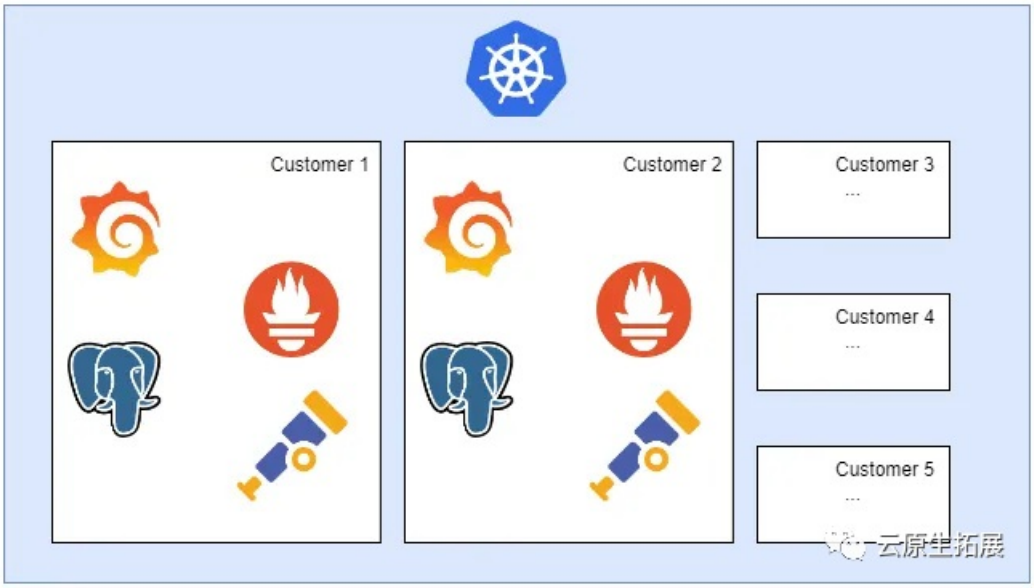
案例

为了更具体，这里有一些例子。

1. 自动化避免重复多次部署

如果您所在的团队部署并授予公司每个团队访问完整工具堆栈来收集和可视化其指标的权限，则您需要为每个团队手动部署以下堆栈：

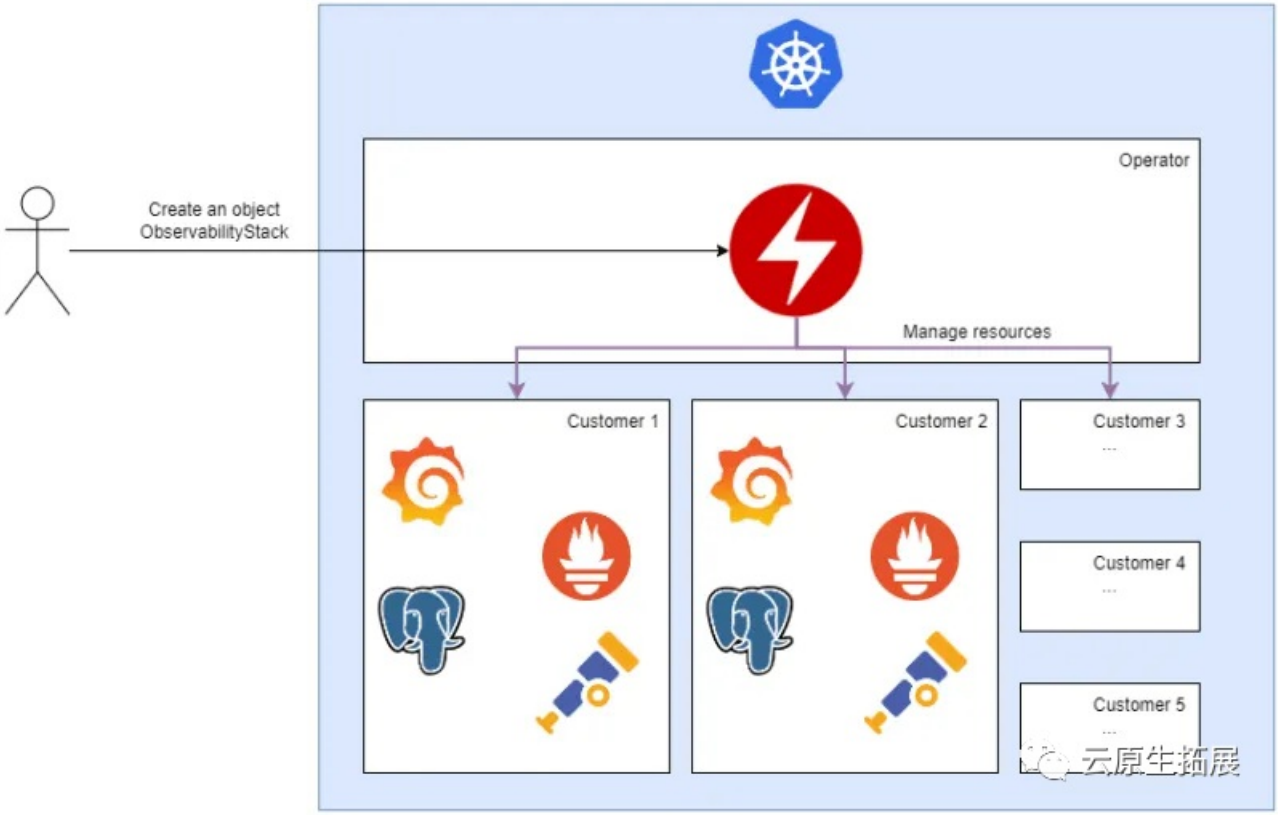
- prometheus
- OpenTelemetry
- Grafana
- Postgres



在这种情况下，您可以创建一个 Operator 来让它管理，而不是重复相同的任务来部署所有这些应用程序和所有相关组件（服务、配置映射、服务帐户...）

这种任务为您提供了一个 CRD 来包含所有可以更新的配置。

例如，CRD 可以命名为 `ObservabilityStack`，每次创建该资源的新实例时，它都会自动创建您定义的所有资源。



2. 自动化避免人工错误

已经管理过 `StatefulSet` 的人知道我会说什么。

当我们管理一些应用程序（尤其是带有卷的应用程序）时，我们可能需要按照特定的顺序执行一些特定的任务，以便创建、更新或删除某些内容。

因此，如果您只有 3 或 4 个此类资源，则使用 bash 脚本或 ansible 脚本来完成此操作可能是正确的。但如果你有50个呢？100？或者更多？

在这种情况下，Operator 模式也可以为您提供帮助。如果您能够定义需要执行的所有任务，它们将在所需的情况下执行。

因此，在更新过程中，该应用程序将不再依赖于您。它会让你的生活更轻松。

3. 自动化配置

在此示例中，假设您在一个管理 Nginx 的团队中，该 Nginx 公开了您公司的所有 API。所有 API 和 nginx 都位于同一个 Kubernetes 集群中。部署新端点后，您需要在所有环境中使用新端点更新配置文件。此外，您的公司喜欢微服务，因此您每周都会有新的 API 和更新。

其中一些还被重命名、移动甚至删除。但您并不总是处于循环之中，因此，如果一个 api 不再工作，您会收到电话以了解发生了什么情况。

正如您已经了解的那样，Operator 是在这种情况下为您提供帮助的解决方案。由于您能够跟踪所有集群上的资源，因此您可以查看是否添加、重命名或删除了某些部署！因此，有了这个，您就可以在发生此类事件时触发，并且可以更新您的配置文件！

总结

通过所有这些示例，我想您已经理解了该模式的原理和用途。

现在，如果我们想更进一步，我们需要把手放在代码中。所以我们将在后续的文章中看到如何创建一个 Operator！

我希望它能对你有所帮助，如果你有任何问题（没有愚蠢的问题）或某些点你不清楚，请不要犹豫，在评论中添加你的问题。