容器监控

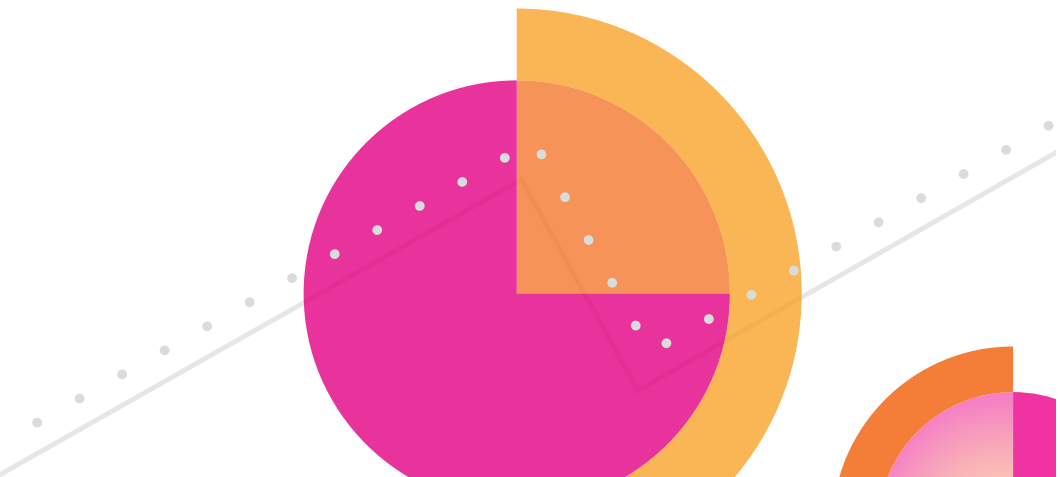
基本指南

Kubernetes 监控初学者指南

自 2013 年引入这一概念以来，容器已经成为 IT 世界的热门话题。很容易看到其中原因：应用程序容器技术正在彻底改变应用程序开发，给开发过程带来了以前无法想象的灵活性和效率。

各个企业纷纷接纳容器。据 Gartner 称，到 2020 年，超过一半的全球企业将在生产中运行容器化应用程序，而现在这一比例还不到 20%。IDC 预测，到 2021 年，超过 95% 的新微服务将部署在容器中。这种大规模采用清楚地表明，组织需要采用基于容器的开发方法来保持竞争力。

为此，让我们看看容器化涉及到什么，以及您的组织如何利用它来获得优势。

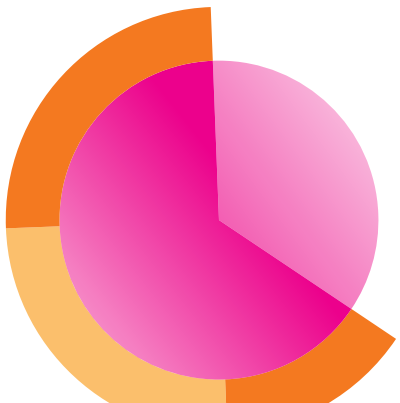


什么是容器?

理解容器概念最简单的方法是考虑它的同名物。物理容器是用来将货物从一个地方存放和运输到另一个地方的容器。

软件容器执行类似的功能。它允许您将应用程序的代码、配置文件、库、系统工具和执行该应用程序所需的所有其他东西打包成一个独立的单元，以便您可以在任何地方移动和运行它。

此外，容器支持"微服务"方法，将应用程序分解为单功能模块，只有在需要时才能访问。这允许开发人员在需要更改时修改和重新部署特定的服务，而不是整个应用程序。



为什么容器如此重要?

容器解决了 IT 操作中一个非常常见的问题: 无论软件部署在哪里, 都可以让软件可靠而一致地运行。当应用程序从一个计算环境移动到另一个计算环境时, 例如从筹划到生产, 如果操作系统、网络拓扑、安全策略或环境的其他方面不同, 它可能会遇到问题。容器将应用程序与其环境隔离开来, 抽象出这些环境差异。

在容器出现之前, 虚拟机 (VM) 是在单个服务器上运行许多独立应用程序的主要方法。像容器一样, 虚拟机抽象出机器的底层基础架构, 这样硬件和软件的变化就不会影响应用程序的性能。但是每台虚拟机如何做到这一点却有很大的不同。

虚拟机抽象硬件, 将一台物理服务器转换成几台虚拟服务器。它是通过在虚拟机管理程序上运行来实现的, 虚拟机管理程序本身运行在一台名为"主机"的物理计算机上。虚拟机管理程序本质上是一个模拟主机资源 (CPU、RAM 等) 的协调系统, 使虚拟机或"来宾机器"可以使用它们。应用程序和运行它们所需的一切, 包括库和系统二进制文件, 都包含在来宾机器中。每台来宾机器还包括自己的完整操作系统。例如, 运行四个虚拟机的服务器, 除了虚拟机管理程序协调它们之外, 还会有四个操作系统。这对于一台机器的资源来说是一个很大的需求, 事情可能会很快陷入停滞, 最终限制了一台服务器可以运行的虚拟机数量。

另一方面, 容器在操作系统级别是抽象的。单个主机操作系统在主机上运行 (可以是物理服务器、虚拟机或云主机) , 容器 (使用像 Docker 引擎这样的容器化引擎) 与其他容器共享操作系统的内核, 每个容器都有自己独立的用户空间。这里的开销比虚拟机少得多, 因此, 容器比虚拟机轻得多, 资源效率也高得多, 从而可以更好地利用服务器资源。

部署容器的 5 个好处

基于容器的基础设施提供了许多好处。
以下是最大的五个好处。

1. **交付速度** — 安装在虚拟机上的应用程序通常需要几分钟才能启动。容器不必等待操作系统启动，所以它们可以在几分之一秒内启动。它们运行速度也更快，因为它们使用的主机操作系统资源更少，并且创建、克隆或销毁只需几秒钟。所有这些都对开发过程产生了巨大的影响，使组织能够更快地将软件推向市场，修复错误并添加新功能。

2. **开发运维优先** — 容器的速度、小占地面积和资源效率使它们成为开发运维环境的理想选择。基于容器的基础设施使开发人员能够在他们喜欢的平台上尽可能快速高效地工作，而不必在非业务关键任务上浪费时间。

3. **易接近** — 如前所述，容器只打包应用程序及其依赖项。这使得在 Windows、Linux 或 Mac 硬件上移动和可靠运行容器变得容易。容器可以在裸机或虚拟服务器上运行，也可以在公共或私有云中运行。如果您需要将应用程序从一个公共云环境移动到另一个公共云环境，这也有助于避免供应商锁定。

4. **增强的可扩展性** — 容器往往很小，因为它们不像虚拟机那样需要单独的操作系统。一个容器的大小通常是几十兆字节，而一个虚拟机可以是几十千兆字节，大约是一个容器大小的 1000 倍。这种效率允许您在单个主机操作系统上运行更多的容器，提高了可扩展性。

5. **一致性** — 因为容器在内部保留了所有的依赖关系和配置，所以无论容器部署在哪里，它们都可确保开发人员能够在一致的环境中工作。这意味着开发人员不必浪费时间解决环境差异问题，而是可以专注于解决新的应用功能。这也意味着，当需要投入使用时，您可以将同一个容器从开发阶段带到生产阶段。

Kubernetes 和容器基础知识

要开始容器编排，您需要使用专门的软件来部署、管理和扩展容器化应用程序。当今最早也是最受欢迎的选择之一是 Kubernetes，这是一个由 Google 开发的开源自动化平台，现在由 Cloud Native Computing Foundation 管理。

Kubernetes 可以通过简化容器管理、自动更新和最大限度地减少停机时间来显著增强开发过程，这样开发人员就可以专注于改进应用程序并为其添加新功能。为了更好地理解这一点，让我们来看看 Kubernetes 的基本组件以及它们是如何协同工作的。

Kubernetes 使用在其独特语言中定义的多层抽象。Kubernetes 有很多部分。这个列表并不详尽，但是它提供了一个简化视图，让您了解硬件和软件在系统中是如何表示的。

节点：在 Kubernetes 行话中，任何一台“工作机”都是一个节点。它可以是云提供商 (例如 AWS 或 Microsoft Azure) 上的物理服务器或虚拟机。节点最初被称为“杂役”，这让您知道它们的用途。它们接收并执行从主节点分配的任务，并包含管理资源和向容器分配资源所需的所有服务。

主节点：这是协调所有工作节点的机器，也是您与 Kubernetes 互动的点。所有分配的任务都源于此。

集群：集群代表一个主节点和几个工作节点。集群将所有这些机器整合成一个强大的单元。容器化的应用程序被部署到集群中，集群将工作负载分配给不同的节点，随着节点的添加或删除而转移工作。

Pod：Pod 表示打包在一起并部署到节点的容器集合。Pod 中的所有容器共享本地网络和其他资源。它们可以像在同一台机器上一样互相交谈，但它们仍然彼此隔离。同时，Pod 将网络和存储与底层容器隔离开来。

单个工作节点可以包含多个 Pod。如果某个节点发生故障，Kubernetes 可以向正常运行的节点部署替换 Pod。

尽管 Pod 可以容纳许多容器，但建议它们只包装需要的容器：一个主过程和它的辅助容器，称为“侧车”。无论个体需求是什么，Pod 都会作为一个单元扩展，过度填充的 Pod 会消耗资源。

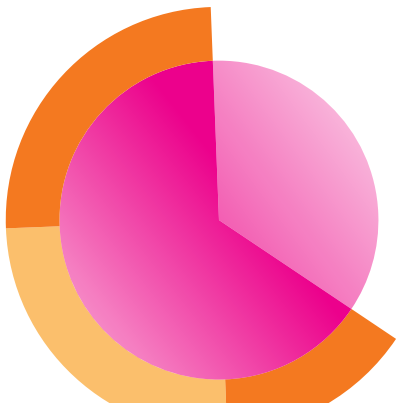
部署：Kubernetes 没有直接将 Pod 部署到集群，而是使用了一个称为“部署”的附加抽象层。 部署使您能够指定要同时运行的 Pod 副本的数量。一旦它将该数量的 Pod 部署到集群中，它将继续监视它们，并在失败时自动重新创建和重新部署 Pod。

入口：Kubernetes 将 Pod 与外部世界隔离开来，因此您需要为任何想要公开的服务打开通信通道。这就是叫做“入口”的另一个抽象层。有几种方法可以向集群添加入口，包括添加负载平衡器、节点端口或入口控制器。

典型的 Kubernetes 部署是什么样的?

熟悉 Kubernetes 的基本组件应该会让您对系统如何处理容器编排有所了解，但是更深入的理解需要可视化这些组件中的每一个。在这里，我们将使用 Google 工程师 Daniel Sanche 的一个例子，来看看如何使用 Kubernetes 将应用程序部署到 Google Cloud 的集群中。

Sanche 的教程使用 Gitea (开源 git 托管服务) 作为部署的应用程序，但是正如他指出的，这个演练实际上可以用于任何应用程序。



创建集群

设置 Kubernetes 环境时，两个命令至关重要: *kubectl* 和 *gcloud*。*kubectl* 是与 Kubernetes API 交互的主要工具，用于创建和管理软件资源，例如 Pod 和部署。但是，由于 Kubernetes 与平台无关，因此 *kubectl* 命令不能直接从您选择的云提供商配置节点，因此需要第三方工具。例如，如果 Google Cloud 是您的提供商，您可以使用 Google Kubernetes 引擎的 gcloud 命令来配置您的节点。

一旦您设置了 Kubernetes 环境，这些命令将用于创建由三个节点组成的默认集群:

```
gcloud container clusters create my-cluster
--zone us-west1-a
gcloud container clusters get-credentials my-
cluster \
--zone us-west1-a
```

您的集群现在可以在 Google Cloud 控制台的 GKE 部分看到。您作为节点配置的虚拟机将出现在 GCE 部分。

部署应用程序

现在，您可以开始为您的实时集群分配资源。虽然您可以用 *kubectl 添加命*令交互地完成这一任务，但 Sanche 建议通过将 Kubernetes 的所有资源写入 YAML 文件来完成。这使您能够将集群的整个状态记录在易于维护的文件中，并将托管服务所需的所有指令与实际代码一起保存，从而简化管理。

要使用 YAML 文件向集群添加 Pod，请创建一个名为 gitea.yaml 的文件，其中包含以下内容：

```
apiVersion: v1
kind:Pod
metadata:
name: gitea-pod
spec:
containers:
   - name: gitea-container
   image: gitea/gitea:1.4
```

这段代码声明您正在创建 Pod，名为 “gite a-pod” ，在 Kubernetes API 的 v1 版本中定义。它包含一个名为 “gitea-container” 的容器。 最后一行定义了要运行的容器图像。这里的图像是在 gitea/gitea 库中标记为 1.4 的图像。Kubernetes 告诉内置容器运行时定位该容器图像并将其添加到 Pod 中。

接下来，通过执行以下命令将 YAML 文件应用到集群：*kubectl apply -f gitea.yaml*。

Kubernetes 将读取文件并将 Pod 添加到集群中。您可以通过运行 *kubectl* get pods 命令来查看新的 Pod。这将返回有关 Pod 状态、它是否已经重新启动以及它已经运行多长时间的数据。

您还可以通过运行命令 *kubectl logs -f gitea-pod* 来查看容器的标准输出，该命令将返回如下内容：

```
Generating /data/ssh/ssh_host_ed25519_key...
Feb 13 21:22:00 syslogd started:BusyBox v1.27.2
Generating /data/ssh/ssh_host_rsa_key...
Generating /data/ssh/ssh_host_dsa_key...
Generating /data/ssh/ssh_host_ecdsa_key...
/etc/ssh/sshd_config line 32:Deprecated option
UsePrivilegeSeparation
Feb 13 21:22:01 sshd[12]:Server listening on :: port 22.
Feb 13 21:22:01 sshd[12]:Server listening on 0.0.0.0
port 22.
2018/02/13 21:22:01 [T] AppPath: /app/gitea/gitea
2018/02/13 21:22:01 [T] AppWorkPath: /app/gitea
2018/02/13 21:22:01 [T] Custom path: /data/gitea
2018/02/13 21:22:01 [T] Log path: /data/gitea/log
2018/02/13 21:22:01 [I] Gitea v1.4.0+rc1-1-gf61ef28
built with: bindata, sqlite
2018/02/13 21:22:01 [I] Log Mode:Console(Info)
2018/02/13 21:22:01 [I] XORM Log
Mode:Console(Info)
2018/02/13 21:22:01 [I] Cache Service Enabled
2018/02/13 21:22:01 [I] Session Service Enabled
2018/02/13 21:22:01 [I] SQLite3 Supported
2018/02/13 21:22:01 [I] Run Mode:Development
2018/02/13 21:22:01 Serving [::]:3000 with pid 14
2018/02/13 21:22:01 [I] Listen:
http://0.0.0.0:3000
```

部署

如前所述，直接在 Kubernetes 内部署 Pod 并不典型，而是使用部署抽象层来代替。为此，您需要后退一步，删除以前用 *kubectl delete -f gitea.yaml* 命令创建的 Pod，以便您可以通过部署层重新创建它。

接下来，回到您最初创建的 YAML 文件，并进行如下所示的修改：

```
apiVersion: extensions/v1beta1
kind:部署
metadata:
name: gitea-deployment
spec:
replicas:1
selector:
matchLabels:
app: gitea
template:
metadata:
labels:
app: gitea
spec:
containers:
- name: gitea-container
image: gitea/gitea:1.4
```

这段代码的前九行定义了部署本身，其余的行定义了部署将管理的 Pod 模板。第 6 行 (副本) 是最关键的信息，因为它告诉 Kubernetes 您想要运行多少个 Pod。

现在，您可以使用命令 *kubectl apply -f gitea.yaml* 来应用修改后的 YAML 文件。

再次键入 *kubectl get pods*，查看正在运行的 Pod。要验证部署信息，请输入 *kubectl get deployments*。

Kubernetes 部署的优势之一是，如果某个 Pod 出现故障或被删除，它将自动重新部署。要查看实际情况，请通过键入 *kubectl delete pod <podname>* 删除您刚刚部署的 Pod，您应该会看到一个新的 Pod 添加到您的集群中。

有关部署应用程序、添加入口以便您可以通过浏览器访问的更多信息，请参阅 Sanche 的完整教程。

您如何有效地监控 Kubernetes?

尽管容器给 IT 组织带来了诸多好处，但它们也可以使基于云的应用程序管理更加复杂。它们带来的一些挑战包括：

- **明显的盲点** — 容器被设计成一次性使用。因此，他们在应用程序和底层硬件之间引入了几个抽象层，以确保可移植性和可扩展性。在常规监控方面，这一切都造成了一个明显的盲点。
- **记录需求增加** — 如此多相互依赖的组件易于移植，增加了维护遥测数据的需求，以确保应用程序、容器和编排平台的性能和可靠性。
- **可视化的重要性** — 容器和容器编排带来的规模和复杂性要求能够可视化环境，以便立即了解基础架构的运行状况，同时能够放大和查看容器、节点和 Pod 的运行状况和性能。正确的监控解决方案应该提供此工作流。
- **不要将开发运维留在黑暗中** — 可以用闪电般的速度缩放和修改容器。这种加速的部署速度使得 DevOps 团队更难跟踪应用程序性能如何在部署中受到影响。

良好的容器监控解决方案将使您能够通过将容器数据与其他基础设施数据统一起来，提供更好的上下文化和根本原因分析，从而保持在动态的基于容器的环境之上。让我们看看如何为 Docker 提供多层监控，Docker 是最流行的容器实施：

主机： 可以监控集群中的物理和虚拟机的可用性和性能。要跟踪的关键指标包括内存、CPU 使用率、已用交换空间和存储利用率。这应该是任何容器监控工具的核心能力。

容器： 对容器整体和单独的可见性至关重要。监控工具可以提供关于当前运行的容器数量、使用最多内存的容器以及最近启动的容器的信息。它还可以洞察每个容器的 CPU 和内存利用率及其网络 I/O 的健康状况。

应用程序端点： 在典型的基于容器的环境中，每个应用程序服务将在一个或多个容器上运行。理想情况下，应用程序监控应该在容器、Pod 和整个系统的级别上执行。

开始**使用**

容器是您的开发“兵器库”中的一个强大工具，但是，了解您的容器环境如何工作以及工作得如何非常重要。有关更多信息，请在线访问我们，了解我们如何帮助您开始使用容器、编排和监控。

splunk>

2020-Splunk-The-Essential-Guide-to-Container-Monitoring-113-EB-web