115Kubernetes 系列（一零八）揭秘 Kubernetes 中的 OOM Killer：追踪内存问题

介绍

您是否遇到过 Kubernetes 中可怕的 OOM（内存不足）杀手？这个关键组件在集群内有效管理内存资源方面发挥着至关重要的作用。然而，当 Pod 超出其内存限制并消耗过多内存时，OOM Killer 就会介入，终止该 Pod 以便为其他关键进程释放内存。在这篇博客中，我们将揭开 OOM 杀手的神秘面纱，深入研究其内部工作原理，并学习如何追踪导致 OOM 杀手的内存问题。那么，让我们深入了解一下吧！

理解 OOM Killer

OOM Killer 是 Kubernetes 中的一个重要机制，有助于维护系统的稳定性并防止内存耗尽。当内存资源严重不足时，它充当最后一道防线。在这种情况下，OOM Killer 会识别导致内存过载的进程或 Pod，并将其终止，以便为系统的其余部分释放内存。 OOM Killer 通过牺牲一个进程，防止系统完全崩溃，保证集群的整体稳定性。

OOM Kill 是如何触发的

当 Kubernetes 中的 pod 超出其指定的内存限制时，它会触发 OOM 事件。容器运行时（例如 Docker）向 Kubernetes kubelet 报告内存使用情况。 kubelet 反过来监视所有 pod 的内存使用情况，并将其与各自的限制进行比较。如果 pod 超出其限制，kubelet 会启动 OOM Killer 来终止有问题的 pod，从而为其他关键工作负载释放内存资源。

了解内存指标和 OOM 决策

为了对 OOM Kill 做出明智的决策，OOM Killer 依赖于从 cAdvisor（容器顾问）获取并公开给 Kubernetes 的内存指标。OOM 杀手使用的主要指标是“container_memory_working_set_bytes”。考虑到容器主动使用的内存页，它表示无法驱逐的内存的估计。该指标充当 OOM Killer 决定是否应终止 pod 的基准。

区分内存指标

虽然“container_memory_usage_bytes”似乎是监视内存利用率的明显选择，但它包括缓存项目，例如文件系统缓存，这些项目可以在内存压力下被逐出。因此，它不能准确反映 OOM 杀手观察到的内存并对其采取行动。另一方面，“container_memory_working_set_bytes”提供了更可靠的内存使用情况指示，与 OOM 杀手监控的内容保持一致。它专注于无法轻易回收的内存。

如何调试内存消耗？

要跟踪应用程序中的内存增长，您可以监视提供内存使用信息的特定文件。通过将以下代码片段部署为应用程序的一部分，您可以在调试模式下调用它来打印内存使用情况：

```
const fs = require('fs');

// Function to read memory usage
function readMemoryUsage() {
  try {
    const memoryUsage = fs.readFileSync('/sys/fs/cgroup/memory/memory.usage_in_bytes', 'utf8');
    console.log(`Memory Usage: ${memoryUsage}`);
  } catch (error) {
    console.error('Error reading memory usage:', error);
  }
}

// Call the function to read memory usage
readMemoryUsage();
```

在这段代码中，我们利用 `fs.readFileSync` 方法来同步读取 `/sys/fs/cgroup/memory/memory.usage_in_bytes` 文件的内容。使用 'utf8' 编码读取文件，将数据解释为字符串。

`readMemoryUsage` 函数读取文件并将内存使用情况记录到控制台。如果在读取过程中发生错误，也会被捕获并记录。

请注意，访问系统文件 `/sys/fs/cgroup/memory/memory.usage_in_bytes` 通常需要提升权限。确保使用必要的权限或以特权用户身份运行 Node.js 脚本。

获取节点堆使用情况的代码

```
const fs = require('fs');

// Function to track memory growth
function trackMemoryGrowth() {
  const memoryUsage = process.memoryUsage();
  console.log(`Memory Usage (RSS): ${memoryUsage.rss}`);
  console.log(`Memory Usage (Heap Total): ${memoryUsage.heapTotal}`);
  console.log(`Memory Usage (Heap Used): ${memoryUsage.heapUsed}`);
}

  trackMemoryGrowth();
});
```

如何使用 kubectl top 监控 K8s 中的内存和 CPU?

“kubectl top”命令是 Kubernetes 中的一个强大工具，可让您监控集群中 pod 和节点的资源使用情况。它提供有关内存和 CPU 利用率的实时信息，使您能够识别潜在的瓶颈、解决性能问题并就资源分配做出明智的决策。我们来探讨一下如何使用 kubectl top 命令来监控 pod 和节点资源使用情况。

监控 Pod 资源使用情况：

监控 Pod 的内存和 CPU 使用情况，可以使用以下命令：

```
kubectl top pod
```

此命令提供当前命名空间中所有 pod 的资源使用情况概览。显示Pod名称、CPU使用率、内存使用率以及相应的资源利用率百分比。

如果你想关注特定的 pod，可以使用 pod 名称作为参数：

```
kubectl top pod <pod-name>
```

此命令提供指定 pod 的详细资源使用信息。

监控节点资源使用情况

要监控集群中节点的内存和CPU使用情况，可以使用以下命令：

```
kubectl top node
```

此命令为您提供集群中所有节点的资源使用情况概览。它提供有关节点名称、CPU 使用率、内存使用率以及相应的资源利用率百分比的信息。

与监控 Pod 类似，您可以指定特定节点来检索详细的资源使用信息：

```
kubectl top node <node-name>
```

总之，“kubectl top”命令是一个很有价值的工具，可让您监控 Kubernetes 集群中 pod 和节点的内存和 CPU 使用情况。通过该命令，您可以洞察资源利用率、检测性能瓶颈、优化资源分配，保障应用的高效运行。

等等，如果我想要我的 docker 环境类似情况怎么办？

`docker stats` 显示从路径 /sys/fs/cgroup/memory 收集的内存数据

```
# similar to top
docker stats --no-stream <container id>
```

在 Linux 上，Docker CLI 通过从总内存使用量中减去缓存使用量来报告内存使用量。 API 不执行此类计算，而是提供总内存使用量和缓存量，以便客户端可以根据需要使用数据。缓存使用情况定义为 cgroup v1 主机上 memory.stat 文件中 total_inactive_file 字段的值。

在 Docker 19.03 及更早版本上，缓存使用情况被定义为 cache 字段的值。在 cgroup v2 主机上，缓存使用情况定义为 inactive_file 字段的值。

memory_stats.usage 来自 /sys/fs/cgroup/memory/memory.usage_in_bytes 。 memory_stats.stats.inactive_file 来自 /sys/fs/cgroup/memory/memory.stat 。

给我打印 docker 容器内存使用情况的节点代码

要使用 docker stats，请使用 Node.js 中的 docker-stats-api 库并打印容器使用情况，您可以按照以下步骤操作：

通过运行以下命令将 docker-stats-api 库安装为 Node.js 项目中的依赖项：

```
npm install docker-stats-api
```

定义一个函数来打印容器的使用情况：

```
function printContainerUsage(containerId) {
  dockerStats.getStats(containerId)
    .then(stats => {
      console.log(`Container Usage (CPU): ${stats.cpu_percent}`);
      console.log(`Container Usage (Memory): ${stats.mem_usage}`);
    })
    .catch(error => {
      console.error('Error retrieving container stats:', error);
    });
}
```

这是完整的示例：

```
const DockerStats = require('docker-stats-api');

const dockerStats = new DockerStats();

function printContainerUsage(containerId) {
  dockerStats.getStats(containerId)
    .then(stats => {
      console.log(`Container Usage (CPU): ${stats.cpu_percent}`);
      console.log(`Container Usage (Memory): ${stats.mem_usage}`);
    })
    .catch(error => {
      console.error('Error retrieving container stats:', error);
    });
}

const containerId = 'YOUR_CONTAINER_ID';
printContainerUsage(containerId);
```

通过执行此代码，您将能够使用 Node.js 中的“docker-stats-api”库检索并打印特定容器的 CPU 和内存使用情况。请记住将“YOUR_CONTAINER_ID”替换为您要监控的容器的实际 ID。

总结

通过了解 Kubernetes 中 OOM killer 的角色和功能，我们获得了有关集群内内存管理的宝贵见解。在本博客中，我们探讨了 OOM killer 如何触发 OOM 终止，并讨论了内存指标（特别是“container_memory_working_set_bytes”）在做出 OOM 决策时的重要性。有了这些知识，您就可以有效地监控和解决导致 OOM 终止的内存问题。