Dieser Artikel beschreibt ein Tool für das Monitoring von Ausführungsumgebungen im Continuous-Integration-Umfeld.

Wie der Name (Kiss - keep it simple and straightforward) schon andeutet, handelt es sich bei dem Tool um ein leichtgewichtiges, einfach gehaltenes Werkzeug nach dem Motto: möglichst wenig Aufwand bei möglichst hohem Nutzen.

Wozu brauche ich KissMon?

Im Gegensatz zu herkömmlichen Monitoring-Ansätzen wie Nagios oder Zenoss betrachtet KissMon nicht nur Devices, sondern auch Umgebungen. Im CI-Kontext ist das z.B. eine Testumgebung, die dazu dient, die zu einer Anwendung definierten Regressionstests auszuführen.

Um den Umgebungsgedanken bewerkstelligen zu können, unterscheidet KissMon nicht nur Devices, sondern auch Aspekte zu einem Device, die überwacht werden können. Z.B. kann es zu einem Server mehrere Datenbank-instanzen geben, die zu verschiedenen Umgebungen gehören können. Angenommen die DB-Instanz DB1 gehört zur Testumgebung T1, die zweite Instanz DB2 – auf demselben Server - zur QS-Umgebung Q1. Wenn jetzt z.B. der Tablespace in DB1 vollläuft, dann wird im Monitoring kein Alert auf Serverebene, sondern auf Umgebungsebene (hier T1) ausgelöst, da der Server an sich ja keine Probleme hat und damit die Umgebung Q1 nicht betroffen ist.

Beim herkömmlichen Monitoring würde stattdessen zum Server quasi ein rotes Lämpchen aufleuchten, das signalisiert, dass an dem Server irgendetwas nicht in Ordnung ist. Ich weiß damit aber noch nicht, welche Umgebungen dadurch tatsächlich betroffen sind. Das Problem ist dabei, dass bei herkömmlichen Monitoring-Ansätzen eine Gruppierung nur auf Device-Ebene möglich ist, was nicht feingranular genug ist, wenn man davon ausgeht, dass heute auf einem Server beliebig viele Anwendungen parallel betrieben werden.

Zusammenspiel mit dem CI-Server

Die Umgebungen werden auch in Jenkins gepflegt. Wenn Jenkins auf einer bestimmten Umgebung ein Deployment durchführt, um z.B. eine bestimmte Revision für den Test zur Verfügung zu stellen, dann informiert Jenkins über den Production State KissMon darüber, dass die Umgebung gerade für einen bestimmten Zeitraum im Wartungs-modus ist. Im Zustand Wartung ist das Alerting für die jeweilige Umgebung abgeklemmt, und im Dashboard wird für die Umgebung der Wartungsmodus angezeigt.

Anwendungsfälle

Folgendes sind mögliche Anwendungsfälle für den Einsatz von KissMon:

- Projekte mit mehreren Umgebungen
- Projekte mit vielen unabhängigen Deployment-Units

Der Einsatz von KissMon kann ab zwei Umgebungen hilfreich sein und ist ab ca. vier Umgebungen definitiv notwendig.

Die Technologie zu KissMon

Das Frontend mit dem Monitoring-Dashboard ist mit AngularJS realisiert, das Backend mit Spring. Die Kommunikation – was naheliegend ist – erfolgt über REST-Schnittstellen. Die Anwendung ist leichtgewichtig (z.B. ist für den Betrieb keine DB notwendig), kann entweder stand-alone oder im Application Server (falls vorhanden) deployed werden.

KissMon ist non-intrusive

Non-intrusive bedeutet hier, dass auf den zu beobachtenden Devices und Applikationen keinerlei Eingriffe erforderlich sind, d.h. es müssen z.B. keine Agenten installiert werden.
KissMon kann damit auch dort verwendet werden, wo keine Root-Rechte vorhanden sind, was dann z.B. der Fall ist, wenn Entwicklung und Administration organisatorisch getrennt sind. Die CI-Umgebungen werden zwar von den Entwicklern genutzt, nicht aber von ihnen aufgesetzt. KissMon ist Bestandteil des Entwicklungsprozesses, nicht der Produktivumgebung.

KissMon-Dashboard

Das Dashboard ist eine freundliche grüne Box. Wenn alles ok ist, kann ich das schon von weitem erkennen. Detailinformationen können via Drill-down auf Device-, Aspekt- oder Umgebungsebene erfragt und beobachtet werden. Über die Checks lassen sich auch die Ressourcen der einzelnen Umgebungen transparent machen.

DSL

Zur Definition von Umgebungen gibt es eine DSL, die ein einfaches und schnelles Definieren ermöglicht. KissMon bildet diese auf ein Objektmodell ab und ermöglicht dadurch eine einfache und schnelle Definition von Umgebungen.

Erweiterbarkeit

Mit Java-Mitteln können eigene Checks geschrieben werden; auf den Zielsystemen muss, wie eingangs schon gesagt, nichts installiert werden. Die Checks können hierarchisch strukturiert werden, d.h. es kann einen Check C1 für ein Device und einen zweiten Check C2 für eine bestimmte Applikation auf dem Device definiert werden. Zu einer bestimmten Umgebung kann ich dann definieren, dass ein Alert erfolgen soll, wenn C1 oder C2 feuern.

Das Wesentliche im Überblick

- umgebungsorientiertes Monitoring
- Wartungsfenster für das Deployment
- non-intrusive, d.h. keine Root-Rechte erforderlich
- logische Sicht statt physische