

corewire

Docker & Kubernetes Troubleshooting

Einführung

Folien-Hinweis

- `Space`, `Page down`: Nächste Folie
- `Page up`: Vorherige Folie
- `ESC`, `o`: Übersicht

[Zur Kapitelübersicht](#)

Vorstellungsrunde

-  Name
-  Job / Aufgabenbereich
-  Vorerfahrungen
-  Erwartungen an den Kurs

Pausen

- Mittagspause: 12 - 13 Uhr
- 5-10 min Pause ca. stündlich

Agenda - Tag 1

1. Troubleshooting Einführung
2. Docker Troubleshooting
3. Kubernetes Developer Troubleshooting

Agenda - Tag 2

1. Kubernetes Developer Troubleshooting
2. Kubernetes Admin Troubleshooting
3. Externe Dienste & Cloud-Integration

Ein Kollege/eine Kollegin kommt zu dir und sagt:

"Es geht nicht."

Was machst du?

- Was geht nicht?
- Seit wann?
- Was hast du geändert?
- Zeig mal.
- Welche Fehlermeldung erscheint?
- Hast du in die Logs geschaut?
- Neustarten.
- Wer ist betroffen?
- Funktioniert es bei anderen?
- Ist es Produktion?

Kategorie	Typische Frage / Beispiel
Problem verstehen	"Was genau geht nicht?"
Kontext erfassen	"Seit wann?"
Auswirkung verstehen	"Wer ist betroffen?"
Informationen sammeln	Logs, Status, Fehlermeldung
Reproduzieren	"Zeig mir, was du machst."
Hypothesen bilden	Netzwerk, Anwendung
Maßnahmen durchführen	Neustart, Rollback, Änderung
Eskalieren	Plattformteam, Entwicklerteam

Von vielen Fragen zur Struktur

Es passiert sehr viel auf unterschiedlichen Ebenen.

Vom Gefühl zum Fehlerbild

Aus "Es geht nicht" muss ein konkretes Problem werden:

- Was wurde erwartet?
- Was ist tatsächlich passiert?
- Wer oder was ist betroffen?
- Seit wann tritt es auf?
- Was funktioniert weiterhin?

Beispiel: unscharf vs. präzise

Unscharf:

"Die Anwendung geht nicht."

Präzise:

"Seit 10:15 erhalten Nutzer:innen beim Speichern HTTP-500.
Lesen funktioniert weiterhin. Betroffen ist nur Produktion."

Ein Kollege/eine Kollegin kommt zu dir und sagt:

"Seit 10:15 erhalten Nutzer:innen beim Speichern HTTP-500. Lesen funktioniert weiterhin. Betroffen ist nur Produktion."

Was machst du?

- Wie kritisch ist dieser Dienst?
- Sind alle Nutzer:innen oder nur Teilgruppen betroffen?
- Gab es Änderungen zu diesem Zeitpunkt?
- Welche Unterschiede gibt es zwischen Produktion und Test-Systemen?

Zusammenfassung: Was ist passiert?

- Besseres Verständnis
- Fragen werden spezifischer
- Der Suchraum wird kleiner

Was ist Debugging?

Debugging ist das strukturierte Reduzieren von Unsicherheit.

Wann wenden wir es an?

- Wenn Erwartung und beobachtetes Verhalten nicht übereinstimmen
- Wenn die Ursache unklar ist
- Wenn Symptome auf mehreren Ebenen sichtbar sind
- Wenn Entscheidungen nicht auf Vermutung basieren sollen

Kurz: Immer dann, wenn wir aus Unsicherheit belastbares Wissen machen müssen.

Debugging vs. Incident-Response

Debugging	Incident-Response
Ziel: Ursache verstehen	Ziel: Auswirkung schnell begrenzen
Fokus: Erkenntnisgewinn	Fokus: Service-Stabilisierung
Ergebnis: überprüfbare Erklärung	Ergebnis: Betrieb kurzfristig wiederhergestellt
Zeithorizont: mittel bis langfristig	Zeithorizont: akut, unter Zeitdruck

Beides gehört zusammen: Incident-Response stabilisiert, Debugging liefert die nachhaltige Lösung.

Debugging als Zyklus

1. Problemstellung schärfen
2. Beobachten
3. Hypothese bilden
4. Experiment durchführen
5. Ergebnis erklären und entscheiden

Problemstellung schärfen

- Was genau ist die Abweichung?
- Unter welchen Bedingungen tritt sie auf?
- Was wohin funktioniert es noch?

Beobachten

- Fakten sammeln
- Mehrere Signale kombinieren
- Zeitbezug herstellen
- Reproduzierbarkeit prüfen

Hypothese bilden

- Hypothesen formulieren
- Erwartete Beobachtung pro Hypothese benennen
- Wenige priorisierte Hypothesen statt vieler Vermutungen
- Wahrscheinlichkeiten und Risiken transparent machen

Experiment durchführen

- Kleinsten aussagekräftigen Test wählen
- Vorab definieren, wie Ergebnisse interpretiert werden
- Pro Test nur eine Variable ändern
- Ergebnis festhalten: bestätigt, widerlegt oder unklar

Ergebnis erklären und entscheiden

- Was haben wir gelernt?
- Welche Entscheidung folgt daraus?
- Reicht ein Fix, braucht es einen Workaround oder einen weiteren Zyklus?
- Erkenntnis dokumentieren und für spätere Fälle nutzbar machen

Zur Kapitelübersicht

- Nächstes Kapitel: [Docker Troubleshooting](#)