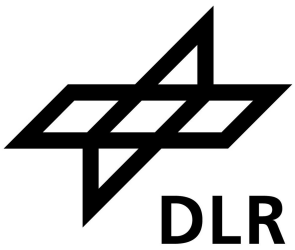

Bachelorarbeit
zur Erlangung des akademischen Grades

Bachelor of Science (B. Sc.)

im Bachelorstudiengang Informatik
der Fakultät Informatik und Medien

Automatisierte Anwendung von Chaos Engineering Methoden zur Untersuchung der Robustheit eines verteilten Softwaresystems

vorgelegt von
Brian Hampel

Leipzig, den 1. Oktober 2021

Erstprüfer: Prof. Dr. Karsten Weicker
Zweitprüfer: Dr. Alexander Weinert

Zusammenfassung

Verteilte Softwaresysteme bringen ein sehr komplexes Verhalten unter echten Einsatzbedingungen mit sich, meist resultiert dies auch in sehr komplexen Fehlerzuständen, die durch den Betrieb unter widrigen Netzwerkbedingungen wie beispielsweise hohen Latenzen und zunehmenden Paketverlusten entstehen. Diese Fehlerzustände können mit herkömmlichen Softwaretestverfahren wie Unit- und Integrationstests nicht mehr hinreichend provoziert, getestet und verifiziert werden. Mit der Methode des Chaos-Engineerings werden komplexe Chaos-Szenarien entworfen, die es ermöglichen dieses unbekannte Verhalten der Software in Grenzfällen strukturiert zu entdecken.

Am Beispiel einer verteilten Software, die bereits seit über 10 Jahren am Deutschen Zentrum für Luft- und Raumfahrt (DLR) entwickelt wird, werden Chaos-Engineering-Methoden angewandt und sowohl konzeptuell in existierende Softwaretestverfahren eingeordnet als auch praktisch in einer Experimental-Cloud-Umgebung erprobt. Innerhalb eines Experteninterviews mit den RCE-Entwicklern wird ein Chaos-Szenario entworfen, in der die Robustheit der Software mit Chaos-Experimenten auf die Probe gestellt wird. Aufbauend auf einem Softwareprojekt zur automatischen Erstellung von RCE-Testnetzwerken, wird eine Softwarelösung entwickelt die eine automatische Ausführung von Chaos-Szenarien innerhalb der Experimental-Cloud-Umgebung ermöglicht. Anschließend wird das aus den Experteninterviews resultierende Chaos-Szenario in der Praxis durchgeführt. Abschließend werden die Erkenntnisse aus der Ausführung des Chaos-Szenarios vorgestellt und weiterführende Fragestellungen und Arbeiten aufgezeigt.

Selbständigkeitserklärung

Ich erkläre hiermit, dass ich die vorliegende Graduierungsarbeit ohne Hilfe Dritter und ohne Benutzung anderer als der angegebenen Quellen und Hilfsmittel verfasst habe. Alle den benutzten Quellen wörtlich oder sinngemäß entnommene Stellen sind als solche einzeln kenntlich gemacht.

Diese Arbeit ist bislang keiner anderen Prüfungsbehörde vorgelegt und auch nicht veröffentlicht worden.

Ich bin mir bewusst, dass eine falsche Erklärung rechtliche Folgen haben wird.

Leipzig, 1. Oktober 2021 Brian Hampel

Inhaltsverzeichnis

1 Einleitung

Motivation

Durch die in den letzten Jahren steigende Popularität der Microservice-Architektur und die damit wachsende Komplexität der Softwaresysteme fällt es immer schwerer zuverlässige Aussagen über das Verhalten eines Gesamtsystems zu treffen. Gerade das Verhalten in Grenzfällen, beispielsweise unter widrigen Netzwerkbedingungen mit hoher Latenz und Paketverlusten ist bei steigender Anzahl der Einzelkomponenten nicht mehr trivial nachvollziehbar. Viele große Technologie Firmen wie beispielsweise der Online-Streamingdienst Netflix nutzen die explorative Fehlersuche die auf Experimenten in Produktionsumgebungen aufsetzt, um das Verhalten Ihrer Plattform in der Praxis herauszufinden. Dieser Ansatz der Fehlersuche mit Hilfe von Experimenten nennt sich **Chaos-Engineering**. [6] Mit dieser Methode wird durch die gezielte Manipulation von Einzelkomponenten versucht, Erkenntnisse zum Verhalten der Gesamtapplikation zu gewinnen. Dabei kann Chaos-Engineering wie eine präventive Maßnahme wirken um mögliche Schwachstellen innerhalb der Applikation strukturiert zu finden und frühzeitig in der Entwicklungsphase zu erkennen.

Chaos-Engineering kann allerdings nicht nur für Software, die nach dem Abbild der Microservicearchitektur geschaffen wurde, interessant sein. Verteilte Softwaresysteme wie die Software Remote Component Environment die am Deutschen Zentrum für Luft- und Raumfahrt e.V. entwickelt wird, weisen dabei ähnliche Problem- und Fragestellungen auf. Mit herkömmlichen Softwaretestverfahren kann dabei nicht mehr zufriedenstellend getestet werden, wie sich die Applikation unter echten Einsatzbedingungen verhält.

2 Grundlagen

In diesem Kapitel werden in Abschnitt 2.2 grundlegende Begriffe erläutert, einfache Standardsoftwaretests in Abschnitt 2.1 vorgestellt und in Abschnitt 2.4 ein erster Einblick in Chaos Engineering gegeben. Weiterhin wird in Abschnitt 2.3 der Begriff Containerorchestrierung eingeführt.

2.1 Softwareentwicklung und Testverfahren

Aufgrund der stetig wachsenden Komplexität von Software werden (teil-)automatisierte Testmethoden immer wichtiger. Dabei dienen Softwaretests der Reduzierung von Fehlern, die während des Softwareentwicklungsprozesses entstehen.

Für jede Software existiert eine Spezifikation, diese kann explizit oder implizit stattfinden. Eine explizite Spezifikation beschreibt konkrete Anforderungen und Umgebungsbedingungen unter denen eine Software funktionieren muss. Die Anforderungen können dabei vielfältig sein, von einer zugesicherten Reaktionszeit, über eine Anforderung unter bestimmten Umgebungsbedingungen weiterhin zu funktionieren bis hin zu einer zugesicherten Genauigkeit einer Berechnung. Eine implizite Spezifikation ist die Beschreibung der Anforderungen, die direkt aus der Implementierung heraus stattfindet.

Die Spezifikation kann in einzelnen Softwaretests ausgedrückt und strukturiert überprüft werden. Innerhalb dieser Arbeit wird zwischen drei Arten von Softwaretests unterschieden.

Unit-Tests Testen eines einzelnen Moduls mit Hilfe von bekannten Eingabe- und Ausgabewerten ohne die Kopplung zu anderen Modulen oder Funktionen der Software

Integration-Tests Testen der Zusammenarbeit verschiedener Module die miteinander integriert werden (bspw. die Berechnung von Daten in Modul A, Verwendung der Daten in Modul B)

End-to-End-Tests Test eines kompletten Abfrage- / Anwendungszyklus durch die gesamte Applikation

Fehler in neuen Erweiterungen der Software können durch eine automatisierte Testausführung bedeutend schneller gefunden werden. In der Praxis wird dies durch eine Continuous-Integration-Pipeline (CI-Pipeline) umgesetzt.

Continuous Integration beschreibt den Vorgang der häufigen Integration von Änderungen in den bereits vorhandenen Quellcode. Diese Veränderungen entstehen hierbei durch verschiedene Entwickler, die gleichzeitig an einem Softwareprojekt arbeiten. Die Arbeitsweise mit CI-Pipelines wird in drei Phasen beschrieben. [2]

Build Kompilieren der Software mit hinzugefügten Änderungen

Test Testen der Software anhand der Spezifikation

Merge Anschließende Integration in den Code. Dieser Arbeitsschritt steht am Ende der Pipeline und wird nach der erfolgreichen Überprüfung der Änderungen in den beiden vorausgehenden Phasen manuell ausgeführt.

Mit den genannten Testmethoden wird immer eine spezifische Funktion auf ihre Arbeitsweise innerhalb der Spezifikation geprüft. Die Überprüfung resultiert stets in einer Booleschen Antwort, True oder False.

Es existieren viele weitere Verfahren und Methoden zum Testen eines Softwareprojekts, die allerdings nicht Bestandteil dieser Arbeit sind. Weitere Informationen können dem Guide to the Software Engineering Body of Knowledge [7] entnommen werden.

2.2 Verteilte Software

In der frühen Zeit der Informationstechnik wurden hauptsächlich zentralisierte Lösungen genutzt um beispielsweise Massendaten zu verarbeiten oder Simulationen zu berechnen. Die allseitige Vernetzung der Informationstechnik förderte dabei die Entstehung von Applikationen die fortan nicht mehr nur auf einem zentralem Computer liefen, sondern eine oder mehrere Aufgaben koordiniert über das Netzwerk lösen können. Verteilte Systeme sind wie folgt definiert.

> A distributed system is a system whose components are located on different networked computers, which communicate and coordinate their actions by passing messages to one another from any system.
>
> (Maarten van Steen, Andrew S. Tanenbaum [16])

Dies ist heutzutage besonders bei großen Applikationen wie der Streaming Plattform von Netflix sichtbar, die gegenüber den Nutzern als eine Applikation / Service auftritt, aber im Grunde aus tausenden kleiner Dienste besteht. Ohne eine verteilte Architektur der Softwarelösung wäre eine Skalierung auf Millionen von Nutzern die den Dienst jeden Tag gleichzeitig nutzen, nicht möglich. [15] [4]

Verteilte Systeme bringen allerdings auch ein sehr komplexes Verhalten und unterschiedlichste Fehlerzustände mit sich. Mit herkömmlichen Softwaretestverfahren wie in Abschnitt 2.1 beschrieben, kann man ein komplexes Fehlverhalten des Gesamtsystems nicht mehr hinreichend testen und verifizieren.

2.3 Containerorchestrierung

Mit der weiter voranschreitenden Ablösung von virtuellen Maschinen zum Bereitstellen von Softwareapplikationen durch Container-Images, wurden auch vielfältige Orchestrierungs- und Managementlösungen für Container entwickelt.

Ein Container-Image ist ein unveränderliches Abbild einer Software in einer spezifizierten Version, das zusätzlich die notwendige Laufzeitumgebung zur Ausführung enthält.

Eine Container-Orchestrierung übernimmt die Bereitstellung, Konfiguration, Überwachung und Modifikation von Containerressourcen. Meist können Nutzer mit Hilfe von YAML-Beschreibungen komplette Applikationsstacks mit allen notwendigen Komponenten beschreiben und mit nur einem Befehl zur Verfügung stellen.

Einer der meistgenutzten Containerorchestrierungslösungen, das Softwareprojekt Kubernetes, wurde 2014 von Google unter einer Open Source Lizenz weltweit zur Verfügung gestellt. Die Software ermöglicht Nutzern flexibel Applikationen hochverfügbar und skalierbar in Public-, Private- und Hybrid-Cloudumgebungen bereitzustellen. Im Gegensatz zu physikalischen oder virtuellen Servern wird die gesamte Applikationsumgebung als ein Zielzustand beschrieben. Zur Applikationsumgebung gehören beispielsweise Datenbanken, Webserver oder ein Dienst der eine Webschnittstelle für das Frontend einer Applikation bereitstellt. Der Orchestrator versucht die Applikation anschließend entsprechend der Beschreibung bereitzustellen und zu diesem Zielzustand zu gelangen.

2.4 Chaos Engineering

Chaos Engineering ist eine Methode, um Informationen über die Robustheit eines gegebenen verteilten Softwaresystems zu sammeln und zu verifizieren.

Um diese Informationen mithilfe von Chaos Engineering zu sammeln, wird in vier Schritten vorgegangen:

- Hypothese zum stabilen Zustand aufbauen
- Real-World-Events variieren
- Blast-Radius minimieren
- Experimente automatisiert ausführen

Diese vier Schritte werden Principles of Chaos Engineering genannt. [6]

Der erste Schritt besteht aus dem Aufbau einer Hypothese über das zu vermessende System. Diese Hypothese ist eine Annahme über Rahmenbedingungen des Systems unter denen das System seine Spezifikation nicht mehr erfüllen kann. Über das in Abschnitt 3.1 beschriebene wissenschaftliche Integrationssystem könnte zum Beispiel folgende Hypothese aufgestellt werden:

> Sobald die Netzwerklatenz zwischen zwei Instanzen 500ms überschreitet kann das Integrationssystem seine Spezifikation nicht mehr erfüllen.

Für den Rest dieser Arbeit wird davon ausgegangen, dass eine Hypothese immer in der Form "Sobald Real-World-Event X geschieht, kann das System die Spezifikation Y nicht mehr erfüllen." beschrieben wird. Weiterhin wird davon ausgegangen, dass es möglich ist, automatisiert zu überprüfen, ob das System die Spezifikation Y erfüllt.

Der nächste Schritt ist die Auswahl von variierenden Real-World-Events. Unter diesem Begriff werden innerhalb dieser Arbeit Ereignisse verstanden, die im Produktionsbetrieb eines Systems auftreten. Mögliche Real-World-Events wären beispielweise eine fehlerhafte Eingabe eines Nutzers oder die schlechte Verbindung zu einem Dienst des verteilten Systems. Auch Probleme aus der Vergangenheit und deren Ursachen werden berücksichtigt.

Unter der Minimierung des Blast-Radius wird verstanden, dass kein zusätzlicher Schaden an der Infrastruktur oder benachbarter Dienste herbeigeführt werden soll. Da bei der Einführung von Chaos-Engineering die Robustheit des Systems weitestgehend unbekannt ist,

kann in einem Produktionssystem schwer eingeschätzt werden, welche Folgen die Durchführung eines Chaos-Szenarios hat. Innerhalb dieser Arbeit werden aus diesem Grund alle Chaos-Szenarien innerhalb einer Testumgebung ausgeführt.

Der letzte Schritt stellt sicher, nicht nur eine Momentaufnahme der gerade vorliegenden Softwareversion zu erhalten, sondern eine permanente Überprüfung, die unter den gleichen Bedingungen und Annahmen stattfindet. Die automatische Ausführung kann beispielsweise durch eine Integration in eine CI-Pipeline oder durch die Aufnahme in einen Testplan stattfinden.

Es wird zwischen drei Kategorien, in die Chaos-Experimente eingeordnet werden, unterschieden:

System Chaos Beschreibt dabei das Herunterfahren oder Zerstören von kompletten Servern oder einzelnen Containern. Hierbei wird zufällig ein laufendes System in Produktion heruntergefahren um eine bessere Idee davon zu bekommen, wie gut das Umschalten von Knoten zu Knoten in Umgebungen mit hochverfügbaren Systemen funktioniert. Ein typischer Vertreter dieser Kategorie ist das Softwareprojekt "Chaos Monkey"das von Netflix stammt. [6]

Environment Chaos In dieser Kategorie wird die zu untersuchende Software häufig containerisiert, um von außen wirkende Einflüsse und Abhängigkeiten zu reduzieren. Chaos-Experimente beeinflussen hier die Umgebung der Applikation wie Netzwerkverbindungen, Hardwareauslastung (File I/O, CPU, RAM) oder auch die künstliche Abweichung der Systemuhr. Zwei Softwareprojekte die in diese Kategorie fallen sind Chaos-Orca und Chaos Mesh. [18] [8]

Software Chaos Exceptions oder Stacktraces gehören in Softwareprojekten zur Normalität und müssen korrekt abgefangen und behandelt werden. Über spezielle Chaos-Module, die in das Softwareprojekt integriert werden, können geplant Exceptions bei bestimmten Funktionsaufrufen geworfen werden. In dieser Kategorie wird also direkt in die Software und deren interne Ausführung eingegriffen und die Robustheit der Fehlerbehandlung überprüft. Mit Hilfe von ChaosMachine, kann man diese Art der Experimente in Java / JVM Softwareprojekten durchführen. [20]

Im Rest dieser Arbeit wird ausschließlich Environment Chaos betrachtet.

3 Betrachtetes System

Nach der Einführung der notwendigen Begriffe und Beschreibungen von modernen Softwaretestverfahren in Kapitel 2 wird innerhalb dieses Kapitels das wissenschaftliche Integrationssystem Remote Component Environment (RCE) in Abschnitt 3.1 vorgestellt, sowie ein Einblick in den aktuellen Testprozess der Software (Abschnitt 3.2) in einem Softwareentwicklungsteam am DLR gegeben. Dabei werden mit Hilfe eines Experteninterviews (Abschnitt 3.3, 3.4, 3.5) Informationen über die aktuellen Defizite im Testingprozess erhoben und mögliche Lösungsansätze (Abschnitt 3.6) diskutiert.

3.1 Remote Component Environment

RCE (Remote Component Environment) ist eine verteilte Workflow-basierte Integrationsumgebung für Ingenieure und Wissenschaftler um komplexe Multi-Disziplinäre Systeme (Schiffe, Satelliten, Flugzeuge) zu designen und zu simulieren. Dabei können Anwender ihre eigene Design- und Simulationssoftware in RCE integrieren. Der Umgang mit komplexen Systemen benötigt viele Experten mit unterschiedlichster Fachexpertise. Dabei kommen eine Vielfalt an Softwaretools zum Einsatz die mit RCE innerhalb der Arbeitsgruppen geteilt werden und in einem automatisierten und verteilten Workflow ausgeführt werden können. [11] RCE stellt dafür die folgenden Kernfunktionen bereit: [13]

Integration Die Kopplung von externer Expertensoftware in Workflows ist eine Kernfunktionalität von RCE. Jedes Tool muss nur einmal integriert werden und kann dann über standardisierte In- und Outputs im verteilten System genutzt werden.

Automation Während der Ausführung von Workflows ist kein zusätzlicher User-Input erforderlich. RCE kann auf Hochleistungssystemen wie speziellen Compute-Nodes in Rechenzentren betrieben und an vorhandene Umgebungen angeschlossen werden um aufwendige Simulationen auf einer dafür vorgesehenen Hardware auszuführen. Ein Abkoppeln der eigenen Instanz, die Workflows konfiguriert und steuert, kann jeder Zeit vorgenommen werden.

Data Management Ergebnisse aus Simulationen, Analysen und Berechnungen sind für Wissenschaftler und Ingenieure essentiell. Sämtliche von Tools generierten Daten werden in RCE vorgehalten und sind über eine grafische Oberfläche auslesbar.

Collaboration RCE Instanzen können über das Internet oder das lokale Netzwerk zu einem Peer-to-Peer Netzwerk verbunden werden.

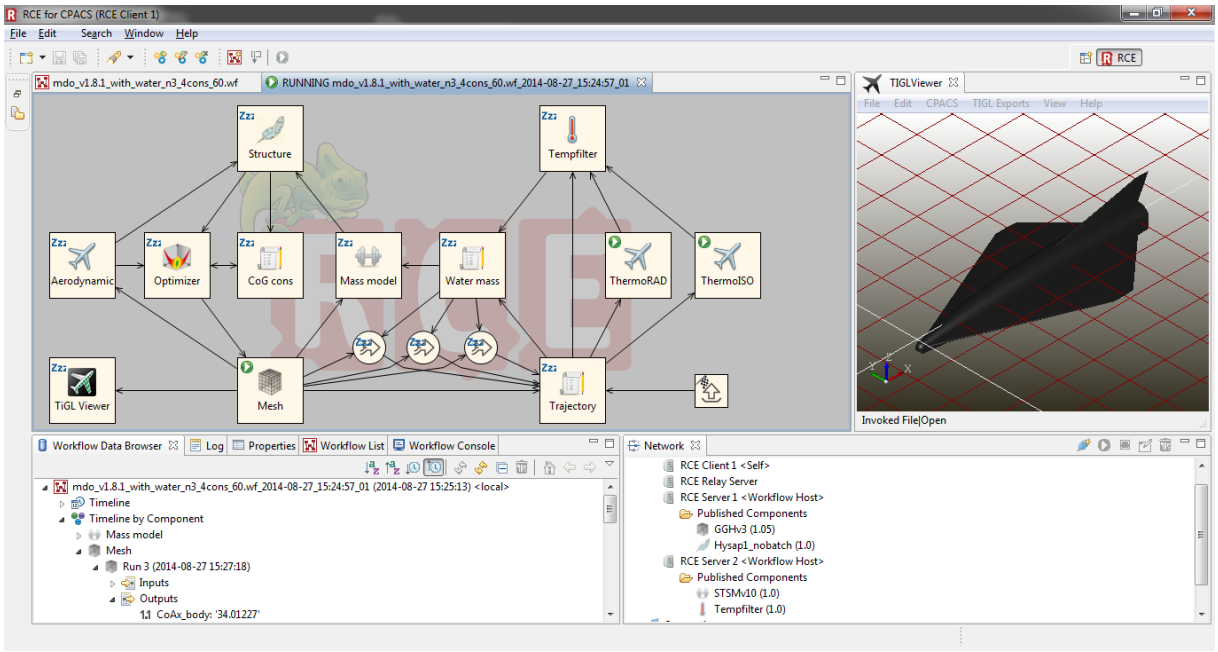

Abbildung 3.1: Ablauf und Ansicht eines Workflows mit integrierten Tools zur Optimierung von Luftfahrzeugen

Unter dem Begriff **Tool** ist eine externe Fachsoftware zu verstehen, die von einem fachspezifischen Anwender geschrieben wurde und in die RCE-Instanz integriert wurde. Ein **Workflow** ist dabei die Verkettung von Tools mit klar spezifizierten In- und Outputs. Innerhalb eines Workflows wird dabei auch die Reihenfolge und der Ort der Ausführung des Tools bestimmt.

Ein **Localnet-Relay** ist eine RCE-Instanz die zur lokalen Vernetzung innerhalb einer Institution standardmäßig genutzt wird. Hierbei werden zwischen den Instanzen detaillierte Informationen ausgetauscht, beispielsweise Informationen über andere Instanzen innerhalb dieses Netzwerks.

Das **Uplink-Relay** ist eine RCE-Instanz, die ausschließlich der Kopplung verschiedener RCE-Netzwerke dient, die beispielsweise in unterschiedlichen Institutionen betrieben werden. Das Uplink-Relay bietet eine zusätzliche Isolation der Netzwerke voneinander, indem nur notwendige Informationen für das Teilen und Ausführen eines Tools, über eine

verschlüsselte Verbindung ausgetauscht werden. Weitere Informationen wie die Netzwerktopologie der verbundenen Netzwerke sind dabei nur für die Nutzer im jeweiligen Segment sichtbar.

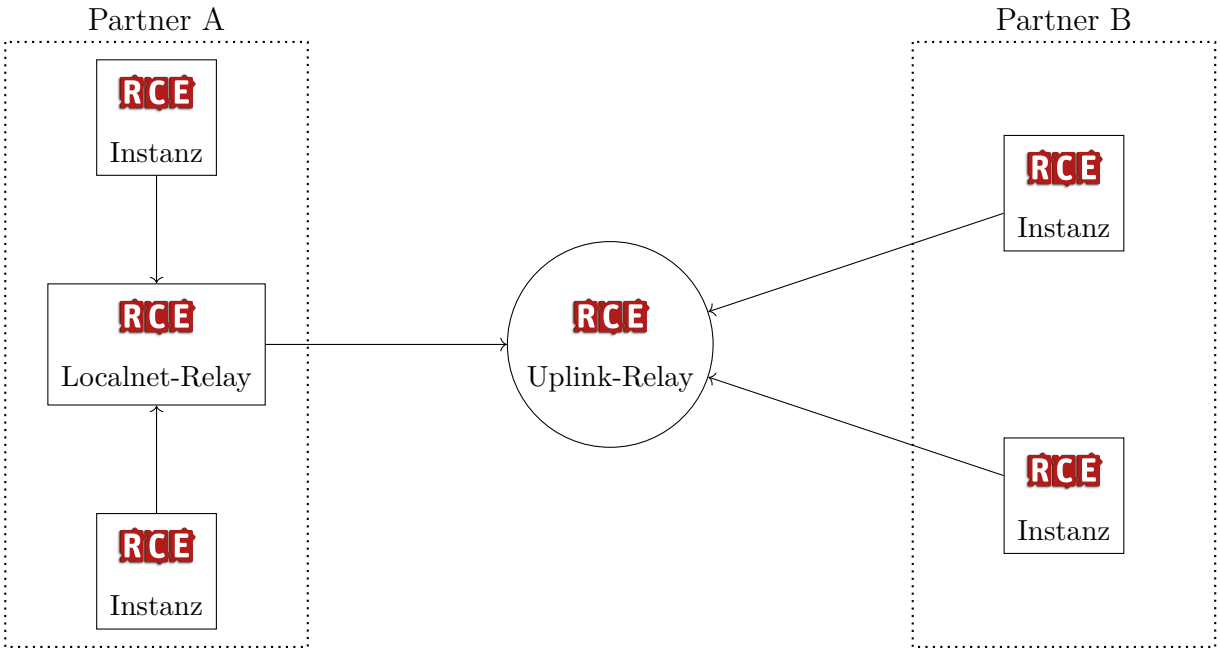

Abbildung 3.2: Schematische Darstellung gekoppelter RCE-Netzwerke

Weitere Informationen können dem RCE-Admin-Guide entnommen werden. [12]

3.2 Testing von RCE Releases

Neue RCE-Release Versionen werden einem aufwendigen und umfangreichen Testprozess unterworfen. Dabei bedient sich das RCE-Entwicklerteam den in der Softwareentwicklung gängigen Methoden, um Fehler möglichst früh zu finden und in kleineren Iterationsschritten lösen zu können.

- Unit-Tests
- CI-Pipelines
- Issue-Tracking
- automatisierte Softwarebuilds
- Behavior Driven Development (BDD)-Tests
- manuell durchgeführte Tests anhand von softwaregestützten Checklisten

Im aktuellen Release 10.2.4 der Software RCE, stellen rund 1700 Unittests sicher, dass einzelne Teile der Software wie vorgesehen funktionieren. BDD-Tests ergänzen diese mit einer verständlicheren und umgangssprachlichen Version der Tests. Dabei werden verschiedene Szenarien umgangssprachlich in 'Wenn-Dann' Szenarien beschrieben und mit dem echten Code modulübergreifend getestet:

```
...
  When  stopping all instances
  And   starting all instances
  And   waiting for 5 seconds

  Then  instance "NodeA" should see these components:
        | NodeA | rce/Switch   | shared:public                  |
        | NodeA | rce/Joiner   | shared:GroupA:0123456789abcdef |
        | NodeA | rce/Database | local-only                     |
...
```

Listing 3.1: BDD-Test, Neustart von RCE-Instanzen, Quelle: [10]

Ein weiterer wichtiger Baustein sind die manuellen Softwaretests anhand von softwaregestützten Checklisten. Zur Unterstützung wird hierfür die Softwarelösung TestRail [19] genutzt. TestRail ermöglicht den manuellen Softwaretestingprozess zu koordinieren, zu strukturieren und auszuwerten. Einzelne Testphasen können hier gezielt für einzelne Releases anhand von vorher definierten Templates und auf Basis bekannter Problemstellungen aus der Historie erstellt werden. Innerhalb der letzten Testphase wurden insgesamt 153 verschiedene Testfälle durchgeführt und geprüft.

3.3 Methode Experteninterview

Mit Hilfe eines semistrukturierten Experteninterviews, das einzeln mit RCE-Entwicklern als Interviewpartner durchgeführt wurde, soll ein tieferer Einblick in den Softwaretestprozess und dessen mögliche Schwächen ermöglicht werden. Dabei sollen vor allem Stellen innerhalb der Applikation identifiziert werden, die mit den bisher genutzten Testmethoden nicht oder nur schlecht abgedeckt werden können.

Vor jedem Interview wurde eine Einleitung in das Thema Chaos Engineering durchgeführt, dabei wurden insbesondere die Methodik der Durchführung von Chaos-Experimenten und mögliche Arten von Experimenten zur Manipulation von Containern, Instanzen und Serversystemen erläutert, die in Sektion 2.4 vorgestellt wurden.

Anschließend fand eine Auswertung der Antworten statt. Diese werden nach Umsetzbarkeit und notwendigen Implementierungsmaßnahmen bewertet und priorisiert.

3.4 Fragestellungen entwerfen

Um einen detaillierten Einblick in die möglichen Schwächen des Testprozesses zu bekommen, wurden folgende Fragen während des Interviews gestellt:

1. Welche konkreten Funktionen / Komponenten von RCE können derzeit nur eingeschränkt getestet werden?

2. Wird die Zuverlässigkeit bei Problemen des auszuführenden Systems (Netzwerkausfälle, CPU-Auslastung) bereits getestet?

3. Welche Chaos-Experimente wären als Ergänzung zum aktuellen Softwaretestprozess interessant um eine Aussage über die Zuverlässigkeit von einer neuen RCE Version zu treffen?

4. Wie oft muss eine konkrete Funktion getestet werden?

Die Zielstellung des Interviews ist es, bekannte Probleme im Softwaretestprozess zu erkennen und diese in einer Diskussion zu einem Chaos-Szenario zu entwickeln. Wenn das Szenario nur einmalig getestet werden soll, muss entschieden werden ob sich der Implementierungsaufwand gegenüber den gewonnenen Erkenntnissen im Verhältnis stehen. Innerhalb der Arbeit sind vor allem Szenarien interessant, die häufig genutzt und in der jetzigen Ausgangssituation schlecht nachzustellen und zu testen sind.

Die Auswertung der Antworten erfolgt anonymisiert, aus Zeitgründen ist kein vollständiges Transkript angefertigt worden.

3.5 Resultate aus Interview

Innerhalb der Experteninterviews wurden drei RCE-Entwickler befragt. Bei der Befragung kamen eine Vielzahl an möglichen Testszenarien zum Vorschein. Im aktuellen Zustand konnten diese nicht in der Form eines Softwaretests getestet und durchgeführt werden. Gerade der manuelle Testprozess ist sehr umfangreich. Dieser wird mit der Unterstützung von digitalen Testplänen durchgeführt die über die Software TestRail verwaltet werden.

Anschließend wird ein verteiltes Netzwerk aufgebaut und unterschiedlichste Tests durchgeführt. Alle Instanzen befinden sich aus Netzwerksicht nah beieinander (meist am gleichen Switch) und die Netzwerkschnittstellen können nur mit hohem Aufwand und wenig kontrolliert manipuliert werden.

Die Entwickler sehen ein großes Potenzial in der Präzision und Granularität, in denen einzelne Chaos-Szenarien geplant und durchgeführt werden können. Ein besonderer Vorteil ist dabei die kontrollierte Umgebung, die bei jedem Testzyklus genau der vorherigen Umgebung entspricht, bei der lediglich die Version der Software innerhalb des Containers verändert wird. Als besonders kritisch wurde allerdings auch die Automatisierung der ablaufenden Szenarien gesehen. Diese verhindert, dass einzelne Szenarien gar nicht ausgeführt werden oder schlichtweg innerhalb des Testprozesses vergessen werden. Gleichzeitig würde eine Automatisierung aber auch eine große Arbeitserleichterung darstellen, da vorher manuell erhobene Daten wie Log-Dateien nun automatisch für den Entwickler nach der Ausführung eines Szenarios eingesammelt und bereitgestellt werden können.

Ebenfalls wurden die unterschiedlichen Kategorien der Chaos-Experimente angesprochen, die in Sektion 2.4 vorgestellt wurden. Für eine erste Integration in den Testprozess lag dabei der Fokus vor allem auf Experimente der Kategorie Zwei - Environment Chaos. Experimente aus der ersten Kategorie werden bereits mit wenig Aufwand in der manuell aufgebauten Umgebung getestet. Experimente der Kategorie Drei wären für die Interviewpartner ebenfalls interessant, allerdings würde hier die Integrations- und Implementierungszeit in die bereits vorhandene Software den zeitlichen Rahmen dieser Arbeit sprengen. Gezielte Tests zum Verhalten der Applikationen unter extremer CPU-, RAM-Auslastung gibt es derzeit nicht.

Ein besonders interessanter Aspekt war, das „chaotische“ Testverfahren bereits im manuellen Testprozess enthalten waren, diese aber einen hohen Aufwand bei der Durchführung oder Einrichtung besitzen und meist durch die vorhandene Infrastruktur sehr begrenzt waren. Das gleichzeitige, aber minimal zeitversetzte Starten von möglichst vielen parallelen Workflows ist ein Beispiel für die chaotischen Testverfahren die bereits jetzt genutzt werden.

Aus Sicht der Interviewpartner sind die folgenden Chaos-Szenarien besonders interessant, geordnet nach der Häufigkeit der Nennung, Umsetzbarkeit und Implementierungsaufwand.

- Ausführen eines Workflows unter extremen Netzwerkbedingungen. (Simulation einer gestörten Netzwerkverbindung mit hohen Paketverlusten, Latenzen und Jitter)
- Ausführen eines Workflows unter vollständiger Partitionierung des Netzwerks

- Stresstesten eines Workflowstarts durch unterschiedliche Netzwerkbeeinträchtigungen

3.6 Integration von Chaos-Engineering

Aufgrund der Antworten aus den Experteninterviews kann Chaos-Engineering eine sinnvolle Erweiterung für den jetzigen Softwaretestingprozess darstellen. Gerade bei der Simulation einer echten verteilten Umgebung über mehrere Standorte hinweg, kann die neue Methode einen großen Vorteil bieten um unvorhergesehenes Verhalten der Software noch vor Release einer neuen RCE-Version zu entdecken.

Nach der Auswertung des Experteninterviews hat sich das folgende Chaos-Szenario zur testweisen Integration in den Softwaretestprozess als besonders geeignet herausgestellt. Das Chaos-Szenario wird anhand der Principles of Chaos Engineering, die in Sektion 2.4 eingeführt wurden, wie folgt beschrieben.

Hypothese zum stabilen Zustand aufbauen Sobald eine Netzwerkpartitionierung während eines laufenden Workflows eintritt, kann das System keine erfolgreiche Wiederherstellung der Netzwerkverbindungen vornehmen und nicht in eine konsistente Netzwerksicht zurückkehren.

Real-World-Events variieren Nach dem Start des Workflows wird zwischen allen beteiligten Instanzen und dem Uplink-Relay für die Dauer der Ausführung das Netzwerk partioniert. Eine Kommunikation von Instanz zu Localnet-Relay ist nicht mehr möglich.

Blast-Radius minimieren Dieses Szenario soll ausschließlich in einer Testumgebung innerhalb einer Experimental-Cloud-Umgebung getestet werden. Somit besteht keine Gefahr für eine produktiv genutzte Umgebung.

Experimente automatisiert ausführen Das Chaos-Szenario wird mit Hilfe einer Ablaufsteuerung automatisch ausgeführt. Diese Ablaufsteuerung wird in Sektion 4.4.5 detailliert vorgestellt.

Netzwerklatenzen, Packet Loss und Netzwerkpartitionierung wie sie in echten Nutzungsszenarien auftreten, konnten bisher noch nicht vor dem Release der Software getestet und simuliert werden. Daher besteht hier auch die größte Wahrscheinlichkeit unvorhersehbares und unerwünschtes Verhalten der Software zu finden.

Aus den Interviews kann abgeleitet werden, dass Chaos-Engineering vielseitig innerhalb des Entwicklungsprozesses integriert werden kann. Sinnvoll wäre eine Nutzung der definierten Szenarien bereits während der Entwicklung, beispielsweise wenn ein Entwickler etwas an der Funktionalität eines Uplink-Relays verändert hat und diese Änderung überprüfen möchte. Weiterhin ist es ebenfalls von großem Vorteil die Chaos-Szenarien innerhalb der Release-Testphase zu nutzen um ein kurz vor der Veröffentlichung stehendes Update auf Robustheit zu testen.

4 Konzepte des Chaos-Engineering am Beispiel

In diesem Kapitel wird das Wissen aus Kapitel 2 mit den Erkenntnissen aus den Interviews (Sektion 3.5) verknüpft und in die Praxis umgesetzt.

Weiterhin wird eine Komponente (Sektion 4.4) entwickelt, die einen bereits vorhandenen Softwarestack (Sektion 4.1.2) zur automatischen Erstellung von RCE-Testnetzwerken um Chaos-Engineering Funktionen erweitert.

4.1 Ausgangssituation

Es besteht bereits ein Softwareprojekt am DLR, das eine automatische Erstellung von RCE-Testnetzwerken innerhalb einer Private-Cloud-Umgebung ermöglicht. In diesem Abschnitt wird die Systemumgebung beschrieben und der Aufbau sowie die Funktionsweise dieser Software erläutert.

4.1.1 Systemumgebung

Innerhalb der Abteilung Intelligente und Verteilte Systeme im Institut für Softwaretechnologie wird ein Experimental-Cloud -System betrieben. Dieses besteht aus insgesamt 10 Servern, die den Aufbau einer Private-Cloud-Computing Infrastruktur ermöglichen.

Es stehen drei Arten von Serverknoten zur Verfügung:

Compute Serverknoten zur Berechnung und Bearbeitung CPU-intensiver Workloads. Diese Server stellen 128 CPU-Kerne und rund 400 GB RAM pro System bereit.

Storage Serverknoten zur Speicherung von größeren Datenmengen, stellen 30 TB nutzbaren Speicher zur Verfügung

GPU Serverknoten zur Beschleunigung von Berechnungen im Artifical Intelligence / Machine Learning Umfeld. Für diese Aufgaben werden NVIDIA Tesla V100 GPUs bereitgestellt.

Alle Server befinden sich im gleichen Netzwerk, eine Kommunikation mit dem Internet ist nur über einen zusätzliches Filtersystem möglich. Innerhalb dieser Arbeit werden ausschließlich Server aus der Kategorie Compute und Storage genutzt.

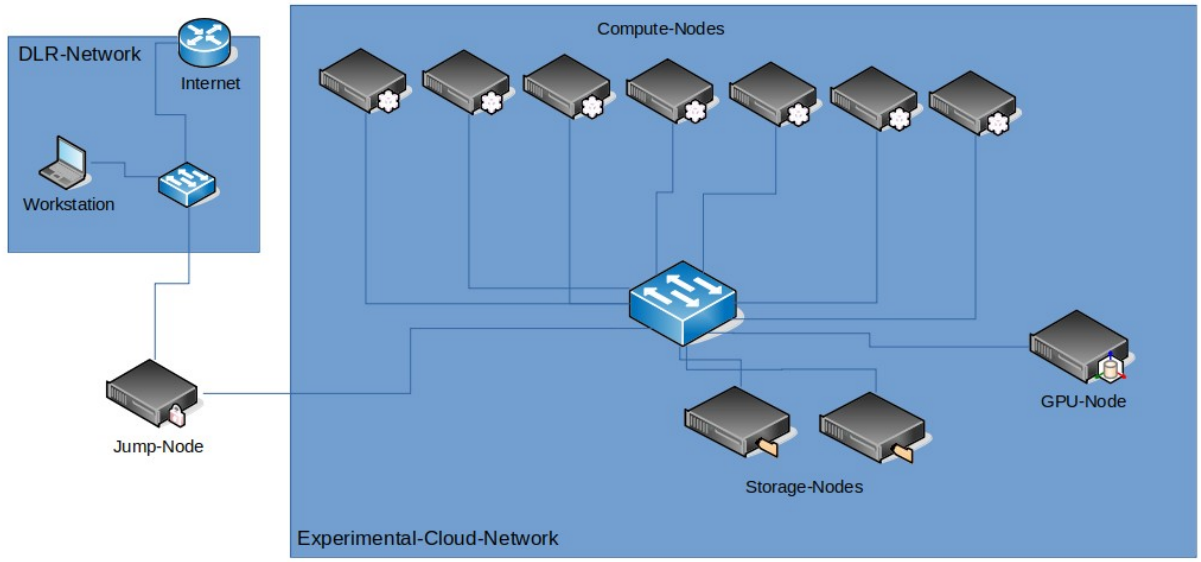

Abbildung 4.1: Aufbau Experimental-Cloud-Umgebung

Im Vorfeld dieser Arbeit wurde ein Kubernetes Cluster auf Basis von k3s aufgesetzt. Diese Distribution bietet im Gegensatz zu einer vollständigen Kubernetes Installation, vielfältige Vorteile wie geringeren Wartungsaufwand, schnelleren Aufbau, schnelle Updatezyklen und die Reduktion auf Kernfunktionen. [17] Durch umfangreiche Automatisierung über die Software Ansible kann zum Beispiel im Fehlerfall die Umgebung zeitnah frisch aufgesetzt werden, um wieder in einen funktionsfähigen Zustand zu gelangen.

4.1.2 Automatisierte Erstellung von Testnetzwerken

Das Projekt „rce-test-network“ stellt eine Software mit mehreren Diensten bereit, die den Aufbau eines RCE-Netzwerks automatisieren. Über einen veränderbaren Bauplan kann das Testnetzwerk an die Anforderungen des Testplans angepasst werden.

Testnetzwerke werden innerhalb des Kubernetes Clusters in **Namespaces** logisch voneinander getrennt. Ein Namespace ist ein separater Namensraum, in dem Namen für Ressourcen nur einmalig vergeben werden können. Damit kann ein RCE-Entwickler in der Praxis sein Testnetzwerk, innerhalb seines eigenen Namensraums aufbauen und verändern lassen ohne andere Testnetzwerke zu beeinflussen.

Vor jedem Release der Software RCE werden eine Reihe von manuellen Tests ausgeführt, die eine langwierige Vorbereitung erfordern. In jeder Testphase werden auf den Entwickler-Workstations manuell mehrere RCE-Instanzen gestartet, die anschließend manuell vernetzt werden. Dieser Umstand konnte bereits größtenteils durch das Softwareprojekt „rce-test-network“ gelöst werden, das innerhalb der Experimental-Cloud-Umgebung ganze Testnetzwerke nach einem vom Benutzer definierten Bauplan aufsetzt und vorkonfiguriert. Die Entwickler können sich nach der Erstellung nun mit ihrer RCE-Instanz über eine vordefinierte Schnittstelle mit dem Testnetzwerk verbinden und Tools oder Workflows ausführen. Damit erleichtert das Softwareprojekt den manuellen Testprozess erheblich, da eine große Zeitersparnis, eine bessere Reproduzierbarkeit und ein von Release zu Release exakt gleicher Aufbau der Testnetzwerke erreicht wird. Die Software selbst ist in zwei Microservices aufgeteilt: Manager und Operator.

4.1.3 Microservices

Die notwendigen Aufgaben zur automatischen Erstellung eines RCE-Testnetzwerks wurden auf zwei Microservices aufgeteilt.

Manager

Der Manager besitzt weitreichende Rechte innerhalb der Kubernetes Umgebung. Er ist dafür zuständig, die bereits erstellten Testnetzwerke zu verwalten. Dies beinhaltet die Erstellung, Modifikation und die Löschung von erstellten Netzwerken, sowie das Validieren und Weiterreichen der Netzwerkbeschreibung als auch die Erstellung des Operators für dieses Testnetzwerk. Dabei hält der Manager so wenig Daten wie möglich über Testnetzwerke vor. Lediglich der Netzwerkname und die interne Adresse, um den Operator für spätere Aktionen zu erreichen. Diese Funktionalitäten werden nach außen über eine HTTP-API bereitgestellt.

Operator

Der Operator wird vom Manager im Testnetzwerk erstellt und ihm wird die bereits durch den Manager überprüfte Netzwerkbeschreibung übergeben. Er ist für die Erstellung, Konfiguration und Verwaltung der RCE-Instanzen anhand der zur Verfügung gestellten Netzwerkbeschreibung zuständig. Weiterhin werden über vordefinierte Schnittstellen Informationen zu jeder Instanz bereitgestellt, die beispielsweise die Konfiguration und das Debug-Log einer RCE-Instanz für den Manager verfügbar machen. Auch der Operator stellt die Funktionalitäten über eine HTTP-API bereit, die der Manager über das interne Netzwerk konsumieren kann.

4.1.4 Systemarchitektur

Die Gesamtapplikation ist nach dem Vorbild der Microservice-Architektur umgesetzt. Diese Softwarearchitektur ermöglicht eine einfache Erweiterbarkeit mit zusätzlichen Diensten und eine gute Daten- und Funktionstrennung der einzelnen Dienste.

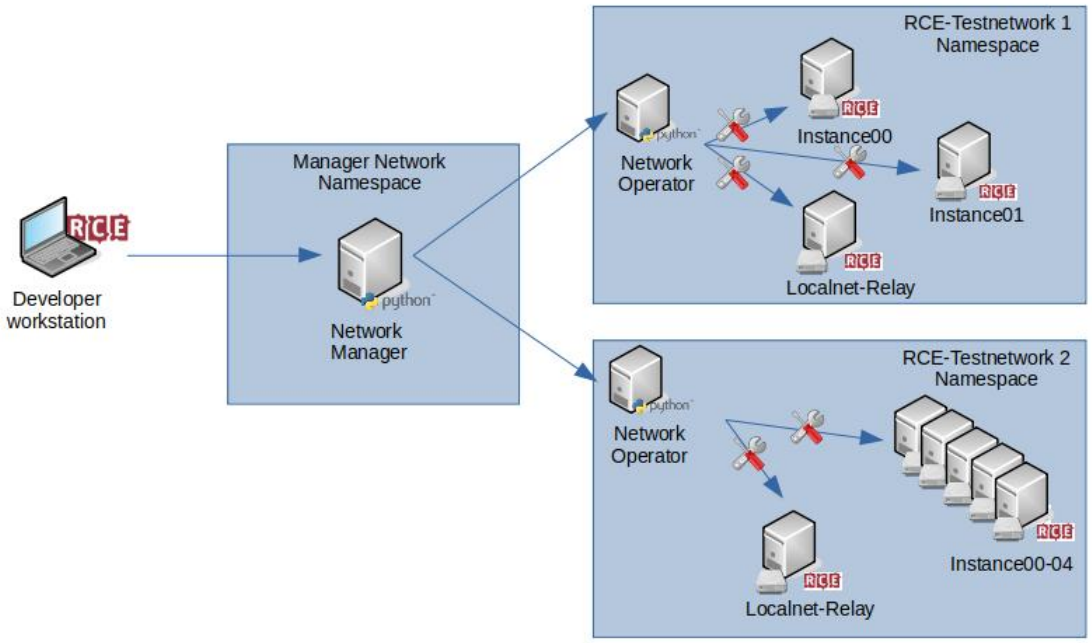

Abbildung 4.2: Kommunikation des Manager und Operator zur Erstellung von Testnetzwerken

In Abbildung 4.2 ist der Kommunikationsfluss von der Entwickler-Workstation zu Manager und anschließend zum Operator. Von einer Entwickler-Workstation wird die Netz-

werkbeschreibung an den im Experimental-Cloud-System laufenden Manager geschickt. Dieser erstellt einen für das Testnetzwerk verfügbaren Namespace sowie den dazugehörigen Operator der von nun an für das Testnetzwerk verantwortlich ist. Anhand der vom Manager übergebenen Netzwerkbeschreibung baut der Operator das Testnetzwerk auf.

Alle Anfragen die ein spezifisches Testnetzwerk betreffen, wie beispielsweise das Abrufen von konkreten Informationen zu RCE-Instanzen die in einem Testnetzwerk verfügbar sind, werden über eine Proxy-Funktion des Managers an den verantwortlichen Operator weitergeleitet.

4.1.5 Netzwerkbeschreibung

Ein zu erstellendes Testnetzwerk wird mit einem JSON-Dokument im folgenden Format beschrieben.

```
{
  "uplink-relay": {
    "image": "rce:10.2.4",
    "config": {
        ...
      }
    }
  },
  "localnet-relay": {
    "image": "rce:10.2.4",
    "config": {
        ...
    }
  },
  "instances": {
    "image": "rce:10.2.4",
    "replicas": 5,
    "config": {
        ...
    }
  }
}
```

Listing 4.1: JSON Beschreibung eines RCE-Testnetzwerks

Auf der ersten Ebene gibt es drei unterschiedliche JSON-Objekte die innerhalb der Experimental-Cloud-Umgebung unterschiedlich ausgeprägte RCE-Instanzen anlegen. Diese Ausprägungen wurden in Abschnitt 3.1 genauer vorgestellt.

Innerhalb der JSON-Objekte befinden sich drei weitere JSON-Parameter die wie folgt beschrieben werden:

image Dieser Parameter erlaubt die Angabe eines RCE-Container-Images

replicas Die Anzahl der erzeugten RCE-Instanzen

config Innerhalb dieses Objektes können RCE-spezifische Konfigurationsparameter mitgegeben werden. Weitere Informationen können dem User Guide entnommen werden. [14]

Die Angabe von unterschiedlichen Container-Images erlaubt die Ausführung von unterschiedlichen RCE Versionen innerhalb eines Testnetzwerks.

Derzeit kann mit dieser Beschreibung nur ein Testnetzwerk mit einer Sterntopologie erstellt werden. Dies stellt für den Testprozess kein Problem dar, da dies die Referenztopologie für RCE-Netzwerke ist. Im Abschnitt 6 wird der Gedanke zu weiteren Topologien noch einmal aufgegriffen.

4.2 Anforderungen an die zu entwickelnde Software

Um Chaos-Engineering Methoden in das vorhandene Software-Engineering Projekt zu integrieren, muss eine Erweiterung der bereits vorhandenen Dienste stattfinden.

Aus den Principles of Chaos Engineering (Sektion 2.4) den Experteninterviews (Sektion 3.5) sowie der Beschreibung der Ausgangslage in Abschnitt 4.1 werden die folgenden Anforderungen an eine Erweiterung des vorhandenen Softwareprojekts entnommen:

- Der Nutzer kann Chaos-Szenarien flexibel und strukturiert beschreiben
- Der Nutzer kann Chaos-Experimente anpassen
- Der Nutzer kann Chaos-Experimente zu einem beliebigen Zeitpunkt starten und stoppen
- Das System kann den RCE-Netzwerkzustand überprüfen
- Das System ist in der Lage Chaos-Szenarien automatisiert auszuführen
- Die Erweiterung sollte als Microservice in die vorhandene Systemlandschaft integriert werden

4.3 Erweiterung des vorhandenen Gesamtsystems

Aus Kapitel 4.1 geht hervor, auf welcher Softwaretechnologie aufgesetzt wird und welche vorhandenen Systeme erweitert werden. Innerhalb dieses Kapitels wird die Erweiterung des Softwareprojekts zur automatischen Erstellung von RCE Netzwerken vorgestellt. Auch am Kubernetes System muss eine Erweiterung durchgeführt werden um die Durchführung von Chaos-Experimenten zu unterstützen.

4.3.1 Chaos Mesh

Kubernetes selbst bietet keine Möglichkeit zur Durchführung von Chaos-Experimenten oder zur absichtlichen Drosselung von Netzwerkverbindungen. In Kapitel 2.4 werden bereits Softwareprojekte als Vertreter der unterschiedlichen Chaos-Kategorien vorgestellt.

Chaos-Mesh ist ein Sandbox-Projekt der Cloud-Native-Computing-Foundation (CNCF). [8] Es wurde am 28.07.2020 aufgenommen, nur sieben Monate nach der Veröffentlichung der Software auf Github. Die Kernfunktionalität ist das Erweitern von Kubernetes um die technische Planung und Durchführung von Chaos-Experimenten auf bereits bestehenden Containern. Dabei integriert es sich in die vorhandene Infrastruktur und das Deployment von Ressourcen innerhalb der Cloud-Umgebung. Bereits bestehende Tools zur Steuerung von Ressourcen in Kubernetes wie zum Beispiel kubectl, können hiermit auch zur Erstellung von Chaos-Experimenten gentzt werden. Ermöglicht wird das durch Custom-Resource-Definitions (CRDs), die die Kubernetes API um benutzerdefinierte Ressourcen erweitern, die anschließend wie Standardkomponenten verwaltet werden. [5]

Chaos-Mesh bringt die Unterstützung von folgenden Chaos-Experiment-Typen mit [1]:

PodChaos Aktives Manipulieren von Containern, Herunterfahren oder randomisierte Löschung von Containern

Network Faults Detailierte Manipulation von Netzwerkverbindungen an definierten Containern zu bestimmten Zielen. Unterstützung von Netzwerklatenz, Paketverlust, Jitter, Netzwerkpartitionierung, Bandbreitenbeschränkungen

Stress Scenarios Auslastung der CPU oder des Arbeitsspeichers innerhalb eines Containers

File I/O Faults Latenz oder Durchsatz eines Lese- / Schreibzugriffs begrenzen, simulieren von langsamen Storage

Time Faults Künstlicher Zeitdrift der Uhren innerhalb eines spezifischen Containers

Um möglichst vielfältige Chaos-Szenarien zu ermöglichen, werden zusätzlich zeitgesteuerte Abläufe sowie Workflows durch Chaos-Mesh unterstützt, in denen die verschiedenen Arten von Experimenten parallel oder in Reihe ausgeführt und wiederkehrende Experimente geplant werden können.

Aufgrund der weitreichenden Unterstützung von verschiedenen Chaos-Experimenten und guter Integration in die bereits vorhandene Experimental-Cloud-Umgebung wird Chaos-Mesh zur technischen Umsetzung der Experimente benutzt. Die Installation erfolgt über ein durch die Entwickler bereitgestellte Installationspaket (HELM-Chart) in Version 2.0. [9]

4.4 Chaos-Operator Microservice

Innerhalb dieser Sektion wird der Chaos-Operator Microservice vorgestellt, der den vorhandenen Softwarestack um direkte Möglichkeiten zur Erstellung von Chaos-Experimenten erweitert. Weiterhin wird eine Ablaufsteuerung in Abschnitt 4.4.5 vorgestellt. Die Ablaufsteuerung führt Chaos-Szenarien aus, die nach dem Vorbild der Principles of Chaos-Engineering (Abschnitt 2.4) geschaffen werden.

4.4.1 Erweiterung der Systemarchitektur

Um Chaos-Experimente in das vorhandene Softwareprojekt zu integrieren, wird die bereits vorhandene Systemarchitektur um einen weiteren Microservice erweitert.

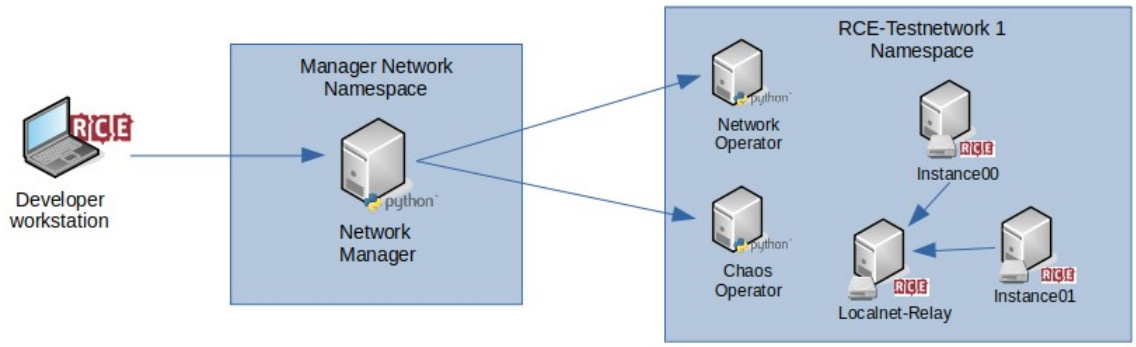

Abbildung 4.3: Erweiterung der Systemarchitektur mit dem Chaos-Operator Microservice

Wie in Abbildung 4.3 dargestellt, wird innerhalb jedes Testnetzwerks ein Chaos-Operator erstellt. Die Aufgabe des Chaos-Operator ist das Erstellen, Löschen und Konfigurieren von Chaos-Experimenten innerhalb des eigenen Testnetzwerks. Das bietet den Vorteil, dass nur Daten über das eigene Testnetzwerk gespeichert und verwaltet werden. Dadurch wird ebenfalls eine unabsichtliche Manipulation von anderen Testnetzwerken ausgeschlossen.

4.4.2 Erweiterung der Schnittstellen

Die bereits vorhandenen Webschnittstellen des Manager und Operator werden für die Integration des Chaos-Operator erweitert.

Um die Funktionen des Chaos-Operator von außen für den Nutzer erreichbar zu machen, wird die Proxy-Funktion des Manager um eine weitere Route für den Chaos-Operator erweitert. (Abbildung 4.3)

Der Operator wird um eine Schnittstelle für Probes erweitert, die eine technische Lösung für die Überprüfung einer Chaos-Szenario Hypothese bieten. Die Funktionsweise der Probes können Abschnitt 4.4.4 entnommen werden.

Der Chaos-Operator stellt folgende Funktionen bereit.

- Erstellen und Löschen eines Chaos-Experiments
- Starten und Stoppen eines Chaos-Experiments
- Ausgabe von Detailinformationen der erstellten Chaos-Experimente

Die neu geschaffenen Schnittstellen ermöglichen, ein Chaos-Experiment innerhalb eines RCE-Testnetzwerks manuell anzulegen und damit aktiv die Netzwerkkonnektivität von RCE-Instanzen beeinflussen.

4.4.3 Beschreibung eines Chaos-Experiments

Ein Chaos-Experiment wird als Chaos-Mesh Ressource mit einer umfangreichen Beschreibung in YAML dargestellt. Für einige Experimente, wie das Hinzufügen von Netzwerklatenz zwischen Containern, müssen weitere Parameter die, die Verbindungsqualität parametrisieren übergeben werden.

Der Chaos-Operator soll dem Nutzer auch hier eine möglichst einfache Möglichkeit bieten, ein Chaos-Experiment zu erstellen. Für diesen Zweck wird eine vereinfachte Beschreibung

eines Experiments im JSON Format genutzt. Für weitere Parameter, die nicht für den reinen Testablauf von außen veränderbar sein müssen, wurden Standardwerte mit den RCE-Entwicklern abgestimmt und innerhalb der Software definiert.

Mit der prototypischen Implementierung werden drei Typen von Chaos-Experimenten unterstützt, die zukünftig für das RCE-Entwicklungsteam die meiste Relevanz für Chaos-Szenarien besitzen.

Delay Verzögerung des Netzwerkverkehrs

```
{
    "id": "exp_delay",
    "type": "delay"
    "latency_in_ms": 200,
    "from_label_selector": "uplink-relay",
    "to_label_selector": "compute-instance"
}
```

Listing 4.2: JSON-Beschreibung eines Delay Chaos Experiments

Netzwerkpartitionierung Trennung eines logischen Netzwerks in mehrere Subnetzwerke

```
{
    "id": "exp_partition",
    "type": "partition"
    "from_label_selector": "uplink-relay",
    "to_label_selector": "compute-instance"
}
```

Listing 4.3: JSON-Beschreibung eines Partition Chaos Experiments

Packet Loss Paketverluste zwischen einem oder mehreren Containern

```
{
    "id": "exp_packetloss",
    "type": "loss"
    "percent_package_loss": 50,
    "from_label_selector": "uplink-relay",
    "to_label_selector": "compute-instance"
}
```

Listing 4.4: JSON-Beschreibung eines Packetloss Chaos Experiments

Innerhalb der Definition werden Labels verwendet, um die Quelle sowie das Ziel des Chaos-Experiments auszuwählen. Zur Auswahl werden die Namen der Rollen der jeweiligen RCE-Instanz genutzt. In diesem Beispiel wird jeweils die Verbindung vom Uplink-Relay zu den einzelnen RCE-Instanzen eingeschränkt.

Die Entscheidung für diese drei Experiment-Typen erfolgte basierend auf den Erkenntnissen aus dem Experteninterview die in Sektion 3.5 vorgestellt werden. Im weiteren Verlauf dieser Arbeit wird ausschließlich das Chaos-Experiment, das eine Netzwerkpartitionierung in einem RCE-Netzwerk durchführt, benutzt. Die Nutzung der Netzwerkpartitionierung als Beispiel für Chaos-Experimente illustriert dabei hinreichend das Konzept, da die weiteren Chaos-Experimente auf gleichem Wege in ein Chaos-Szenario integriert werden können.

4.4.4 Probes

Ein Kernkriterium für die Durchführung eines Chaos-Szenarios ist die Überprüfung der formulierten Hypothese. Dies wird innerhalb dieser Arbeit mit der Hilfe einer **Probe** durchgeführt. Eine Probe (deutsch „Sonde“) ist ein Softwaremodul zur Überprüfung einzelner oder mehrerer Spezifikationen einer Hypothese des Chaos-Szenarios. In dem in Abschnitt 3.6 definierten Chaos-Szenario ist die Spezifikation dass nach dem Eintreten des Chaos-Experiments, die Netzwerksicht einzelner oder mehrerer Instanzen nicht mehr konsistent ist.

Die Netzwerksicht von RCE-Instanzen ist konsistent wenn die folgenden drei Bedingungen erfüllt sind.

1. Alle RCE-Instanzen sehen die gleichen Nachbarn
2. Alle RCE-Instanzen sehen die gleichen freigegebenen Tools
3. Die Gesamtanzahl der sichtbaren Instanzen stimmt mit der Anzahl der Instanzen aus der Infrastrukturbeschreibung überein. (Abschnitt 4.1.5)

Die notwendigen Informationen erhält der Operator dazu über eine gesicherte Netzwerkverbindung zur RCE-Konsole. Über das Kommando „components list“ auf der jeweiligen Instanz kann der Operator die notwendigen Informationen abholen. Alle Sichten werden miteinander verglichen und mit den vorhandenen Netzwerkparametern abgeglichen. Insofern diese übereinstimmen hat die Probe den Status `passed` andernfalls den Status `failed`. Die Antwort der Probe Schnittstelle nutzt das folgende Format.

```
{
    "status": "passed",
    "debug": {
        ...
    }
}
```

Listing 4.5: JSON Beschreibung einer erfolgreichen Probe

Im Debug-Teil der JSON Beschreibung, werden weitere Detailinformationen zur jeweiligen Sicht der Instanzen hinterlegt. Mit diesen Informationen kann ein RCE-Entwickler den Status der Probe zu einem späteren Zeitpunkt besser nachvollziehen.

Innerhalb dieser Arbeit wird lediglich eine Probe vorgestellt, die zur Überprüfung der Hypothese in Abschnitt 3.6 aufgestellt wird.

4.4.5 Ablaufsteuerung

Fast alle Komponenten um ein Chaos-Szenario automatisiert im Experimental-Cloud-System auszuführen sind nun vorhanden. Es fehlt nun lediglich die Steuerung des Ablaufs eines Chaos-Szenarios. In Kapitel 3.6 wurde ein Chaos-Szenario an einem Testfall beschrieben, der aus den Experteninterviews als besonders wertvoll hervorging. Die Beschreibung beinhaltet aber nur den Teil, der getestet wird und unter welchen Einschränkungen durch Chaos-Experimenten dieser Test stattfindet. Die Ablaufsteuerung für Chaos Engineering **Chaos-Sequencer** wird das vollständige Chaos-Szenario ausführen. Hierzu nutzt es die bereits vorliegenden und neu geschaffenen Schnittstellen am Experimental-Cloud-System um Ressourcen anzulegen, den Status des RCE-Netzwerks abzurufen, Chaos Experimente anzulegen und schlussendlich die Umgebung zu löschen.

Der Ablauf eines Chaos-Szenarios wird in drei Phasen aufgeteilt.

- Pre-Chaos-Phase
- Chaos-Phase
- Post-Chaos-Phase

Innerhalb der Pre-Chaos-Phase werden die Grundvoraussetzungen für die Durchführung eines Chaos-Szenarios geschaffen. Die notwendige Infrastruktur wird aufgebaut und durch eine Probe auf Funktionsfähigkeit überprüft. Anschließend werden die definierten Chaos-Experimente im pausierten Zustand angelegt.

In der Chaos-Phase werden benutzerdefinierte Aktionen, wie das Starten eines RCE-Workflows, ausgeführt. Im Anschluss werden die in der Pre-Chaos-Phase angelegten Chaos-Experimente gestartet und der erwartete Zustand des Netzwerks überprüft. Nach dem definierten Zeitraum (Chaos Duration) wird die Chaos-Phase mit dem Pausieren der Chaos-Experimente gestoppt.

An die Deaktivierung der Chaos-Experimente schließt sich die Post-Chaos-Phase an. Dem Testnetzwerk wird ein konfigurierbarer Zeitraum (Infrastructure Recovery Time) gegeben um sich selbstständig wieder von den Chaos-Experimenten zu erholen. Im Anschluss wird ein letztes Mal überprüft, ob der erwartete Zustand des RCE-Netzwerks vorliegt. Zum Schluss werden die Debug-Logs aller Instanzen eingesammelt, für den ausführenden Benutzer gespeichert und das RCE-Testnetzwerk gelöscht.

Der Abbildung 4.4 kann die konkrete Arbeitsweise entnommen werden.

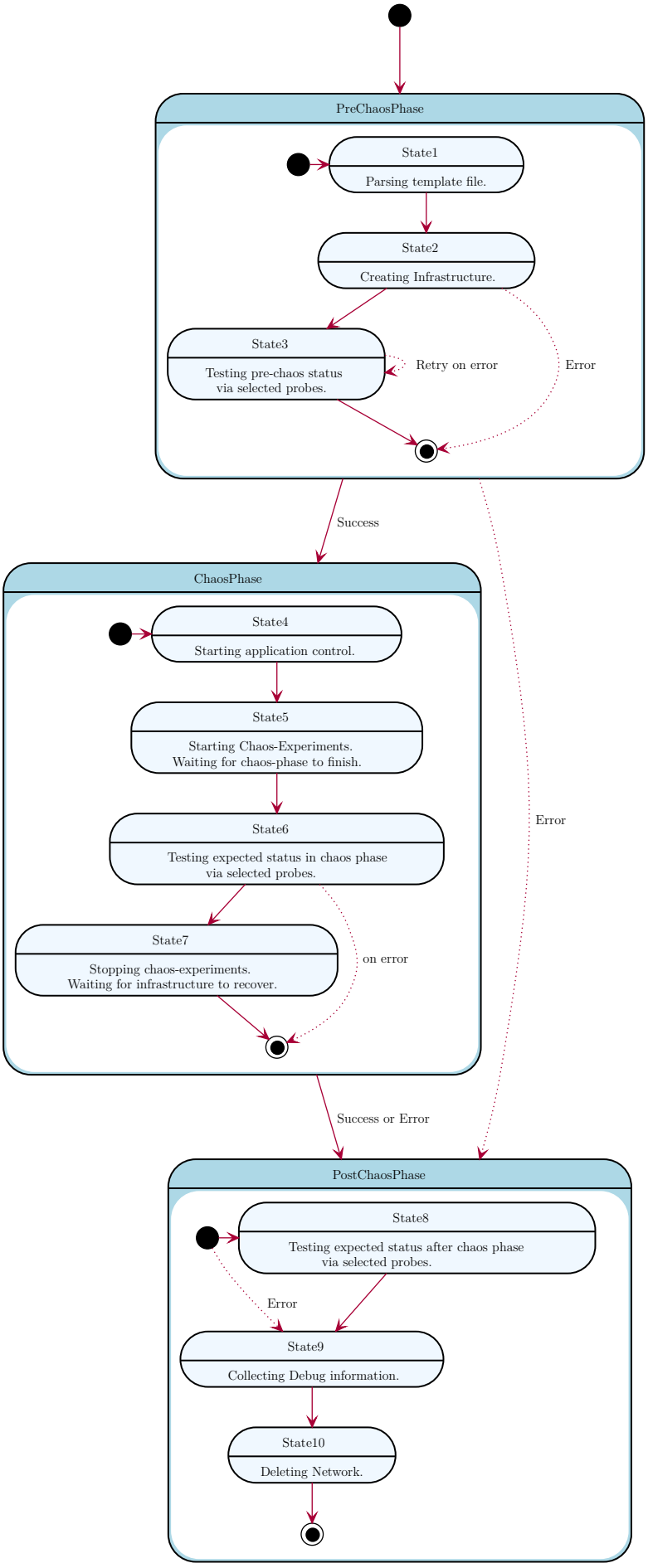

Abbildung 4.4: Ablauf Chaos-Szenario - UML-Zustandsdiagramm

5 Evaluierung und Diskussion

Innerhalb dieses Kapitels wird das System nun praktisch erprobt, sowie das in Kapitel 3 entworfene Chaos-Szenario in ein Format gebracht, dass der Chaos-Sequencer zur Ausführung des Chaos-Szenarios nutzen kann. Weiterhin werden die Resultate des Chaos-Szenarios vorgestellt.

5.1 Geplantes Chaos-Szenario

Ein Chaos-Szenario wurde anhand der Principles of Chaos-Engineering bereits in Abschnitt 3.6 beschrieben und das Experimental-Cloud-System wurde in Abschnitt 4.4.3 um die Möglichkeit der Ausführung von Chaos-Experimente erweitert. Das Szenario muss nun in eine Form gebracht werden, welches der Chaos-Sequencer intepretieren und ausführen kann.

5.1.1 JSON Beschreibung eines Chaos-Szenarios

Auch das vollständige Chaos-Szenario wird als JSON-Dokument dargestellt. Innerhalb der Beschreibung werden bereits vorhandene und vorgestellte Teile benutzt, wie beispielsweise die Beschreibung der Infrastruktur aus Abschnitt 4.1.5 und die Definition von Chaos-Experimenten aus Abschnitt 4.4.3. Aber auch weitere notwendige Bestandteile werden der Beschreibung eines Chaos-Szenarios hinzugefügt, wie beispielsweise die Beschreibung von Probes die in Abschnitt 4.4.4 vorgestellt wurden. Im Folgenden werden die einzelnen Bestandteile der Definition eines Chaos-Szenarios beschrieben.

Grundstruktur der Beschreibung eines Chaos-Szenarios

Die Grundstruktur der Beschreibung des Chaos-Szenarios sieht im Konkreten wie folgt aus:

```
{
  "chaos-scenario": {
    "id": "chaos-scenario-2342",
    "chaos_duration_in_s": 60,
    "infrastructure_recovery_time_in_s": 120,
    "probes": {...},
    "infrastructure": {...},
    "application_control": {...},
    "experiments": [...]
}
```

Listing 5.1: Grundstruktur der JSON Beschreibung eines Chaos-Szenarios

Innerhalb dieser Grundstruktur finden sich die essentiellen Bestandteile eines Chaos-Szenarios sowie die einzelnen konfigurierbaren Zeiträume, die in Abschnitt 4.4.5 vorgestellt werden, wieder. Die einzelnen Objekte sind dabei selbsterklärend oder werden folgend, wie das Objekt `application_control`, in diesem Abschnitt genauer vorgestellt.´

Beschreibung von Probes

Manche Hypothesen sind nicht durch eine Probe allein überprüfbar. Dies wurde innerhalb der Beschreibung ebenfalls berücksichtigt. Die Probes jeder einzelnen Phase des Chaos-Szenarios können unabhängig voneinander beschrieben werden. Innerhalb der JSON-Beschreibung ist dies wie folgt umgesetzt:

```
{
    "probes": {
      "pre": {
        "network_views_consistent": "true",
        "workflow_running": "false"
      },
      "chaos": {
        "network_views_consistent": "false",
        "workflow_running": "true"
      },
      "post": {
        "network_views_consistent": "true",
        "workflow_running": "true"
      }
    }
}
```

Listing 5.2: JSON Beschreibung der Probes innerhalb der Phasen eines Chaos-Szenarios

Innerhalb der Struktur kann zu jeder einzelnen Probe ein erwartetes Ergebnis in der konkreten Phase definiert werden. Zum Beispiel ist anzunehmen, dass während einer Netzwerkpartitionierung die Netzwerkverbindung einer RCE-Instanz verloren geht und sie so die Verbindung zu ihren Nachbarn verliert. In der Chaos-Phase eines Chaos-Szenarios ist das Fehlschlagen einer Probe in diesem konkreten Beispiel ein erwartetes Ergebnis.

Beschreibung von benutzerdefinierten Befehlen

Am Anfang der Chaos-Phase werden benutzerdefinierte Befehle innerhalb des RCE-Netzwerks ausgeführt. Dies kann beispielsweise das Starten eines Workflows sein. Innerhalb der JSON-Beschreibung werden diese Schritte unter dem Objekt `application_control` wie folgt definiert:

```
{
  "chaos-scenario": {
    ...
    "application_control": {
      "actions": [{
          "endpoint": "/cmd",
          "payload": {
            "command": "wf start filetransfer.wf"
          },
          "instance": "localnet-relay"
        },
        {
        ...
        }]
    },
    ...
}
```

Listing 5.3: JSON Beschreibung eines Chaos-Szenarios - Befehle im RCE-Testnetzwerk

Über den Parameter `endpoint` werden unterschiedliche API-Endpunkte gewählt. Das JSON-Objekt innerhalb von `payload` wird anschließend in der Phase zu dem definierten Endpunkt geschickt und damit eine Aktion innerhalb des Testnetzwerks ausgelöst. Beispielweise könnte damit nicht nur ein Befehl innerhalb einer ausgewählten RCE-Instanz ausgeführt werden, sondern auch eine Instanz neu gestartet, gelöscht oder die Konfiguration der Instanz verändert werden.

Beschreibung der Infrastruktur und Chaos-Experimente

In die Beschreibung des Chaos-Szenarios wird ebenfalls die schon bereits vorhandene Infrastrukturbeschreibung, die in Abschnitt 4.1.5 vorgestellt wird, integriert. Die Beschreibung der Chaos-Experimente aus Abschnitt 4.4.3, wird ebenfalls integriert.

Gesamtbeschreibung des Chaos-Szenarios

Alle vorgestellten Teile werden zu einer vollständigen Beschreibung des Chaos-Szenarios aus Abschnitt 3.6 zusammengeführt. Diese sieht wie folgt aus:

```
{
  "chaos-scenario": {
    "id": "chaos-scenario-2342",
    "chaos_duration_in_s": 60,
    "infrastructure_recovery_time_in_s": 120,
    "probes": {
      "pre": {
        "network_views_consistent": "true"
      },
      "chaos": {
        "network_views_consistent": "false"
      },
      "post": {
        "network_views_consistent": "true"
      }
    },
    "application_control": {
      "actions": [{
          "endpoint": "/cmd",
          "payload": {
            "command": "wf start filetransfer.wf"
          },
          "instance": "localnet-relay"
        }]
    },
    "infrastructure": {
      "uplink-relay": {
        "image": "rce:10.2.4",
        ...
      },
      "localnet-relay": {
        "image": "rce:10.2.4",
        ...
```

```
      },
      "instances": {
        "image": "rce:10.2.4",
        "replicas": 5,
        ...
      }
  },
  "experiments": [{
      "id": "exp-partition",
      "type": "partition",
      "from_label_selector": "localnet-relay",
      "to_label_selector": "compute-instance",
    }]
}
```

Listing 5.4: JSON Beschreibung eines Chaos-Szenarios

Die Auslassungpunkte im Objekt `infrastructure` stehen für die benutzerdefinierte Konfiguration die den einzelnen Rollen der RCE-Instanzen mitgegeben werden können. Innerhalb der Konfiguration wird beispielsweise die notwendige Wiederverbindung nach einem Netzwerkfehler von RCE-Instanzen konfiguriert. Durch diesen Konfigurationsparameter können sich RCE-Instanzen vom geplanten Chaos-Experiment wieder erholen. Die Beschreibung wird anschließend in der Datei „chaos-scenario.json“ auf dem lokalen Computer im Verzeichnis des Chaos-Sequencers abgelegt.

5.2 Durchführung des entworfenen Chaos-Szenarios

Die RCE Releaseversion 10.2.4 wurde am 18.09.2021 veröffentlicht, so dass die Erkenntnisse und die implementierte Software leider nicht in diese Testphase mit einfließen konnten. Dies resultierte aus einer, zu diesem Zeitpunkt noch nicht hinreichend funktionsfähigen Implementierung der Software die in dieser Arbeit vorgestellt wird. Innerhalb der Softwareentwicklung von RCE gibt es keinen festgelegten Releasezyklus. Grundsätzlich zeichnet sich aber ein Trend von rund drei Monaten ab.

Allerdings wurden in der Testphase von 10.2.4 bereits einzelne Bestandteile dieser Arbeit manuell benutzt um einen ersten Eindruck der Möglichkeiten von Chaos-Engineering zu bekommen. Dazu gehörte die automatische Erstellung eines Testnetzwerks und das manuelle Anwenden von Chaos-Experimenten auf einzelne RCE-Instanzen.

Für die Überprüfung des Gesamtsystems das innerhalb dieser Arbeit vorgestellt wird, wurde eine nachträgliche Testphase geschaffen die sich an den Release der Version 10.2.4 anschließt.

5.2.1 Ausführung mit Chaos-Sequencer

Die Ablaufsteuerung Chaos-Sequencer, die bereits in Abschnitt 4.4.5 vorgestellt wurde, ist eine in Python 3 entwickelte Software, die auf der lokalen Workstation eines RCE-Entwicklers ausgeführt wird. Für die Ausführung der Software wird das DLR-interne Software Repository heruntergeladen und anschließend notwendige Abhängigkeiten der Software nachinstalliert.

Nun kann das Szenario mit folgendem Aufruf ausgeführt werden:

```
> python3 main.py --namespace rcetestnet0
```

Listing 5.5: Aufruf Chaos-Sequencer mit definiertem Testnetzwerknamen

Anschließend führt die Software vollständig automatisiert das beschriebene Chaos-Szenario im Testnetzwerk `rcetestnet0` aus und spricht mit den Schnittstellen die innerhalb des Experimental-Cloud-Systems bereitgestellt wurden. Der Chaos-Sequencer gibt dem Nutzer innerhalb der Ausführung stets Rückmeldung zum aktuellen Status der Ausführung:

```
### Chaos-Sequencer. Test RCE with a predefined chaos-scenario.
0: Parsing scenario file.
   SUCCESS
1: Creating infrastructure.
   Waiting for infrastructure to be up.
   SUCCESS
2: Testing pre-chaos status via selected probes
   probe-result: passed
   SUCCESS
3: Creating chaos experiments.
   creating experiment with id exp-partition-rcetestnet0.
   SUCCESS
4: Starting application control.
   "Executing: 'filetransfer.wf'"
   "Workflow Id: 57c41398-8221-49cb-a9ca-f8d7e8e40f73"
   SUCCESS
5: Activating Experiments. Waiting for chaos-phase to finish.
   Waiting for 60s in chaos-phase as defined.
   SUCCESS
```

```
6: Testing expected status in chaos phase via selected probes
   probe-result: failed
   SUCCESS
7: Deactivating chaos-experiments.
   Waiting some time for infrastructure to recover.
   SUCCESS
8: Test post-chaos status via selected probes.
   probe-result: passed
   SUCCESS
9: Collecting Debug information.
   Debug-Logs saved to ./chaos-scenario-2342/
   SUCCESS
10: Deleting network.
   SUCCESS
Chaos-Scenario result: SUCCESS.
```

Listing 5.6: Ausgaben auf dem Terminal für Nutzer - Chaos Sequencer

Wenn alle Ausgaben mit `SUCCESS` quittiert wurden, ist das Chaos-Szenario abgeschlossen. Das Testnetzwerk wird im Anschluss gelöscht und Debug-Informationen zu jeder erzeugten Instanz werden für den RCE-Entwickler lokal innerhalb eines Ordners zur besseren Nachvollziehbarkeit abgespeichert.

5.2.2 Validierung

Um sicherzustellen, dass die RCE-Instanzen tatsächlich manipuliert werden und das System einen Fehlerzustand der RCE-Instanzen feststellen kann, wurde ein Negativtest durchgeführt. Hierzu wird das Chaos-Szenario gezielt in einen Fehlerzustand versetzt.

Innerhalb der RCE-Instanzen wird dafür die Funktion der automatischen Wiederverbindung bei einem Verbindungsfehler deaktiviert und das Szenario noch einmal ausgeführt.

Durch die Deaktivierung der Wiederverbindung werden die Instanzen in einen Zustand versetzt, der nicht der Spezifikation innerhalb der Hypothese entspricht, da nach der Chaos-Phase das RCE-Netzwerk wieder vollständig wiederhergestellt sein muss. Hierbei könnte es zwei mögliche Resultate geben, die auf einen Fehler innerhalb der Funktionsweise des Systems hinweisen:

Probes erkennen einen Fehlerzustand nicht Die innerhalb des Systems implementierte Probe `network_views_consistent` kann einen Fehlerzustand in der Netzwerksicht der Instanzen nicht feststellen.

Chaos-Experimente beeinflussen die Instanzen nicht Chaos-Experimente werden erfolgreich innerhalb des Testnetzwerks angelegt, wirken sich aber nicht auf die Netzwerkverbindungen der RCE-Instanzen aus

Das Ergebnis der Probe wurde explizit so definiert, dass eine Zustandsänderung stattfinden muss um die korrekte Funktionsweise der Erkennung eines Fehlerzustandes sicherzustellen. Sollte diese Erkennung nicht funktionieren, würde die Netzwerktrennung innerhalb der Chaos-Phase nicht detektiert werden und dadurch das Chaos-Szenario in der Phase 6 „Testing expected status in chaos phase via selected probes“ fehlschlagen.

Sollten Chaos-Experimente das Netzwerk einer RCE-Instanz nicht beeinflussen, würden wir den Fehlschlag ebenfalls in Phase 6 sehen, da dann keine Netzwerktrennung stattfindet und die Probe den Zustand `passed` zurückliefert.

Im Fall, dass beide oben beschriebenen Fehlerfälle zutreffen, können zusätzlich die gesammelten Logdateien herangezogen werden um den Zustand der Applikation manuell zu prüfen und sicherzustellen, dass eine Netzwerktrennung auf Applikationsebene detektiert wurde. Das Ergebnis der zweiten Ausführung sieht wie folgt aus:

```
### Chaos-Sequencer. Test RCE with a predefined chaos-scenario.
0: Parsing scenario file.
   SUCCESS
1: Creating infrastructure.
   Waiting for infrastructure to be up.
   SUCCESS
2: Testing pre-chaos status via selected probes
   probe-result: passed
   SUCCESS
3: Creating chaos experiments.
   creating experiment with id exp-partition-rcetestnet0.
   SUCCESS
4: Starting application control.
   "Executing: 'filetransfer.wf'"
   "Workflow Id: 57c41398-8221-49cb-a9ca-f8d7e8e40f73"
   SUCCESS
5: Activating Experiments. Waiting for chaos-phase to finish.
   Waiting for 60s in chaos-phase as defined.
   SUCCESS
6: Testing expected status in chaos phase via selected probes
   probe-result: failed
   SUCCESS
7: Deactivating chaos-experiments.
   Waiting some time for infrastructure to recover.
   SUCCESS
```

```
8: Test post-chaos status via selected probes.
   probe-result: failed
   FAILED
9: Collecting Debug information.
   Debug-Logs saved to ./chaos-scenario-2342/
   SUCCESS
10: Deleting network.
   SUCCESS
Chaos-Scenario result: FAILED.
```

Listing 5.7: Ausgaben auf dem Terminal für Nutzer - Chaos Sequencer Validierung

Aus dem Listing 5.7 kann entnommen werden, dass ein Fehler in Phase 8 detektiert wurde. Dieses Ergebnis ist erwartet und zeigt, dass mit einer deaktivierten Wiederverbindung sowohl die Netzwerkverbindung von RCE-Instanzen durch das Chaos-Experiment getrennt wird, als auch die korrekte Detektierung des fehlerhaften Zustands durch die eingesetzte Probe. Auch in den zusätzlich geprüften Debug-Informationen der einzelnen RCE-Instanzen konnte eine Netzwerktrennung durch das Chaos-Experiment nachvollzogen werden.

5.3 Resultate

Im vorhergehenden Abschnitt zur Funktionsweise und Validierung des Gesamtsystems wurde gezeigt, dass das entworfene Gesamtsystem wie erwartet arbeitet. Basierend auf den Anforderungen in Abschnitt 4.2 sind folgende Ziele erreicht wurden:

- Der Nutzer kann ein Chaos-Szenario für RCE in einer maschinenlesbaren Form ausdrücken
- Der Nutzer kann Chaos-Experimente auf eine Testumgebung anwenden, pausieren und zu einem beliebigen Zeitpunkt starten
- Der Nutzer ist in der Lage, eine angegebene Spezifikation durch eine Probe zu überprüfen
- Der Nutzer kann das Chaos-Szenario automatisch und ohne Interaktion ausführen
- Der Nutzer wird in der Auswertung von Fehlern durch die automatische Erhebung und Speicherung von Debug-Informationen unterstützt

- Der Nutzer kann Applikationsbefehle, wie das Starten eines Workflows innerhalb des Testnetzwerks ausführen

Durch die neuen Möglichkeiten, die den RCE-Entwicklern nun zur Verfügung stehen, können weitere Erkenntnisse zum Verhalten der verteilten Applikation RCE in Grenzsituationen und Fehlerfällen gesammelt werden. Allerdings erfordert das Vorbereiten eines Chaos-Szenarios mit der Implementierung der jeweiligen Probe zusätzliche Implementierungszeit. Daher kann insgesamt vor allem durch Chaos-Szenarien profitiert werden, die mit jedem Release neu ausgeführt werden müssen (Regressionstests), um beispielsweise die korrekte Wiederherstellung der Verbindung nach einem Netzwerkausfall zu überprüfen. Für Chaos-Szenarien die nur sehr selten oder einmalig ausgeführt werden, muss eine klare Abwägung stattfinden, ob die einmalige Erkenntnis den zusätzlich anfallenden Implementierungsaufwand rechtfertigt.

Chaos-Engineering weist aber auch Grenzen auf. Im Gegensatz zu den Testverfahren die bisher verwendet werden, kann keine genaue Aussage über die Herkunft eines Fehlers gemacht werden. Die Analyse des beobachteten Fehlverhaltens der Applikation bleibt damit weitestgehend dem Entwickler selbst überlassen. Der Entwickler kann immerhin von der automatischen Erhebung der Debug-Informationen profitieren, die sonst manuell zusammengeführt werden müssten.

6 Fazit

Mit der in dieser Arbeit vorgestellten Erweiterung des bereits vorhandenen Softwareprojekts „rce-test-network“ kann Chaos-Engineering weitergehend in den Softwaretestprozess integriert werden. Dabei konnte mit Hilfe der Principles of Chaos-Engineering ein Testszenario, basierend auf den Erkenntnissen des Experteninterviews in Abschnitt 3.5, beschrieben und anschließend in eine maschinenlesbare Form gebracht werden. Dies ermöglicht eine automatische Ausführung der Chaos-Szenarien, bis hin zu einer Detektion von unerwartetem Verhalten und dem automatischen Erheben von Debug-Informationen, die den Entwickler bei der Untersuchung des festgestellten Fehlverhaltens unterstützen. Durch die Ausführung des Szenarios wurde sichergestellt, dass die automatische Wiederverbindung bei einem Netzwerkausfall und die Herstellung einer konsistenten Netzwerksicht in RCE-Version 10.2.4 funktioniert.

Komplexere Testfälle, die eine künstliche Manipulation der Netzwerkschnittstelle benötigen, wurden vorher nur sehr selten genutzt, da sie einen erheblichen manuellen Aufwand durch die Vorbereitung und Ausführung erzeugen und nur mit zusätzlicher Abstimmung der Teilnehmer einer Testphase durchgeführt werden können. Mit der Nutzung von modernen Cloud-Technologien wie Kubernetes und der Integration von Chaos-Mesh (Abschnitt 4.4.3) können weitaus größere Testnetzwerke erzeugt und Instanzen, mit einem einzigen Aufruf für das Testszenario manipuliert werden.

Die Integration von Chaos-Engineering schafft für die RCE-Entwickler in der Zukunft die Möglichkeit interessante neue Testfälle zu erstellen, die durch bereits eingesetzte Testverfahren (Abschnitt 3.2) nicht möglich sind. Auch neue Versionen die sich kurz vor Release befinden, können so mit geringem Aufwand auf die korrekte Funktionsweise innerhalb der spezifizierten Rahmenbedingungen getestet werden.

Damit zeigt diese Arbeit, dass Chaos-Engineering auch für verteilte Software, die nicht nach dem Bild der Microservice Architektur entwickelt wurde, einen Nutzen bieten kann.

Offene Probleme und weitere Arbeiten

Auch wenn die Integration von Chaos-Engineering viele neue Testszenarien ermöglicht und die Vielfalt an verfügbaren Testszenarien erhöht, so gibt es einige weitere offene Frage- und Problemstellungen die innerhalb der Bearbeitungsphase dieser Arbeit entstanden sind.

RCE-spezifische Chaos-Experimente Die innerhalb dieser Arbeit vorgestellten Chaos-Experimente beschränken sich auf die Simulation von Netzwerkproblemen die im echten Einsatz beim Nutzer auftreten können. Weitere spezifisch für die Applikation zugeschnittene Chaos-Experimente könnten in der Zukunft ebenfalls einen erheblichen Mehrwert bieten. Dabei könnte man beispielweise die Registrierung von Tools durch ein anhaltendes An-und-Abmelden auf Robustheit testen.

Chaos-Sequencer als Python-Modul Der Chaos-Sequencer stellt eine bequeme Möglichkeit dar, ein vorher entworfenes Chaos-Szenario automatisiert auf einem lokalen Computer auszuführen. Die in Abbildung 4.4 vorgestellte Ausführung in Phasen beschränkt das Design von komplexeren Chaos-Szenarien unter Umständen zu sehr. Denkbar wäre hier, statt der starren Vorgabe der Phasen, durch die Entwickler gescriptete Abläufe zu ermöglichen. Das könnte beispielhaft mit einer Python-Schnittstelle umgesetzt werden, in der typische Aufrufe an das Experimental-Cloud-System auf eine einfacher zu nutzende Art und Weise abstrahiert werden.

Vielfältigere Parametrisierung von Probes Probes liefern derzeit bereits eine Antwort ob die Applikation innerhalb einer Spezifikation arbeitet. In der Beschreibung des Chaos-Szenarios 4.4.5 kann ein RCE-Entwickler einen erwarteten Zustand der Probe hinterlegen. Eine genauere und umfangreichere Parametrisierung von Probes wäre hierbei aber sehr hilfreich. Als Beispiel kann hier eine Probe dienen, die prüfen soll, ob ein spezifischer Workflow gerade läuft. Dabei könnte man derzeit den Namen des Workflows entweder fest in der Probe verankern oder allgemein nach einem laufenden Workflow suchen und dieses Ergebnis zurückliefern. Eine Erweiterung der Probebeschreibung um zusätzliche Parameter, wie den Namen eines Workflows, könnte hierbei für eine größere Flexibilität der Probes sorgen.

Bessere Nachvollziehbarkeit der Wirkung von Chaos-Experimenten Der Status eines Chaos-Experiments kann zu jeder Zeit über die Webschnittstelle des Chaos-Operator nachvollzogen werden. Hierbei erlangt der RCE-Entwickler lediglich die Information ob ein Chaos-Experiment derzeit pausiert ist oder aktuell läuft. Durch eine zusätzliche Überwachung der RCE-Instanzen innerhalb des Experimental-Cloud-Systems

könnten genauere Systemmetriken wie beispielsweise die CPU- oder Arbeitsspeicher-Auslastung erfasst werden. In einem Fehlerfall können die genutzten Systemressourcen in einem Graphen anschaulich dargestellt und beobachtet werden.

Zusätzliche Netzwerktopologien ermöglichen Das Softwareprojekt `rce-test-network` unterstützt nur die automatische Erstellung eines Netzwerks mit einer Sterntopologie. RCE hat bezüglich der Netzwerktopologie keine Restriktionen, so das auch weitere Topologien wie ein teilvermaschtes RCE-Netzwerk bei Nutzern vorkommen könnten. Um auch diesen Verwendungszweck besser abzudecken und weitere vielfältigere Chaos-Szenarien zu unterstützen, muss eine generalisierte Netzwerkbeschreibung entworfen werden. Diese gibt dem Nutzer die Freiheit, jede einzelne Verbindung zwischen RCE-Instanzen innerhalb der Beschreibung zu definieren.

Literaturverzeichnis

[1] Chaos Mesh, Types of Chaos-Experiments. https://chaos-mesh.org/docs/ [Aufgerufen am 29. Sep. 2021].

[2] Gitlab Docs, CI/CD Pipelines . https://docs.gitlab.com/ee/ci/pipelines/ [Aufgerufen am am 20. Sep. 2021].

[3] HELM, the package manager for Kubernetes. https://helm.sh/ [Aufgerufen am 20. Sep. 2021].

[4] Number of Netflix paid subscribers worldwide from 3rd quarter 2011 to 2nd quarter 2020. https://www.statista.com/statistics/250934/quarterly-number-of-netflix-streaming-subscribers-worldwide/ [Aufgerufen am 29. Sep. 2021].

[5] The Kubernetes Authors. Kubernetes Documentation - CustomResourceDefinition. https://kubernetes.io/docs/tasks/extend-kubernetes/custom-resources/custom-resource-definitions/ [Aufgerufen am 21. Sep. 2021].

[6] Ali Basiri, Niosha Behnam, Ruud de Rooij, Lorin Hochstein, Luke Kosewski, Justin Reynolds, and Casey Rosenthal. Chaos Engineering. *IEEE Software*, 33(3):35–41, May 2016.

[7] Pierre Bourque, R. E Fairley, and IEEE Computer Society. *Guide to the Software Engineering Body of Knowledge*. 2014.

[8] CNCF Chaos Mesh Authors. Chaos-Mesh, a powerful Chaos Engineering Platform for Kubernetes. https://chaos-mesh.org/ [Aufgerufen am 28. Aug. 2021].

[9] CNCF Chaos Mesh Authors. Chaos Mesh Helm Chart, Installation. https://github.com/chaos-mesh/chaos-mesh/tree/release-2.0/helm/chaos-mesh [Aufgerufen am 30. Aug. 2021].

[10] The RCE Developers. Source of BDD-Tests. https://github.com/rcenvironment/rce/blob/9df689f8fe9ab706cf39e10541992e53774caa2e/de.rcenvironment.supplemental.testscriptrunner.scripts/resources/scripts/ComponentTests.feature [Aufgerufen am 30. Aug. 2021].

[11] DLR. RCE - Remote component environment. https://rcenvironment.de [Aufgerufen am 24. Aug. 2021].

[12] DLR. RCE Admin Guide. https://updates-external.sc.dlr.de/rce/10.x/products/standard/releases/latest/documentation/windows/admin_guide.pdf [Aufgerufen am 21. Sep. 2021].

[13] DLR. RCE Features. https://rcenvironment.de/pages/features.html [Aufgerufen am 24. Aug. 2021.

[14] DLR. RCE User Guide. https://updates-external.sc.dlr.de/rce/10.x/products/standard/releases/latest/documentation/linux/user_guide.pdf[Aufgerufen am 25. Aug. 2021].

[15] Nicola Dragoni, Schahram Dustdar, Stephan Larsen, and Manuel Mazzara. Microservices: Migration of a Mission Critical System. *IEEE Transactions on Services Computing*, PP, April 2017.

[16] Andrew S. Tanenbaum Maarten van Steen. *Distributed Systems*. 2017.

[17] Rancher. k3s - lightweight kubernetes. https://k3s.io [Aufgerufen am 21. Aug. 2021].

[18] Jesper Simonsson, Long Zhang, Brice Morin, Benoit Baudry, and Martin Monperrus. Observability and Chaos Engineering on System Calls for Containerized Applications in Docker. *Future Generation Computer Systems*, 122:117–129, September 2021.

[19] Gurock Software. TestRail: Comprehensive Test Case Management. https://www.gurock.com/testrail/ [Aufgerufen am 20. Sep. 2021].

[20] Long Zhang, Brice Morin, Philipp Haller, Benoit Baudry, and Martin Monperrus. A Chaos Engineering System for Live Analysis and Falsification of Exception-handling in the JVM. *IEEE Transactions on Software Engineering*, pages 1–1, 2019.

Abbildungsverzeichnis

Listings