Technology
Arts Sciences
TH Köln

Evaluierung und Optimierung einer Kubernetes Architektur

am Beispiel vom Unternehmen Ambient Innovation: GmbH

Bericht zum Praxisprojekt

ausgearbeitet von

David Bajon

vorgelegt an der

Technischen Hochschule Köln
Campus Gummersbach
Fakultät für Informatik und
Ingenieurwissenschaften

im Studiengang
Medieninformatik

Betreuer: Prof. Dr. Roman Majewski
Technische Hochschule Köln

Dipl. Marius Burfey
Ambient Innovation GmbH

Inhaltsverzeichnis

Kurzfassung

Kubernetes (K8s) gehört zu einer der beliebtesten Technologien um Anwendungen zu deployen, zu skalieren und zu verwalten. Sie hat sich so stark etabliert, dass die fünf größten Cloud-Anbieter eigene „Managed Kuberentes“ Dienste entwickelt haben und diese vermieten. Trotzdem kann das Administrieren und Optimieren von Clustern zu einer Herausforderung werden. Im Rahmen dieses Projektes sollte dem Unternehmen Ambient Innovation: GmbH (Ambient) temporär bei dieser Aufgabe geholfen werden. Um ein zufriedenstellendes Ergebnis zu erreichen, wurden Tools für Alerting, Monitoring, Backups und Autoscaling implementiert, bestehende Cluster Architekturen umgebaut und Kostenoptimierung vorgenommen. Die umgesetzten Änderungen haben zu einer Verbesserung in Bezug auf Stabilität und Kosten der Umgebung und zu einem strukturierterem und leichterem Administrieren, für das zugehörige Team des Unternehmens geführt. Die Durchführung des Projektes hat ein Fundament für ein potentielles Anknüpfen einer Bachelorarbeit in einem fortführendem oder verwandtem Themengebiet geschaffen.

Akronyme

Ambient Ambient Innovation: GmbH. 2, 5, 7, 8, 11, 17, 18, 19, 20, 21, 23, 24, 26, 28, 29, 32, 34

API Application Programming Interface. 9, 18, 25

ASG Auto Scaling Group. 26, 29, 32

AWS Amazon Web Services. 7, 9, 13, 21, 22, 23, 25, 26, 27, 28, 29, 30, 32

AZ Availability Zone. 28

CD Continuous Delivery. 18, 23, 32

CI Continuous Integration. 18, 23, 32, 33

CNAME Canonical Name. 28

CPU Central Processing Unit. 5, 9, 12, 13, 14, 15, 16, 18, 30, 31, 32

DevOps Development and Operations. 34

DNS Domain Name System. 24, 27, 28

EBS Elastic Block Store. 25

EC2 Elastic Compute Cloud. 7, 8, 9, 12, 14, 20, 21, 23, 24, 25, 26, 27, 28, 29, 30, 32, 33

EKS Elastic Kubernetes Service. 26, 28, 29, 33

EPD Entsorung Punkt DE. 16

ETCD /etc distributed. 11, 12, 13, 17, 18, 20, 21, 22, 35

HA High Availability. 13, 35

HPA Horizontal Pod Autoscaler. 30, 31

HTTP Hypertext Transfer Protocol. 28

HTTPS Hypertext Transfer Protocol Secure. 28

IP Internet Protocol. 25, 28

IT Information technology. 5

K8s Kubernetes. 2, 12, 16, 17, 18, 19, 20, 21, 23, 24, 25, 26, 27, 29, 30, 32, 35

mCPU milli Central Processing Unit. 16, 33

MiB Mebibyte. 16

NLB Network Load Balancer. 28

OS Operating System. 12

RAM Random-Access Memory. 5

RDS Relational Database Service. 23

REST Representational State Transfer. 10

RKE Rancher Kubernetes Engine. 11, 12, 26

S3 Simple Storage Service. 7, 18, 20, 21, 22

SMS Short Message Service. 19

SSH Secure Shell. 8, 10, 21

TCP Transmission Control Protocol. 9

UI User Interface. 10, 12, 21, 26, 27, 29, 30

URL Uniform Resource Locator. 17, 19, 25

usw. und so weiter. 27

UX User Experience. 10

VPC Virtual Private Cloud. 26

YAML YAML Ain't Markup Language. 9, 12, 15, 20, 21, 23, 24, 27

z.B. zum Beispiel. 7, 17, 20, 27, 33, 34

1 Einleitung

Dieses Praxisprojekt wurde in Kooperation mit der Agentur Ambient durchgeführt. Durch diese Relation ergibt sich auch das eigentliche Thema des Projektes, die Optimierung und Evaluierung der Kubernetes Architektur des genannten Unternehmens. Im Folgendem wird zuerst die Ausgangssituation und das Ziel erläutert, darauf folgt die Dokumentation der praktischen Durchführung des Themas der Arbeit. Hierfür wird das Angewandte zusammengefasst, Probleme dargestellt und einzelne Entscheidungen begründet. Im Fazit folgt die persönliche Bewertung bezüglich der Erfüllung des Ziels, sowie ein Ausblick auf eine mögliche Fortsetzung.

1.1 Problemfeld & Kontext

Aktuell betreibt Ambient, eine Agentur für Information technology (IT), App- und Webentwicklung, mehrere Kubernetes Cluster mit Rancher 2. Zum Hosten der einzelnen Nodes wird primär der Clouddienst Amazon Web Services genutzt, in Ausnahmen auch Microsoft Azure oder die Open Telekom Cloud. Diese Architektur wird verwendet, um sowohl interne Firmendienste, als auch externe Kundenanwendungen zu betreiben. Im ursprünglichen Zustand verfügten die einzelnen Cluster nur über Rancher's minimalistisches Monitoring Dashboard. Dieses besteht lediglich aus einer Darstellung der durch Kubernetes reservierten Ressourcen (Central Processing Unit (CPU), Random-Access Memory (RAM) & Anzahl der Pods) in Relation zu den maximal verfügbaren Ressourcen. Auch das Alerting war sehr minimal ausgelegt und informierte die Firma nur über Probleme in einem einzigen Cluster. Mit diesen Bedingungen ist es für Ambient schwierig, die Gesundheit der Cluster und Dienste zu überwachen und bei Ausfällen oder Störungen schnell reagieren zu können oder gar zu verhindern. Zudem die von den Entwicklern definierten Ressource Requests und Limits der einzelnen Kubernetes Deployments/Pods waren willkürlich gesetzt und folgten standardisierten oder geschätzten Prognosen, anstatt dienstspezifischen, evaluierten und gemessenen Produktivwerten. Die Unzuverlässigkeit der gesetzten Werte resultiert in einem weiterem Problem, denn die Hardwarespezifikationen der einzelnen Nodes basieren auf diesen. Deswegen sind diese möglicherweise nicht wirtschaftlich gewählt und optimiert. Dies kann wiederum dazu führen, dass die Firma für Ressourcen bezahlt, welche von den einzelnen Diensten gar nicht in Anspruch genommen werden oder genommen werden können.

1.2 Ziel

Das Ziel dieses Projektes war es, dem Unternehmen Ambient dabei zu helfen, mehr Transparenz bezüglich der Gesundheit der eigenen Kubernetes Architektur zu ermöglichen.

Die Reaktionszeiten bei Ausfällen oder Störungen sollen möglichst minimalisiert werden und ein Ablaufprozess für Problemfälle soll eingeführt werden. Auch eine wirtschaftliche Optimierung der Kubernetes Infrastruktur in Bezug auf Ressourcenverbrauch und Hardwarezusammensetzung der einzelnen Nodes soll im Rahmen dieses Projektes erreicht werden.

1.3 Durchführung

Das Thema des Projektes wurde in einzelne Unteraufgaben aufgeteilt, um so eine leichtere bzw. modulare Abarbeitung zu ermöglichen sowie eine Grundstruktur für die Problematik zu schaffen. Über die Ausarbeitung dieser einzelnen Aufgaben wird in den folgenden Kapiteln berichtet. Die Reihenfolge der einzelnen Aufgaben entspricht nicht derer, in welcher sie ursprünglich abgearbeitet worden sind, sondern wurde angepasst, um ein möglichst gutes Leseverständnis zu schaffen.

2 Implementierung Monitoring

Für das Monitoring eines Kubernetes Clusters existieren mehrere Technologien: einige sind bereits Teil von Kubernetes selber (Kubelet, cAdvisor, Kube-state-metrics, Kubernetes Dashboard), andere müssen explizit im Cluster implementiert werden (Prometheus, Jaeger)[1]. Zum Start des Praxisprojektes hatte sich das Unternehmen bereits dafür entschieden eine Rancher 2 Integration für Prometheus zu nutzen. Prometheus selbst wurde schon 2012 veröffentlicht[2], die zu implementierende Integration wurde jedoch erst mit der Version 2.2 von Rancher am 26. März 2019 veröffentlicht[3]. Dieser Zeitpunkt legte somit auch den Start des praktischen Teiles dieses Projektes fest, mit welchem dann in der Folgewoche begonnen wurde. Das Unternehmen entschied sich für diese Integration, da der Implementierungs- und Verwaltungsaufwand für das Monitoring möglichst klein gehalten werden sollte. Desweiteren stellt die Integration noch zusätzlich die durch Prometheus gesammelten Daten in der Rancher Verwaltungsoberfläche dar[4], lässt sie für andere Funktionalitäten von Rancher wiederverwenden (zum Beispiel (z.B.) Metric basierende Alerts)[5] und stellt vordefinierte Grafana Dashboards zur Verfügung. Dies waren weitere Gründe, die zu dieser Entscheidung geführt haben.

2.1 Rancher Upgrade auf 2.2

Ambient nutzt eine Amazon Web Services (AWS) Elastic Compute Cloud (EC2) t2.xlarge Instanz, um Rancher in einem Docker Container zu hosten. Zum Start des Projektes befand sich dieser jedoch noch auf Version 2.1.5 und beinhaltete deswegen noch nicht die Prometheus und Grafana Integration. Deswegen musste zuerst ein Upgrade durchgeführt werden. Neben dem Rancher Container wird zudem ein zweiter Container mit dem Dockerhub Image ambient-innovation/rancher2-backup[6] auf dieser Maschine betrieben. Dieser sorgt dafür, dass mit Hilfe von crontab und crond in periodischen Abständen Backups der Rancher Instanz erstellt und in einem AWS Simple Storage Service (S3) Bucket abgespeichert werden.

Der Prozess für das Update wurde sehr ähnlich wie in der entsprechenden Dokumentation von Rancher[7] ausgeführt, allerdings mit ein paar zusätzlichen Schritten, aufgrund des zusätzlichen Backup Containers. Der komplette Ablauf bestand aus folgenden Schritten:

[1] vgl. (TheKubernetesAuthors, 2020d)
[2] vgl. (PrometheusAuthors)
[3] vgl. (rancherio-gh m, 2019)
[4] vgl. (Rancher, j)
[5] vgl. (Rancher, a)
[6] vgl. (ambientinnovation, 2020b)
[7] vgl. (Rancher, i)

1. Aufbauen einer Secure Shell (SSH) Verbindung zur EC2 Maschine
2. Stoppen des Rancher2 Containers
3. Stoppen des Backup Containers von Ambient (zusätzlich)
4. Erstellen eines Data Containers des gestoppten Rancher Containers
5. Erstellen eines Backup Tarballs aus dem erstellten Data Container
6. Pullen des Images „rancher/rancher:latest“
7. Entfernen des alten Rancher2 Containers (zusätzlich)
8. Entfernen des alten Ambient Innovation Backup Containers (zusätzlich)
9. Starten eines Rancher2 Containers
10. Starten eines Ambient Innovation Backup Containers (zusätzlich)

Direkt nach dem Update war die Verwaltungsoberfläche von Rancher über den Browser nicht mehr erreichbar. Den Logs konnte man entnehmen, dass es ein Problem mit dem SSL Zertifikat gab. Durch inspizieren des Containers mit Hilfe von „docker exec“ konnte festgestellt werden, dass das SSL-Zertifikat sich nicht im Container befand. Denn es stellte sich heraus, dass das SSL-Zertifikat im Filesystem des Docker Hosts durch ein leeres Verzeichnis ersetzt worden ist und deswegen nach dem Mounten des Volumes dieses auch im Container fehlte. Durch Löschen des Verzeichnisses, erneutem Hinterlegen des Zertifikates im Pfad des Volumemounts und Neustarten des Containers konnte das Problem gelöst werden.

2.2 Aktivierung des Monitorings

Die Integration für das Monitoring sollte zuerst auf drei Kubernetes Clustern aktiviert werden. Diese waren Prod (Produktivumgebungen von Projekten), Test (Testumgebungen von Projekten) und Shared (firmeninterne Anwendungen). Für einzelne Kunden dedizierte Cluster wurden hierbei vorerst ignoriert. Die Aktivierung erfolgte basierend auf der Dokumentation von Rancher[8]. Nach einigen Minuten wurde durch Helm Charts die Rancherversion von Prometheus und Grafana deployed. Im Anschluss waren erste Metriken im Rancher Dashboard zu sehen und die Grafana Oberfläche konnte von dort aus aufgerufen werden.

Nach genauerer Inspektion wurde jedoch festgestellt, dass in allen drei Fällen jeweils nur die Metriken von genau einer Maschine verfügbar waren. Bei dieser Maschine handelte es sich um diejenige, auf welcher der Pod „prometheus-cluster-monitoring“ gehostet war. Durch Analyse der Logs dieses Pods konnte festgestellt werden, dass der entsprechende Container des Pods versuchte Verbindungen zu anderen Diensten aufzubauen, diese jedoch meistens fehlschlugen. Im nächsten Schritt wurde versucht ein

[8] vgl. (Rancher, d)

besseres Verständnis aufzubauen, wie der Datenaustausch der Metriken zwischen den einzelnen Maschinen in Prometheus erfolgt. Es stellte sich heraus, dass Rancher mit der Aktivierung der Integration auch ein Deamon Set „exporter-node-cluster-monitoring“ deployed hat und somit auch einen entsprechenden Pod auf allen Maschinen des Clusters. In der zugehörigen YAML Ain't Markup Language (YAML) Datei des Deamon Sets war konfiguriert, dass ein Container „exporter-node“ einen Hostport (9796) über eine Transmission Control Protocol (TCP) Verbindung konfiguriert hatte. Mit der Annahme, dass über diese Verbindung die Metriken zur Verfügung gestellt werden, wurde eine Anpassung der AWS EC2 Security Groups vorgenommen. Nach der Anpassung durften die einzelnen EC2 Instanzen eines Clusters über eine TCP Verbindung (Port 9796) kommunizieren. Für andere Quellen blieb dieser Port weiterhin blockiert. Mit dieser Änderung wurde das Problem behoben und die Metriken aller Maschinen waren im Rancher Dashboard und in Grafana verfügbar. Die Notwendigkeit diesen Port freizugeben, konnte zum Zeitpunkt der Einrichtung allerdings in keiner Dokumentation von Rancher gefunden werden. Inzwischen wird dies jedoch als Hinweis in der Anleitung zum Aktivieren des Monitorings[9] erwähnt.

Ein weiteres Problem ergab sich einige Stunden nach der Installation des Monitorings, als erste Entwickler darüber berichteten, sie können Dienste nicht erreichen, ihre Pipelines würden fehlschlagen und man könnte generell nicht deployen. Betroffen waren hiervon das Shared- und Test-Cluster. Auch der Versuch die Cluster direkt mit Hilfe der kubeconfig Datei, anstatt über die Rancheroberfläche zu erreichen, endete oft in einem Timeout oder einer sehr langen Wartezeit pro genutzten Befehl. Die CPU Metriken der EC2-Instanzen indizierten keine Probleme und die benötigten Ressourcen für das Monitoring waren wie in der Dokumentation[10] angeben, verfügbar. Trotzdem ist die Ursache auf eine Überlastung der Instanz zurückzuführen. Die genaue Ursache und Problematik zu verstehen und eine Lösung hierfür zu finden dauerte mehre Tage und wird genauer im Kapitel Optimierung bestehender Cluster Architektur auf S.11 behandelt.

Durch Zufall wurde ausserdem entdeckt, dass über die Rancher Application Programming Interface (API) herausgefunden werden konnte, welche Node als Kube-Apiserver Endpoint eines Clusters von Rancher genutzt wird und dass dieser automatisch gewechselt wird, sobald die Node offline ist. Um wieder Kommunikation mit dem Cluster zu ermöglichen, wurde entsprechende EC2-Instanz über die AWS Konsole neu gestartet, Rancher wechselte zu einem Kube-Apiserver einer anderen Node und für einen kurzen Moment konnte wieder mit dem Cluster kommuniziert werden, bis sich der Zustand wieder verschlechterte. In einem späteren erneuten Versuch wurde diese Zeitspanne ausgenutzt, um das Monitoring zu deaktivieren und den zugehörigen Namespace „cattle-prometheus“ zu löschen. Auch in anderen Fällen, bei welchen das Monitoring aktiviert wurde und es zur selben Situation führte, wurde dieser Umweg angewandt. Das Monitoring blieb daraufhin solange deaktiviert, bis die Änderungen aus dem Kapitel Optimierung bestehender Cluster Architektur auf S.11 durchgeführt waren.

[9] vgl. (Rancher, d)
[10] vgl. (Rancher, h)

In einigen Fällen konnte das Deaktivieren jedoch nicht vollständig vollzogen werden, so blieb das Löschen des Namespaces stecken im Status „Terminating“. Mit Hilfe eines Issues auf GitHub[11] wurde eine Lösung gefunden, die nicht darin bestand, kubectl zum Löschen zu nutzen. Stattdessen wurde direkt mit dem Representational State Transfer (REST) Kube-Apiserver kommuniziert. Um diesen für uns erreichbar zu machen, wurde wie in der Dokumenation von Kubernetes[12] beschrieben, kubectl genutzt, um einen Reverse Proxy zu dieser zu starten. Im Anschluss wurde ein curl Befehl abgesendet, um den Namespace zu löschen. Die Ausführung sah zusammengefasst wie folgt aus:

```
$ kubectl proxy
Starting to serve on 127.0.0.1:8001

$ kubectl get namespace cattle-prometheus -o json >  cattle-prometheus.json

$ sed -i "/"kubernetes"/d" cattle-prometheus.json

$ curl -H "Content-Type: application/json" -X PUT –data-binary
@cattle-prometheus.json 127.0.0.1:8001/k8s/clusters/cluster-id/api/v1/
namespaces/cattle-prometheus/finalize
```

In anderen Fällen wurden nach dem Deaktivieren und erneutem Aktivieren des Monitoring nicht immer alle benötigten Pods automatisch vom Rancher deployed. Dieser Bug wurde mittlerweile in Version 2.2.2 behoben. Bis dahin wurde eine manuellen Lösung angewandt, welche in einem Github Issue[13] gefunden wurde. Bei dieser war es notwendig zuerst sämtliche Spuren des Monitorings zu entfernen, dann eine SSH Verbindung zum Rancher Server aufzubauen und dort das betroffene Cluster zu bearbeiten und die Information zu entfernen, dass das Monitoring jemals aktiviert worden ist.

2.3 Personalisierung der Grafana Dashboards

Ursprünglich war geplant, die durch das Medieninformatikstudium gewonnene Expertise in User Interface (UI) & User Experience (UX) zu nutzen, um eigene Grafana Dashboards basierend auf den Bedürfnissen der Firma zu entwerfen. Im Nachhinein stellte sich jedoch heraus, dass die angepasste Version von Grafana für Rancher diese Funktionalität nicht wie üblich besitzt. Darüber, ob diese zu einem späterem Zeitpunkt folgen wird beziehungsweise geplant ist, konnten keine Informationen gefunden werden.

[11] vgl. (slassh, 2018)
[12] vgl. (TheKubernetesAuthors, 2020e)
[13] vgl. (Logan, 2019)

3 Optimierung bestehender Cluster Architektur

Wie bereits in einem früheren Kapitel behandelt wurde, hat die Aktivierung des Monitorings nach einer gewissen Zeit bei zwei Clustern zu starken Performance Problemen geführt und musste deaktiviert werden. Um das Problem besser verstehen zu können, wird zunächst am Beispiel des Test-Clusters die ursprüngliche Architektur und Konfiguration aller mit Rancher Kubernetes Engine (RKE) betriebener Cluster von Ambient genauer beschrieben. Für RKE Cluster, welche Nodes durch einen Infrastructure Provider starten, werden diese durch Node Pools[14] definiert. Mit diesen ist es möglich die Anzahl der gewünschten Nodes pro Pool und die Node Roles (/etc distributed (ETCD), Control Plane, Worker) zuzuweisen. Zu dem Zeitpunkt der Problemanalyse bestand das erwähnte Cluster aus 2 Node Pools. Einem mit 3 Nodes und allen Rollen und einem mit nur der Rolle Worker und einer Node.

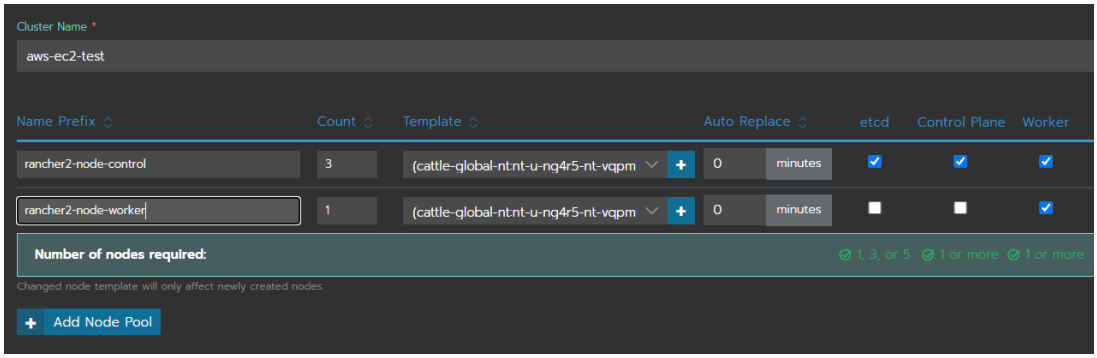


Abbildung 3.1: Test-Cluster Node Pools
Quelle: Eigene Darstellung, Bildschirmkopie

Auch das Shared- und Prod-Cluster hatten einen ähnlichen Aufbau. Es gab jeweils einen Node Pool mit der festen Anzahl 3, welcher alle Rollen zugewiesen hatte und einen zweiten Node Pool, welcher nur als Worker diente und nach Bedarf flexibel skaliert werden konnte. Die Idee der Firma war es, auf genau 3 Instanzen einen ETCD zu betreiben, um somit das Cluster trotzdem betriebsfähig zu machen, sollte einer davon ausfallen. Weil die Nodes freie Rechenkapazitäten haben, sollten diese auch noch zusätzlich als Worker genutzt werden.

Das Mischen der Rolle Control Plane/ETCD mit Worker, ohne sicherzustellen, dass systemrelevante Prozesse nicht eingeschränkt werden, wird als einer der Gründe für die

[14] vgl. (Rancher, e)

Performance Probleme nach dem Aktivieren des Monitoring vermutet. In der besagten Konfiguration konnten bis zu 100% der verfügbaren CPU Leistung der EC2-Instanz durch Kubernetes Pods in Anspruch genommen werden. Sollte dieser Fall eintreten, würde nicht nur die Geschwindigkeit der Pods eingeschränkt werden, sondern auch die von anderen Prozessen, welche sich auf diesem Host befinden. Diese wären zum Beispiel Kubelet, Kube-Apiserver, Docker Container-Runtime oder der Operating System (OS) Kernel. Es wird angenommen, dass dieses Szenario durch Nutzen des Prometheus Monitoring eingetreten ist, welches zusätzliche Auslastung in Form von Pods (zum Beispiel Prometheus Server oder Node Exporter) auf der Instanz verursachte.

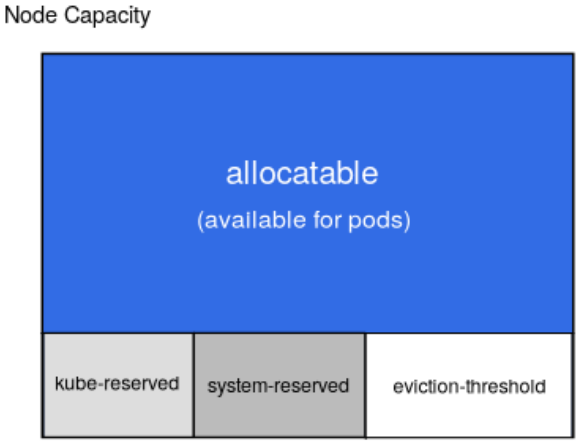


Abbildung 3.2: K8s Node Allocatable
Quelle: (TheKubernetesAuthors, 2020c)

Dieses Problem kann laut einem Eintrag der Kubernetes Dokumentation[15] durch das Feature Node Allocatable verhindert werden. Dieses ermöglicht es durch definieren eines kube-reserved und system-reserved Wertes (siehe Abbildung 3.2 auf S.12) Prozesse welche nicht als Kubernetes Pods gestartet wurden vor einem Mangel an Rechenleistung durch Auslastung von Pods zu beschützen. Rancher startet Prozesse wie den Kubelet von alleine, jedoch wäre es möglich gewesen diese Argumente über eine Anpassung der RKE Konfiguration[16] über die Rancher UI als YAML durchzuführen. Diese Anpassung wurde jedoch nicht vollzogen, weil diese Option erst zu einem viel späteren Zeitpunkt entdeckt wurde.

Stattdessen wurden dem zweiten Node Pool drei weitere Nodes hinzugefügt, welche nur die Rolle Worker hatten. Dies ermöglichte es, den ersten Node Pool zu bearbeiten und diesem die Rolle Worker zu entziehen, damit diese Node ausschließlich als Control Plane und ETCD dienten und nicht von Auslastungen von Pods gefährdet werden konnten. Diese Entscheidung wurde getroffen, weil in der Dokumentation von Kuber-

[15] vgl. (TheKubernetesAuthors, 2020c)
[16] vgl. (Rancher, b)

netes[17] in beiden Optionen für High Availability (HA) Cluster dedizierte Worker Nodes dargestellt werden und nicht mit Control Planes oder ETCDs gemischt werden. Die erste Option des Eintrags mit "Stacked ETCD" (siehe Abbildung 3.3 auf S.13) wurde als Richtwert für die Anpassungen an den Clustern gewählt. Es wurde sich gegen die

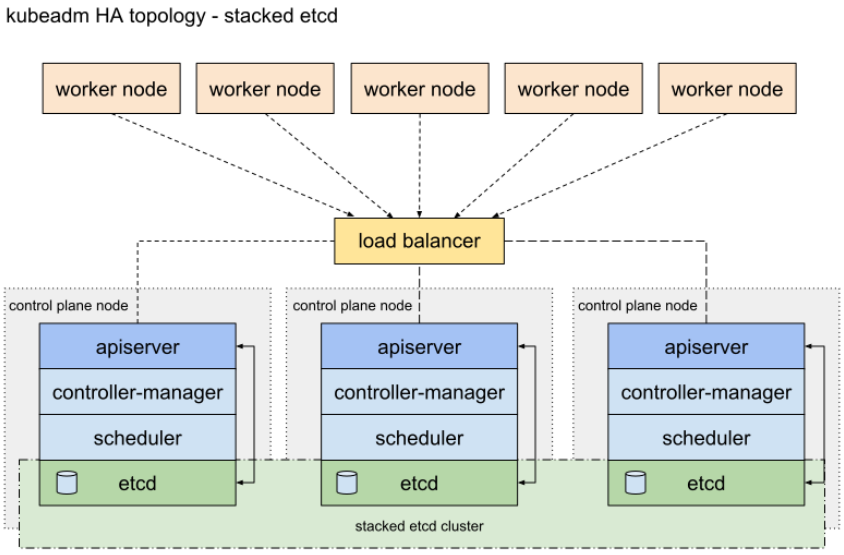


Abbildung 3.3: kubeadm HA topology - stacked ETCD
Quelle: (TheKubernetesAuthors, 2020b)

Option mit dem "External ETCD" entschieden, weil diese 3 weitere Nodes benötigt hätte, auf welche aus Kostengründen verzichtet werden sollte. Die umgesetzte Lösung (siehe Abbildung 3.4 auf S.14) unterschied sich zum Beispiel aus der Dokumentation in dem Punkt, dass sie keinen Load Balancer für die kube-Apiserver beinhaltete. Stattdessen wird immer der kube-Apiserver angesprochen, welcher von Rancher als apiEnpoint hinterlegt ist. Durch Rancher wurde die Architektur erweitert, sodass eine direkte Kommunikation von außen mit dem kube-Apiserver nicht möglich war. Zwischen einem Nutzer und einem kube-Apiserver befanden sich 3 weitere Komponenten (Authentication Proxy, Cluster Controller und Cluster Agent). Dadurch war es nicht wie üblich möglich, einen Load Balancer zu betrieben, der den Traffic zwischen den verfügbaren kube-Apiservern verteilt und welcher direkt angesprochen werden kann. Zu einem späterem Zeitpunkt veröffentlichte Rancher eine Anleitung[18] wie dies trotz ihrer Architektur möglich ist.

Nach dem Abschluss des praktischen Teiles dieses Projektes wurde eine andere Option entdeckt, welches Teil des ursprüngliche Problems gewesen sein kann. Die Nodes, die ursprünglich Worker, ETCD und Controlplanes zugleich waren, hatten den Instance Type t2.large. Laut der AWS Dokumentation[19] sammeln alle Instanzen des Types t CPU Credits und haben eine Baseline Utilization. Ist die CPU Auslastung kleiner als

[17] vgl. (TheKubernetesAuthors, 2020b)
[18] vgl. (Kim, 2019)
[19] vgl. (AWS, a)

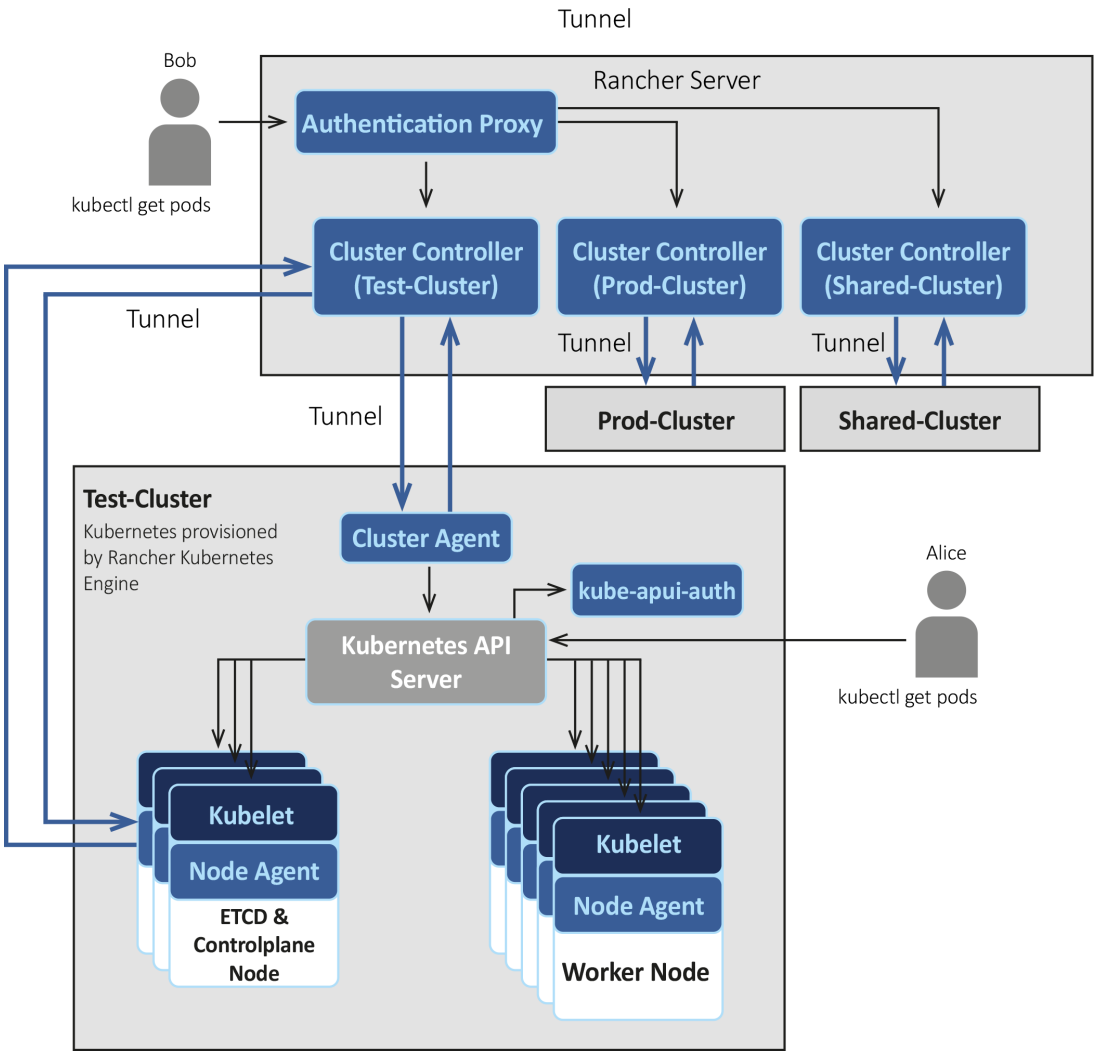


Abbildung 3.4: Umgesetzte Noderole Topologie
Quelle: Eigene Grafik

der definierte Baseline Utilization Wert, dann werden Credits gesammelt, ist sie größer, dann werden Credits ausgegeben. Besitz die entsprechende EC2-Instanz keine Credits mehr, so wird ihre CPU Leistung, falls nicht anders konfiguriert, auf den Baseline Wert gedrosselt, um zusätzliche Kosten zu vermeiden. Also hätte es in diesem konkreten Fall sein können, dass die Nodes auf 30% (t2.large Baseline Utilization) ihrer maximalen Leistung gedrosselt wurde, was zur Folge hatte, dass nicht mehr genug Rechenleistung zur Verfügung stand, um die Dienste der Nodes, unter anderem den kube-Apiserver, mit gewünschter Geschwindigkeit zu betreiben. Da dieser Anhaltspunkt niemals untersucht wurde, weil er zu einem viel späterem Zeitpunkt entdeckt wurde, handelt es sich hierbei lediglich um eine Spekulation einer möglichen Ursache.

4 Optimierung der K8s Resource Requests und Limits

Durch die Optimierung bestehender Cluster Architektur auf S.11 konnte das Monitoring aktiviert werden und befand sich in einem stabilem Zustand ohne negative Auswirkungen auf das Cluster. Im nächsten Schritt wurde einige Tage gewartet, damit sich genügend Daten ansammelten. Diese sollten verwendet werden, um auszuwerten, ob die von Entwicklern gesetzten Requests und Limits sinnvoll waren oder ob Optimierungspotential besteht.

Im ersten Schritt wurden die Ressource Werte, welche in den YAML Dateien der Deployments definiert wurden, in einer Excel Tabelle pro Cluster (siehe Tabelle 4.1, 4.2 & 4.3 auf S.42-44) zusammengefasst. Dies sollte einer besseren Übersicht für das folgende Vorgehen dienen. Hierbei stellte sich heraus, dass verschiedene Deployments unterschiedlich mit der Vergabe dieser Werte umgingen. Manche Deployments hatten gar keine Werte, manche nur teilweise und manche unterschieden nicht zwischen Request und Limit. Hieraus resultiert, dass die Deployments in unterschiedlichen Quality of Service[20] betrieben wurden. Mit dieser Erkenntnis wurde Rücksprache mit dem Mentor für dieses Projekt, als auch mit einzelnen Entwicklern gehalten. Es stellte sich heraus, dass man sich nicht bewusst war, dass Quality of Service existiert und wie dieser überhaupt zustande kommt. Darauf folgend wurde zusammen mit dem Betreuer entschieden, dass es das Ziel ist, keine Pods mehr im BestEffort Zustand laufen zu lassen und dass alle Deployments im Burstable Zustand laufen sollten. Auf den Guaranteed Quality of Service sollte möglichst verzichtet werden, um so nicht zu viele Ressourcen zu reservieren, damit diese anderen Deployments je nach Situation zur Verfügung stehen. Der erste Wunsch war es, alle Deployments so zu optimieren, dass der Request leicht größer war als das, was das Deployment mindestens brauchte, um im Normalzustand zu laufen. Das Limit sollte so gewählt werden, dass zu Zeiten mit durchschnittlich hoher Auslastung die Deployments Freiraum nach oben haben, um mehr Ressourcen für sich in Anspruch zu nehmen. Allerdings trotzdem stark genug eingeschränkt werden, um die Gesundheit der Node nicht zu gefährden.

Für diese Optimierung wurden die bereits erstellten Excel Tabellen mit den bisherigen CPU/Memory Requests & Limits erweitert. Bei den zusätzlichen Werten handelte es sich um mit Grafana manuell ausgelesene Daten, durchschnittliche Verbrauchswerte einzelner Deployments (CPU/Memory). Mit Hilfe dieser und der Untersuchung, wie hoch einzelne Deployments zu Höchstzeiten verbrauchen können, wurden die Tabellen erneut erweitert, um grob geschätzte neue mögliche Requests und Limits. Dieser Vorgang dauerte einige Tage und wurde zuerst für das Shared-, dann das Test- und an-

[20] vgl. (TheKubernetesAuthors, 2020a)

schließend für das Prod-Cluster durchgeführt. Im späteren Verlauf des Projektes wurde noch eine Migration von dem Projekt Entsorung Punkt DE (EPD) aus dem Zentek-Cluster in das Test- und Prod-Cluster durchgeführt. Nach der Migration wurden auch diese Deployments optimiert und der Tabelle hinzugefügt. Abschließend wurden alle Requests, jeweils vor und nach der Optimierung, pro Cluster summiert und separat dargestellt (siehe Tabelle 4.1 auf S.16). Zusätzlich wurde ein Feld angelegt, durch welches eingesehen werden kann, wie viel % der vorherigen Reservierungen durch die Anpassungen frei geworden sind.

Tabelle 4.1: Zusammenfassung Ressourcenoptimierung

Cluster	Request CPU (alt)	Request CPU (neu)	Gesparter CPU Request	Request Memory (alt)	Request Memory (neu)	Gesparter Memory Request
Shared	4.050	1.055	74,0 %	13.381	10.360	22,6 %
Test	7.825	292	96,3 %	18.178	6.700	63,1 %
Prod	4.400	255	94,2 %	7.640	6.430	15,8 %

CPU und Memory Werte sind jeweils in milli Central Processing Unit (mCPU) und Mebibyte (MiB) angegeben.

Als Resultat dieser Optimierung konnten im Anschluss einige Maschinen abgestellt werden, jedoch nicht im selben Prozentsatz. Grund dafür war, dass man immer noch auf plötzliche höhere Auslastungen vorbereitet sein wollte und abseits der K8s Deployments auch noch andere Prozesse Ressourcen der Maschinen in Anspruch nehmen, die nicht eingerechnet sind.

5 Implementierung Alerting

Vor der 2.2 Version hatte Rancher eine minimalistische Alert Funktionalität integriert, Diese konnte jedoch nur die Gesundheit der drei System Services ETCD, Scheduler und Controller Manager überwachen. Ambient hatte diese Alerts jedoch nur in einem Cluster eingerichtet und diese waren mit keinem Notifier verbunden, deswegen wurde nicht über aktive Alerts benachrichtigt. Durch das Rancher Upgrade auf 2.2 auf S.7 wurde die Funktionalität erweitert. Eigene Alerts konnten z.B. auf Projektbasis mit Hilfe von Metriken, also den Daten die das Monitoring sammelt, definiert werden [21].

In Absprache mit der Firma wurde entschieden, dass Alerts, die nur einzelne Entwickler Teams betreffen, sie in deren Slack oder Mattermost Channel benachrichtigen sollen. Für Alerts, welche das ganze Cluster, also einzelne Maschinen oder Teile des K8s Ecosystems betreffen, sollte es einen eigenen Kommunikationskannal geben. Hierfür wurde ein eigener Mattermost Channel 'Hosting Alerts' erstellt, in welchen nur die Mitarbeiter eingeladen wurden, welche auch im Fall eines Problemes agieren würden. Für das Einrichten von Notifiern stellt Rancher mehrere Integrationen[22] zur Verfügung. Geplant war für Slack, die Slack Integration und für Mattermost die Webhook Integration zu nutzen. Im Falle von Slack konnte diese ohne Probleme eingerichtet werden. Es wurde in Slack eine Webhook Uniform Resource Locator (URL) erstellt, welche man in der Integration von Rancher hinterlegen konnte und auch den Channel definieren konnte. Bei Mattermost ergab sich beim Ausprobieren der Webhook Integration von Rancher jedoch ein Problem. Der Webhook wurde erstellt und auch in Rancher konfiguriert, jedoch kam niemals ein Alert in Mattermost an. Wie sich herausstellte, erwartet Mattermost einen ganz speziellen Payload, weswegen die normale Webhook Integration nicht funktionierte. Weitere Recherche ergab, dass der erwartete Payload von Mattermost die identische Struktur hatte, wie der von Slack. Mit dieser Erkenntnis wurde ausprobiert, die Slack Integration zu nutzen, jedoch einen Webhook URL von Mattermost anzugeben und es funktionierte.

Mit funktionsfähigen Notifiern galt es als nächstes die Alerts zu definieren. Mit dem Upgrade auf Rancher 2.2 wurden auch die Liste an bereits vordefinierten Alerts (basierend auf Metriken) erweitert. In Absprache mit dem Mentor wurde entschieden, dass diese Liste ausreichend ist und das wichtigste bereits enthält. Es galt jedoch diese noch in Bezug auf Empfindlichkeit mit angepassten Werten zu individualisieren und den entsprechenden Notifiern zuzuweisen. Neben den drei Alerts für die Gesundheit der System Services ETCD, Scheduler und Controller Manager beinhaltete jedes Cluster jetzt auch folgende Alerts:

[21] vgl. (Rancher, g)
[22] vgl. (Rancher, f)

- ETCD Member haben keinen Leader
- ETCD Leader wurde in 3 Minuten mindestens 3 mal neu gewählt
- ETCD Datenbank nähert sich einem Verbrauch von 1500 MB
- Einzelne Node erreichte eine CPU Auslastung von 100%
- Einzelne Node hat weniger als 20% Memory verfügbar
- Einzelne Node wird in weniger als 24h keinen Speicherplatz mehr haben

Der Großteil der Deployments in den Clustern ist mit K8s-Healthchecks ausgestattet, damit sobald es ein Problem mit einem Pod gibt, dieser möglichst schnell von K8s ausgetauscht wird. Allerdings kann es vorkommen, dass in manchen Fällen auch ein Neustarten eines Pods nicht ein Problem löst. Für diesen Fall soll das entsprechende Team möglichst schnell informiert werden, damit dieses entsprechend handeln kann. Um dies zu erreichen, wurden zusätzlich Alerts auf Projektbasis eingerichtet. Sollte ein Deployment unter 50% der gewünschten Replicas pro Deployment fallen, schlägt ein Alert an, welcher den zugehörigen Slack beziehungsweise Mattermost Channel des jeweiligen Teams benachrichtigt.

5.1 Probleme mit einzelnen Alerts

Nach längerer Beobachtung stellte sich heraus, dass der Alert welcher benachrichtigen soll, wenn der System Service ETCD nicht mehr funktionsfähig ist, täglich mitten in der Nacht ausgelöst worden ist. Hierfür konnte lange Zeit weder eine Lösung, noch eine sinnvolle Erklärung gefunden werden. Durch Recherche konnte aber ein ähnlicher Fall gefunden werden, welcher in einem GitHub Issue[23] dokumentiert wurde. In dem Ticket wurde das Problem zurück geleitet auf das automatisierte Backupen der ETCD Datenbank in einen S3 Bucket. Während des Backups kann es zu Timeouts von Requests an den ETCD kommen, was wiederum den Alert auslösen kann. Auch im Falle von Ambient war es sehr wahrscheinlich, dass dies die Ursache sein könnte, denn genau dieses Feature wurde auch im Rahmen dieses Projektes in ETCD Backups zu S3 auf S.20 konfiguriert und aktiviert. Nach manueller Anpassung des Thresholds des Alerts über die API, wie als Lösungsvorschlag in dem Issue beschrieben wurde, konnte dieses Problem letztendlich gelöst werden.

Ein weiterer Alert, welcher öfters "false positives"meldete, war der Alert, welcher zuständig war zu berichten, wenn weniger als 50% der Replicas eines Deployment sich in gesundem Status befinden. Hierfür konnte zuerst keine Ursache ermittelt werden, jedoch fiel mit der Zeit auf, dass diese Alerts anschlugen, sobald eine Gitlab Continuous Integration (CI)/Continuous Delivery (CD) Pipeline des zugehörigen Projektes durchlief und ein Deployment aktualisierte (neues Image). Wie sich herausstellte, hing das Auslösen des Alerts damit zusammen, wie Deployments bei Ambient aktualisiert

[23] vgl. (Mattox, 2019)

werden. Für diesen Prozess wurde eine firmeninterne eigene Lösung programmiert[24], welche zu einem Docker Image gebaut wird und auf Docker Hub[25] gehostet ist. Dieses Image wird meistens für den letzten Job in den GitLab Pipelines genutzt, um ein Deployment zu aktualisieren. Das Prinzip des Image ist, mit „kubectl patch“ eine Environment Variable (FORCE_RESTART_AT) eines Deployments mit dem aktuellen Timestamp zu aktualisieren, damit K8s in einem Rolling Upgrade Prozess alle Pods des Deployments austauscht und das Image neu herunterladet, sollte es eine neuere Version geben. Das Problem welches hierbei entsteht ist, dass K8s für diesen Vorgang ein neues ReplicaSet erstellt und dieses langsam hoch skaliert, während das alte ReplicaSet runter skaliert wird. Der Alert wiederum bezieht sich auf jedes ReplicaSet individuell. Sobald also das neue ReplicaSet einen desired Zustand von 1 hat, wird der Alert ausgelöst, da der erste Pod nicht sofort verfügbar ist. Zusammengefasst heißt das, dass es für paar Sekunden den Zustand 0/1 gibt bei dem neuem ReplicaSet und dieser Zeitraum ausreicht, um den Alert auszulösen. Nachdem keine Lösung für diese Problem gefunden werden konnte und es nicht gewünscht war, den Deployment Prozess für alle Projekte zu überarbeiten, weil der Alert suboptimal war, wurde dieser Alert vorübergehend für alle Projekte wieder deaktiviert.

5.2 Externes Monitoring für Deployments

Für Ambient ist es allerdings wichtig, informiert zu werden, sollte ein Deployment Probleme haben und nicht mehr funktionsfähig sein. Es galt eine Alternative für den deaktivierten Alert zu finden. Hierbei wurde sich von Seiten der Geschäftsführung für den externen Dienstleister uptime.com entschieden. Dieser überprüft in definierten Zeitabständen von unterschiedlichen Standorten aus, ob ein Dienst unter einer definierten URL erreichbar ist. Es wurden im nächsten Schritt alle zu überwachenden Dienste dort konfiguriert und entsprechende Mattermost, Slack und Short Message Service (SMS) Notifier wurden den Alerts zugewiesen. Ein Problem ergab sich bei der Einrichtung des Mattermost Notifiers, da uptime.com lediglich SMS und Slack unterstützt. Weil Mattermost und Slack für ihre Integrationen den selben Payload erwarten, wurde auf der Plattform uptime.com der Mattermost von Ambient als Slack hinterlegt. Es war nicht möglich dies über die Formulare der Webseite selbst zu erreichen, da die diese eine frontendseitige Validierung hatte. Um dies zu umgehen, wurde der Post Request beim Anlegen einer Slack Integration betrachtet. Danach wurde die URL im Payload des Requests durch die gewünschte Mattermost URL ersetzt und mit Hilfe des curl command abgesendet. Dies war nur möglich, weil es keine backendseitige Validierung der URL von uptime.com gab. Nach kurzem Testen stellte sich heraus, dass der Mattermost Notifier, auch wenn er vom Typ Slack angelegt war, fehlerfrei funktionierte. Nach einigen Wochen meldete sich der Support von uptime.com, dass die konfigurierte Integration entfernt wurde, weil die URL ungültig sei und fragte wie das Hinzufügen durchgeführt wurde. In einer Antwort wurde der Prozess beschrieben mit der Bitte, dies zu erlauben, weil es trotzdem funktioniert. Diese Erlaubnis wurde erteilt und der Mattermost der Firma wurde erneut als Slack Notifier hinzugefügt.

[24] vgl. (ambientinnovation, 2020a)
[25] vgl. (ambientinnovation, 2020b)

6 ETCD Backups zu S3

Mit der Version 2.2 hat Rancher ein neues Feature eingeführt, mit welchem es möglich ist, ETCD Snapshots zu erstellen und diese als Grundlage für ein Backup eines Clusters zu nutzen. Bis zu diesem Zeitpunkt nutze Ambient eine eigene Lösung[26]. Diese funktioniert so, dass in regelmäßigen Abständen ein Cron Job ausgeführt wird, welcher durch alle K8s Objekte des Clusters iteriert und für jedes eine YAML Datei generiert und in einen S3 Bucket abspeichert. Sollte es zu Problemen mit dem Cluster kommen, könnten diese genutzt werden, um die K8s Objekte neu zu erstellen oder es könnte ein neues Cluster kreiert werden, auf welchem dann alle Objekte neu erstellt werden. Dieser Lösungsansatz ist zwar funktionsfähig, allerdings nicht automatisiert für den Fall eines Ausfalles. Im Notfall würde es zum einem viel manuellen Aufwand erfordern und zum anderem würde es oft darauf hinaus laufen, dass das Cluster neu aufgebaut werden müsste. Im Rahmen dieses Projektes galt es, das neue Rancher Feature, welches durch Version 2.2 eingeführt wurde, am Beispiel eines neu erstellen Sandboxclusters zu testen. Sollte sich die Lösung als zuverlässig erweisen, sollte diese für die drei EC2 Cluster implementiert werden.

Für das Testen wurde mit Rancher ein neues EC2 Cluster und ein S3 Bucket erstellt. Mit Hilfe der Dokumentation[27] von Rancher wurde der S3 Bucket als Snapshot Ziel hinterlegt und wiederkehrende Snapshots wurden konfiguriert. Zum Testen wurde die Periode zum Erstellen der Snapshots auf eine Stunde eingestellt. Insgesamt sollte das Feature mit drei Szenarien getestet werden.

- Im ersten Test galt es festzustellen, ob ein Backup durch Einspielen eines ETCD Snapshots, K8s Objekte zu ihren ursprünglichen Zuständen wiederherstellt. Hierfür wurden einige Deployments manipuliert (z.B. Image Name verändert, Environment Variablen verändert) und funktionsunfähig gemacht. Danach wurde ein Backup mit dem neuen Rancher Feature initialisiert und alles wurde zum gewünschten Ausgangszustand wiederhergestellt.

- Im zweiten Test wurden manuell die EC2 Instanzen heruntergefahren (Simulation eines Ausfalls der Maschinen), welche für das Verwalten des Clusters (ETCD Controll Plane Nodes) zuständig sind. Ein Cluster aus diesem Zustand wiederherzustellen ist ohne ein Backup und ohne Zugriff auf den ETCD nicht mehr möglich. Darauf folgend wurde über die Rancher Oberfläche ein Backup aus dem S3 initialisiert. Dies führte zu einem Misserfolg und das Cluster war immer noch defekt. Nach mehreren Versuchen wurde eine Lösung gefunden, welche in der Lage war, das Cluster wiederherzustellen. Hierzu mussten erstmals alle kaputten

[26] vgl. (ambientinnovation, 2020c)
[27] vgl. (Rancher, c)

ETCD & Controll Plane Nodes in der Rancher Oberfläche entfernt werden, dann musste eine neue Maschine dem Cluster hinzugefügt werden, welche als ETCD Controll Plane dienen sollte. Diese wurde jedoch nicht erfolgreich dem Cluster hinzugefügt, da es keine ETCD & Controll Plane Node gab, bei welcher sie sich anmelden konnte. Deswegen blieb diese im Status „Provisioning“ stecken. Abschließend musste ein Backupprozess aus dem S3 ETCD Snapshot gestartet werden. Dies führte dazu, dass Rancher das Cluster reparierte und wieder funktionsfähig machte. Dieser Prozess konnte keiner Dokumentation entnommen werden und wurde durch Ausprobieren im Sandboxcluster entdeckt.

- Im dritten Test sollte geprüft werden, ob es möglich ist, ein ETCD Snapshot auf ein anderes Cluster anzuwenden. Dieses Szenario wäre hilfreich, um zum Beispiel eine identische Kopie zu erstellen oder als absolute Notfallmaßnahme, um ein Cluster neu aufzubauen, sollte ein bestehendes Cluster sich nicht mehr in einem angemessenen Zeitrahmen reparieren lassen. Hierfür wurde ein zweites Sandboxcluster erstellt und der selbe S3 Bucket für ETCD Snapshots hinterlegt. Beim Versuch ein Backup zu initialisieren, wurden jedoch die Snapshots des alten Clusters nicht in der Oberfläche angezeigt und somit war der Test vorerst nicht möglich. Auch der nächste Lösungsversuch hatte keinen Erfolg. Bei diesem wurde sich über eine SSH Verbindung zu einer ETCD Node verbunden, der Snapshot wurde heruntergeladen und lokal auf die Festplatte in das Verzeichnis gespeichert, welches für die Snapshots vorgesehen ist. Erneut wurde der Snapshot in der Rancher UI nicht angezeigt. Mit diesen zwei Fehlschlägen ist die Annahme entstanden, dass es in Rancher eine Zugehörigkeit von Snapshot und Cluster gibt, welche das Vorhaben nicht ermöglicht. Durch diese Erkenntnis wurde der Test abgebrochen und es wurde ein Feature Request[28] im Gitlab von Rancher erstellt, mit der Bitte, dass ETCD Snapshots clusterübergreifend genutzt werden können.

Der Verzicht auf die bisherige Backup-Lösung von Ambient (Cron Job erstellt YAML Dateien) ist wegen dem gescheiterten dritten Test nicht möglich. Trotzdem wurde das Feature für die drei bestehenden EC2 Cluster als zusätzliche Maßnahme konfiguriert und aktiviert. Wie sich herausstellte, ist es für das S3 Snapshot Feature notwendig, dass der K8s Cloud Provider für AWS aktiviert ist. Dies war nur bei zwei der drei Clustern der Fall. Bei den Clustern wo dieser bereits aktiv ist, konnte das Snapshot Feature ohne Probleme aktiviert werden. Bei dem dritten wurde versucht die Cloud Provider Funktionalität im Nachhinein zu aktivieren. Dies führte dazu, dass das Cluster kaputt ging. Sobald für das Cluster in der Rancher UI der Cloud Provider AWS aktiviert worden ist, fing Rancher an, die Nodes des Clusters auszutauschen. Durch automatisches Entfernen von allen bestehenden ETCD & Controll Plane Nodes, wurde ein Zustand erreicht, in dem keine neuen Nodes mehr angemeldet werden konnten und das Cluster keine funktionsfähigen Nodes mehr hatte. Während das Cluster kaputt war, wurde mit einem Team aus drei Leuten und mit Hilfe der YAML Dateien aus der Backup Lösung von Ambient manuell eine Kopie des Clusters rekonstruiert. Der große Unterschied zum alten Cluster war, dass bei diesem das Cloud Provider Feature von

[28] vgl. (Bajon, 2019)

Anfang an auf AWS gesetzt war. Sobald das neue Cluster voll funktionsfähig war, wurde der AWS Load Balancer auf das neue Cluster umgestellt und im Anschluss wurde auch bei diesem das S3 ETCD Snapshot Feature aktiviert.

7 Migration Zentek-Cluster in Test- & Prod-Cluster

Vor dem Start dieses Projektes hat Ambient für den Kunden Zentek eine Migration ihrer Dienste in die AWS Cloud durchgeführt. Hierfür wurden die einzelnen Dienste des Unternehmens lauffähig in Docker Containern gemacht und es wurde eine CI/CD erstellt, welche die Images automatisch erstellt und in ein eigenes Registry pusht. Danach wurde für diesen Kunden in Rancher ein eigenes K8s Cluster auf EC2 Basis aufgesetzt und sämtliche Dienste (von der Test- und Produktivumgebung) wurden dorthin migriert. Einige Zeit später wurde entschlossen, dass das Cluster wieder aufgelöst werden sollte und die zugehörigen Deployments sollten in das bestehende Test- als auch Prod-Cluster von Ambient migriert werden. Dies galt es im Rahmen dieses Projektes zu erledigen.

Diese Aufgabe wurde in zwei Phasen erledigt, zuerst wurden alle Dienste der Testumgebung umgezogen und danach die der Produktivumgebung. Es wurde errechnet, dass weil die Dienste der Testumgebung eine sehr kleine Auslastung haben, diese ohne Änderung der Anzahl der Nodes des Test-Clusters, deployed werden können. Für beide Phasen war der Ablauf sehr ähnlich und beinhaltete mindestens folgende Schritte:

1. Exportieren der K8s Objekte des bereits bestehenden Clusters als YAML Dateien. Dies beinhaltete Namespaces, Deployments, Ingresse, Secrets, Configmaps, Services, Peristant Volumes und Persistant Volume Claims.

2. Anpassen der Secruity Group der zugehörigen Relational Database Service (RDS) Datenbank, damit Nodes des Clusters eine Verbindung aufbauen dürfen.

3. Erstellen der K8s Objekte im Cluster zu welchem migriert wird. Dies erfolgte durch die exportierten YAML Dateien und mit Hilfe des kubectl apply -f Befehls.

4. Hoch-skalieren der einzelnen Deployments.

5. Überprüfen der Logs der einzelnen Container auf mögliche Fehler und Beseitigen dieser.

6. Anpassen der Hosts Datei auf dem eigenem Rechner, damit alle Domains der Dienste auf den Loadbalancers des Clusters zeigen, wohin umgezogen werden soll

7. Mit dem Browser die Domains der jeweiligen Dienste besuchen und die Funktionalität der Dienste verifizieren.

8. Aktualisierung der Domain Name System (DNS) Records der betroffenen Dienste auf den Loadbalancers des Clusters zu welchem umgezogen wurde.

In der zweiten Phase, in welcher die Dienste der Produktivumgebung umgezogen werden sollten, galt es einige zusätzliche Punkte zu beachten.

- Im Gegensatz zur Testumgebung haben die Dienste eine größere Auslastung durch die aktive Nutzung. Es war daher erforderlich, das Cluster zuerst mit zusätzlichen Nodes zu erweitern, bevor die Dienste deployed werden konnten.

- In der Produktivumgebung von Zentek existiert ein Dienst „EPD-Admin“, welcher zur Stoßzeit Ressourcen von bis zu zwei EC2 t2.xlarge Instanzen verbraucht. Das Prod-Cluster von Ambient basierte zu diesem Zeitpunkt nur auf t2.large Instanzen, verfügte weder über Cluster Autoscaling (Node Anzahl), noch über Horizontal Pod Autoscaling (Deployment Replica Anzahl). Ein früherer Versuch im alten Cluster, den Dienst „EPD-Admin“ zusammen mit anderen Diensten auf Nodes zu mischen, führte zu häufigen Ausfällen einzelner Dienste. Ursache dafür waren wahrscheinlich suboptimale Ressourcen Request & Limits. Dies war ein Problem, was sich möglicherweise mit einer Optimierung dieser und der Einführung von Autoscaling hätte beheben lassen können. Um den Umzug möglichst problemfrei durchführen zu können, wurde entschieden, die alte Struktur beizubehalten. Es wurden zwei dedizierte t3.xlarge Instanzen dem Cluster hinzugefügt. Diese Nodes wurden mit einem Taint und einem Label versehen. Die YAML Datei des Deployments wurde mit der zugehörigen Toleration und nodeAffinity versehen. Damit wurde garantiert, dass die besagten zwei Nodes nur diesem Dienst zur Verfügung stehen und die Deployments bevorzugt auf diese Nodes gescheduled werden.

- Anders als die Testumgebung, beinhaltet die Produktivumgebung einen Jenkins Server, welcher im Sekundentakt Jobs im Cluster startet. Diesen galt es ebenfalls in das neue Cluster zu migrieren. Der Jenkins Server ist mit dem K8s Plugin konfiguriert. Dieses ermöglicht ihm, für jeden Job für dessen Laufzeit einen eigenen Pod im Cluster zu starten und diesen wieder zu löschen, sobald er durchgelaufen ist. Insgesamt gibt es hunderte unterschiedliche Jobs, welche auf diese Art und Weise gestartet werden, jedoch wird für alle Pods das selbe YAML Template mit den selben Requests Limits genutzt. Eine Unterscheidung der Jobs, welcher wie viele Ressourcen benötigt, wäre ohne zusätzlichen Aufwand nicht möglich. Aus diesem Grund hatten im alten Cluster auch diese Jobs eine dedizierte EC2 t2.xlarge Node, um ohne Drosselung laufen zu können und andere Dienste nicht zu beeinträchtigen. Ein früherer Versuch, diese Jobs auf den allgemeinen Worker Nodes des Clusters laufen zu lassen, führte zu einem Desaster und etlichen Ausfällen von Diensten. Dies lag an der schwer möglichen Definition von Requests und Limits bei der überaus hohen Anzahl der Arten von Jobs. Auch in diesem Fall wurde entscheiden, diese Struktur vorerst zu übernehmen, um möglichst problemfrei den Umzug durchzuführen. Es wurde eine dedizierte t3.large Maschine hinzugefügt, welche mit entsprechendem Taint und Label versehen worden ist, damit auf dieser nur die Jobs laufen. Ein weiteres Problem des Jenkins Servers

war, dass dieser ein AWS Elastic Block Store (EBS) über ein Persistant Volume Clain gemountet hat. EBS Volumes ermöglichen es jedoch nicht an unterschiedliche EC2 Instanzen gleichzeitig gemountet zu werden. Die Entscheidung für dieses Problem war, dass der Umzug zunächst ohne des Jenkins Servers vollzogen wird. Im Anschluss würde der Jenkins Server mit einer Downtime von ungefähr 2 Stunden migriert. Dass hierbei eine Downtime von mehreren Stunden entstehen könnte, war im Voraus bewusst und wurde als nicht kritisch eingeschätzt. Grund dafür war, dass die Jobs im Nachhinein abgearbeitet werden würden und nur eine Verzögerung aber kein Verlust entstehen würde. Für die Durchführung wurde der Jenkins Server herunter-skaliert, damit das EBS Volume unmounted wird und im neuem Cluster wieder hoch-skaliert, damit dort dann das Volume gemounted wird. Sobald der Jenkins Server im neuem Cluster online war, wurde dessen Funktionalität überprüft. Es wurde festgestellt, dass dieser versuchte Jobs zu starten, diese aber nicht als Pods im Cluster erstellt wurden. Die Ursache hierfür war zu Beginn nicht bekannt. Nach längerer Recherche konnte in der Wiki des GitLab Projektes von Zentek eine kurze Dokumentation (siehe Abbildung 4.1 auf S.40) gefunden werden. Mit Hilfe dieser wurden einige Werte das Jenkins K8s Plugins akualisiert. Zum einem wurde die K8s API URL mit der des neuen Clusters ersetzt. Es wurde im neuem Cluster ein "Kubernetes server certificate key"generiert, welcher in Jenkins hinterlegt wurde. Abschließend wurde noch die Jenkins Tunnel Internet Protocol (IP) Addresse aktualisiert. Hierbei handelt es sich um die interne K8s IP-Addresse und den Port welche dem K8s Service des Jenkins Server Deployments von K8s automatisiert zugewiesen worden ist. Nach Aktualisierung dieser fing der Jenkins Server an, die Jobs als Pods im neuem Cluster, exklusiv auf der dafür dedizierten Node zu starten und arbeitete alle verpassten Jobs im Nachhinein ab.

Nachdem der Umzug vollzogen war, blieb das alte Cluster noch für ungefähr eine Woche online. Es diente für diesen Zeitraum als ein Backup, auf welches schnell gewechselt werden konnte, sollte es zu Problemen kommen. Nach dieser Zeitperiode wurde das Zentek Cluster komplett aufgelöst.

8 Ersetzen der RKE Cluster durch EKS Cluster

Das Shared-, Test- und Prod-Cluster wurden ursprünglich mit der RKE und AWS EC2 Nodes betrieben. Jedoch gab es zwei Gründe, die zu der Entscheidung führten, dass diese drei bestehenden Cluster neu als AWS Elastic Kubernetes Service (EKS) Cluster errichtet werden sollten. Der erste Grund war, dass Ambient schon länger unzufrieden war mit den zusätzlichen Hürden und Problemen, die eine Administration von K8s Master Nodes mit sich brachte. Deswegen wurde auch bereits vor diesem Projekt über einen Wechsel diskutiert. Der zweite Grund war, dass Ambient Innovation sich wünschte, den K8s Cluster-Autoscaler zu nutzen, um nach Bedarf die Anzahl der Maschinen in den verschiedenen Clustern automatisch anzupassen. Das Thema wird detailierter in Implementierung des K8s Cluster Autoscalers auf S.29 behandelt werden. Ein anderer Mitarbeiter hatte sich bereits davor mit der Thematik beschäftigt, hatte dabei aber keine Lösung gefunden, dies mit RKE Clustern umzusetzen. Zu einem späteren Zeitpunkt, am 6 Juli 2020, als der praktische Teil dieses Projektes bereits abgeschlossen war, wurde die Dokumentation von Rancher um eine Anleitung[29]zur Nutzung des K8s Cluster Autoscalers unter RKE erweitert.

Der Umstieg wurde zuerst für das Shared-Cluster durchgeführt, dann für dass Test-Cluster und zum Schluss für das Prod-Cluster. Für jeden Umstieg wurden grob folgende Schritte durchgeführt:

1. Cluster Initialisieren:
   Für die Initialisierung des Clusters wurde sich für den Weg entschieden, dies über die Rancher 2 Benutzeroberfläche zu tun. Der Grund dafür war, um möglichst schnell das Cluster in Rancher einzubetten und sich nicht mit diesem Thema im Nachhinein beschäftigen zu müssen. Der Prozess hierfür bestand daraus, in der Oberfläche anzuklicken, dass man ein Cluster hinzufügen will und daraufhin den Typ EKS auszuwählen. Im nächsten Schritt mussten zusätzliche Angaben hinterlegt werden, wie AWS Credentials, die gewünschte K8s Version, Virtual Private Cloud (VPC), Subnetze und eine Konfiguration für ein Node Pool, bestehend aus EC2 Instanzen. Mit dem Hinterlegen der zusätzlichen Angeben und einem Bestätigen wurde ein Prozess gestartet der etwa 30-40 Minuten dauerte. Sobald dieser abgeschlossen war, konnte man in der Rancher UI das neue Cluster einsehen. Dieses verfügte über einen Node Pool, welcher durch eine AWS Auto Scaling Group (ASG) skalierbar war.

2. Architektur der Nodes des alten Clusters aufbauen:
   Ein Problem, welches erst im Prozess des Initialisierens des ersten Clusters be-

[29]vgl. (Luse, 2020)

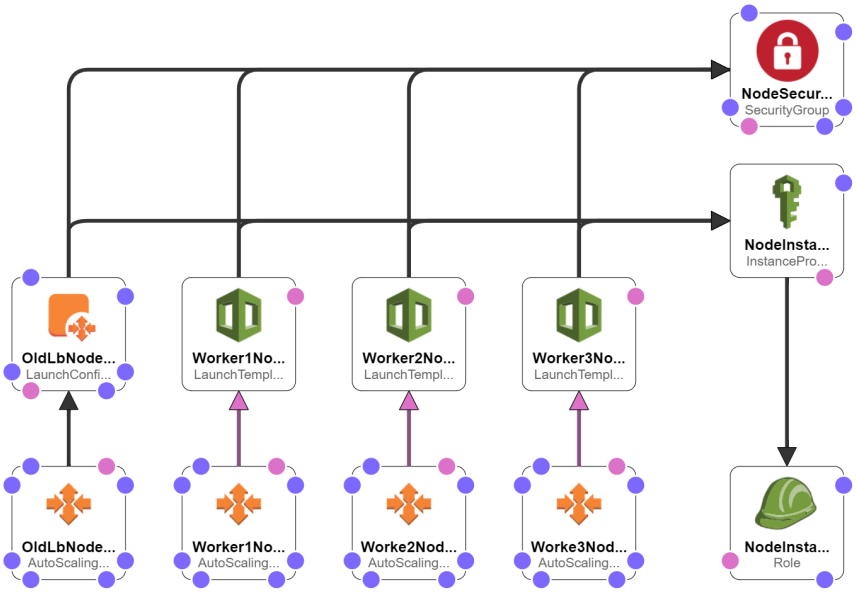


Abbildung 8.1: Cloudformation Designer: Prod-Cluster Template

merkt wurde war, dass über die Rancher Oberfläche es lediglich möglich ist, einen einzigen Node Pool zu erstellen. Auch durch späteres Editieren des Clusters über die UI konnten keine weiteren hinzugefügt werden. Mit dieser Einschränkung wäre es jedoch nicht möglich gewesen, die Architektur der alten Cluster nachzustellen. Eine Recherche hat ergeben, dass Rancher im Hintergrund AWS Cloudformation nutzt, um das Cluster und die Node Group anzulegen. In der AWS Konsole im Service Cloudformation wurde der jeweils entsprechende Stack gefunden, welcher den initialen Node Pool generiert hatte. Über den Cloudformation Designer war es anschließend möglich, das Template des Stacks als YAML Datei einzusehen und zu bearbeiten. Nach einer Analyse des bestehenden Templates und einer Evaluation der benötigten Änderungen wurden drei neue individuelle YAML Templates (eines pro Cluster) angefertigt. Der Cloudformation Designer bildete jedes der Cluster visuell dar (siehe Abbildung 8.1 auf S.27). Mit jedem dieser Templates war es möglich, die komplette Node Infrastruktur des alten Clusters in einem Cloudformation Stack abzubilden. Im nächsten Schritt wurde das Template geladen und mit den gewünschten offenen Eingabeparametern (z.B. Node Pool Name, EC2 Instanz Typ, Subnetze und so weiter (usw.)) befüllt, sodass CloudFormation die unterschiedlichen Node Groups der Cluster einrichtete.

3. K8s Objekte deployen und testen:
   Im nächsten Schritt wurden alle K8s Objekte der alten Cluster in den entsprechend neuen Clustern deployt getestet. In Migration Zentek-Cluster in Test- & Prod-Cluster auf S.23 wurde bereits thematisiert, wie bei einer Migration von Diensten im Rahmen dieses Projektes vorgegangen wurde. Hierfür wurde der selbe Workflow genutzt mit dem Unterschied, dass noch nicht die DNS Einträge

geändert wurden, weil davor zusätzlich ein Network Load Balancer eingerichtet werden musste.

4. Network Load Balancer einrichten:
   In den alten Clustern wurden alle EC2 Instanzen in der selben Availability Zone gehostet. Um sich besser vor Ausfällen beim Eintreten einer Störungen in einer Availability Zone (AZ) abzusichern, wurde beschlossen, dass im Rahmen des Umstiegs auf EKS Cluster auch die Anpassung erfolgen sollte, sodass die EC2 Instanzen der einzelnen Node Pools in unterschiedlichen AZs gehostet werden. Hierfür wurde pro Cluster ein neuer Network Load Balancer (NLB) gemäß der Anleitung von AWS[30] eingerichtet, welcher Traffic in alle 3 Subnetze weiterleiten konnte.

5. DNS Einträge auf den AWS Loadbalancer des neuen Clusters umstellen:
   Auf einige der betroffenen Domain Name Registrys hat Ambient keinen Zugriff und Anpassungen erfordern oft längere Prozesse mit dem Kunden. Dies war ein Hindernis für eine möglichst gleichzeitige Migration aller Deployments. Damit der NLB für mehrere Subnetze zuständig sein kann, stellt AWS selber eine Subdomain pro NLB aus, auf welchen über einen Canonical Name (CNAME) DNS Eintrag verweisen werden kann. Für einen Kunden stellte das in der Vergangenheit ein Problem dar. Diesem wurde die IP Addresse von dem Network Interface von einer Zone (1A) übermittelt. Dadurch konnte dieser einen A Record DNS Entry setzen. Der Nachteil hierbei ist allerdings, dass wenn das Cluster aus Nodes aus unterschiedlichen AZs besteht und keine der Nodes aus dieser AZ ist, der Traffic über die eben genannte IP Adresse nicht im Cluster ankommen würde. Um das Problem mit dem langwierigen Prozess von DNS Anpassung durch Kunden temporär zu lösen, wurde die CloudFormation Templates um ein zusätzlichen Node Pool erweitert. Dieser Node Pool sollte aus genau einer statischen Node bestehen die nur in Zone 1A sein darf. Gleichzeitig wurde die Hypertext Transfer Protocol (HTTP) und Hypertext Transfer Protocol Secure (HTTPS) Target Group des jeweiligen alten NLB so aktualisiert, damit dort nur noch die eben genannte statische Node eingetragen ist. Mit dieser Anpassung pro Cluster wurde der komplette Traffic von allen Domains ohne DNS Anpassung über den alten NLB in die neuen Cluster umgeleitet. Die Nutzung des alten Loadbalancers mit nur einer Node als Target ist extrem anfällig für einen Komplettausfall und sollte die eine statische Node versagen, würde gar kein Traffic mehr über diesen Loadbalancer in das Cluster gelangen. Hierbei handelte es sich nur um eine temporäre Lösung, um den Prozess zu beschleunigen. Im Nachhinein wurden mit den Kunden Anpassungen der DNS Einträge auf einen CNAME des neuen NLB koordiniert und schließlich auch die komplette Node Group wieder entfernt, sobald diese nicht mehr notwendig war.

6. Alte Cluster auflösen:
   Nachdem der Traffic für ungefähr eine Woche in die neuen Cluster gelenkt wurde und es zu keinen Problemen oder Beschwerden geführt hat, wurden die alten Cluster aufgelöst.

[30] vgl. (Anthony, 2019)

8.1 Implementierung von Managed EKS Node Groups

Einige Monate später, am 18. November 2019, erweiterte AWS das Produkt EKS mit dem Feature Managed Node Groups[31]. Obwohl die Node Groups der EKS Cluster von Ambient inzwischen durch die CloudFormation Templates verwaltet wurden, hatte die EKS eigene Lösung Vorteile (leichter wartbar, übersichtlicher) und ein Wechsel sollte unternommen werden. Der Wechsel wurde durchgeführt, indem neue Node Groups mit identischen Parametern (Größe, Anzahl, Labels, Taints, Volumes etc.) erstellt wurden. Danach wurden die alten Nodes gedrained, damit die Deployments sich auf die neuen Nodes verlagern. Abschließend wurden die alten Node Groups entfernt. Das Hinzufügen einer Node Group war sehr einfach und über die AWS UI möglich. Hierfür musste in das entsprechende Cluster navigiert werden und dann konnte über einen Button die Node Group hinzugefügt werden. Nach Ausfüllen zusätzlicher Angaben wie (Name, Größe, Anzahl etc.) wurden automatisch Elemente wie EC2-Instanzen, ASGs und EC2-Startvorlagen erstellt. Nodes wurden automatisch gestartet und dem Cluster hinzugefügt. Bei der Konfiguration der Node Groups ist man jedoch auf einige vorbestimmte Parameter beschränkt und kann nicht alles individualisieren. Deswegen ist es zum Beispiel nicht möglich gewesen, die Nodes als Spot Instanzen zu starten oder Node Taints zu definieren. Eine Lösung hierfür wurde erreicht, indem die entsprechende EC2-Statvorlage im Nachhinein angepasst wurde und die Nodes wieder entfernt und neu hinzugefügt worden sind.

8.2 Implementierung des K8s Cluster Autoscalers

Einer der Gründe für den Umstieg auf EKS war unter Anderem, damit der K8s Cluster Autoscaler implementiert werden kann. Dieser ermöglicht, dass je nach Auslastung des Clusters die entsprechenden Nodes automatisch hoch oder runterskaliert werden. Für den Cluster Autoscaler gibt es ein Helm Chart, welches etwa im „kubernetes-charts“ Helm Catalog gelistet ist. Da dieser Catalog bereits in Rancher hinterlegt war, war auch das entsprechende Helm Chart für alle Cluster verfügbar. Die Installation wurde durch Auswählen des Helm Charts und Eingeben von Werten über die answers.yaml durchgeführt. Zu diesen Werten gehörten unter Anderem AWS Credentials, die AWS Region, die Sensibilität, Namen der EC2 ASGs und die dazugehörige maximale und minimale Anzahl an Nodes pro ASG. Sobald diese bestätigt waren, wurden die entsprechenden K8s Objekte erstellt. In einigen inszenierten Tests wurde überprüft, ob der Autoscaler funktionierte. Dies war dar Fall.

[31] vgl. (Hausenblas, 2019)

9 Implementierung von Horizontal Pod Autoscaling

Im Kapitel Implementierung des K8s Cluster Autoscalers auf S.29 wurde zwar der K8s Cluster Autoscaler implementiert, aber zu diesem Zeitpunkt wurde dieser nur aktiv, wenn Deployments addiert oder entfernt wurden. Denn nur dann wurden die benötigten Ressourcen verändert. Das volle Potential wird jedoch erst genutzt, wenn man diesen mit einer Lösung für horizontales Autoscaling kombiniert. Um dies zu erreichen, sollte der Horizontal Pod Autoscaler (HPA) eingesetzt werden. Mit diesem ist es möglich basierend auf Auslastung von Deployments die Anzahl der Replicas automatisch zu skalieren. Dies hat zur Folge, dass die benötigten Ressourcen für das Cluster sich anpassen würden, was wiederum eine vertikale Skalierung des Clusters durch den K8s Autoscaler verursachen würde.

Im Rahmen dieses Projektes galt es zuerst den K8s Metric Server zu installieren, damit HPAs definiert werden konnten. Dann die Deployments zu identifizieren, welche Schwankungen in ihrer Auslastung haben, die groß genug sind, dass es sich lohnen würde für diese HPA zu nutzen. Für diese sollten im Anschluss K8s HPA Objekte definiert werden. Für genannten Prozess wurde sich an dem User Guide [32] von AWS orientiert. Nach einer Installation des Metric Server schien dieser zuerst nicht zu funktionieren, was wie sich herausstellte an der Firewall der einzelnen EC2-Instanzen lag. Durch eine Anpassung der EC2 Security Groups konnte dieses Problem gelöst werden. Für die Analyse der Workloads wurde das im Kapitel 2.1 auf S.7 implementierte Monitoring genutzt. Für diesen Prozess wurde jedoch nur das Prod-Cluster untersucht, da die Dienste welche im Test-Cluster betrieben werden nur einer kleinen Nutzergruppe bekannt sind und es nicht erwartet wird, dass diese große Schwankungen in Auslastung verzeichnen. Für das Prod-Cluster stellte sich heraus, dass die meisten Dienste eine relativ gleichmäßige Auslastung verzeichnen, mit Ausnahme der Dienste des Kunden Zentek. Deren Auslastung hatten je nach Wochentag und Uhrzeit große Differenzen. Deswegen wurde sich entschieden, nur für diese Deployments HPAs zu definieren und für die restlichen Dienste darauf zu verzichten, da es dort kaum Auswirkungen hätte. Das Erstellen der HPAs wurde über die Rancher UI durchgeführt. Bei der Einrichtung eines HPAs musste eine minimale und maximale Anzahl an Replicas definiert werden. Zusätzlich musste noch mindestens eine Metrik bestimmt werden, auf Basis welcher der HPA die gewünschten Replicas bestimmt. Weil bei allen betroffenen Workloads die CPU Auslastung der relevante Faktor ist, wurde diese als Metrik für die Skalierung gewählt. Für diese Metrik musste ein Breakpoint in Prozent bestimmt werden, bei welchem die positive oder negative Skalierung erfolgen würde. 100% entspricht hierbei dem gesetzten CPU Request pro Container. In Optimierung der K8s Resource Re-

[32] vgl. (AWS, b)

quests und Limits auf S.15 wurden die Requests für Deployments so definiert, dass diese möglichst nah an dem Wert sind, was das Deployment braucht, um im Normalzustand uneingeschränkt zu funktionieren. Dies würde bedeuten, das für den Metric Server die Deployments immer mindestens eine CPU Auslastung haben würde, die relativ nah an 100% ist. Deswegen wurde für alle Deployments die einen HPA haben sollten, der CPU Request verdoppelt. Damit würden diese Deployments im Normalbetrieb ohne zusätzliche Auslastung vom Metric Server mit 50% Auslastung interpretiert werden. Zudem wurden die HPA Metriken so eingestellt, dass ab einer durchschnittlichen CPU Auslastung von 75%, also einer 1,5 fachen Auslastung wie im Normalzustand, der HPA die Replicaanzahl des Deployments hochskalieren soll.

10 Optimierung der GitLab Runner

GitLab Runner haben die Aufgabe, CI/CD Jobs von GitLab Projekten abzuarbeiten. Dabei gibt es mehrere Möglichkeiten diese zu konfigurieren, beziehungsweise zu betreiben. Bei Ambient wurde sich vor längerer Zeit für den Weg entschieden, mit Hilfe eines Helm Charts in einem dedizierten K8s Cluster, einen GitLab Runner Manager zu deployen. Dieser Manager startet pro zu bearbeitenden Job einen neuen K8s Pod, der als kurzlebiger GitLab Runner dient und nach Abarbeiten seines Jobs wieder gelöscht wird. Als Ressourcen standen hierfür durchgängig 2x t2.xlarge AWS EC2-Instanzen zur Verfügung. Dabei werden in der Regel nur während der Arbeitszeiten Jobs gestartet, was ungefähr 60% der Zeit der bezahlten Ressourcen unnötig macht. Des Weiteren konnte es in seltenen Fällen dazu kommen, dass so viele Jobs gleichzeitig abgearbeitet wurden, dass die CPU Leistung der Nodes an ihr Limit kamen und die Jobs gedrosselt wurden. Dies resultierte in längeren Laufzeiten, was wiederum in Beschwerden von Mitarbeitern resultierte. Von Ambient wurde sich eine Lösung gewünscht, welche automatisiert dafür sorgt, dass immer genug Ressourcen zur Verfügung stehen, damit alle CI Jobs in voller Geschwindigkeit abgearbeitet werden können, aber auch, dass so möglich wie wenig bezahlt werden muss für ungenutzte Ressourcen.

Zuerst wurde wie in 2.7.2 auf S.29 beschrieben ein K8s Cluster Autoscaler eingerichtet und die bereits vorhandene EC2 ASG genutzt. Nach Testen der Änderungen stellte sich heraus, dass die ASG nicht hoch skalierte, wenn die Kapazitäten auf einer Node knapp wurden. Stattdessen wurden weitere Pods auf dieser Node gestartete. Grund dafür war, dass die Pods, welche durch den GitLab Runner Manager gestartet wurden, keine Ressource Request hatten. Dies hatte zur Folge, dass der Cluster Autoscaler nicht wusste, wann die Node zu wenig Platz hat, um einen weiteren Gitlab Runner Job zu erlauben. Das Helm Chart des GitLab Runner Managers ermöglicht es für die Runner die er startet, universell Requests zu definieren. Dies wurde im nächsten Schritt unternommen. Das nächste Problem was entstand war, dass beim Definieren eines identischen Requests für alle Jobs, diese viel weniger oder viel mehr Ressourcen benötigen könnten, als sie reservierten. Wenn ein Job weniger Ressourcen benötigt als der definierte Request, dann werden unnötigerweise Ressourcen blockiert auf der Node. Dies kann dazu führen, dass unnötigerweise eine zusätzliche Node hochskaliert wird. Wenn ein Job weit aus mehr Ressourcen benötigt als der definierte Request, dann kann es dazu kommen, dass zu viele Jobs auf einer Node landen bevor hoch skaliert wird und einzelne Jobs gedrosselt werden.

Verbessert wurde diese Situation durch das Deployen von 3 weiteren GitLab Runner Managern. Sodurch war es möglich zumindest 4 unterschiedliche Requests zu definieren. Jedem dieser Jobs wurde ein anderer GitLab Runner Tag zugewiesen. Dadurch würde jeder nur CI Jobs starten, welche mit seinem exclusiven Tag markiert sind. Der

Ansatz hat funktioniert und die Nodes wurden zwar nicht perfekt skaliert, aber zu einem Grad, dass es zufriedenstellend war. Ein sodurch erreichtes Ziel war, dass wenn keine CI Jobs laufen, die Anzahl der Instanzen komplett auf 0 skaliert wurde. Dies war das primären Ziele des Umbaus.

Ein Thema war jedoch noch suboptimal. Es wurde immer mit kompletten Nodes der Größe t2.xlarge skaliert. Wenn z.B. gerade keine Jobs liefen und jemand einen kleinen Job gestartet hat, wurde direkt eine Instanz dieser Größe gestartet, obwohl gegebenenfalls 1/8 gereicht hätte. Dieses Problem wurde damit gelöst, dass zwei weitere Managed EKS Node Pools angelegt wurden. Einmal für Nodes der Größe t2.medium und einmal für t2.large. Dem Bootsrap Befehl der 3 EC2 Launchtemplates wurde außerdem ein Argument hinzugefügt. Dieses verursachte, dass die Nodes nach dem Start mit ihrer entsprechenden Größe gelabelt werden. Diese Labels wurden genutzt, um durch die Helm Charts einen NodeSelector definieren zu können. Das Endergebnisse war eine Lösung, in welcher CI Jobs basierend auf ihrem Tag von einem bestimmten Gitlab Runner Manager aufgenommen wurde. Dieser startet wiederum einen Pod, der einen spezifischen Request hat und mit einem spezifischen Nodeselector versehen ist, welcher die gewünschte Nodegröße bestimmt. Die Tags und ihre entsprechenden Requests mit NodeSelector wurden wie folgt definiert:

- low-load: 50mCPU auf t2.medium (bis zu 8 mal pro Node)
- normal-load: 700mCPU auf t2.large (bis zu 2 mal pro Node)
- high-load: 1400mCPU auf t2.large (eigene Node pro Job)
- very-high-load: 3000mCPU auf t2.xlarge (eigene Node pro Job)

Mit diesen Werten könnten bis zu ungefähr 8 low-load Jobs auf einer t2.medium Instanz laufen und es wurde nur eine t2.medium Instanz gestartet, wenn nicht mehr benötigt wurde. Größere Nodes wurden erst gestartet, wenn diese für Jobs mit größerer Auslastung notwendig waren.

Das Zuweisen der richtigen Tags zu den jeweiligen Jobs sollte Aufgabe der entsprechenden Entwicklerteams werden. Diese kennen ihre CI Jobs am besten und können im Code über die .gitlab-ci.yaml die Tags ihrer Jobs definieren. Im Prozess des initialen Umstiegs wurde den Teams jedoch geholfen. Bereits nach einigen Tagen wurde durch Feedback von Entwicklerteams festgestellt, dass die neue Lösung viel besser ist. Auch einige Monate nach der Umsetzung läuft die Lösung immer noch zuverlässig und verursacht ungefähr 90% weniger Betriebskosten wie zuvor.

11 Fazit und Zukunftsaussichten

Seit der Umsetzung der Änderungen an der Architektur von Ambient sind inzwischen mehrere Monate vergangen. Der neue Zustand hat sich als große Verbesserung dargestellt. Besonders durch das Alerting, welches schon einige Male ausgelöst worden ist, konnte viel schneller reagiert werden. Auch fest definierte Prozesse für unterschiedliche Problemszenarien helfen in der Hinsicht, möglichst schnell bei Problemen zu handeln. An einigen Stellen konnten sogar durch das Praxisprojekt die Betriebskosten für das Unternehmen gesenkt werden. Das Ziel wurde deswegen aus meiner Sicht vollständig erfüllt und auch das Unternehmen stuft dieses Projekt als vollen Erfolg ein. Selbst in meinem Alltag im Unternehmen merke ich immer noch die positiven Auswirkungen auf die Architektur, welche das Projekt mit sich gebracht hat.

Für mich selber hat sich durch das Projekt vieles in meiner täglichen Arbeit bei Ambient geändert. Inzwischen bin ich dort nicht mehr primär als Software Entwickler aktiv, sondern arbeite überwiegend im Development and Operations (DevOps) Bereich.

Eine Fortsetzung oder Anknüpfung an das Projekt, z.B. etwa im Rahmen der Bachelorarbeit, sehe ich als eine gute Möglichkeit an. Hier gäbe es zum einen die Option die Thematik Operations beziehungsweise DevOps stärker literarisch zu behandeln. Sei es mit den Themen des Praxisprojektes oder anderen aus diesem Gebiet. Eine weitere Möglichkeit könnte aber auch sein, praktisch das Thema Infrastructure as Code in das besagte Unternehmen einzuführen. Zum Beispiel ist es aktuell noch bei Ambient ein langer manueller Akt, die Infrastruktur aufzubauen. Sollte es zu einer Katastrophe kommen und ein Recovery Prozess notwendig sein, würde dieser viel Zeit in Anspruch nehmen. Mit Technologien wie Terraform könnte dies jedoch automatisiert werden.

Abbildungsverzeichnis

Tabellenverzeichnis

Literaturverzeichnis

[ambientinnovation 2020a] AMBIENTINNOVATION: *ai-rancher2-upgrade @ONLINE*. `https://github.com/ambient-innovation/rancher2-upgrade/`. Version: Oktober 2020. – Zuletzt aufgerufen am 17. Januar 2019

[ambientinnovation 2020b] AMBIENTINNOVATION: *ambientinnovation/rancher2-upgrade @ONLINE*. `https://hub.docker.com/r/ambientinnovation/rancher2-upgrade/`. Version: Oktober 2020. – Zuletzt aufgerufen am 17. Januar 2019

[ambientinnovation 2020c] AMBIENTINNOVATION: *k8s-backup @ONLINE*. `https://github.com/ambient-innovation/k8s-backup`. Version: Oktober 2020. – Zuletzt aufgerufen am 17. Januar 2019

[Anthony 2019] ANTHONY, Cornell: *Using a Network Load Balancer with the NGINX Ingress Controller on Amazon EKS @ONLINE*. `https://aws.amazon.com/de/blogs/opensource/network-load-balancer-nginx-ingress-controller-eks/`. Version: August 2019. – Zuletzt aufgerufen am 17. Januar 2019

[AWS a] AWS: *CPU credits and baseline utilization for burstable performance instances @ONLINE*. `https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/burstable-credits-baseline-concepts.html`. – Zuletzt aufgerufen am 17. Januar 2019

[AWS b] AWS: *Horizontal Pod Autoscaler @ONLINE*. `https://docs.aws.amazon.com/eks/latest/userguide/horizontal-pod-autoscaler.html`. – Zuletzt aufgerufen am 17. Januar 2019

[Bajon 2019] BAJON, David: *Make Restore Snapshot possible across clusters @ONLINE*. `https://github.com/rancher/rancher/issues/20051/`. Version: Mai 2019. – Zuletzt aufgerufen am 17. Januar 2019

[Hausenblas 2019] HAUSENBLAS, Michael: *Extending the EKS API: Managed Node Groups @ONLINE*. `https://aws.amazon.com/de/blogs/containers/eks-managed-node-groups/`. Version: November 2019. – Zuletzt aufgerufen am 17. Januar 2019

[Kim 2019] KIM, Chris: *Reserve Compute Resources for System Daemons @ONLINE*. `https://rancher.com/learning-paths/building-a-highly-available-kubernetes-cluster/`. Version: September 2019. – Zuletzt aufgerufen am 17. Januar 2019

[Logan 2019] Logan: *Cluster monitoring cannot be restored once disabled @ONLINE*. https://github.com/rancher/rancher/issues/19209#issuecomment-479404791. Version: März 2019. – Zuletzt aufgerufen am 17. Januar 2019

[Luse 2020] Luse, Catherine: *Cluster Autoscaler with AWS EC2 Auto Scaling Groups @ONLINE*. https://github.com/rancher/docs/blob/master/content/rancher/v2.x/en/cluster-admin/cluster-autoscaler/amazon/_index.md. Version: Juli 2020. – Zuletzt aufgerufen am 17. Januar 2019

[rancherio-gh m 2019] m rancherio-gh: *Release v2.2.0 @ONLINE*. https://github.com/rancher/rancher/releases/tag/v2.2.0/. Version: März 2019. – Zuletzt aufgerufen am 17. Januar 2019

[Mattox 2019] Mattox, Matthew: *etcd alerts during backups @ONLINE*. https://github.com/rancher/rancher/issues/19474. Version: April 2019. – Zuletzt aufgerufen am 17. Januar 2019

[PrometheusAuthors] PrometheusAuthors: *Prometheus Overview @ONLINE*. https://prometheus.io/docs/introduction/overview/. – Zuletzt aufgerufen am 17. Januar 2019

[Rancher a] Rancher: *Cluster Alerts @ONLINE*. https://rancher.com/docs/rancher/v2.x/en/monitoring-alerting/v2.0.x-v2.4.x/cluster-alerts. – Zuletzt aufgerufen am 17. Januar 2019

[Rancher b] Rancher: *Cluster Configuration @ONLINE*. https://rancher.com/docs/rancher/v2.x/en/cluster-admin/editing-clusters/. – Zuletzt aufgerufen am 17. Januar 2019

[Rancher c] Rancher: *Configuring Recurring Snapshots @ONLINE*. https://rancher.com/docs/rancher/v2.x/en/cluster-admin/backing-up-etcd/#configuring-recurring-snapshots/. – Zuletzt aufgerufen am 17. Januar 2019

[Rancher d] Rancher: *Enabling Cluster Monitoring @ONLINE*. https://rancher.com/docs/rancher/v2.x/en/monitoring-alerting/v2.0.x-v2.4.x/cluster-monitoring/#enabling-cluster-monitoring/. – Zuletzt aufgerufen am 17. Januar 2019

[Rancher e] Rancher: *Launching Kubernetes and Provisioning Nodes in an Infrastructure Provider @ONLINE*. https://rancher.com/docs/rancher/v2.x/en/cluster-provisioning/#launching-kubernetes-and-provisioning-nodes-in-an-infrastructure-provider/. – Zuletzt aufgerufen am 17. Januar 2019

[Rancher f] Rancher: *Notifiers @ONLINE*. https://rancher.com/docs/rancher/v2.x/en/monitoring-alerting/v2.0.x-v2.4.x/notifiers/. – Zuletzt aufgerufen am 17. Januar 2019

[Rancher g] Rancher: *Project Alerts @ONLINE*. https://rancher.com/docs/rancher/v2.x/en/monitoring-alerting/v2.0.x-v2.4.x/cluster-alerts/project-alerts/. – Zuletzt aufgerufen am 17. Januar 2019

[Rancher h] Rancher: *Resource Consumption @ONLINE*. https://rancher.com/docs/rancher/v2.x/en/monitoring-alerting/v2.0.x-v2.4.x/cluster-monitoring/#resource-consumption/. – Zuletzt aufgerufen am 17. Januar 2019

[Rancher i] Rancher: *Upgrading Rancher Installed with Docker @ONLINE*. https://rancher.com/docs/rancher/v2.x/en/installation/other-installation-methods/single-node-docker/single-node-upgrades/. – Zuletzt aufgerufen am 17. Januar 2019

[Rancher j] Rancher: *Viewing Metrics @ONLINE*. https://rancher.com/docs/rancher/v2.x/en/monitoring-alerting/v2.0.x-v2.4.x/cluster-monitoring/viewing-metrics/. – Zuletzt aufgerufen am 17. Januar 2019

[slassh 2018] slassh: *deleting namespace stuck at "Terminatingßtate @ONLINE*. https://github.com/kubernetes/kubernetes/issues/60807#issuecomment-408599873. Version: Juli 2018. – Zuletzt aufgerufen am 17. Januar 2019

[TheKubernetesAuthors 2020a] TheKubernetesAuthors: *Configure Quality of Service for Pods @ONLINE*. https://kubernetes.io/docs/tasks/configure-pod-container/quality-service-pod/. Version: November 2020. – Zuletzt aufgerufen am 17. Januar 2019

[TheKubernetesAuthors 2020b] TheKubernetesAuthors: *Options for Highly Available topology @ONLINE*. https://kubernetes.io/docs/setup/production-environment/tools/kubeadm/ha-topology/. Version: Mai 2020. – Zuletzt aufgerufen am 17. Januar 2019

[TheKubernetesAuthors 2020c] TheKubernetesAuthors: *Reserve Compute Resources for System Daemons @ONLINE*. https://kubernetes.io/docs/tasks/administer-cluster/reserve-compute-resources/. Version: Oktober 2020. – Zuletzt aufgerufen am 17. Januar 2019

[TheKubernetesAuthors 2020d] TheKubernetesAuthors: *Tools for Monitoring Resources @ONLINE*. https://kubernetes.io/docs/tasks/debug-application-cluster/resource-usage-monitoring/. Version: Oktober 2020. – Zuletzt aufgerufen am 17. Januar 2019

[TheKubernetesAuthors 2020e] TheKubernetesAuthors: *Using kubectl proxy @ONLINE*. https://kubernetes.io/docs/tasks/access-application-cluster/access-cluster/#using-kubectl-proxy/. Version: Oktober 2020. – Zuletzt aufgerufen am 17. Januar 2019

Anhang

7b. Configure Kubernetes Plugin with Jenkins

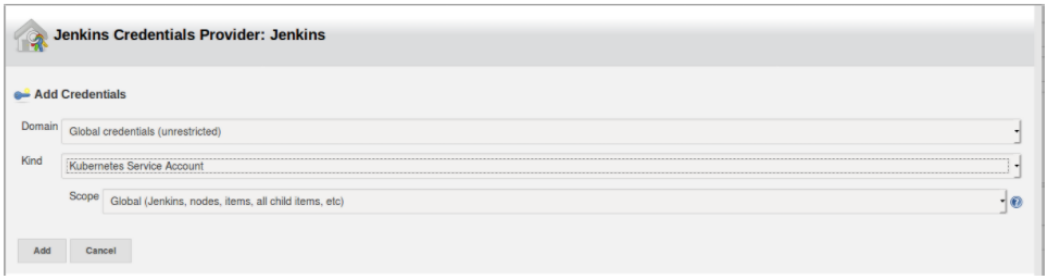
- Go to the URL https://jenkins-jobs.zentek.ambient-innovation.com/manage ==> Configure System ==> Scroll to the Cloud Section ==> Select Add a new cloud and Kubernetes
- Add the following Configuration:

Name: `Zentek Kubernetes Cluster`

Kubernetes URL: API EndPoint from Rancher "Go to the Rancher Server, Choose the Cluster ==> View in API ==> Search for `apiEndpoint`". You **MUST** enter a port in this URL, https://XYZ.eu-central-1.eks.amazonaws.com must be entered as https://XYZ.eu-central-1.eks.amazonaws.com:443.

Kubernetes Namespace: epd "Namespace in which pods are deployed"

Credentials: Add ==> Kind: Kubernetes Service account



Jenkins URL: https://jenkins-jobs.zentek.ambient-innovation.com

Jenkins tunnel: 10.43.252.157:50000 "Service IP of Jenkins:50000", you can get Service IP from Service Discovery in Rancher.

Images: Kubernetes Pod Template

Name: jobs

Namespace: epd

Labels: jobs "This pod template label must be jobs because the jenkins-jobs are configured to run on pod templates with the label jobs"

Containers: Container Template Name: jnlp "The name must be jnlp, this is a jenkins-agent configuration in order to make it work this way"

Docker image: registry.ambient-innovation.com/zentek/epd/base-image:develop

Always pull image: check

Abbildung 4.1: EPD GitLab Wiki Jenkins Eintrag
Quelle: Eigene Darstellung, Bildschirmkopie

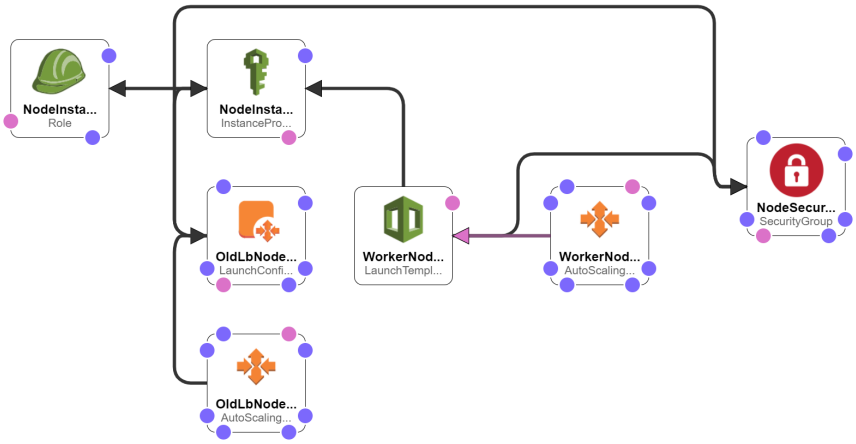


Abbildung 4.2: Cloudformation Designer: Test-Cluster Template
Quelle: Eigene Darstellung, Bildschirmkopie

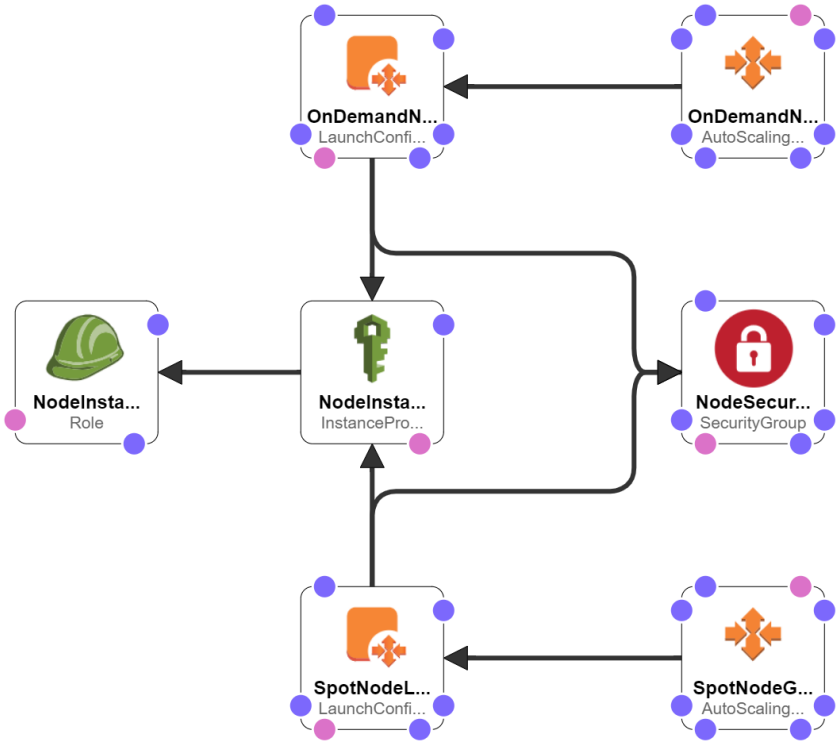


Abbildung 4.3: Cloudformation Designer: Shared-Cluster Template
Quelle: Eigene Darstellung, Bildschirmkopie

Test Cluster

	Pod	avg. CPU current	Memory current	CPU request	CPU limit	Memory request	Memory limit	new CPU req	new CPU limit	new Mem req	new Mem limit
Stroeer	web	0,0000	278	0,500	0,500	500	500	0,001	0,500	300	500
Stroeer-special	web	0,0000		0,500	0,500	500	500	0,001	0,500	300	500
efs-provisioner	efs-provisioner	0,0017	10	-	-	-	-	0,002	0,050	50	100
cert-manager	cert-manager	0,0022	13	-	-	-	-	0,003	0,100	50	100
ai-portal	celery-beat-wo	0,0180	190	0,300	0,500	1.000	1.000	0,001	0,100	250	500
	web	0,0639	290	0,050	0,300	100	768	0,005	0,500	400	800
ai-web	ai-web	0,0014		0,050	0,250	100	300	0,002	0,250	250	400
ec-kassenbuch	beat-worker	0,0010	142	-	-	-	-	0,002	0,250	150	300
	web	0,0023	85	-	-	-	-	0,002	0,250	100	200
ec-web	beat-worker	0,0134	150	0,150	0,150	75	150	0,002	0,250	170	300
	web	0,0500	125	0,050	0,250	100	200	0,050	0,250	170	300
wahnsinn	django	0,0016	100	0,050	0,250	100	300	0,001	0,250	100	300
	react	0,0000	24	-	-	320	640	0,001	0,100	50	100
kaiserkraft	cms	0,0013	80	0,050	0,250	100	300	0,002	0,250	100	300
poolbook	breat-worker	0,0663	237	0,200	0,200	512	512	0,070	0,200	250	500
	rabbit	0,0096	78	-	-	-	-	0,010	0,200	100	200
	web	0,0036	356	0,050	0,300	300	1.000	0,005	0,300	400	800
olz-logistiksyster	backend	0,0006	259	0,050	0,300	75	600	0,001	0,300	270	500
	beat-worker	0,0010	153	0,200	0,500	200	500	0,001	0,300	160	300
fm-community-te	backend	0,0003	166	0,250	0,250	400	400	0,001	0,250	200	400
	frontend	0,0001	66	0,100	0,300	128	512	0,001	0,300	100	300
	prisma	0,0033	135	0,200	0,200	400	400	0,005	0,200	250	400
portal	backend	0,0002	149	0,100	0,250	150	300	0,001	0,250	150	300
	frontend	0,0011	23	0,100	0,100	350	350	0,001	0,100	50	150
empto	backend	0,0010	298	0,050	0,750	300	1.500	0,001	0,750	300	1.500
	celerybeat	0,0000	91	-	-	100	300	0,001	0,100	100	300
	celeryflower	0,0007	93	0,200	0,200	100	300	0,001	0,200	100	300
	celeryworker	0,0008	150	0,200	0,200	100	300	0,001	0,200	150	300
	frontend	0,0003	35	0,100	0,100	256	256	0,001	0,100	50	150
	rabbitmq	0,0117	90	0,050	0,200	200	300	0,020	0,200	100	300
empto-review	backend	0,0300	159	0,050	0,250	100	300	0,050	0,250	160	300
	frontend	0,0000	30	0,200	0,200	550	700	0,001	0,200	50	150
empto-stage	backend	0,0010	392	0,050	0,500	100	1.024	0,001	0,500	400	1.024
	celerybeat	0,0000	93	0,100	0,250	100	300	0,001	0,250	100	300
	celeryflower	0,0007	95	0,150	0,300	100	200	0,001	0,300	100	200
	celeryworker	0,0008	170	0,200	0,200	300	300	0,001	0,200	170	300
	frontend	0,0003	37	0,100	0,100	512	512	0,001	0,100	50	150
	rabbitmq	0,0103	86	0,050	0,200	200	300	0,020	0,200	100	300
epd test	abfaelle	0,0001	21	0,150	0,150	300	300	0,001	0,150	25	300
	admin	0,0000	21	0,250	1,000	1.000	4.000	0,001	1,000	25	4.000
	autoverschrott	0,0001	22	0,15	0,15	300	300	0,001	0,15	25	300
	containerdiens	0	28	0,25	0,5	500	2000	0,001	0,5	30	2000
	containerdiens	0,0001	17	0,125	0,25	500	500	0,001	0,25	25	500
	elektroschritt	0,0001	12	0,15	0,15	300	300	0,001	0,15	15	300
	entsorgung-cu	0	46	0,5	1	1000	4000	0,001	1	50	4000
	entsorgung-pa	0	37	0,25	0,5	1000	2000	0,001	0,5	50	2000
	entsorgung-po	0	0	0,25	0,25	1000	2000	0,001	0,25	10	2000
	entsorgung-sta	0	0	0,125	0,25	500	1000	0,001	0,25	10	1000
	hal-api	0	0	0,25	0,5	1000	2000	0,001	0,25	10	2000
	mongodb	0,003	75	0,25	1	500	1000	0,005	1	75	1000
	muell	0	0	0,15	0,15	300	300	0,001	0,15	10	300
	schrott	0	0	0,25	0,5	1000	2000	0,001	0,5	10	2000
	schrott-static	0,0001	20	0,125	0,25	250	500	0,001	0,25	20	500
	schrottauto24	0	0	0,15	0,15	300	300	0,001	0,15	10	300
			Total:	7,825		18.178		0,292		6.700	
			New percentage:					0,963		0,631422	

Tabelle 4.1: Resource Optimierung: Test-Cluster

Prod Cluster

	Pod	avg. CPU current	Memory current	CPU request	CPU limit	Memory request	Memory limit	new CPU req	new CPU limit	new Mem req	new Mem limit
empto-prod	backend	0,0030	269	0,050	0,500	300	1.024	0,001	0,500	300	1.024
	celerybeat	0,0000	91	0,100	0,250	100	300	0,001	0,250	100	300
	celeryflower	0,0007	93	0,200	0,200	100	300	0,001	0,200	100	300
	celeryworker	0,0008	155	0,200	0,200	100	300	0,001	0,200	160	300
	frontend	0,0003	35	0,100	0,100	512	512	0,001	0,100	50	512
	rabbitmq	0,0117	90	0,050	0,200	200	300	0,020	0,200	100	300
wahnsinn	django	0,0016	172	0,050	0,250	100	300	0,001	0,250	180	300
	react	0,0000	24	-	-	320	640	0,001	0,250	50	640
poolbook	breat-worker	0,0663	315	0,100	0,300	128	1.024	0,150	0,300	320	1.024
	rabbit	0,0100	158	-	-	-	-	0,010	0,200	160	300
	portal	0,0036	492	0,400	0,400	1.024	1.024	0,005	0,400	500	1.024
Stroeer	web	0,0005	302	0,150	0,300	500	500	0,001	0,300	300	500
Stroeer-special	web	0,0005	300	0,300	0,300	500	500	0,001	0,300	300	500
holz-reinmann	backend	0,0006	498	0,050	0,500	75	800	0,001	0,500	500	800
	beat-worker	0,0030	245	0,200	0,500	200	500	0,001	0,500	250	500
ec-kassenbuch	beat-worker	0,0010	142	-	-	-	-	0,002	0,250	170	300
	web	0,0010	98	0,300	0,500	150	300	0,002	0,500	100	300
ec-web	beat-worker	0,0013	182	0,150	0,150	75	150	0,002	0,250	190	300
	web	0,0010	242	0,050	0,500	100	200	0,001	0,500	250	500
mice-booking	celery-beat-worker	0,0001	254	0,400	0,400	850	850	0,001	0,400	260	850
	celery-flower	0,0001	128	0,100	0,100	200	200	0,001	0,100	150	200
	rabbitmq	0,0092	89	0,050	0,150	150	250	0,010	0,150	100	250
	web	0,0070	599	0,050	0,500	100	1.000	0,008	0,500	600	1.000
mice-booking-test	celery-beat-worker	0,0001	300	0,400	0,400	850	850	0,001	0,400	300	850
	celery-flower	0,0001	123	0,100	0,100	200	200	0,001	0,100	125	200
	rabbitmq	0,0092	89	0,050	0,150	150	250	0,010	0,150	100	250
	web	0,0070	256	0,050	0,500	100	1.000	0,008	0,500	260	1.000
calcexporter	react	0,0000	23	0,250	0,250	100	128	0,001	0,250	25	128
dennisgilliam	react	0,0000	35	0,250	0,250	100	128	0,001	0,250	50	128
spendenverwaltungsportal	demo	0,0072	87	0,200	0,200	256	256	0,008	0,200	90	256
webseite	wagtail	0,0018	287	0,050	0,250	100	300	0,002	0,250	290	400
			Total:	4,400		7.640		0,255		6.430	
			New percentage:					0,942045		0,158376	

Tabelle 4.2: Resource Optimierung: Prod-Cluster

Shared Cluster

	QoS	Pod	avg. CPU current	Memory current	CPU request	CPU limit	Memory request	Memory limit	new CPU req	new CPU limit	new Mem req	new Mem limit
Redash	guaranteed	adhoc-worker	0,0006	279	0,250	0,250	500	500	0,001	0,050	300	500
	best efford	redis	0,0004	10	-	-	-	-	0,001	0,050	50	100
	guaranteed	scheduled-worke	0,0006	179	0,250	0,250	500	500	0,001	0,050	250	500
	guaranteed	scheduler	0,0022	271	0,250	0,250	500	500	0,003	0,100	300	500
	guaranteed	server	0,0000	655	0,250	0,250	1.953	2.441	0,001	0,050	1.000	2.000
Sentry	burstable	sentry-worker	0,0040	308	0,100	0,500	100	750	0,005	0,100	500	1.000
	burstable	sentry-cron	0,0014	96	0,100	0,200	100	200	0,002	0,100	100	200
	burstable	sentry	0,0079	347	0,300	0,300	300	500	0,150	0,300	400	800
	best efford	redis	0,0024	33	-	-	-	-	0,003	0,100	50	200
Portal	guaranteed	celery-beat-worke	0,0020	123	0,500	0,500	300	300	0,003	0,100	200	300
	guaranteed	web	0,0050	347	0,500	0,500	1.000	1.000	0,300	0,500	400	1.000
Owncloud	burstable	owncloud	0,0332	900	-	-	3.900	3.900	0,040	0,250	1.000	
	best efford	redis	0,0017	3	-	-	-	-	0,002	0,100	10	
Gitlab	burstable	gitlab	0,5000	3.896	1,000	2,000	4.000	6.000	0,500	1,000	4.000	6.000
Sonarqube	best efford	sonarqube	0,0077	1.471	-	-	-	-	0,008	0,100	1.500	
Mail	guaranteed	webmail	0,0000	85	0,050	0,050	100	100	0,001	0,050	100	200
Verdaccio	burstable	verdacio-verdacio	0,0000	69	0,500	0,500	128	256	0,030	0,100	100	200
Efs-Provisioner	best efford	efs-provisioner	0,0000	12	-	-	-	-	0,001	0,050	50	100
Cert-Manager	best efford	cert-manager	0,0022	31	-	-	-	-	0,003	0,100	50	100
				Total:	4,050		13.381		1,055		10.360	
				New percentage:					0,740		0,226	

Tabelle 4.3: Resource Optimierung: Shared-Cluster

Eidesstattliche Erklärung

Ich versichere, die von mir vorgelegte Arbeit selbständig verfasst zu haben.

Alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten oder nicht veröffentlichten Arbeiten anderer entnommen sind, habe ich als entnommen kenntlich gemacht. Sämtliche Quellen und Hilfsmittel, die ich für die Arbeit benutzt habe, sind angegeben.

Die Arbeit hat mit gleichem Inhalt bzw. in wesentlichen Teilen noch keiner anderen Prüfungsbehörde vorgelegen.

Köln, 18. Januar 2021

David Bajon