УНИВЕРЗИТЕТ „СВ. КИРИЛ И МЕТОДИЈ" ВО СКОПЈЕ

ФАКУЛТЕТ ЗА ИНФОРМАТИЧКИ НАУКИ
И КОМПЈУТЕРСКО ИНЖЕНЕРСТВО

Војдан Здравко Ќорвезироски

АНАЛИЗА И ДИЗАЈН НА ПОВЕЌЕНАМЕНСКА АРХИТЕКТУРА НА БЕЗСЕРВЕРСКИ ПЛАТФОРМИ ЗА ТРАНСПАРЕНТНИ ПРЕСМЕТКИ ВО ОБЛАК-РАБ ЕКОСИСТЕМОТ

Докторски труд

Скопје, 2025

УНИВЕРЗИТЕТ „СВ. КИРИЛ И МЕТОДИЈ” ВО СКОПЈЕ

ФАКУЛТЕТ ЗА ИНФОРМАТИЧКИ НАУКИ
И КОМПЈУТЕРСКО ИНЖЕНЕРСТВО

Војдан Здравко Ќорвезироски

АНАЛИЗА И ДИЗАЈН НА ПОВЕЌЕНАМЕНСКА АРХИТЕКТУРА НА БЕЗСЕРВЕРСКИ ПЛАТФОРМИ ЗА ТРАНСПАРЕНТНИ ПРЕСМЕТКИ ВО ОБЛАК-РАБ ЕКОСИСТЕМОТ

Докторски труд

Скопје, 2025

Докторанд:
Којдан Здравко Ќорвезироски

Тема:
Анализа и дизајн на повеќенаменска архитектура на безсерверски платформи за транспарентни пресметки во облак-раб екосистемот

Ментор:
проф. д-р Соња Филипоска
Факултет за информатички науки и компјутерско инженерство, УКИМ

Комисија за одбрана:
проф. д-р Марјан Гушев (претседател)
Факултет за информатички науки и компјутерско инженерство, УКИМ

проф. д-р Соња Филипоска
Факултет за информатички науки и компјутерско инженерство, УКИМ

проф. д-р Владимир Трајковиќ
Факултет за информатички науки и компјутерско инженерство, УКИМ

проф. д-р Анастас Мишев
Факултет за информатички науки и компјутерско инженерство, УКИМ

вонр. проф. д-р Павле Вулетиќ
Електротехнички факултет, Универзитет во Белград

Научна област:
Компјутерски науки и инженерство

Датум на одбрана:
(Ако датумот не е утврден, се додава дополнително рачно)

Војдан Здравко Ќорвезироски

АНАЛИЗА И ДИЗАЈН НА ПОВЕЌЕНАМЕНСКА АРХИТЕКТУРА НА БЕЗСЕРВЕРСКИ ПЛАТФОРМИ ЗА ТРАНСПАРЕНТНИ ПРЕСМЕТКИ ВО ОБЛАК-РАБ ЕКОСИСТЕМОТ

– А п с т р а к т –

Со постојаниот напредок на полето на софтверот и хардверот, компјутерите станаа неизбежен дел од секојдневниот живот како за лична, така и за деловна употреба. Нивната сè поголема компактност овозможи секоја индивидуа и претпријатие да поседува сопствен пресметковен капацитет, во форма на паметни телефони, персонални компјутери и сервери. Сепак, ваквата поставеност на нештата подразбира дека за зголемување на пресметковниот капацитет потребно е купување дополнителна, помодерна, а неретко и скапа опрема. Со воведувањето на парадигмата за пресметување на облак, целта е овие проблеми да станат минато и да се овозможи на корисниците и претпријатијата да го добијат потребниот пресметковен потенцијал во моментот кога тој им е потребен, налик на останатите комунални услуги во домот како струја или вода. Имајќи предвид дека облакот е апстрактен концепт, кој всушност само претставува инфраструктура во надлежност на некој друг, овозможи давателите на услуги да претставуваат сопствени пресметковни решенија. Ваквиот пристап има и негативна страна, изостанувањето на унифициран интерфејс за комуникација води до ситуација различни даватели на услуги да создаваат сопствени решенија кои не се компатибилни со оние останатите, ограничувајќи го изборот на корисниците. Дополнително, изнајмените ресурси во облакот, во општ случај, се наплаќаат без разлика дали тие се користат или не, што има дополнителни негативни финансиски импликации.

Безсерверското пресметување е нова пресметковна парадигма, моментално карактеристична за облакот, која има голем потенцијал да ја реализира визијата за пресметковниот капацитет како само уште една комунална услуга, притоа намалувајќи ги и финансиските трошоци. Спротивно на името, сервери сè уште постојат, но тие се целосно транспарентни, со што се овозможува едноставно поставување и користење софтвер, без грижа за инфраструктурата на која тој се извршува. Сепак, целосниот потенцијал на безсерверското пресметување е сè уште неискористен, бидејќи моменталните безсерверски решенија страдаат од несоодветни перформанси, ограничена компатибилност меѓу себе и се фокусирани на извршување на една локација – во облакот или на работ од мрежата, дополнително ограничувајќи ги сценаријата во кои може да се користи оваа парадигма.

Оваа докторска дисертација предлага нов пристап во безсерверското пресметување преку анализа и дизајн на повеќенаменска архитектура на безсерверски платформи за транспарентни пресметки во облак-раб екосистемот. Целта е да се овозможи создавање облак-раб екосистем сочинет од различни инфраструктури, на различни локации (облак и раб), поседувани од различни чинители, обединети во една целина која ги апстрахира овие детали од крајните корисници и одговара на нивните потреби. Во дисертацијата се воведува користење сосема нова извршна околина за безсерверски функции и се дефинираат напредни стратегии за меѓусебно поврзување хетерогени инфраструктури. Овие аспекти овозможуваат надминување на идентификуваните досегашни проблеми и играат улога на темели во архитектурата преку која се реализира визијата за транспарентни пресметки низ унифициран облак-раб екосистем.

Клучни зборови: пресметување во облак, пресметување на работ од мрежата, екосистем, безсерверско пресметување, функции како услуга, оркестрација, дистрибуирани системи.

Vojdan Zdravko Kjorveziroski

ANALYSIS AND DESIGN OF A MULTIPURPOSE ARCHITECTURE FOR SERVERLESS PLATFORMS AIMED AT TRANSPARENT EXECUTION IN THE EDGE-CLOUD CONTINUUM

– Abstract –

The continuous advancements both in terms of software and hardware have made computing devices an omnipresent aspect of daily life, both for personal and business use. Miniaturization of hardware components has allowed every individual and corporation to be responsible for their own computing capacity through the use of personal computers, smartphones, and servers. Nevertheless, this approach requires continuous procurement of new and expensive devices to keep up with the pace of increasing computing demands. With the advent of cloud computing, these problems should have been overcome, allowing computing capacity to be available on demand, without requiring the possession of expensive equipment, thus transforming computing power into just another utility, such as electricity or tap water. Considering that the cloud is an abstract concept which at its core represents a physical infrastructure managed by a third party, many cloud providers offer ready-made services to provide more user-friendly interfaces. This approach also has significant drawbacks, mainly in terms of limited to non-existent compatibility between services offered by different cloud providers, resulting in a severe vendor lock-in effect. Additionally, cloud resources, in most cases, are billed no matter whether they are actually used or remain idle, leading to negative financial ramifications.

Serverless computing is a new computing paradigm, with its most popular implementations currently limited to the cloud. Serverless computing has a real potential of realizing the vision of treating computing capacity just as another utility, while also reducing the overall cost, currently associated with other cloud services. Contrary to its name, servers are still involved, but they are completely transparent, allowing seamless provisioning and usage of software, without needing to take care of any infrastructure components. However, the full potential of serverless computing is still not completely realized, since many serverless solutions are hindered by subpar performance, limited interoperability, and restricted set of execution locations (either in the cloud or at the edge close to the end-users, but not both), resulting in a severe vendor lock-in.

This doctoral thesis proposes a novel approach to serverless computing through analysis and design of multipurpose architecture for serverless platforms aimed at transparent execution in the edge-cloud continuum. The end-goal is the creation of such an edge-cloud continuum, an environment comprised of heterogenous infrastructures placed in different locations (both in the cloud and at the edge), owned by independent third parties, yet unified through a common interface. This thesis introduces a completely new execution environment for serverless functions and describes advanced connectivity strategies, allowing trusted communication between infrastructures no matter their geographical location or network conditions. The proposed approaches overcome the identified open issues in the area of serverless computing and act as foundational pillars to the architecture through which the vision of transparent computations across a unified edge-cloud continuum can finally be realized.

Keywords: cloud computing, edge computing, continuum, serverless computing, function as a service, orchestration, distributed systems.

Посвета

За моите родители Здравко и Оливера и мојата сестра Мирна, без чијашто поддршка овие страници немаше да бидат возможни. За тоа и за уште многу друго им благодарам.

Изјавувам дека докторскиот труд го изработев самостојно, дека уредно ги цитирам сите користени извори и литература и дека трудот не е користен во рамките на други универзитетски студии или за стекнување на друго звање.

Војдан Ќорвезироски

Изјавувам дека електронската верзија на докторскиот труд е идентична со отпечатениот докторски труд.

Војдан Ќорвезироски

Содржина

1. Вовед ... 11
2. Преглед на литературата ... 19
2.1. Методологија на прегледот и класификациска рамка ... 21
2.1.1. Цел на прегледот ... 21
2.1.2. Класификациска рамка ... 22
2.1.3. Класификација преку примена на класификациската рамка ... 23
2.2. Преглед на темите од интерес ... 25
2.2.1. Ефикасност ... 25
2.2.2. Распределба на задачи ... 29
2.2.3. Платформи за безсерверско пресметување на работ од мрежата ... 32
2.2.4. Континуум ... 37
2.2.5. Безсерверски апликации ... 38
2.2.6. Тестирање перформанси ... 42
2.2.7. Безбедност, интегритет и регулативи кај безсерверското пресметување ... 45
2.2.8. Решенија со отворен код ... 47
2.3. Отворени прашања и насока на истражување ... 49
3. Анализа на постојни безсерверски платформи ... 50
3.1. Избор и адаптација на свита за безсерверски тестови ... 51
3.2. Анализа на еднојазлени безсерверски платформи на работ од мрежата ... 51
3.2.1. Избор на платформи за тестирање ... 51
3.2.2. Методологија на извршување тестови ... 53
3.2.3. Споредбена анализа ... 55
3.3. Анализа на повеќејазлени безсерверски платформи на работ од мрежата ... 60
3.3.1. Kubernetes како сеприсутен оркестратор и неговата врска со безсерверското пресметување . 60
3.3.2. Избор на безсерверски платформи за тестирање ... 62
3.3.3. Избор на Kubernetes дистрибуции за тестирање ... 62
3.3.4. Методологија на извршување тестови ... 63
3.3.5. Споредбена анализа ... 65
3.4. Ретроспекција кон резултатите ... 74
4. Алтернативни извршни околини за безсерверски функции ... 76
4.1. Подемот на WebAssembly и преминувањето на серверската страна ... 77
4.2. WebAssembly како нова безсерверска извршна околина ... 78
4.3. Интеграција помеѓу традиционални контејнерски околини и WebAssembly ... 79
4.3.1. Избор на WebAssembly извршни околини ... 79
4.3.2. Свита за тестирање WebAssembly безсерверски функции ... 80
4.3.3. Процес на извршување на тестовите ... 81
4.4. Испитување на перформансите во пракса ... 82
4.4.1. WasmEdge ... 82
4.4.2. Wasmtime ... 84
4.4.3. Wasmer ... 85
4.4.4. Стандарден пристап со контејнери ... 86

4.5. Избор на најсоодветна WebAssembly извршна околина за безсерверски функции 87
4.6. Отворени прашања 90
5. Оркестрација на WebAssembly безсерверски функции 92
5.1. Осврт кон постојните напори за WebAssembly оркестрација 92
5.1.1. WasmCloud 93
5.1.2. Spin 93
5.1.3. Можности за интеграција 94
5.2. Патот до унифицирано оркестрирање на безсерверски функции 94
5.2.1. Проширување на постојните извршни околини за контејнери 94
5.2.2. WebAssembly поддршка во Kubernetes 95
5.2.3. Стратегија за валидација на решението 97
5.3. Валидација на решението за оркестрација 99
5.3.1. Сериско извршување 99
5.3.2. Подигнување функции 101
5.3.3. Паралелно извршување 103
5.3.4. Големина на софтверски артефакти 104
5.4. Придобивки од примена на решението 105
5.4.1. WebAssembly како дел од MEC 106
5.4.2. MEC управувач со платформи 107
5.4.3. MEC управувач со инфраструктура за виртуелизација 107
5.4.4. MEC платформа 107
5.4.5. MEC апликации 107
5.5. Значењето на WASM оркестрацијата за облак-раб екосистемот 108
6. Транспарентно мрежно поврзување низ облак-раб екосистемот 109
6.1. Преглед на постојни технологии за виртуелни приватни мрежи и нивното место во облак-раб екосистемот 111
6.2. Анализа на VPN решенија за воспоставување екосистеми меѓу облакот и работ 112
6.2.1. Критериуми за избор на решенија 112
6.2.2. Стратегија за евалуација на кандидатите 116
6.2.3. Дефинирање тестови за симулација на облак-раб екосистем низ различни мрежи 120
6.3. Споредба на ефикасноста на VPN решенијата при обезбедување транспарентно поврзување во облак-раб екосистемот 122
6.3.1. TCP пропусен опсег 123
6.3.2. UDP пропусен опсег 125
6.3.3. Време на одговор при користење на програмскиот интерфејс на Kubernetes 126
6.4. Импликации врз воспоставувањето транспарентен облак-раб екосистем 127
6.5. Избор на најсоодветно решение 129
7. Предлог архитектура за транспарентни пресметки во облак-раб екосистемот 131
7.1. Врската меѓу пресметковните заедници и облак-раб екосистемот 133
7.2. Функционални барања за имплементација на облак-раб екосистемот 134
7.3. Дизајн на архитектура за облак-раб екосистемот 136
7.3.1. Вмрежување меѓу заедниците во екосистемот 136
7.3.2. Оркестрација меѓу заедниците и во рамките на екосистемот 137

7.3.3. Кластерирање во рамките на заедниците 138
7.3.4. Поддршка за различни извршни околини во рамките на заедниците 139
7.4. Карактеристики на предложената архитектура 140
7.5. Референтна имплементација на пресметковен екосистем врз различни облак-раб заедници 140
7.5.1. Kubernetes како решение за кластерирање 141
7.5.2. Извршни околини низ унифицираниот екосистем 142
7.5.3. Оркестрација низ унифицираниот екосистем 143
7.5.4. Унифицирано вмрежување 145
7.6. Валидација на референтната имплементација 146
7.7. Унифицираниот екосистем од аспект на крајните корисници 148
7.8. Дискусија 149
8. Заклучок 152
9. Користена литература 158

Листа на слики

Слика 1.1 Хронолошки развој на пресметковните технологии во последните децении ... 13
Слика 2.1 Класификациска рамка сочинета од теми и соодветни категории ... 22
Слика 2.2 Најпопуларни безсерверски платформи со отворен код ... 48
Слика 3.1 Споредба на перформансите при сериско извршување, едно барање во даден момент ... 55
Слика 3.2 Споредба на перформансите при паралелно извршување, со променлив број на истовремени извршувања ... 56
Слика 3.3 Број на истовремени процеси за опслужување барања при паралелно извршување ... 58
Слика 3.4 Споредба на перформансите за извршување кај различните режими на работа на FaasD ... 59
Слика 3.5 Доцнење при ладен старт и сериско извршување кај AWS Greengrass ... 60
Слика 3.6 Просечно доцнење во секунди при ладен старт на контејнери за секоја од трите тестирани платформи ... 66
Слика 3.7 Алтернативен приказ на доцнењето при ладен старт за секоја од платформите ... 67
Слика 3.8 Просечно доцнење при ладен старт за секоја од функциите при различните платформи ... 67
Слика 3.9 Вкупен број на извршувања за секоја функција во период од 5 минути ... 68
Слика 3.10 Просечно време на одговор за секоја функција при сериско извршување ... 69
Слика 3.11 Вкупно време потребно за опслужување 20 барања користејќи 1 инстанца од функцијата ... 70
Слика 3.12 Број на паралелни барања опслужени од една инстанца на контејнер ... 71
Слика 3.13 Просечно време на одговор при паралелни повици ... 73
Слика 3.14 Промена во бројот на инстанци во зависност од бројот на барања во секунда ... 74
Слика 4.1 Големина (во килобајти) на секоја од контејнерските слики за тестирање на WasmEdge (интерпретација наспрема AOT) ... 83
Слика 4.2 Големина (во килобајти) на секоја од контејнерските слики за тестирање на Wasmtime (JIT наспрема AOT) ... 85
Слика 4.3 Големина (во килобајти) на секоја од контејнерските слики за тестирање на Wasner (JIT наспрема AOT) ... 86
Слика 4.4 Големина (во килобајти) на секоја од контејнерските слики за тестирање на distroless и Debian:bullseye ... 87
Слика 4.5 Споредба на средното време на AOT извршување меѓу петте поддржани извршни околини (WE-WasmEdge; WT-Wasmtime; WR-Wasmer; DL-distroless; DB-debian:bullseye) ... 88
Слика 4.6 Споредба на средното време на интерпретација (WasmEdge) и JIT (Wasmtime, Wasmer) меѓу трите поддржани извршни околини (WE-WasmEdge; WT-Wasmtime; WR-Wasmer) ... 89
Слика 5.1 Процес на поставување нова WASM безсерверска функција ... 96
Слика 5.2 Споредба на средното време на одговор (во секунди) кај WASM (Spin) и OpenFaaS ... 100
Слика 5.3 Време на одговор (во секунди) при различни сериски извршувања ... 100
Слика 5.4 Споредба на средното време за подигнување функции помеѓу WASM (Spin) и OpenFaaS ... 102
Слика 5.5 Споредба на средното време на одговор при првото повикување на функциите по нивното подигнување ... 102
Слика 5.6 Средно време на одговор при паралелно извршување на функциите ... 103
Слика 5.7 Време на одговор (во секунди) при различни паралелни извршувања ... 104
Слика 5.8 Предлог мапирање на софтверски алатки во различни компоненти од MEC архитектурата ... 106
Слика 6.1 Споредба помеѓу TCP пропусниот опсег и искористеноста на процесорот за секое од сценаријата ... 123
Слика 6.2 Споредба помеѓу UDP пропусниот опсег и искористеноста на процесорот за секое од сценаријата ... 125
Слика 6.3 Средно време на одговор за Kubernetes API серверот при користење на различните VPN решенија ... 126
Слика 6.4 Средни вредности за пропусниот опсег при користење TCP ... 127
Слика 6.5 Средни вредности за пропусниот опсег при користење UDP ... 128
Слика 7.1 Приказ на хиерархиските врски меѓу различните компоненти од архитектурата ... 134
Слика 7.2 Главни столбови во архитектурата за унифициран облак-раб екосистем ... 136

Листа на табели

Табела 1.1 Преглед на организацијата на остатокот од дисертацијата 18
Табела 2.1 Класификација на трудови кои се осврнуваат на само една тема од рамката 24
Табела 2.2 Класификација на трудови кои се осврнуваат на повеќе од една тема од рамката 24
Табела 2.3 Трудови поврзани со темата „Ефикасност“ 26
Табела 2.4 Трудови поврзани со темата „Распределба на задачи“ 29
Табела 2.5 Карактеристики на безсерверски платформи 32
Табела 2.6 Трудови поврзани со темата „Континуум“ 37
Табела 2.7 Трудови поврзани со темата „Безсерверски апликации“ 38
Табела 2.8 Трудови поврзани со темата „Тестирање перформанси“ 43
Табела 2.9 Трудови поврзани со темата „Безбедност, интегритет и регулативи кај безсерверското пресметување“ 45
Табела 2.10 Трудови поврзани со темата „Решенија со отворен код“ 47
Табела 2.11 Анализа на користените безсерверски платформи со отворен код 49
Табела 3.1 Тест параметри и нивен опис 54
Табела 3.2 Информации за околината за тестирање 63
Табела 3.3 Параметри за извршување на секој од тестовите 64
Табела 4.1 Споредба меѓу разгледуваните WebAssembly извршни околини 80
Табела 4.2 Опис на функциите дел од свитата за тестирање 80
Табела 4.3 Споредба помеѓу времето на извршување при интерпретација и AOT компајлирање кај WasmEdge 83
Табела 4.4 Споредба помеѓу времето на извршување при JIT и AOT компајлирање кај Wasmtime 84
Табела 4.5 Споредба помеѓу времето на извршување при JIT и AOT компајлирање кај Wasmer 85
Табела 4.6 Споредба помеѓу времето на извршување на функциите при користење на distroless и debian:bullseye контејнерските слики како основа 87
Табела 5.1 Информации за користената свита на тестови 98
Табела 5.2 Споредба на стандардната девијација и коефициентот на варијација при сериско извршување 101
Табела 5.3 Споредба на стандардната девијација и коефициентот на варијација при паралелно извршување 104
Табела 5.4 Споредба на големината на OCI сликите за WASM и OpenFaaS (во килобајти) 105
Табела 6.1 Споредба на VPN решенија и нивна класификација во однос на поставените критериуми ... 115
Табела 7.1 Информации за користената инфраструктура 147
Табела 7.2 Информации за околината за евалуација 148

1. ВОВЕД

Со еволуцијата на личното пресметување и отворањето на интернетот кон пошироката јавност во последните децении од 20 век, компјутерите станаа неизбежен, а и незаменлив дел од секојдневието. Достапниот пресметковен капацитет овозможи сосема нови можности, како за претпријатијата така и за индивидуалните корисници. Со ова широко се отворија вратите на дигитализацијата, овозможувајќи им на компаниите да поставуваат сопствена инфраструктура која ја користат за истражување и подобрување на нивните продукти, водење на претпријатието, но и интеракција со надворешниот свет преку новите комуникациски алатки. Ваквиот развој доведе и до демократизација на пресметковниот капацитет, каде што тој повеќе не е луксуз само за одредени привилегирани групи, туку е достапен и за индивидуалните корисници преку личните компјутери. Сепак, желбата за сè поголеми пресметковни можности, со овој пристап, значи купување на сè повеќе и повеќе опрема, чиешто ефикасно користење е далеку од осигурано и всушност зависи од моменталните потреби кои драстично варираат со времето.

Следниот чекор од оваа еволуција, овозможен од постојаниот напредок во развојот на хардверот и компјутерските мрежи е појавувањето целосно нова парадигма во пресметувањето, пресметувањето во облак. На овој начин, во првата деценија од 21 век, уште еднаш комплетно се промени перцепцијата за компјутерските пресметки, овозможувајќи процесирачката моќ, а и самиот процес на обработка на податоци да се разгледува само како уште една услуга достапна во секој момент, без ограничување, налик на традиционалните комунални услуги во домот [1]. Секој има на располагање токму онолку пресметковен капацитет колку што е потребен во моментот. Со текот на времето, приватниот сектор, индустријата, но и академската заедница го дадоа својот придонес на ова поле преку развивање најразлични приватни, јавни, споделени и хибридни облаци [2]. Сепак, за да им се овозможи и на обичните корисници да уживаат во придобивките на облакот, стана јасно дека ќе биде потребно воведување нивоа на апстракција. Поедноставувањето на работата со првичните облаци се појави во форма на три понуди, популаризирани во втората деценија од 21 век: инфраструктура како услуга (англ. infrastructure as a service, IaaS), платформа како услуга (англ. platform as a service, PaaS) и софтвер како услуга (англ. software as a service, SaaS) [3]. IaaS го претставува најниското ниво на апстракција, овозможувајќи им на корисниците да изнајмат складиште (англ. storage), пресметковен и/или мрежен капацитет, по потреба, најчесто во форма на виртуелни машини кои потоа може да бидат користени за најразлична намена и градење сопствена инфраструктура. PaaS оди чекор понатаму и е првично наменет за програмерите, елиминирајќи ја потребата тие самите да се грижат за одржувањето на инфраструктурата (инсталација, ажурирање, закрпување). Наместо тоа, PaaS нуди готови алатки кои директно може да се искористат при процесот на развој на софтвер. Популарни примери за алатки кои може да се понудат во форма на PaaS денес се: бази на податоци, услуги за известувања, редици за размена на пораки (англ. message queues) и др. На крајот, SaaS, којшто примарно е насочен кон крајните корисници, го нуди највисокото ниво на апстракција. Во овој случај, услугата на располагање е комплетен производ, спремен за директно користење од страна на крајните корисници, без притоа да има каква било потреба од самостојно одржување, ажурирање, закрпување или, пак, управување. Сите овие аспекти се во надлежност на самиот давател на услуги.

Претходно споменатите IaaS, PaaS и SaaS, иако најпопуларни, сепак не се единствените вакви понуди денес. Идејата за затскривање на потенцијално комплицираните задачи од крајните корисници се покажа како многу успешна и доведе до појавата на * (сè) како

услуга [4]. Овој воопштен термин се однесува на услуги кои го ослободуваат крајниот корисник од извршување на некоја несакана и макотрпна задача, едноставно предавајќи ја одговорноста за неа на професионален давател на услуги во облакот. Главните придобивки од ваквиот пристап се зголемената леснотија при управување и намалувањето на потребното време за пласирање нов продукт на пазарот, како резултат на брзото, но и квалитетно завршување на заднинските задачи од страна на третите даватели на услуги.

Од перспектива на очекуваниот надоместок за сè поприсутните * како услуга понуди, најчесто мерењето на потрошувачката се заснова на многу мали временски интервали, како на пример на ниво на час, минута или, пак, во последно време дури и секунда. Иако ова оди во прилог на крајните корисници, сепак, се плаќа дури и за времето кога дадената пресметковна задача (англ. workload) не извршува конкретна работа, т.е. не опслужува корисници.

Поедноставување на процесот на развој на нови функционалности за програмерите, но и надминување на проблемите во поглед на неефикасноста може да се реализира со помош на безсерверското пресметување (англ. serverless computing[1]). Безсерверското пресметување е релативно нова парадигма која своите зачетоци ги има во средината на втората деценија од 21 век, со зголемено прифаќање на почетокот од третата (од 2020 година па до денес). Покрај навидум невообичаеното име на ваквиот нов пристап, се разбира, сè уште се користат класични сервери под нечија надлежност. Формално, безсерверското пресметување се дефинира како симбиоза од функција како услуга (англ. function as a service, FaaS) и заднина како услуга (англ. backend as a service, BaaS) [5]. Со прифаќање на безсерверското пресметување, доаѓа до преформулирање на една од класичните дилеми на програмерите, т.е. повеќе не станува збор за тоа „каде да се постави решението“, туку за тоа „кои функционалности да ги има решението“. На овој начин, тие можат да се посветат на она што го прават најдобро, пишување код и воведување нови можности во софтверот. Споредено со останатите како услуга понуди кои се во широка употреба, безсерверското пресметување нуди уште повисоко ниво на апстракција, грижејќи се за комплексните аспекти како што се: поставувањето на инфраструктурата, управувањето со ресурси, распределбата на задачите, па дури и хоризонталното и вертикално скалирање. Комбинирањето на ваквите FaaS можности од една страна заедно со BaaS услугите кои ги нудат често користените заднински функционалности од друга страна, им овозможува на развивачите на софтвер целосно да заборават на серверите, правејќи ја безсерверската апстракција реалност. Дополнително, со поединечно наплаќање за секое извршување на дадена функција и можноста да се намалат активните инстанци до 0 во случај кога нема дојдовни барања, се овозможува елиминирање на трошоците за временските периоди кога софтверот не обработува никакви податоци. Ваквиот пристап е од особен интерес не само за крајните корисници, туку и за давателите на услуги, кои така можат поефикасно да опслужуваат повеќе задачи, користејќи ги истиот хардвер и инфраструктура.

Првите јавни FaaS понуди се појавија на отворениот пазар во 2015 година, со претставувањето на Lambda услугата од страна на Amazon [6], првично ориентиран кон веб програмерите. Инспирирани од брзиот успех и здобиената популарност, за кратко време се појавија дополнителни решенија [7], [8], [9]. FaaS решенијата со отворен код се

[1] Дополнителен превод на англискиот термин „serverless computing“, покрај „безсерверско пресметување“ е и „пресметување без грижа за сервери“, термин кој исто така се среќава во литературата. Иако еквивалентни изрази, во продолжение на дисертацијата ќе се користи исклучиво „безсерверско пресметување“, со цел осигурување конзистентност.

исто така широко распространети денес [10], [11], [12], до тој степен што одредени комерцијални даватели на услуги се имаат одлучено да ги користат како основа за развој на сопствен FaaS продукт, како што е случајот со OpenWhisk и IBM [13]. Друг позитивен пример е и публикувањето на готови решенија со отворен код [14], овозможувајќи корист на поширокатa заедница, а не само непосредните клиенти. Слика 1.1 го визуелизира хронолошкиот развој на различните пресметковни технологии низ последните децении.

Слика 1.1 Хронолошки развој на пресметковните технологии во последните децении

Иако безсерверското пресметување своите зачетоци ги влече од областа на развојот на веб апликациите, ова не е единственото поле каде што тоа се применува. Интернетот на нештата (англ. Internet of Things, IoT) е друг концепт со енормен пораст во популарност последниве години. Примената на безсерверското пресметување при пресметки поврзани со IoT е особено корисно, земајќи ги предвид задачите базирани на настани во реално време (англ. real-time event-based workloads) карактеристични за ваквите средини [15], [16]. За целосна имплементација на оваа визија, потребно е да се разрешат новите предизвици кои се јавуваат како резултат на тоа што IoT околините имаат поразлични побарувања од централизираниот клиент-сервер модел карактеристичен за класични сценарија. Иако облакот постојано се применува и при работа со IoT, сепак, пренесувањето на податоците од изворните уреди до оддалечените сервери за обработка во многу краток временски рок отсекогаш важело за предизвик. Со други зборови, облакот е одлично решение за апликации кои треба да бидат користени од луѓе и каде времето на добивање одговор во опсегот од стотици милисекунди, па и неколку секунди, е прифатливо. Но, кога станува збор за IoT апликации кои работат во реално време и каде е потребна комуникација од типот машина-со-машина, неопходно е намалување на времето на одговор [17]. Проблемите од аспект на големото доцнење (англ. latency) и бавното време на одговор ќе бидат уште поизразени во иднината, како што сè повеќе и повеќе IoT уреди ќе станат дел од домовите и индустријата.

Потенцијалното решение, околу кое се има оформено значаен интерес од страна на академската заедница, е поместувањето на пресметковниот капацитет поблиску до изворот на самите податоци. Пресметувањето на работ од мрежата (англ. edge computing) има за цел да го намали мрежното доцнење преку поставување пресметковна инфраструктура блиску до изворните уреди, нудејќи брз пренос и обработка на временски зависни податоци. Ваквиот пристап има потенцијал да овозможи уште повеќе придобивки за временски чувствителни IoT пресметки, преку примената на безсерверското пресметување на работ од мрежата. На овој начин, класичната парадигма „испрати ги податоците до кодот“, која се карактеризираше со неприфатливо доцнење за временски осетливи задачи и големи трошоци за пренос на податоците се трансформира во

„испрати го кодот до податоците“ [18]. Со ова се нудат и дополнителни придобивки, како што е зголемување на приватноста на крајните корисници, што е и едно од барањата во најновите правни регулативи од ова поле [19].

Голем број на даватели на услуги ги имаат прилагодено сопствените портфолија и ги имаат проширено оригиналните безсерверски продукти да работат и на работ од мрежата, како на пример AWS Greengrass [20] и Azure IoT Hub [21]. Достапни се и значителен број на решенија со отворен код изградени како адаптација на постојни решенија за работ од мрежата или, пак, развиени како целосно нови платформи наменети исклучиво за ова сценарио.

Облакот и работ од мрежата не може да се гледаат како издвоени и независни локации за пресметување. Во основата станува збор за една комплексна релација каде што и двете парадигми, заедно, градат целосно нов инфраструктурен екосистем кој ќе овозможи примена на комплетно нови технологии и кориснички сценарија во иднината. Како и во секој комплексен екосистем составен од голем број зависни компоненти, така и во овој случај се поставуваат значајни прашања околу координацијата. Еден голем страв кој постепено станува и реалност е креирањето изолирани островчиња од поединечни помали екосистеми овозможени од страна на давателите на услуги, опфаќајќи голем број на услуги, но ограничувајќи ја интероперабилноста со останати решенија, што негативно влијае врз конкурентноста. Цената за ваквото однесување, како во фигуративна, така и во буквална смисла, ја плаќаат крајните корисници за кои миграцијата од една кон друга инфраструктура е макотрпна, скапа, па дури и неизвесна поради немањето унифициран интерфејс за апликативни програми (англ. application programming interface, API). Потребна е фокусирана работа за избегнување на ваквите трендови преку дефинирање отворени интерфејси за меѓусебна комуникација меѓу различните продукти и услуги, без разлика на инфраструктурата и нејзината локација.

Дополнително прашање кое неизбежно се поставува при дискусијата за поврзување на различни пресметковни локации е стратегијата за вмрежување. Администраторите и систем архитектите, во отсуство на формално дефинирана рамка, се препуштени сами на себе да обезбедат безбедно и робусно решение кое истовремено би ги задоволило потребите за изолација и големи податочни брзини.

Осврнувајќи се на пресметковниот капацитет достапен на различните локации, на работ од мрежата и на облакот, од интерес е дали едноставно пресликување на типичната архитектура на облакот може да се постигнат посакуваните резултати и на работ од мрежата. Од есенцијално значење е сознанието дека постојните решенија кои го уживаат удопството на облакот во поглед на количеството ресурси и брзи мрежни врски не работат соодветно и на далеку поограничена инфраструктура, при нивното поставување на работ од мрежата. Неопходно е разгледување и воведување алтернативни решенија, чиишто оптимизации ќе ја отсликуваат очекуваната околина за работа. Сепак, ваквиот пат е трнлив и може лесно да се скршне во насока каде што различните стратегии за работ и облакот би станале премногу меѓусебно различни, без можност за нивно обединување. За избегнување вакво сценарио, треба уште од самиот почеток да се има стремеж кон тесна интеграција преку единствен интерфејс за комуникација. Ваквата интеграција би овозможила унифицирано контролирање на задачите, криејќи ги спецификите за хардверот, локацијата на извршување или користените технологии.

Земајќи ја предвид дискутираната потреба од поширока компатибилност и обединувањето на работ од мрежата со облакот во тековно фрагментираната околина на ова поле, се доаѓа до едно главно прашање: Дали е возможно креирање платформа која

би ги надминала досегашните ограничувања и би овозможила транспарентни пресметки во облак-раб екосистемот, обединувајќи различни инфраструктури и даватели на услуги во процесот. Оттука произлегува и првата хипотеза на оваа дисертација:

- *Хипотеза 1: Со помош на последните научни достигнувања во релевантните области и преку нивно комбинирање во заедничка целина се овозможува имплементација на базична функционална платформа за пресметување која би ги надминала ограничувањата на досегашните алтернативни решенија.*

Се разбира, ваквата платформа би требало да понуди решение за асиметричноста на достапните ресурси во облакот од една страна и работ на мрежата од друга преку специјализирани и адаптирани пресметковни решенија кои би работеле на пониско ниво. На овие аспекти директно се надоврзува втората хипотеза:

- *Хипотеза 2: Тековните извршни околини кои се користат за покренување безсерверски функции не ги задоволуваат барањата од аспект на брзина на извршување и доцнење при нивната примена на работ од мрежата. Потребна е оптимизација на постојните или примена на целосно нови алтернативни извршни околини за надминување на овие проблеми, што би довело и до задоволување на барањата кои ги поставуваат IoT уредите со своите пресметковни задачи засновани на настани.*

Користењето соодветни извршни околини за различните локации не го решава проблемот на фрагментација, специфичен за големите дистрибуирани инфраструктури. На крајот од краиштата, без унифицирано вмрежување, работ на мрежата и облакот се сè уште само изолирани острови. Потребно е нивно робусно поврзување, занемарувајќи ги специфичните аспекти од локалната мрежна архитектура или географската поставеност. Решението мора да биде изградено на принципот на еластичност, овозможувајќи лесно додавање, но и намалување на бројот на вклучени уреди во мрежата, на што се надоврзува и третата хипотеза:

- *Хипотеза 3: Новите решенија за воспоставување на надмрежи може да се користат за создавање на ефикасно меѓуповрзување на целосно различни пресметковни околини, како од аспект на нивните перформанси, така и во однос на нивната географска поставеност. На овој начин се овозможува креирање целосно ново ниво на апстракција над физичките мрежи, што е неопходен чекор за воспоставување на посакуваниот облак-раб екосистем.*

Имајќи го предвид импресивниот број на пресметковни инфраструктури поставени на различни географски локации и изградени врз специфични технологии кои треба да се меѓусебно поврзани, неопходно е повисоко ниво на апстракција кое ќе се погрижи за затскривање на различностите. Аспекти како: кои извршни околини се достапни, на кој начин треба да се врши интеракција, на која локација се наоѓаат; треба да се целосно транспарентни за крајните корисници преку унифицирано API. Суштински е тоа што поради различната природа на инволвираните инфраструктури, не се очекува една извршна околина да ги задоволи сите потреби, па потребна е нивна соодветна комбинација. Хипотезата 4 се осврнува токму на овој аспект:

- *Хипотеза 4: Виртуелните машини и контејнерите како извршни околини може да се комбинираат заедно со нови и поефикасни алтернативи кои се подобро прилагодени за работ на мрежата со цел креирање една унифицирана платформа која ќе понуди флексибилност преку избор на најсоодветната технологија за извршувањето на дадена задача.*

Истражувањето спроведено во насока на претходните хипотези води кон севкупна анализа и дизајн на повеќенаменска архитектура на безсерверски платформи за транспарентни пресметки во облак-раб екосистемот, нудејќи отворен каталог на алатки и апликации. Ова е всушност и последната хипотеза на дисертацијата:

- *Хипотеза 5: Со комбинација на нови и постојни софтверски решенија со отворен код може да се изгради отворена платформа за пресметување која не само што ќе овозможи пресметување на различни локации користејќи различни извршни околини, туку и ќе промовира реискористување на постојното знаење преку отворен каталог на веќе дефинирани алатки и апликации.*

Дисертацијата е организирана на следниов начин: во второто поглавје се врши преглед на актуелната литература, со цел идентификување на актуелните трендови и отворени прашања од полето на безсерверското пресметување на работ од мрежата. Посебно внимание се обрнува на концептот на интернет на нештата и потенцијалното влијание на безсерверското пресметување во иднина. Поглавјето 3 е посветено на компаративна споредба на постојни комерцијални и отворени еднојазлени и повеќејазлени безсерверски платформи, утврдувајќи ги нивните сличности, разлики, но и перформансни карактеристики. Надоврзувајќи се на добиените резултати, поглавјето 4 ја разгледува идејата за воведување нови извршни околини за извршување на безсерверски функции, претставувајќи го WebAssembly како соодветно решение. Имајќи ја предвид неопходната интеграција со постојните околини, се дискутираат пристапи за интеграција помеѓу WebAssembly и традиционалните контејнери. Апстрахирањето како на контејнерите, така и на WebAssembly под капата на еден ист интерфејс за апликативни програми ја отвора вратата и кон решавање на проблемот на оркестрација на безсерверските функции, особено оние претставени како WASM модули, што е и фокус на поглавјето 5. Во оваа насока се врши адаптација на популарниот Kubernetes оркестратор, воведувајќи можност за покренување безсерверски функции во форма на контејнери и/или WebAssembly модули. Поглавјето 6 се осврнува на проблемот за безбедно, брзо и робусно поврзување на оддалечени инфраструктури, анализирајќи го потенцијалот на новите решенија за виртуелни приватни мрежи кои нудат сестрано поврзување да бидат употребени како унифицирачки подмрежи низ екосистемот. Искористувајќи ги резултатите и заклучоците од претходните поглавја, поглавјето 7 ја претставува комплетната архитектура на мултифункционалната отворена и ефикасна платформа која ќе се протега низ повеќе локации во мрежата, нудејќи униформно API за интеракција. Дополнително, во истото поглавје се врши и осврт на практичната примена на решението, дискутирајќи начини за интеграција со трети, постоечки системи, потврдувајќи ја неговата флексибилност, но и олеснувајќи го користењето од страна на трети корисници преку каталог на достапни безсерверски функции. Дисертацијата завршува со поглавјето 8 каде што се изнесуваат заклучоците и се резимира целокупната нејзина содржина, осврнувајќи се на најважните научни придонеси.

Имајќи ги предвид поставените хипотези на почетокот од дисертацијата и опсежната дискусија понудена низ нејзините различни поглавја, табела 1.1 во продолжение врши поврзување меѓу поединечните хипотези, поглавјата од интерес, како и сопствените публикациите во меѓународни списанија или конференции кои произлегле од соодветното истражување.

Главните придонеси на дисертацијата се следни:

- Со систематско мапирање на релевантната литература, се врши идентификување на последните трендови, решенија и недостатоци од областа на безсерверското

пресметување на работ од мрежата и во облакот, со посебен осврт кон интернетот на нештата [22], [23]. Се изготвува посветена класификациска рамка со конкретни критериуми од областа на перформансите, безбедноста, скалабилноста, можноста за примена на решенијата низ различни околини и начинот на распоредување на задачите. Ваквото систематско мапирање е од особена корист за запознавање со концептот на безсерверското пресметување за нови истражувачи заинтересирани за ова поле, но исто така и за идентификување на последните достигнувања, како и вреднување на останати безсерверски решенија преку претставената класификациска рамка.

- Преку спроведување експерименти и вршење детални емпириски мерења се прави компаративна анализа на постојни безсерверски платформи наменети за работ од мрежата и за облакот [24], [25]. Со цел поверодостојни резултати, се врши дистинкција помеѓу еднојазлени и повеќејазлени безсерверски платформи и тие се анализираат независно, со унифицирана свита од тестови. Дополнително, се обрнува внимание и на аспектот како оркестраторот врз кој е подигнато безсерверското решение влијае врз крајните перформанси, наместо да се разгледува исклучиво безсерверската платформа. Претставената свита на тестови е реупотреблива и може да се користи за тестирање на останати безсерверски платформи во иднина.
- Со симулација на дистрибуирана пресметковна инфраструктура која се протега на работ од мрежата и во облакот, се врши тестирање на перформансите на модерни решенија за воспоставување виртуелни приватни мрежи [26], [27]. Се дефинира прецизна методологија за спроведување на тестовите, како и класификациска рамка за вреднување на постојни и идни софтвери кои може да се користат за сестрано поврзување на уреди од различни локации.
- Претставување на WebAssembly како алтернативна извршна околина за безсерверски функции и потврдување на нејзината ефикасност преку емпириско тестирање користејќи ја претходно дефинираната методологија и свита на тестови, со што се овозможува компаративна анализа меѓу различните технологии и извршни околина. Преку дизајн на нова софтверска компонента, се овозможува интеграција на WebAssembly со останати извршни околини, со цел унифицирано управување и подигнување безсерверски функции преку единствено оркестрациско решение [28], [29], [30], [31], [32].
- Анализа и дизајн на повеќенаменска архитектура на безсерверски платформи за транспарентни пресметки во облак-раб екосистемот која ги надминува недостатоците на досегашните решенија, со можност за поставување како низ нови, така и низ постоечки околини. Евалуација на архитектурата во пракса преку референтна имплементација и тестирање со реални сценарија и апликации [33], [34].

Табела 1.1

Преглед на организацијата на остатокот од дисертацијата

Хипотеза	Поглавје	Публикации
Хипотеза 1	1. Вовед 2. Преглед на литературата	• V. Kjorveziroski, S. Filiposka, and V. Trajkovik, 'IoT Serverless Computing at the Edge: A Systematic Mapping Review', Computers, vol. 10, no. 10, Art. no. 10, Oct. 2021, doi: 10.3390/computers10100130. • V. Kjorveziroski et al., 'IoT Serverless Computing at the Edge: Open Issues and Research Direction', Transactions on Networks and Communications, vol. 9, no. 4, Art. no. 4, Dec. 2021, doi: 10.14738/tnc.94.11231.
Хипотеза 2	3. Анализа на постојни безсерверски платформи	• V. Kjorveziroski, S. Filiposka, and V. Trajkovik, 'Serverless Platforms Performance Evaluation at the Network Edge', in ICT Innovations 2021. Digital Transformation, in Communications in Computer and Information Science. Cham: Springer International Publishing, 2022, pp. 160–172. doi: 10.1007/978-3-031-04206-5_12. • V. Kjorveziroski and S. Filiposka, 'Kubernetes distributions for the edge: serverless performance evaluation', J Supercomput, vol. 78, no. 11, pp. 13728–13755, Jul. 2022, doi: 10.1007/s11227-022-04430-6.
Хипотеза 3	6. Транспарентно мрежно поврзување низ облак-раб екосистемот	• V. Kjorveziroski, C. Bernad, K. Gilly, and S. Filiposka, 'Full-mesh VPN performance evaluation for a secure edge-cloud continuum', Software: Practice and Experience, vol. n/a, no. n/a, doi: 10.1002/spe.3329. • V. Kjorveziroski, A. Mishev, and S. Filiposka, 'Evaluating IPv6 Support in Kubernetes', in 2021 29th Telecommunications Forum (TELFOR), Belgrade, Serbia: IEEE, Nov. 2021, pp. 1–4. doi: 10.1109/TELFOR52709.2021.9653276.
Хипотеза 4	4. Алтернативни извршни околини за безсерверски функции 5. Оркестрација на WebAssembly безсерверски функции	• V. Kjorveziroski and S. Filiposka, 'WebAssembly as an Enabler for Next Generation Serverless Computing', J Grid Computing, vol. 21, no. 3, p. 34, Jun. 2023, doi: 10.1007/s10723-023-09669-8. • V. Kjorveziroski and S. Filiposka, 'WebAssembly Orchestration in the Context of Serverless Computing', J Netw Syst Manage, vol. 31, no. 3, p. 62, Jul. 2023, doi: 10.1007/s10922-023-09753-0. • V. Kjorveziroski, S. Filiposka, and A. Mishev, 'Evaluating WebAssembly for Orchestrated Deployment of Serverless Functions', in 2022 30th Telecommunications Forum (TELFOR), Nov. 2022, pp. 1–4. doi: 10.1109/TELFOR56187.2022.9983733. • V. Kjorveziroski, C. B. Canto, K. Gilly, and S. Filiposka, 'Implementing Multi-Access Edge Computing with Kubernetes', presented at the 14th ICT Innovations Conference, Skopje, North Macedonia, Sep. 2022. • V. Kjorveziroski, 'Framework for a Multipurpose Remotely Accessible Laboratory for Education', presented at the 19th International Conference on Informatics and Information Technologies, Skopje, North Macedonia, May 2022.
Хипотеза 5	7. Предлог архитектура за транспарентни пресметки во облак-раб екосистемот	• V. Kjorveziroski and S. Filiposka, 'Federated architecture for serverless platforms aimed at transparent execution in the edge-cloud continuum', IJCC, vol. 14, no. 1, pp. 115–144, 2025, doi: 10.1504/IJCC.2025.145664. • V. Kjorveziroski, P. V. Vuletic, Ł. Łopatowski, and F. Loui, 'On-Demand Network Management with NMaaS: Network Management as a Service', in NOMS 2022-2022 IEEE/IFIP Network Operations and Management Symposium, Apr. 2022, pp. 1–3. doi: 10.1109/NOMS54207.2022.9789815.

2. ПРЕГЛЕД НА ЛИТЕРАТУРАТА

Целта на ова поглавје е преглед на сродните истражувања во врска со употребата на безсерверското пресметување за пресметковни задачи низ облакот и работ од мрежата со посебен фокус на интернетот на нештата, како и идентификација на сè уште отворените прашања околу воспоставување облак-раб екосистеми. Низ текстот се користи општо прифатената дефиниција за безсерверското пресметување, т.е. комбинацијата од функција како услуга и заднина како услуга, наведена во воведот. Кога станува збор за приближување кон работ, се користи терминот безсерверско пресметување на работ од мрежата за карактеризирање задачи со можност за извршување на инфраструктура во близина на уредите што ги генерираат податоците или директно на самите уреди.

Безсерверското пресметување претставува активно поле на истражување кое привлекува значителен интерес во последните неколку години, а доказ за тоа е големиот број на примарна и секундарна литература објавена досега. Најголемиот дел од трудовите во оваа област се фокусираат на истражувања направени во облакот, класифицирајќи ги безсерверските инфраструктури како нова технологија која може да има значајно влијание врз многу останати полиња и кориснички сценарија, во иднината. За илустрација, во продолжение се претставени неколку позначајни и иновативни примери за употреба на безсерверските инфраструктури, опишани во актуелни научни трудови.

Според Варгесе и др. [35], со дополнителен развој на безсерверската парадигма, таа може да се претвори во соодветна алтернатива за голем број апликации, заменувајќи ги класичните инфраструктури. IoT исто така се идентификува како поле со голем потенцијал за безсерверските технологии, имајќи предвид дека во овој случај пресметките се првенствено базирани на настани. Авторите на [15] го споделуваат истото гледиште и ја претставуваат сопствената визија за безсерверското пресметување како главен двигател на понатамошниот развој на сензорските мрежи на работ од мрежата, во тандем со блоковските вериги (англ. blockchain) и вештачката интелигенција (англ. artificial intelligence, AI). Големата применливост и корисност на безсерверската парадигма е очигледна веќе и денес, со широката распространетост на безсерверски JavaScript функции како значаен дел од модерното веб програмирање. Тие овозможуваат брзо време на одговор до корисници низ целата планета [36] како резултат на нивното извршување на инфраструктура во облак или на работ од мрежата. Дополнителни области за примена на оваа технологија анализираат Шафиј и др. [37] и Хасан и др. [38], вклучително и сценарија за соработка во реално време, податочни анализи, видео обработка, изведување научни пресметки, сервирање модели од машинското учење, паметни градски инфраструктури и чет-ботови.

Преку искористување на можноста за лесно скалирање на задачите, безсерверското пресметување може да најде примена и во обработката на податоци на барање (англ. on-demand data processing) и извршување пресметковни задачи кои бараат енормни ресурси. Во ваквиот контекст, вкупното време на извршување може да се намали преку покренување на истата функција многу пати, паралелно, на различни пресметковни јазли, каде што секоја инстанца би обработила мало подмножество од вкупните податоци. Буја и др. [39] дополнително го прошируваат овој концепт, опишувајќи безсерверски низи од задачи (англ. serverless pipelines) составени од повеќе функции поврзани меѓу себе, со цел да се моделираат комплицирани пресметки за анализа на податоци. Неколку слични примери веќе постојат и во пракса [40], [41]. Особено интересен е фактот што обработката на податоци не мора да се одвива исклучиво на безсерверска платформа во облакот, туку може да биде префрлена и на работ од мрежата,

оптимизирајќи ја мрежната искористеност и времето на одговор, под претпоставка дека достапните пресметковни ресурси би ги задоволиле барањата [42].

Заедничкото за сите претходно споменати сценарија е тоа што оптоварувањето на инфраструктурата е непредвидливо. Со ова се поставува прашањето како инфраструктурата би можела да се справи со наплив од голем број барања, без притоа да се загрозат перформансите на останатите веќе активни задачи. Оттука станува јасна потребата за напредни алгоритми за алокација на ресурси и распределување задачи, кои би можеле да ги земат предвид строгите барања во однос на квалитетот на услуга (англ. quality of service, QoS) дури и во критични ситуации со енормен број на дојдовни барања [39]. Преглед на литературата која се однесува на вакви оптимизации при распределувањето на задачите е достапен во [43]. Примарниот фокус на трудот се системите во облакот, но истите аспекти се релевантни и на самиот раб од мрежата.

Префрлувајќи се на работ од мрежата, авторите на [44] ги потврдуваат придобивките од поместувањето на безсерверското пресметување поблиску до изворот на податоците. Тие го користат терминот облак-раб континуум со цел да опишат транспарентна миграција на задачите за искористување на предностите на двете опции истовремено. Овој термин, заедно со терминот екосистем, се општо прифатени термини во литературата за отсликување повеќе инфраструктури кои се меѓусебно поврзани и се протегаат низ работ од мрежата и низ облакот. Според авторите на [44], обработка на податоците би се правела на работ од мрежата за да се врати првичен одговор за кратко време, а, пак, на облакот би останало да се вршат дополнителни анализи кои не се временски осетливи, како и долгорочно складирање. Хелерштајн и др. [18] ги дискутираат проблемите на ефикасност кои се карактеристични за првата генерација на безсерверските решенија, како ограниченото време на извршување кое го дозволуваат јавните даватели на услуги, бавно иницијално стартување, умерени перформанси при извршување влезно/излезни операции и ограничена поддршка за специјализиран хардвер како на пример графички картички. Решение на проблемот со бавното иницијално стартување нудат Крацке и др. во [5]. Покрај споредбата на предностите и недостатоците на безсерверската парадигма, уникернелите [45] се претставени како поефикасна извршна околина, погодна за извршување функции.

Денес веќе постојат конкретни, широко употребувани безсерверски платформи. Овие платформи се стремат да ги имплементираат сите последни пронајдоци и подобрувања во поглед на безбедноста, распределбата на задачите и перформансите, со цел да се понуди готов безсерверски производ за крајни корисници. Бочи и др. [46] нудат преглед врз постојните безсерверски платформи и ги класифицираат во однос на поддржаните програмски јазици, начини на користење и заднинска технологија. Тие особено се осврнуваат и на безбедносните предизвици, но намерно ги изоставуваат безсерверските платформи наменети за инсталација врз еден единствен јазол. Треба да се земе предвид дека иако самите по себе не се скалабилни, ваквите платформи се сепак важен ресурс и може да ја играат улогата на водилка при развојот на поскалабилни алтернативи и понапредни решенија како од безбедносен аспект, така и во поглед на извршните околини, со селективно реискористување на одредени компоненти. Дополнителна анализа на популарни безсерверски платформи е достапна и во [36], но во овој случај фокусот е на работ од мрежата, отсликувајќи ги разликите во достапните решенија денес.

За жал, иако во литературата се посветува посебно внимание на безбедносни аспекти и изолација [46] во поглед на безсерверското пресметување, програмерите поретко се интересираат за ваквите теми, фокусирајќи го своето внимание на конкретни имплементациски аспекти, како што покажува истражувањето извршено врз одговори на

прашања поврзани со FaaS поставени на StackOverflow сајтот [47]. Сепак, многу безсерверски платформи наложуваат строга изолација помеѓу различните функции кои се извршуваат во даден момент, што е и една од главните предности на безсерверското пресметување во целина, водејќи до намалување на потенцијалните безбедносни закани [38].

Академската заедница го има идентификувано безсерверското пресметување за технологија во подем со можност за употреба во најразлични сценарија, вклучувајќи го тука и интернетот на нештата. Претходно изложената литература го потврдува овој тренд и го отсликува високиот интересот во полето. Сепак, од големо значење е да се направи систематско истражување на тековните достигнувања, предизвици и отворени прашања, со цел да се добие јасна слика за поставеноста на дискутираните хипотези во однос на останатите истражувачки напори.

2.1. Методологија на прегледот и класификациска рамка

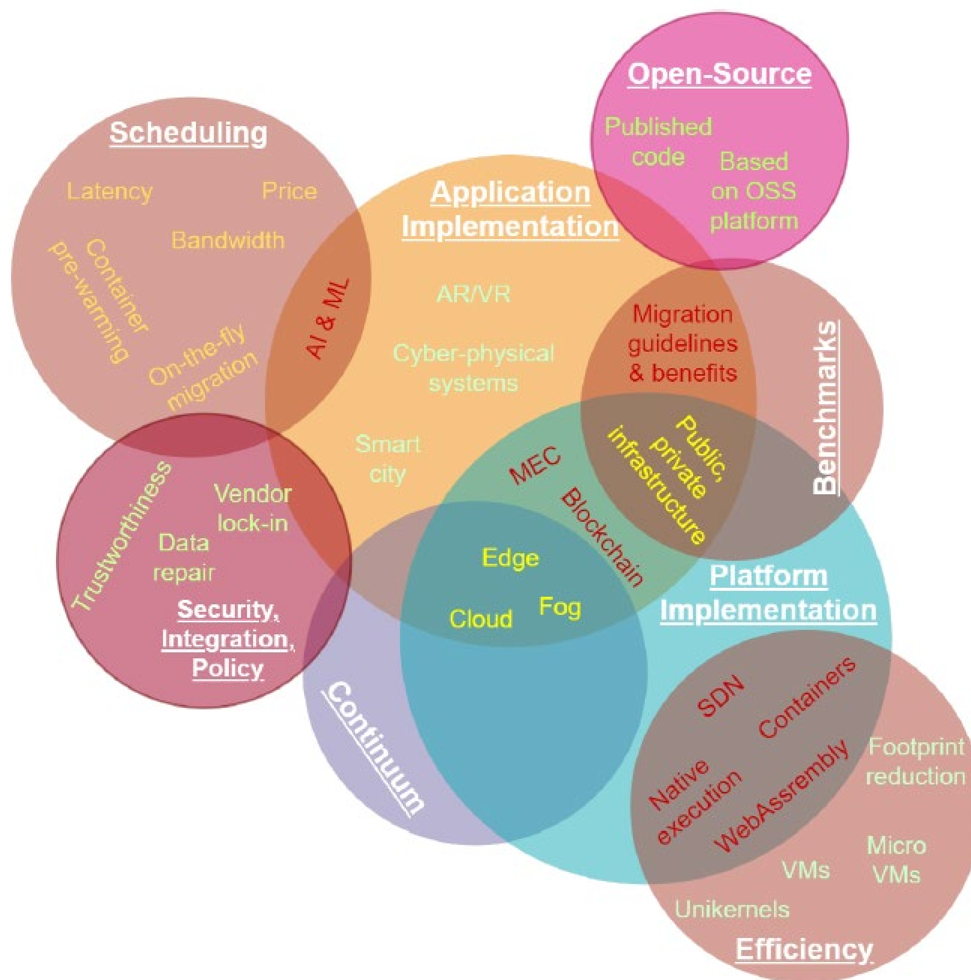
Во продолжение најпрво е дефинирана целта на систематскиот преглед на литературата, а потоа во детали се разгледуваат резултатите од пребарувањето, класифицирањето и анализирањето на релевантните научни трудови. Применетиот истражувачки метод ги следи препораките за изведување систематски прегледи на литературата на Петерсен и др., претставени во [48].

Критериумите врз кои е направен првичниот избор на трудовите се однесуваат на: јазикот на текстот, медиумот на објавување, достапноста на целосниот ракопис и, се разбира, поврзаноста со темата во однос на безсерверското пресметување на работ од мрежата, а во контекст на IoT [22].

2.1.1. Цел на прегледот

Целта на истражувањето е да се анализираат најактуелните научни трудови кои се занимаваат со проучување на безсерверското пресметување како во облакот, така и на работ од мрежата. На самиот почеток се врши испитување на правецот, насоката и природата на актуелните научни трудови од ова поле кои се однесуваат на примена на FaaS или BaaS парадигмите во IoT околина. Потоа, се одредуваат темите од интерес во досегашните истражувања, со посебен осврт на инфраструктурни околини со ограничени перформанси. Резултатите од истражувањето освен што идентификуваат отворени можности за понатамошно истражување, исто така може да им бидат и од корист на нови истражувачи кои се запознаваат со концептите на безсерверското пресметување и пресметувањето на работ од мрежата.

2.1.2. Класификациска рамка



Слика 2.1 Класификациска рамка сочинета од теми и соодветни категории

Како резултат на целосното читање на трудовите, нивното сумирање и придружувањето кратки описни клучни зборови за секој од нив, развиена е класификациска рамка сочинета од 8 теми и 30 категории, дел од различните теми. Слика 2.1 нуди визуелен преглед на оваа класификациска рамка, прикажувајќи го односот помеѓу темите и соодветните категории.

Некои од категориите се намерно придружени на повеќе од една тема, а таков пример се: „edge“, „fog“ и „cloud“ кои се присутни во: „platform implementation“, „application implementation“ и „continuum“ темите. Во случајот со „platform implementation“ и „application implementation“, разликата е јасна, примарниот фокус на даден труд може да биде претставување архитектура на безсерверска платформа наменета за хостирање различни апликации на која било од споменатите локации во мрежата. Од друга страна, пак, авторите може да се имаат посветено на развој на една конкретна безсерверска апликација употреблива во определено сценарио која исто така може да биде инстанцирана на различни локации во мрежата. „Continuum“ темата е наменета исклучиво за трудови во кои се дискутираат различни пристапи за динамичко префрлање задачи низ екосистемот, осврнувајќи се на можните придобивки на нивната имплементација од аспект на доцнењето или пресметковните перформанси. На овој начин, земајќи ги предвид разликите меѓу трите споменати категории („edge“, „fog“ и

„cloud“), класификацијата веднаш ја отсликува суштината на трудот, т.е. дали се поддржани повеќе извршни локации, но администраторот рачно ја носи крајната одлука (без поддршка за „continuum“) или, пак, има динамички систем кој автономно може да ја носи оваа одлука, врз основа на побарувањата на апликациите и тековната состојба (поддршка за „continuum“).

Слична е ситуацијата и со „containers“, „native execution“ и „WebAssembly“ категориите споделени помеѓу „efficiency“ и „ platform implementation“ како главни теми. Во случајот со „efficiency“ темата, споменатите категории се однесуваат на нови пристапи при користењето на овие извршни околини или оптимизации на перформансите применливи за нови или постојни безсерверски решенија. Од друга страна, кога станува збор за „platform implementation“ темата, трите категории служат исклучиво како опис за архитектурата на платформата и преставување на поддржаните извршни околини. Дополнително, „AI & ML“ категоријата е присутна и во „scheduling“ и „application implementation“ темите. Во првиот случај, „scheduling“ темата, присуството на категоријата означува дека вештачката интелигенција се користи во процесот на одлука за најсоодветната локација за извршување на одредена функција, оптимизирајќи различни аспекти како доцнење, цена или пресметковни перформанси, но не наговестува никакво специфично однесување на самата инстанцирана функција. Во вториот случај, „application implementation“, станува збор за сосема спротивното и присуството на категоријата означува дека самата функција, во својата логика има вклучено алгоритми од вештачката интелигенција. Слично објаснување може да се даде и при објаснување на присуството на „MEC“ категоријата во „application implementation“ и „platform implementation“ темите.

Иако има целосно поклопување меѓу категориите придружени на „benchmarking“ и „application implementation“ темите, нивното значење е јасно дефинирано. Од перспектива на „application implementation“, при претставувањето дадена апликација може да се користат приватни и/или јавни инфраструктури за нејзино извршување. Во однос на „benchmarking“, типот на инфраструктура се однесува исклучиво на форматот на тестовите, бидејќи поединечни тестови се насочени кон конкретни инфраструктури, имајќи ја предвид разликата во нивните комуникациски интерфејси. „Migration guidelines & benefits“ категоријата во „application implementation“ случајот се однесува на совети и најдобри практики за тоа како безсерверското пресметување може да придонесе за надминување различни проблеми и отсликува начини како традиционален софтвер може да премине кон безсерверска архитектура. Преостанатиот случај, оној во „benchmarking“ темата, укажува на тоа дека претставеното решение во дадениот труд (апликација, платформа, напреден алгоритам за распределување задачи) е тестирано во поглед на перформансите, споредено со постоечки алтернативи во дадената област.

2.1.3. Класификација преку примена на класификациската рамка

Со користење на претходно опишаната класификациска рамка, сите 64 труда беа категоризирани во однос на темите кои ги опфаќаат. Секој труд е оценет на скала од 0 до 3, зависно од тоа колку е релевантен за соодветната тема. По една примарна тема е доделена на секој труд, означена со задебелена црта под трите ѕвездички кои ја претставуваат оценката. Имајќи го предвид ограничениот простор, трудовите кои со својата содржина опфаќаат само една тема се дадени во табела 2.1. Останатите се дадени во табела 2.2, групирани според нивната примарна тема. Некои од претходно споменатите теми не фигурираат во табела 2.1, бидејќи нема ниту еден труд фокусиран исклучиво на неа.

Табела 2.1

Класификација на трудови кои се осврнуваат на само една тема од рамката

Име на тема	**Трудови**
Application implementation	[15], [16], [18], [35], [37], [38], [39], [47], [49], [50], [51]
Efficiency	[5]
Benchmarks	[52]

Табела 2.2

Класификација на трудови кои се осврнуваат на повеќе од една тема од рамката

Труд	**A. Impl.**[2]	**Eff.**[3]	**Sched.**[4]	**Bench.**[5]	**P. Impl.**[6]	**Cont.**[7]	**SIP**[8]	**OSS**[9]
[53]	★★★	★★☆	☆☆☆	★☆☆	★★☆	☆☆☆	☆☆☆	☆☆☆
[54]	★★★	★☆☆	☆☆☆	★☆☆	☆☆☆	☆☆☆	☆☆☆	☆☆☆
[55]	★★★	☆☆☆	☆☆☆	★☆☆	★★★	☆☆☆	☆☆☆	★☆☆
[56]	★★★	☆☆☆	☆☆☆	★☆☆	☆☆☆	☆☆☆	☆☆☆	★★☆
[57]	★★★	☆☆☆	☆☆☆	★☆☆	☆☆☆	☆☆☆	☆☆☆	☆☆☆
[58]	★★★	☆☆☆	☆☆☆	★☆☆	☆☆☆	☆☆☆	☆☆☆	☆☆☆
[59]	★★★	☆☆☆	☆☆☆	★☆☆	☆☆☆	☆☆☆	☆☆☆	☆☆☆
[60]	★★★	☆☆☆	☆☆☆	☆☆☆	☆☆☆	☆☆☆	★☆☆	☆☆☆
[61]	★☆☆	★★★	★☆☆	★☆☆	★★☆	☆☆☆	☆☆☆	★☆☆
[62]	★☆☆	★★★	☆☆☆	★☆☆	★★☆	☆☆☆	☆☆☆	☆☆☆
[17]	☆☆☆	★★★	★☆☆	★☆☆	★★☆	☆☆☆	☆☆☆	★☆☆
[63]	★★☆	☆☆☆	★★★	★☆☆	★★★	☆☆☆	☆☆☆	★☆☆
[64]	★☆☆	☆☆☆	★★★	★☆☆	☆☆☆	★★☆	☆☆☆	☆☆☆
[43]	★☆☆	☆☆☆	★★★	☆☆☆	☆☆☆	☆☆☆	☆☆☆	☆☆☆
[65]	☆☆☆	★★☆	★★★	★☆☆	☆☆☆	☆☆☆	☆☆☆	★★☆
[66]	☆☆☆	★☆☆	★★★	★☆☆	★★★	★★☆	☆☆☆	☆☆☆
[44]	☆☆☆	★☆☆	★★★	★☆☆	☆☆☆	★★☆	★☆☆	☆☆☆
[67]	☆☆☆	★☆☆	★★★	★☆☆	☆☆☆	★★☆	☆☆☆	☆☆☆
[68]	☆☆☆	☆☆☆	★★★	★☆☆	★★★	☆☆☆	☆☆☆	★☆☆
[69]	☆☆☆	☆☆☆	★★★	★☆☆	★★☆	☆☆☆	☆☆☆	☆☆☆
[70]	☆☆☆	☆☆☆	★★★	★☆☆	★☆☆	★★☆	☆☆☆	☆☆☆
[71]	☆☆☆	☆☆☆	★★★	★☆☆	☆☆☆	☆☆☆	☆☆☆	★☆☆
[72]	☆☆☆	☆☆☆	★★★	★☆☆	☆☆☆	☆☆☆	☆☆☆	★☆☆
[73]	★★☆	☆☆☆	☆☆☆	★★★	☆☆☆	☆☆☆	☆☆☆	★☆☆
[74]	☆☆☆	☆☆☆	☆☆☆	★★★	☆☆☆	☆☆☆	☆☆☆	★☆☆
[75]	☆☆☆	☆☆☆	☆☆☆	★★★	☆☆☆	☆☆☆	☆☆☆	★☆☆
[76]	★★★	☆☆☆	★☆☆	★☆☆	★★★	☆☆☆	☆☆☆	★☆☆

[2] A. Impl. – Application Implementation.

[3] Eff. – Efficiency.

[4] Sched. – Scheduling.

[5] Bench. – Benchmarks.

[6] P. Impl. – Platform Implementation.

[7] Cont. – Continuum.

[8] SIP – Security, Integrity, Policy.

[9] OSS – Open-Source Software.

[77]	★★★	☆☆☆	★☆☆	★☆☆	★★★	★★★	☆☆☆	★★☆
[78]	★★☆	☆☆☆	★★☆	★☆☆	★★★	☆☆☆	☆☆☆	★☆☆
[79]	★★☆	☆☆☆	★☆☆	☆☆☆	★★★	★★☆	☆☆☆	★☆☆
[42]	★★☆	☆☆☆	☆☆☆	★☆☆	★★★	★★★	☆☆☆	★★☆
[80]	★☆☆	☆☆☆	★★☆	★☆☆	★★★	★★★	☆☆☆	★☆☆
[81]	★☆☆	☆☆☆	★☆☆	★☆☆	★★★	★★☆	☆☆☆	★☆☆
[82]	★☆☆	☆☆☆	☆☆☆	★☆☆	★★★	☆☆☆	☆☆☆	★☆☆
[83]	☆☆☆	☆☆☆	★★☆	★☆☆	★★★	★★★	★☆☆	☆☆☆
[84]	☆☆☆	☆☆☆	★☆☆	★☆☆	★★★	★★★	☆☆☆	★☆☆
[85]	☆☆☆	☆☆☆	★☆☆	★☆☆	★★★	★★☆	☆☆☆	★☆☆
[86]	☆☆☆	☆☆☆	★☆☆	★☆☆	★★★	☆☆☆	☆☆☆	★☆☆
[87]	☆☆☆	☆☆☆	☆☆☆	★☆☆	★★★	★★☆	☆☆☆	★☆☆
[88]	☆☆☆	☆☆☆	☆☆☆	★☆☆	★★★	★★☆	☆☆☆	☆☆☆
[89]	☆☆☆	☆☆☆	☆☆☆	★☆☆	★★★	☆☆☆	☆☆☆	★★☆
[90]	☆☆☆	☆☆☆	☆☆☆	★☆☆	★★★	☆☆☆	☆☆☆	★☆☆
[91]	☆☆☆	☆☆☆	☆☆☆	☆☆☆	★★★	★★☆	☆☆☆	☆☆☆
[36]	☆☆☆	☆☆☆	☆☆☆	☆☆☆	★★★	☆☆☆	★☆☆	★☆☆
[92]	☆☆☆	☆☆☆	☆☆☆	☆☆☆	★★★	☆☆☆	☆☆☆	★☆☆
[93]	★☆☆	☆☆☆	★★☆	★☆☆	☆☆☆	★★★	☆☆☆	☆☆☆
[44]	★☆☆	☆☆☆	☆☆☆	☆☆☆	☆☆☆	★★★	☆☆☆	☆☆☆
[94]	☆☆☆	☆☆☆	☆☆☆	☆☆☆	★★★	★★★	☆☆☆	☆☆☆
[46]	★☆☆	☆☆☆	☆☆☆	☆☆☆	☆☆☆	☆☆☆	★★★	★☆☆
[95]	☆☆☆	☆☆☆	☆☆☆	★☆☆	★★☆	★★★	★★★	★★☆
[96]	☆☆☆	☆☆☆	☆☆☆	★☆☆	★★☆	☆☆☆	★★★	★☆☆

Детален осврт на поединечните отворени прашања опфатени со различните трудови, водејќи се по нивната класификација низ темите и категориите е даден во следната секција, „Преглед на отворените прашања“.

2.2. Преглед на темите од интерес

Во делот кој следи е дадена дискусија за главните теми од интерес опфатени со избраните 64 труда, како резултат на опсежниот процес на селекција. Имињата на потпоглавјата ги следат имињата на самите теми, а воедно и имињата на колоните од Табела 2.2, за полесно снаоѓање.

2.2.1. Ефикасност

Подобрувањата во однос на ефикасноста се од особено значење за безсерверското пресметување во целост, имајќи предвид дека ваквите архитектури, барем на најниското ниво, имаат потреба од ефикасно временско мултиплексирање на достапните ресурси. Какви било оптимизации на ова поле директно влијаат не само на давателите на услуги, овозможувајќи им да опслужат повеќе корисници истовремено, туку и на самите крајни корисници, нудејќи им подобри перформанси и услови на користење. Главните барања кои треба да ги исполнува една безсерверска платформа, особено ако е потребно нејзино поставување на работ од мрежата, може да се редуцираат на [61]:

1. можност за инстанцирање безсерверските функции со многу кратко време на извршување, како одговор на некој генериран настан;

2. способност за извршување огромен број безсерверски функции истовремено, независно од корисникот кој го извршил покренувањето;
3. минимално време на одговор, задоволувајќи ги потребите на пресметковните задачи од областа на интернетот на нештата;
4. соодветно справување со постојаните промени во инфраструктурата и високиот степен на ентропија како последица од истовремено извршување различни функции од различни корисници, со исклучително кратко време на извршување и згаснување.

Главниот проблем во областа на ефикасноста, кој воедно има и привлечено значително внимание во академската заедница и индустријата е прашањето како да се намали првичното време потребно за стартување на една безсерверска функција. Оваа загатка уште се нарекува и „проблемот на ладен старт“ (англ. cold start problem). Во потсекциите кои следат направен е детален осврт на најмодерните решенија понудени во литературата досега, почнувајќи со WebAssembly (WASM), но опфаќајќи и други извршни околини како микро виртуелни машини и уникернели. Дискусијата околу безсерверската ефикасност и оптимизации се затвора со опис на неодамнешни напори направени за намалување на целокупната големина на безсерверските функции, притоа заштедувајќи и дисков простор. Во табела 2.3 во продолжение е дадена класификација кој труд на кој од споменатите проблеми се осврнува.

Табела 2.3

Трудови поврзани со темата „Ефикасност“

Категорија	Трудови	Вкупно
Footprint Reduction	[17], [53], [54], [62], [65], [67], [97]	7
Containers	[65], [98], [99]	3
WebAssembly	[17], [61], [62]	3
Native Execution	[53], [98]	2
Micro VMs	[98]	1
SDN	[66]	1
Unikernels	[98]	1
VMs	[98]	1

Проблемот на ладен старт и подгревање контејнери

Проблемот со ладниот старт кај безсерверските архитектури се јавува како директна последица на посакуваното скалирање до 0 инстанци на функции кои во даден момент не се потребни. При наредното повикување на функцијата, претпоставувајќи дека таа нема активни инстанци, мора да се изврши покренување од почеток, водејќи до значително побавно стартување во споредба со случајот во кој се реискористува веќе достапна извршна околина од некој постар повик [16], [62]. Во литературата се предложени различни пристапи за надминување на оваа слабост. Еден од нив е да се чуваат на располагање повеќе „подгреани“ контејнери, спремни за извршување на произволна функција веднаш штом ќе се добие барањето. Вака се избегнува потребата од ад-хок поставување на извршната околина и се елиминира ладниот старт [17], [86]. Иако овој пристап може да се класифицира како потенцијално решение на целокупниот проблем, сепак, води до зголемена потрошувачка на ресурси бидејќи контејнерите се активни без да извршуваат задачи, а и целосно се елиминира една од главните придобивки на безсерверското пресметување, скалирањето до 0. Блага варијација, применета од неколку комерцијални платформи и платформи со отворен код, е да се остави извршната околина активна по завршувањето на дадена функција, со цел во неа

да се постави друга функција која би требало да се инстанцира во блиска иднина. Сепак, треба да се земе предвид дека на овој начин не може да се гарантира изолацијата меѓу функциите кои споделуваат иста извршна околина. Поради тоа овој пристап воведува безбедносни ризици од аспект на протекување чувствителни информации оставени од претходно, и видливи за сите функции што се извршуваат по нив.

Недостатоците на предложените решенија служат како иницијатор за напуштање на контејнерите како примарна извршна околина за безсерверски функции и откривање нови, поефикасни алтернативи. Целта на ваквите напори не е само да се намали иницијалното време за подигнување на функциите, туку исто така да се овозможат подобри перформанси при извршувањето, споредливи со оние кои се достапни при директно извршување на самиот хардвер, без користење дополнителни нивоа на апстракција.

WebAssembly

Еден пример за поефикасна извршна околина за безсерверски функции со огромен потенцијал е WebAssembly. Иако првенствено замислена како веб технологија, што се отсликува и преку самото име, од неодамна стана возможно WASM да се претвори во сестрана околина, не само ограничена на извршување на клиентската страна. Неколку WebAssembly решенија со отворен код веќе може директно да се вгнездат во постојни апликации. Хал и др. [62] го земаат токму ваквиот пристап, имплементирајќи безсерверска платформа инспирирана од архитектурата на OpenWhisk. За разлика од популарниот OpenWhisk, наместо класични контејнери претставеното решение користи WebAssembly за извршување на функциите. Придонесот на авторите дополнително се однесува и на категоризацијата на различни препознаени шаблони при извршување на безсерверски функции:

- еден клиент, повеќе функции;
- повеќе клиенти, една функција;
- повеќе клиенти, повеќе функции.

За секое од сценаријата во трудот е направена анализа на перформансите на WebAssembly во споредба со традиционалната имплементација на OpenWhisk. Резултатите покажуваат дека времето за подигнување на функциите е драстично подобрено, но перформансите при извршување се намалени како во однос на директно извршување врз хардверот, така и во споредба со контејнеризацијата.

Потенцијално решение за намалените перформанси нудат Гадепали и др., најпрво во [17] со претставувањето на прототип имплементација и потоа во [61] со имплементирањето на целосно функционална платформа [100]. Нивниот пристап е различен и не користи JavaScript виртуелна машина како извршна околина за WebAssembly безсерверските функции, бидејќи иако нуди подобро време на стартување, резултира со полоши пресметковни перформанси, како што е покажано во [62]. Наместо тоа, тие развиваат целосно нов WebAssembly компајлер и извршна рамка наречена aWsm [101]. Придобивките од ваквиот чекор се тоа што различни функции може да споделуваат единствена инстанца на извршната околина, заобиколувајќи ги механизмите за распределба на задачи понудени од јадрото на оперативниот систем, како и придружените функционалности за изолација, нудејќи сопствени имплементации на нивно место. Ова резултира со намалување на времето за стартување на функциите до ранг од микросекунди, а нивната мемориска зафатнина притоа се мери во килобајти. Резултатите од [61] каде што е претставена Sledge платформата, покажуваат 4 пати

подобра латентност и пропусна моќ споредено со платформата со отворен код Nuclio [102], која важи за една од најбрзите безсерверски платформи денес.

Иако WebAssembly може да се идентификува како потенцијално решение на проблемот со ладен старт, сè уште постојат отворени прашања кои мора да бидат одговорени пред да се задоволат сите барања во поглед на ефикасна безсерверска извршна околина за работ на мрежата. Платформите кои би ја имплементирале оваа нова технологија треба да се фокусираат на пресметковните перформанси, со цел да не се анулираат какви било подобрувања во поглед на побрзото стартување со побавното време на извршување. Позитивен аспект кој се очекува да биде уште поистакнат во иднина е големата поддршка за различни програмски јазици кои може да бидат компајлирани во WebAssembly модули, овозможувајќи им на програмерите полесно запознавање со технологијата.

Микро виртуелни машини

Микро виртуелните машини се алтернативно решение за проблемот со бавен старт, но и пониското ниво на изолација карактеристично за контејнерите, споредено со традиционалните виртуелни машини. Придобивките во поглед на хардверски поддржана изолација помеѓу различните задачи, придружени со времето на подигнување на функциите мерено во милисекунди ја прават оваа технологија атрактивна за безсерверското пресметување. Првите решенија кои ги користат микро виртуелните машини во пракса се веќе достапни, како од безсерверски аспект, така и од традиционалниот серверски пристап [103]. Amazon Firecracker е пример за една таква инфраструктура, чијашто имплементација на микро виртуелните машини е и дополнително објавена како отворен код со слободна лиценца [104].

Примената на микро виртуелните машини е сè уште предмет на активно истражување, со значителен интерес како од академската заедница, така и од индустријата. Ова е дополнително охрабрување, бидејќи придобивките од нивната примена се јасни како од поглед на брзина на извршување, така и од поглед на изолација [105].

Уникернели

И уникернелите може да се гледаат како решение на проблемите асоцирани со употребата на контејнерите во контекст на безсерверското пресметување. Главниот концепт над кој се потпираат е можноста за пакување на апликацијата заедно со сите нејзини зависности и системски функции потребни за управување со хардверот во една единствена целина. На овој начин, уникернелите може да бидат инстанцирани независно од останатиот софтвер и немаат потреба од поддршка за дополнителен оперативен систем. Ова води до софтверски артефакти кои се драстично помали од традиционалните слики за виртуелни машини, а притоа нудат одлични перформанси при извршувањето.

Авторите на [5] ги идентификуваат уникернелите како извршна околина која во иднина би можела да се користи за безсерверско пресметување. Истражувањето во поглед на уникернели денеска е ограничено и потребни се дополнителни напори на ова поле [106].

Намалување на големината на функциите

Големината на функцијата игра значајна улога во времето потребно за нејзино стартување, особено кога станува збор за пресметковни уреди со ограничени перформанси, како оние на работ од мрежата. Дополнително, најголемиот дел од комерцијалните безсерверски платформи денес наплаќаат и за складирањето на самата извршна датотека за функцијата, што може да достигне стотици мегабајти ако се земат предвид сите нејзини зависности од аспект на трети програмски библиотеки. На овој

проблем не се имуни ниту приватните инфраструктури, бидејќи малиот простор за складирање на уредите на работ од мрежата може да претставува огромно ограничување во поглед на тоа колку функции би можеле да се извршуваат во даден момент. Со цел да се разреши ова прашање, во литературата се предложени неколку оптимизациски техники [54], [97]. Главниот предмет на интерес е како да се спречи големината на функцијата неконтролирано да расте со воведувањето дополнителни програмски зависности. За жал, ова е чест случај при миграцијата од традиционална архитектура кон безсерверското пресметување, каде што претходниот технички долг може да побарува реискористување постари датотеки, неоптимизирани за користење во безсерверски функции.

Повеќе детали околу препорачаните оптимизации кои треба да бидат применети при миграција од традиционална кон безсерверска архитектура и можноста за избегнување одново пишување на целата логика се дадени во „2.3.5.4 Миграција кон безсерверска архитектура“, подолу.

2.2.2. Распределба на задачи

Пресметувањето на работ од мрежата по природа е дистрибуирано, па потребно е да се користат ефикасни алгоритми за распределба на задачи кои би можеле да одлучуваат која задача на кој јазол од мрежата би се извршувала. При одлучувањето, од особено значење е да се земат предвид перформансите на самите уреди на овој дел од мрежата, бидејќи нивниот ограничен складишен простор и пресметковна моќ би можеле да доведат до намалување на бројот на функции активни во даден момент. Најголемиот дел од истражувањата во оваа област се насочени кон развој на алгоритми за оптимизирање на доцнењето, цената и/или пропусниот опсег на функциите. При носењето на одлуката за локацијата на извршување, одредени алгоритми не само што го вклучуваат работ на мрежата, туку го разгледуваат и облакот, на овој начин обезбедувајќи првичен облак-раб екосистем за транспарентно извршување на задачите. Во избраните трудови од ова тема има примери кои се темелат на комерцијални услуги [67], [97], како и онакви наменети за платформи со отворен код [63], [68], [72]. Во табела 2.4 се категоризирани релевантните трудови во однос на карактеристика што ја оптимизираат при носењето одлуки.

Табела 2.4

Трудови поврзани со темата „Распределба на задачи“

Категорија	**Трудови**	**Вкупно**
Latency Optimization	[17], [43], [49], [59], [61], [63], [64], [65], [66], [68], [69], [70], [71], [72], [77], [78], [79], [81], [83], [85], [93], [97]	22
Bandwidth Optimization	[43], [63], [66], [78], [80], [93]	6
AI & ML	[64], [65], [70], [71], [72]	5
Container Prewarming	[65], [71], [72], [76], [86]	5
Price Optimization	[43], [66], [67], [70], [83]	5
On-the-fly Migration	[43], [69], [80]	3

Оптимизации кај комерцијални безсерверски платформи

Елгамал и др. [67] дефинираат алгоритам кој е наменет за AWS портфолиото на безсерверски продукти, со можност за распределба на задачи и на работ од мрежата и во облакот. На работ од мрежата се користи AWS Greengrass услугата, додека, пак, во облакот таа се заменува со AWS Lambda. Можноста за лесно реискористување на функциите низ овие две услуги, без потреба од никаква промена во изворниот код, му дава можност на системот за распределба на задачи да го гледа работ од мрежата како продолжение на облакот. Авторите пристапуваат на ова прашање како проблем на најкраток пат со дадени ограничувања (англ. constrained shortest path problem), барајќи ја најмалата цена на извршување, без притоа да се надмине одреден праг на доцнење. Дополнителни финансиски заштеди може да се реализираат при извршувањето во облак, преку комбинирање повеќе различни функции во една, елиминирајќи ја потребата од последователно врзување на функциите, што носи дополнителни трошоци.

Пеле и др. во [97] исто така имплементираат апстракциски слој помеѓу Greengrass и AWS Lambda со можност за одлучување која инфраструктура да се користи во кои ситуации, темелејќи ја својата одлука врз метрики собрани од уреди за софтверско дефинирано вмрежување (англ. software defined networking, SDN) поставени на различни локации. Авторите опишуваат и дополнителни оптимизации преку групирање на повеќе функции во една, слично на пристапот од [67]. Системот е развиен со користење две нивоа. Првото ниво генерира опис на тоа како би биле групирани различните функции, кои би биле нивните перформансни побарувања и на која инфраструктура би се одвивало извршувањето, сè врз основа на метрики од претходни извршувања и ограничувања зададени од корисникот. Вториот слој ја игра улогата на адаптер одговорен за покренување на генерираниот опис врз конкретна инфраструктура. Во првичниот прототип се поддржани само услуги од портфолиото на AWS, но авторите тврдат дека второто ниво би можело да се прошири за во иднина да работи и со други даватели на услуги, овозможувајќи да се избере понудувачот на најповолни услови.

Оптимизации кај безсерверски платформи со отворен код

За разлика од трудовите кои ги земаат предвид само комерцијалните инфраструктури, оние кои се наменети за независни платформи со отворен код најчесто се фокусираат на оптимизација на доцнењето. Чиконети и др. [68] опишуваат алгоритам кој може да го избере најоптималниот јазол за извршување на дадена функција од низа дистрибуирани јазли, водејќи се од претпоставката дека самата функција е достапна за покренување на секој од нив. Користејќи софтверски дефинирано вмрежување и доделувајќи различни улоги на уредите на работ од мрежата, авторите имплементираат хиерархиски систем за одлуки, што резултира со избор на оној уред кој нуди најмало доцнење при извршување конкретна функција. Прототипот за платформа која го имплементира ова однесување е развиен врз OpenWhisk, а изворниот код е јавно достапен [107].

Алтернативен пристап кој се темели на проширување на постојната Knative безсерверска платформа со отворен код е опишан во [72]. Авторите претставуваат систем кој користи линеарни регресиски модели за предвидување кога дадена функција ќе биде повикана. Нивната идеја е да се елиминираат проблемите асоцирани со ладниот старт на функциите преку предвремено стартување на извршните околини наменети за нив. Слични системи за предвидување се достапни и за останати платформи со отворен код кои често се поставуваат и на работ од мрежата. Агарвал и др. [71] користат учење со поттикнување за да дизајнираат алгоритам за распределба на задачи за Kubeless платформата. Понуденото решение е претставено како алтернатива на вградениот Kubernetes auto-scaler [108] и може да го оптимизира бројот на инстанци од дадена функција во зависност

од тоа колку извршувања има во одреден период. Авторите на [65] го продолжуваат трендот на истражување подобри алгоритми за распределба кај платформите со отворен код, претставувајќи решение наменето за OpenWhisk кое може динамички да влијае врз бројот на процесорски циклуси резервирани за веќе покренатите функции.

Паметни алгоритми за распределување на работ од мрежата

Патман и др. [64] нудат интересна визија за иднината на безсерверските платформи на работ од мрежата, со развивањето алгоритам за распределба на задачи кој ги зема предвид и крајните кориснички уреди, а не само инфраструктурата на работ од мрежата. На овој начин, слободниот пресметковен капацитет на уредите на корисниците во дадена област може да се искористи за извршување безсерверски функции.

Чо и др. [70] исто така го користат машинското учење за дефинирање алгоритми за распределба, но во овој случај тоа се прави во контекст на повеќе-пристапното пресметување на работ од мрежата (англ. Multi-Access Edge Computing, MEC). Инфраструктурата која ги извршува безсерверските функции при ваквиот пристап е колоцирана со самите базни станици за мобилна телефонија. Користејќи алгоритми за длабоко учење со поттикнување задачите се распределуваат помеѓу работ од мрежата и облакот во однос на: бараниот квалитет на услуга, цена и потребни перформанси. Овој тренд на поставување на пресметковна инфраструктура на локациите каде што се наоѓаат базните станици, водејќи се според MEC спецификацијата и потоа избирајќи ја најдобрата локација за извршување е исто така предмет на дискусија во [63]. Она што се оптимизира во овој случај е севкупното време за извршување на функцијата, но земени се предвид и дополнителни ограничувања во поглед на специфичен хардвер кај јазлите, како на пример графички картички.

Миграција при извршување

Може да се заклучи дека постојат голем број различни стратегии кога станува збор за распределување на функции низ различни јазли. Сепак, најголемиот дел од нив се посветени исклучиво на задоволување на барањата на функцијата само при првото нејзино поставување, без да водат сметка што се случува при извршувањето. Кархула и др. [69] нудат решение на овој проблем и опишуваат начин како дадена безсерверска функција може да биде префрлена на друга локација, додека трае нејзиното извршување. Со ваквиот пристап станува возможно да се изврши подобро балансирање на функциите кои се наоѓаат на дадена инфраструктура, надминувајќи потенцијални проблеми како недостаток на меморија или презаситеност на процесорот кои во спротивно би предизвикале предвремено нејзино терминирање или драстично намалување на перформансите. Дополнителна придобивка од сето ова е тоа што истиот пристап може да биде искористен и за привремено паузирање на дадена функција, додека таа е блокирана и чека некоја друга инстанца да заврши пред самата да продолжи да се извршува на истата инфраструктура. Вреди да се спомне дека вакво привремено паузирање е полезно и при чекање да заврши бавна влезно/излезна операција, притоа штедејќи ги ресурсите. Заштедите постигнати со примена на оваа стратегија се значајни како за корисниците така и за давателите на услуги. Корисниците плаќаат помалку, бидејќи нивната функција не е залудно активна, а на давателите на услуги им се овозможува поефикасно временско мултиплексирање на истата инфраструктура. На овој начин се надминува и проблемот на двојно плаќање (англ. double spending problem), каде што на корисникот му се наплаќа за сите функции кои се активни, иако некоја од нив може да е блокирана, чекајќи некоја претходна функција да даде резултат кој потоа би се користел како влез.

2.2.3. Платформи за безсерверско пресметување на работ од мрежата

Практичните имплементации на безсерверски платформи наменети за работ од мрежата се исто така популарна тема во истражувачката заедница. Се нудат голем број на различни пристапи за прилагодување кон побарувањата на работ, како на пример: реискористување на постојни комерцијални решенија, прилагодување популарни безсерверски платформи со отворен код, но и развивање целосно нови решенија. Најголемиот дел од понудените решенија се објавени со слободна лиценца и нивниот код е јавно достапен. На овој начин се овозможува други заинтересирани страни лесно да го реискористат решението или, пак, да го надградат со дополнителна функционалност. Друг сѐ поприсутен тренд е настојувањето да се спојуваат нови безсерверски платформи со постојни комерцијални решенија со помош на средишни слоеви за меѓу-компатибилност. Крајната цел е да стане возможно реискористувањето на постојни функции, без никакви прилагодувања за нивно извршување во новата средина.

Во табела 2.5 се дадени главните карактеристики на сите дискутирани платформи во продолжение заедно со детали во однос на примарната локација на извршување, користената извршна околина, поддржаните програмски јазици и основата врз која е развиено секое од решенијата. Очигледно е дека голем број истражувачи се одлучуваат сопствените безсерверски решенија да ги темелат на постојни платформи, а некои од нив се и со отворен код.

Табела 2.5

Карактеристики на безсерверски платформи

Име	**Локација**	**Околина**	**Јазици**	**Основа**	**Отворен код**
STOIC [81]	Каде било	Директно и контејнери	Кои било	Kubeless	✓ [109]
Serverless IoT [84]	Каде било	Контејнери	Кои било	OpenFaaS	✓[110]
Pigeon [86]	Каде било	Контејнери	Dockerfile	Kubernetes	X
Fog Function [80]	Каде било	Контејнери	Dockerfile	FogFlow	✓ [111]
[76]	Каде било	Контејнери	Кои било	OpenWhisk	X
CSPOT [87]	Каде било	Контејнери	C, Python	Docker	✓ [112]
A3-E [77]	Каде било	Контејнери	Зависи од основа	OpenWhisk, AWS	✓ [113]
[42]	Каде било	Контејнери	Кои било	Kubernetes, AWS Lambda	✓ [114], [115], [116]
Clemmys [99]	Каде било	Контејнери	Кои било	OpenWhisk	X
Hcloud [83]	Каде било	Затворен платформи, контејнери	Python	IaaS и комерцијални решенија	X
[85]	Раб, облак	Контејнери	Кои било	Уреди од работ	✓ [117]
[88]	Раб, облак	Директно и контејнери	Python, Node.js, Java, C, C++	AWS Greengrass и AWS Lambda	X
[91]	Раб, облак	Контејнери	Непознато	LXC	X
EBI-PAI [78]	Раб	Контејнери	Кои било	OpenWhisk	X
tinyFaaS [90]	Раб	Контејнери	Node.js	Docker	✓ [118]
Stack4Things [89]	Раб	Контејнери	Python, Node.js	Qinling, Iotronic	✓ [119]
Kappa [92]	Раб	Calvin околина	Calvin Script, Python	Calvin	X
Sched-Sim [120]	Раб	Контејнери	Кои било	Kubernetes	✓ [120]
Serverless MEC [82]	Раб	Директно	Кои било	Any	✓ [121]

Безсерверски платформи кои поддржуваат повеќе локации на извршување

Последните истражувања во полето на безсерверските платформи се насочени кон поддржување и на облакот и на работ од мрежата како потенцијални извршни локации. За вкупно 13 од избраните платформи возможна е инсталација на повеќе од едно место во мрежата. Примарната цел која треба да се постигне со една таква дистрибуирана платформа е работ на мрежата да се користи за помали пресметки, нудејќи локален, но и брз поглед врз податоците, а облакот да се користи за обработка на поголеми множества, нудејќи глобален поглед врз севкупните податоци [91].

На почетокот од анализата на трудовите од овој дел треба да се спомнат Пинто и др. [84] кои развиваат платформа [110] што содржи посредничка компонента за мерење на вкупното време за извршување на секоја инстанцирана безсерверска функција. Ваквата информација понатаму може да се искористи со цел да се одреди најпогодната локација (облак или раб) за наредното извршување на истата функција. Процесот на одлука се поистоветува со проблемот на „бандитот со повеќе раце“ (англ. multi-armed bandit problem) и понудени се три различни алгоритми за негово решавање, балансирајќи помеѓу извидничките и експлоатациските аспекти. Дополнително и [77] и [83] опишуваат платформи со логика за распределба на задачи низ облак-раб екосистемот. Авторите на [77] ги земаат предвид побарувањата на функциите од аспект на мрежното доцнење, потребната енергија и достапноста при распределбата, додека, пак, во [83] се разгледува и целокупната цена за извршувањето во пари, избирајќи ја најоптималната локација за функцијата од тој аспект. Предложеното решение постојано врши следење на цените на услугите кај различни комерцијални безсерверски платформи и врз основа на овие информации, ја избира најевтината, сè додека перформансните побарувања на функцијата се задоволени. За да се осигура непроменливост на информациите во однос на цените и да се спречат малициозни манипулации од страна на администраторите на платформата, целокупната историја е запишана на складишен систем имплементиран со блоковски вериги. Слично на ова, Жанг и др. [81] претставуваат платформа погодна за извршување на задачи од областа на вештачката интелигенција која може да биде директно поставена на работ или во облакот без никакви дополнителни апстракции, адаптирајќи ја Kubeless безсерверската платформа со отворен код. Имајќи ја предвид специфичноста на задачите од сферата на вештачката интелигенција, при носењето на одлуката се зема предвид и достапноста на специјализиран хардвер на секоја од потенцијалните локации.

Авасалкај и др. [85] нудат уникатно решение кое практично ја елиминира потребата од поставување сопствена сложена и дистрибуирана инфраструктура на работ од мрежата, а и исто така го избегнува и користењето комерцијални даватели на услуги во облакот. Предложеното решение се темели на креирање заедница, каде што корисниците може да одлучат да ги споделат своите лични пресметковни ресурси со останатите. Секоја апликација на платформата е преставена како низа од задачи кои би можеле да бидат извршени на кој било уред моментално достапен на работ од мрежата. Само во случај кога нема доволно ресурси за извршување на работ, може да се користат алтернативни инфраструктури во облакот. Сепак, начинот на кој корисниците би се поттикнале да ги споделуваат локално достапните пресметковни ресурси со заедницата и што би добиле за возврат на тоа, сè уште не е дефиниран од страна на авторите и би бил предмет на идно истражување.

Проширување на постојни безсерверски платформи

Дел од избраните трудови се посветуваат и на подобрување на постојните безсерверски решенија, во некои случаи дури и оние комерцијалните. Пример за ова е Clemmys

платформата [99] која го надополнува OpenWhisk со додавање поддршка за Intel SGX безбедносната технологија, воведувајќи перформансни оптимизации кои го намалуваат негативното влијание од зголемената безбедносна изолација. Крајниот резултат е зголемена безбедност при извршувањето на безсерверските функции со перформанси споредливи со стандардниот OpenWhisk. Користењето на Intel SGX во контекст на безсерверското пресметување во повеќе детали е разгледано во „Безбедност, интегритет и регулативи кај безсерверското пресметување“. Друг пример за подобрување на постојна комерцијална платформа е [88], каде што AWS Lambda се користи за извршување на функции во облакот, а Greengrass на работ од мрежата. Преку автоматско преземање на функциите по потреба, Greengrass уредите може да извршат сопствена реконфигурација, локално складирајќи ги исклучиво оние функции неопходни во дадениот момент со што се заштедува простор. Во овој труд не се користи напредна распределба на задачите и самиот сопственик одлучува дали одредена функција ќе се извршува на работ од мрежата или во облакот. Во случај кога ќе се избере облакот, Greengrass уредите може да служат како посредници во раб-облак комуникацијата.

Алтернативи со отворен код кои може да прилагодат локални уреди да служат за извршување на функции на работ од мрежата се исто така достапни. Трикоми и др. [89] претставуваат платформа [119] која е базирана на Qingling Openstack [122] проектот за безсерверски инфраструктури, овозможувајќи извршување функции на работ од мрежата. Дополнително, понуден е и графички кориснички интерфејс заснован на популарното Node-RED решение [123], овозможувајќи им на корисниците лесно да поврзат различни функции во една целина. Како и кај останатите платформи кои ги користат контејнерите за извршна околина, изолацијата на ниво на поединечно барање не е загарантирана, бидејќи повеќе функции може да бидат извршувани во еден ист контејнер. И Бареси и др. [76] работат на подобрување на постојна безсерверска платформа, во овој случај OpenWhisk, развивајќи споделен перзистентен медиум кој може да се користи при различни повици на една функција, играјќи ја улогата на кеш. На овој начин се елиминира непредвидливоста во однос на локалната состојба која потекнува од тоа дали наредниот повик на функцијата би се извршил во рамките на истата извршна околина или, пак, во нова. Сепак, како и претходно, на овој начин не се гарантира изолација на ниво на барање во сите случаи и крајната одлука за реискористување на извршната околина е препуштена на OpenWhisk.

Средишни слоеви за компатибилност

Со цел полесно прифаќање на дадена новопретставена платформа, одредени решенија имплементираат средишни слоеви кои нудат компатибилност со постојни популарни комерцијални безсерверски продукти, овозможувајќи едноставно реискористување на функциите. Вакви слоеви се имаат имплементирано и во други ситуации во минатото, а можеби најдобриот пример за тоа е широката прифатеност на S3 интерфејсот за управување со складишта за зачувување објекти (англ. object storage) оригинално развиен од AWS. Денес постојат огромен број на комерцијални и отворени решенија кои го користат потполно истиот начин за комуникација како и оригиналното решение, нудејќи компатибилност и од аспект на постојното знаење на програмерите, и од аспект на програмските библиотеки. Авторите на [87] претставуваат систем кој нуди компатибилност со AWS Lambda услугата и може да биде инстанциран на различни уреди, со различни хардверски конфигурации кои се наоѓаат на работ од мрежата или во облакот. Со ова се овозможува постојни AWS Lambda функции само да бидат префрлени на новата платформа, без каква било интервенција во програмскиот код од страна на програмерот. Решението користи контејнери како извршна околина и еден контејнер може да биде реискористен од страна на повеќе функции.

Интелигентна распределба на функции кај безсерверските платформи

Една од техниките за надминување на горливиот проблем на ладен старт, како што беше дискутирано и претходно, е подгревањето на контејнерите. Pigeon рамката [86] е конкретен пример кој го имплементира ваквиот пристап со помош на група од подгреани контејнери со различни хардверски карактеристики, спремни да извршуваат произволни функции кога ќе се појави потреба за тоа.

Чанг и др. во [80] претставуваат платформа која може на интелигентен начин да ја одреди локацијата за извршување на дадена функција, со дополнителна можност и да се изврши миграција во лет од еден јазол на друг. Ова е реализирано преку спојување на сите потенцијални јазли за извршување во едно исто хиерархиско множество во однос на тоа кој јазол е најсоодветен за која задача. Безсерверските функции, штом се јави потребата, се извршувани на најпогодниот јазол.

Интересно, авторите на [42] го промовираат користењето на безсерверски функции во научен контекст, со цел намалување на потребното време за анализа на огромни количества податоци. Во зависност од посакуваната локација за извршување, платформата може да користи: Kubernetes кластери поставени на приватни облаци, Kubernetes кластери на работ од мрежата или, пак, комерцијалната AWS Lambda услуга. Ваквиот пристап е особено корисен при обработка на информации кои може да бидат доверливи и каде што од особено значење е зачувувањето на приватноста на корисниците. Иницијалната обработка и анонимизација може да се изврши на работ од мрежата, пред да се испратат податоците во облакот за дополнителни анализи и агрегација.

Самостојни безсерверски платформи за работ на мрежата

При извршувањето безсерверски задачи на работ од мрежата, може да се идентификуваат два уникатни пристапа. Првиот пристап се заснова на користење постојни уреди веќе достапни на работ од мрежата, потпирајќи се на нивниот слободен пресметковен капацитет, неискористен во даден момент. Во литературата ова уште се нарекува пресметување на работ од мрежата без уреди (англ. deviceless edge computing) [91]. Вториот пристап подразбира поставување нова пресметковна инфраструктура на работ од мрежата со една единствена намена, извршување пресметки врз податоци, самите генерирани од крајни уреди лоцирани на работ. Иако навидум овие два пристапа се поврзани помеѓу себе, сепак вреди да се направи јасна дистинкција, бидејќи во одредена литература важат за комплетно независни ентитети и градбени единки на уште покомплексни хиерархиски инфраструктури каде што работ од мрежата е поделен на повеќе потслоеви.

Пример за платформа исклучиво наменета за работ од мрежата е tinyFaaS [90], [118] и нејзината примарна цел е да биде што е можно полесна од аспект на ресурсни побарувања. Сите уреди кои се дел од платформата се разгледуваат како индивидуални и независни јазли, со што се елиминира потребата од имплементација на комплексни алгоритми за распределување задачи меѓу нив. Функциите се извршуваат во контејнери без да се имплементира изолација на ниво на барање. За повикување на функциите се користи оптимизираниот CoAP протокол наместо традиционалниот HTTP, што дополнително ги намалува пресметковните побарувања од уредите на работ од мрежата.

Пренамената на де-факто стандардизирани решенија во облакот и нивно префрлање на работ од мрежата е исто така популарна тема за истражување денес. Рауш и др. [79] опишуваат како популарниот Kubernetes оркестратор за контејнери може да се прилагоди и да биде основа за безсерверска платформа на работ од мрежата. Иако

Kubernetes доаѓа со вграден распределувач на задачи кој одлучува на кој јазол од кластерот ќе се постави одредена задача, резултатите на авторите покажуваат дека тој не е доволно ефикасен во услови на ограничени перформанси како оние на работ. Проблемот настанува во високодинамични средни каде што треба постојано да се креираат нови контејнери и да се исклучуваат стари. Ова е особено значаен предизвик за безсерверски платформи со токму такви потреби, па неопходно е изнаоѓање алтернативни решенија за заобиколување на проблемот. Едно од предложените решенија, и во овој случај, е користењето подгреани контејнери. Kubernetes се идентификува како потенцијално корисно решение за пресметување на работ од мрежата и од пошироката заедница и индустрија, па денеска постојат специфични Kubernetes дистрибуции наменети токму за извршување на инфраструктура со далеку помали ресурси од она што е стандардно во облакот [124], [125]. На овој начин, Kubernetes кластерите може да се поставаат уште поблиску до корисниците, преку користење скромни уреди, во некои случаи дури и мали сѐ-во-едно компјутери (англ. single-board computers).

Решенија што овозможуваат кој било постоен уред да стане дел од поширока платформа која се протега на работ, имаат потенцијал навистина драстично да го забрзаат и олеснат имплементирањето на безсерверската парадигма на работ од мрежата. Неколку примери за ова веќе постојат и денес, како на пример комерцијалните продукти Greengrass и Azure IoT Hub. Интересот на академската заедница е исто така значителен, па така Персон и др. [92] дискутираат едно вакво решение уште во 2017 година, истата година кога комерцијалниот Greengrass е првпат претставен. Решението на авторите е изградено врз постојната Calvin [126] платформа, која, пак, користи CalvinScript за опишување на функциите. Слично како и сродниот Greengrass, распределбата на задачите се извршува преку обележување на јазлите, а при инстанцирањето дадена функција, се одбира јазолот со посакуваната лабела.

Безсерверски платформи за повеќе-пристапно пресметување на работ од мрежата

Повеќе-пристапното пресметување на работ од мрежата е иницијатива чија цел е обединување на телекомуникациската инфраструктура со инфраструктурата за пресметување преку поставување сервери на локации каде што веќе има базни станици за мобилна телефонија. На овој начин, би се искористило постојното мрежно поврзување, а и во комбинација со постојаниот напредок на телекомуникациските технологии и сѐ поприсутниот 5G, се очекува да се отворат целосно нови кориснички сценарија. Развојот на MEC се одвива во рамките на Европскиот институт за телекомуникациски стандарди (ETSI) [127] и е предмет на значителен интерес од академската заедница.

Последните истражувања одат во насока на интегрирање на безсерверското пресметување со MEC што дополнително би овозможило и ослободување од стегите на комерцијалните даватели на услуги, преку користење отворени и стандардизирани MEC интерфејси за комуникација. Чиконети и др. во [82] ја опишуваат својата визија за една таква интеграција, осврнувајќи се на начини како функциите би можеле да бидат дистрибуирани на различни локации, колоцирани со базните станици. Перформансите на решението се првично испитани со користење симулации. Дополнително, развојот на MEC интерфејс за постојни безсерверски платформи е исто така проект на кој активно се работи [121].

Јанг и др. [78] паралелно даваат свој придонес на ова поле преку реискористувањето на OpenWhisk, имплементирајќи MEC прототип платформа изградена врз софтверски

дефинирани мрежи. Познавањето на самата инфраструктура му овозможува на мобилниот оператор динамички да ги манипулира DNS мапирањата, со цел корисниците секогаш да бидат пренасочени кон најсоодветната функција за нивното барање, слично на [55].

2.2.4. Континуум

Пресметувањето на работ од мрежата, со својата идеја за овозможување пресметки со мало мрежно доцнење, но релативно ограничени перформанси блиску до крајните корисници, не претставува замена за останатите инфраструктури поставени повисоко во хиерархијата. Напротив, за овозможување на најдобро корисничко искуство како и најекономично решение, потребен е начин за обединување на сите постојни инфраструктури преку експлоатација на нивните најдобри аспекти, без разлика дали станува збор за работ или за облакот. Во табела 2.6 е даден преглед на трудовите кои дискутираат решенија што би можеле да бидат поставени на повеќе од една локација истовремено.

Табела 2.6

Трудови поврзани со темата „Континуум“

Категорија	Трудови	Вкупно
Раб, Облак	[64], [66], [67], [70], [77], [79], [81], [85], [87], [88], [91], [97]	12
Раб, Магла, Облак	[42], [44], [80], [83], [84], [93], [94], [95]	8

Остварувањето на визијата за унифициран облак-раб екосистем има потреба како од напредни алгоритми за распределба на задачите, така и од платформи во кои би биле интегрирани ваквите нови алгоритми. Постојано треба да се има предвид деликатната рамнотежа која мора да се воспостави помеѓу работ и облакот. Иако работ нуди помало мрежно доцнење, сепак неговите перформанси се ограничени, па потребен е баланс за придобивките од помалото доцнење да не бидат изгубени како резултат на подолгото извршување. Во насока на ова, Жанг и др. објаснуваат начин за поделба на комплексните задачи [93] при анализа на видео протоци. Повеќе безсерверски функции се дефинирани во низа и одредени се точки на поделба. Сите функции пред точката на поделба се извршуваат на работ од мрежата, во близина на изворот на податоците, а оние после точката се извршуваат во облакот на помошна инфраструктура.

Од аспект на алгоритми за распределба на задачи, веќе се достапни решенија кои може да ги земат предвид слободните ресурси и на работ од мрежата и во облакот и врз основа на тоа да ја направат конечната одлука [67], [70], [97]. Во повеќе научни трудови се опишани и платформи кои може да бидат поставени низ целиот екосистем [77], [80], [81], [83], [84], [88], [91].

Во овој дел се издвојува платформата претставена во [94] која не само што може да биде поставена на повеќе локации истовремено, туку поддржува и различни извршни околини: виртуелни машини, контејнери, па дури и кластери за пресметување со високи перформанси (англ. high performance computing, HPC). Во овој контекст, сите различни околини кои се достапни во рамките на екосистемот се нарекуваат „пилоти“ и програмерите може да изберат на кој пилот би сакале нивната функција да биде извршена. Комуникацијата помеѓу функции извршувани на различни пилоти е овозможена преку брокери на пораки.

2.2.5. Безсерверски апликации

Друга тема за која е забележан голем интерес се безсерверските апликации. Ваквите апликации може да се однесуваат на сајбер физички системи, паметни градови, проширена и виртуелна реалност, но и многу други поврзани области. Иако нивото на техничка софистицираност варира, во сите случаи се користат или приватни или јавни безсерверски платформи, понекогаш поставени дури и на самиот раб од мрежата. Имплементацијата на вакви строго практични апликации дополнително служи како потврда за теоретските решенија кои честопати се предмет на анализа од страна на стручната литература, но, се разбира, и за идентификување целосно нови, претходно непредвидени проблеми.

Во табела 2.7 се прикажани сите трудови кои ја тангираат оваа тема, заедно со релевантните категории дискутирани во нив.

Табела 2.7
Трудови поврзани со темата „Безсерверски апликации“

Категорија	Трудови	Вкупно
Guidelines & Benefits	[15], [16], [18], [35], [37], [38], [39], [43], [44], [46], [47], [50], [54], [56], [61], [62], [79]	17
Public Infrastructure	[18], [35], [37], [49], [51], [54], [55], [57], [58], [59], [73], [77], [128]	13
Private Infrastructure	[18], [35], [37], [51], [53], [55], [57], [58], [76], [77], [128], [129]	12
Edge[10]	[15], [49], [51], [53], [55], [56], [57], [58], [59], [76], [129]	11
AR & VR	[37], [53], [55], [63], [64], [76], [77], [93]	8
AI & ML	[15], [37], [42], [73], [78], [79], [81]	7
Smart City	[42], [49], [54], [59], [76], [80]	6
CPS	[51], [57], [58], [128], [129]	5
MEC	[55], [63], [76], [78], [82]	5
Cloud[11]	[49], [51], [55], [59]	4
Fog[12]	[57]	1
Blockchain	[15]	1

Сајбер-физички системи

Сајбер-физичките системи (англ. cyber-physical systems) претставуваат средини во кои се очекува машините да имаат директна интеракција со околината, преку физички акции. Примери за вакви системи се: паметните струјни системи, паметни медицински помагала, самоуправувачки возила и беспилотни летала (дронови). Потребата од комуникација со исклучително мало доцнење ја прави безсерверската парадигма погодна за употреба во такви средини. Користењето безсерверски архитектури во контекст на сајбер-физичките системи го истражуваат Ган и др. [128], кои опишуваат апликација за контрола на јата од беспилотни летала при извршување некоја комплексна активност. Авторите нудат две имплементациски можности. Во првата централизирана

[10] Означува безсерверски апликации наменети за извршување на работ од мрежата.
[11] Означува безсерверски апликации кои може да се извршуваат и во облак, покрај работ на мрежата и во магла.
[12] Означува безсерверски апликации наменети за извршување во магла.

имплементација дроновите ги испраќаат податоците до единствена платформа која, пак, потоа ги анализира и ги сумира најзначајните информации од нив. Вториот пристап е децентрализиран, каде што пресметките се вршат на хардвер вграден во дроновите како форма на безуредско пресметување на работ од мрежата. Главниот недостаток на централизираниот пристап е потенцијалното преоптоварување на мрежата во случаи кога огромен број на дронови треба да бидат управувани од централизираниот сервер, а воедно и големото доцнење карактеристично за една таква топологија. Од друга страна, пак, користењето на децентрализираниот пристап би значело побрзо трошење на батеријата на самите уреди, бидејќи покрај моторите за летање сега дополнително би требало да се напојуваат и пресметковните компоненти. Ова би довело до помал опсег при летањето, па дури и отсуство на навремена реакција во екстремни случаи, кога процесорот би бил преоптоварен со пресметки врз генерираните податоци наместо со контролата на летање. Овие проблеми може да се надминат со компромис, така што едноставни задачи за кои не е потребна голема пресметковна моќ би се извршувале на самите уреди, намалувајќи го вкупното време на одговор. Спротивно на ова, комплексните функции би се изведувале во облакот. Компромисот помеѓу децентрализираните и централизираните архитектури е важен аспект, релевантен не само за сајбер-физичките системи, туку и за целиот интернет на нештата.

Намаленото доцнење што го нуди работ на мрежата може да биде критично во специфични сценарија. Авторите на [129] опишуваат имплементација на паметно нафтено поле кое користи безсерверско пресметување со цел анализа на податоците генерирани во реално време од страна на нафтените пумпи. Во такви несекојдневни околини, најчесто единствениот начин за добивање на интернет поврзување е преку сателитски врски. Искористувањето на безсерверското пресметување може да придонесе за надминување на предизвиците како што се зголемената латентност, цена, но и намалена достапност кои се асоцираат со овие врски.

Жанг и др. исто така претставуваат несекојдневна безсерверска апликација во [51], со компоненти кои работат и на работ од мрежата и во облакот. Целта на апликацијата е да се користи при вонредни ситуации со ограничен пристапот до останати услуги. Сите нивоа на целосното решение, почнувајќи од собирањето на податоците, преку анализата, складирањето и презентирањето кон крајните корисници преку веб страница, се имплементирани со безсерверски услуги, комбинирајќи ги FaaS и BaaS продуктите на AWS.

Паметни градови

И покрај тоа што границата помеѓу сајбер-физичките системи и одредени аспекти на паметните градови не секогаш е јасна во целост, заедничко за нив е големиот потенцијал за примена на безсерверското пресметување. Главната цел на паметните градови е да се олеснат секојдневните животи на луѓето преку воведување софистицирани IoT уреди во урбаните средини. Овие уреди би можеле да помогнат во однос на управувањето со сметот [49], поефикасното користење на електричната енергија [58], [57] или, пак, да го олеснат пристапот до јавниот транспорт [59].

Паметното рециклирање на отпадот е еден од примерите каде безсерверската парадигма може да понуди огромна скалабилност придружена со ниска цена на користење. При ваквите сценарија се врши периодично испраќање на мал број податоци, но од огромен број дистрибуирани сензори, во што и се состои вистинскиот предизвик. Авторите на [49] претставуваат апликација способна да изврши следење на количеството отпад во градските контејнери, а и истовремено да предупреди при забележување негова

погрешна распределба. Податоците се најпрво обработени на локалните уреди на работ од мрежата користејќи го комерцијалното Azure IoT Hub решение, а потоа се испратени до облакот за поопсежна анализа и долгорочно складирање.

Друго решение кое исто така може директно да влијае врз квалитетот на секојдневието на луѓето преку намалување на сообраќајниот метеж е претставено во [59]. Со користење комерцијални FaaS и BaaS услуги, се оптимизира јавниот превоз со детектирање метеж преку следење околни Bluetooth уреди во близина на постојките. Собраните податоци од уредите на работ од мрежата се испраќаат до BaaS база на податоци на секои 5 минути. Во периодот кога не се испраќаат податоци, релевантните функции се скалираат до 0 инстанци, искористувајќи ги во целост придобивките кои ги нуди безсерверското пресметување од аспект на заштеда на енергија и пресметковни ресурси. Дополнителен интересен пример исто така опишан во литературата е [42], каде што се објаснува употребата на безсерверски функции на работ од мрежата за детектирање дали некоја личност носи заштитна маска или не.

Користењето на безсерверската парадигма е исто така релевантна за енергетскиот сектор, особено поради можноста за ефикасна употреба на огромен број уреди. И во [58] и во [57] се опишуваат паметни апликации за управување со енергетски системи преку користење уреди како во облакот така и на работ од мрежата. Албајати и др. во [57] замислуваат имплементација на таков систем на ниво на цела држава преку модернизација на постојната инфраструктура за мерење на потрошувачката на електрична енергија. Авторите на [58] се надоврзуваат на ова и го имплементираат Toci, решение за детектирање аномалии во енергетските системи преку користење уреди поставени на работ на мрежата, управувани од Greengrass безсерверското решение на AWS. Слично како и претходно, така и во овој случај се користат безсерверски функции низ повеќе хиерархиски нивоа, па така собраните резултати се испраќаат кон AWS Lambda во облакот за дополнителна обработка. На овој начин се овозможува целосното решение, без исклучоци, да ја користи безсерверската архитектура, налик на [51].

Проширена и виртуелна реалност

Системите за проширена и виртуелна реалност се разликуваат од претходно опишаните, бидејќи крајните корисници имаат директна интеракција со нив и какво било доцнење при добивањето одговор влијае врз целокупното искуство при користењето. Дополнително, природата на задачите кои се извршуваат за поддршка на овие кориснички сценарија подразбира широка достапност на специјализиран хардвер, пример графички картички, кои би можеле да помогнат преку забрзување на извршувањето на одредени специфични задачи.

Во оваа насока, Салехе и др. се обидуваат да ги искористат постојните уреди веќе присутни во домаќинствата за дополнително промовирање на системите за проширена и виртуелна реалност [53]. Авторите презентираат платформа која преку користење на специфично дизајнирана JavaScript извршна околина може да извршува безсерверски функции на домашни уреди со цел поддршка на сценарија од проширена реалност/виртуелна реалност. Овие ограничени уреди (од аспект на перформанси) може да се класифицираат како уреди на работ од мрежата, а нивната моќност се зголемува како што расте бројот на достапни уреди. Со поделба на комплексните задачи на поединечни делчиња и нивно распоредување низ голем број уреди станува возможно имплементирањето поопсежни сценарија. Дополнително, во случај да има доволно ресурси, JavaScript околината која поддржува сериско извршување може да се замени со

контејнеризирано решение кое би овозможило изведување повеќе пресметки истовремено.

Идејата за користење на безсерверското пресметување за задачи од проширена/виртуелна реалност исто така ја истражуваат Бареси и др. [55], опфаќајќи и MEC аспекти. Авторите нудат целосно решение кое се потпира на можноста базната станица да го пренасочи барањето до првата достапна инфраструктура соодветна за негово извршување. За имплементацијата користена е OpenWhisk платформата. Споредбите со алтернативни пристапи кои би го употребувале облакот за извршувањето на пресметките покажуваат дека поместувањето кон работ од мрежата нуди значителни придобивки во однос на доцнењето, но и споредливи перформанси во однос на брзината на извршување.

Иако во литературата постојат примери за веќе имплементирани безсерверски апликации од областа на проширената и виртуелната реалност, сепак еден проблем сè уште останува нерешен: достапноста на специјализиран хардвер. Ниту една од платформите денес не поддржува користење графички картички при извршување безсерверски функции на работ од мрежата. Сепак, оваа тема активно се истражува, како што е дискутирано и во [81].

Миграција кон безсерверска архитектура

Безсерверските решенија се широко достапни веќе со години, но, сепак, идејата за нивна примена на работ од мрежата со цел опслужување задачи од интернетот на нештата е далеку понова. Имајќи го предвид огромниот број на апликации од оваа област, секоја со посебни побарувања во однос на цена, перформанси и доцнење, станува јасно дека се потребни убаво дефинирани патокази кои би помогнале при избор на соодветната стратегија за имплементирање на оваа парадигма врз уреди со ограничени перформанси. Пфандцелтер и др. [50] презентираат една таква рамка која може да помогне во дилемата која пресметковна парадигма да се користи при обработка на настани или анализа на големи податоци. Кога станува збор за првиот случај, оној за обработка на настани, препораката е да се користи инфраструктура која е што е можно поблиску до самиот извор на податоците. Авторите заклучуваат дека безсерверските платформи се очигледниот избор, сè додека нема потреба од комплексно управување со состојбата на поединечните функции. Со анализата на настаните на работ од мрежата се постигнува намалено доцнење, а притоа и се избегнува зголеменото време на извршување поради поскромните пресметковни перформанси, бидејќи настаните сами по себе се едноставни. Од друга страна, пак, кога станува збор за покомплицирани податочни протоци, се препорачува облакот, бидејќи побрзото време на извршување успева да ги надмине потенцијалните временски заштеди постигнати со пресметувањето на работ, како резултат на намаленото доцнење.

Дополнителна анализа во однос на придобивките и предизвиците при имплементирање на IoT безсерверско пресметување, со примарен осврт на работ од мрежата, нудат Асланпур и др. во [16]. Нивниот заклучок се поклопува со оној на останатите истражувачи, т.е. дека безсерверското пресметување на работ од мрежата е најдобриот избор кога станува збор за апликации без локална состојба кои примарно вршат анализа на настани од околни уреди. Атрактивноста на оваа парадигма се зголемува во ситуации кога генерираните настани се ретки, овозможувајќи ѝ на инфраструктурата да примени скалирање до 0 на функциите.

Во литературата често се посочува заклучокот дека иако придобивките од безсерверското пресметување се многу примамливи, сепак, се поставува прашањето

како да се изврши миграција на постојни софтверски решенија кон безсерверска архитектура. Авторите на [54] се обидуваат да дадат одговор на ова прашање со предлог насоки за преработка на постоен софтвер. Дополнително, Гросман и др. го споделуваат сопственото искуство стекнато при миграцијата на една традиционална апликација кон безсерверска архитектура во [56]. Главната цел е да се избегне одново пишување на целата програмска логика и да се реискористи што е можно поголем дел од постојниот код. На овој начин би се овозможило посветување повеќе време на извонредно важната оптимизација на решението. Пример за вакво подобрување претставува намалувањето на севкупната големина на функциите, што е особено важно, бидејќи традиционалните апликации имаат тенденција да користат голем број софтверски библиотеки кои придонесуваат кон зголемување на крајната големина на решението. За да се намали времето потребно за првично стартување кое се зголемува како што расте количеството код потребно да се вчита, а и за да се намалат трошоците во однос на складишен простор, Кристидис и др. предлагаат решение за минимизација на безсерверски функции [54]. Опишаниот пристап се темели на намалување на целокупната големина на софтверскиот артефакт кој ја претставува функцијата преку елиминирање на мртов код од програмските библиотеки додадени како зависности, т.е. бришење на код кој не се извршува при работа на функцијата. Имплементацијата е изведена со анализа на тоа до кои сè библиотеки навистина се пристапува при извршувањето и последователно бришење на функционалностите што не се користат во финалната репрезентација. Алтернативен пристап кој исто така доведува до намалување на големината на функциите е даден во [97], каде што авторите дискутираат начини како повеќе функции да бидат агрегирани во една целина, споделувајќи си ги меѓу себе заедничките библиотеки.

Очигледно е дека голем број на трудови се посветени на отсликување на придобивките на безсерверското пресметување, а и постојат многу безсерверски платформи каде што тие може и практично да се испробаат. Сепак, програмерите денес сè уште се соочуваат со проблеми при прилагодувањето на постојни и креирање комплетно нови апликации врз безсерверската архитектура. Вен и др. [47] се занимаваат токму со оваа проблематика и вршат анализа на прашањата од областа на безсерверското пресметување поставени на StackOverflow форумот за програмери. Резултатите покажуваат дека најголемиот дел од прашањата се однесуваат на конкретни имплементациски аспекти, а само 7.9% се општи прашања за основните безсерверски концепти. Преку ова може да се согледа дека денес програмерите веќе се имаат запознаено со основните концепти на безсерверската парадигма, но потребни се повеќе материјали кои би вклучувале поголем број на детали од аспект на конкретни имплементациски решенија и препораки.

2.2.6. Тестирање перформанси

Како резултат на големиот број безсерверски решенија достапни денес, се јавува потребата од тестови кои би овозможиле детална споредба, олеснувајќи го процесот на носење одлука која платформа е погодна во која ситуација. Имплементацијата на тестови дополнително се отежнува поради посебноста на секоја од платформите од една страна, но и потребата да се изврши детална анализа, користејќи ги сите карактеристични функционалности за неа. Во табела 2.8 е дадена распределбата на трудовите во однос на тоа за каков тип безсерверски платформи се наменети.

Табела 2.8

Трудови поврзани со темата „Тестирање перформанси“

Категорија	Трудови	Вкупно
Public Infrastructure	[52], [73], [75], [128]	4
Private Infrastructure	[74], [75], [128]	3

Тестирање комерцијални и отворени безсерверски платформи

Палад и др. [74] евалуираат безсерверски платформи со отворен код кои може да се поставаат на инфраструктура со скромни перформанси на работ од мрежата. Симулирањето на ограничените ресурси е направено со користење само два пресметковни јазла над кои е покренат Kubernetes контејнер оркестраторот. Врз него се инстанцирани Kubeless, OpenWhisk, OpenFaaS и Knative безсерверските платформи со отворен код за кои се испитуваат перформансите, времињата на одговор и ратата на успех при извршување на задачите. Авторите ја користат JMeter [130] алатката за софтверски да дефинираат тестни сценарија, симулирајќи IoT уреди кои постојано испраќаат податоци кон една од безсерверските платформи поставена на Kubernetes јазлите. Сите активности се изведуваат во една иста локална мрежа, со цел намалување на непредвидливите влијанија од користењето јавна, споделена, мрежа. Резултатите покажуваат дека Kubeless нуди најкратко време на одговор, висока рата на успех при извршување голем број функции во краток период, како и број на извршени задачи во единица време споредлив со останатите решенија.

Дас и др. во [75] исто се осврнуваат на анализа на перформансите на популарни безсерверски платформи, но овојпат од друга перспектива, фокусирајќи се на комерцијални решенија. Тестовите на работ од мрежата се изведени со користење на AWS Greengrass и Azure IoT Hub решенијата кои овозможуваат инсталација на соодветните извршни околини врз скромни уреди во сопственост на самиот корисник. На овој начин функциите од соодветните комерцијални FaaS услуги во облакот може да бидат едноставно пренесени на работ од мрежата. Ваквата стратегија овозможува директна споредба на резултатите добиени при извршување на работ од мрежата со оние добиени од извршување во облакот, бидејќи станува збор за целосно исти функции и единствената променлива е локацијата на извршување. Резултатите само дополнително ги потврдуваат очекувањата што и останатите истражувања на оваа тема ги дискутираат, дека работ на мрежата нуди драстично подобри перформанси во однос на мрежното доцнење споредено со облакот. Целата лепеза на користени тестови е јавно публикувана [131] и е поделена во три различни категории:

- конверзија на говор во текст;
- препознавање слики;
- симулирање на паметен сензор со периодично генерирање нумерички вредности.

Публикувањето на изворниот код за тестовите може да биде од голема помош на останати истражувачи кои би сакале да ги реискористат или во иднина дури и а ги прошират тестовите. Сепак, треба да се има предвид дека оваа задача е отежната од фактот што комерцијалните даватели на услуги имаат специфични програмски интерфејси за интеракција, што значи дека користењето врз некоја трета платформа, која не е опфатена од авторите, би побарувало дополнително прилагодување на изворниот код.

Ким и др. [73] имаат сличен пристап како оној прикажан во [75], со развој на свита од тестови за безсерверското пресметување, поделени во 4 различни категории. Тестовите како и претходно може да се извршуваат исклучиво врз комерцијални безсерверски

платформи. Она што е различно е дека дел од тестовите се микро тестови, т.е. не станува збор за целосна апликација што симулира дадено сценарио, туку за конкретна, најчесто добро позната, алатка која тестира само одреден аспект од пресметковниот систем, како на пример iperf или dd. Концептот на микро тестови е широко раширен во литературата и овозможува тестирање на суровите хардверски перформанси на уредите над кои е инсталирана платформата, како на пример пропусната моќ на мрежата, број на влезно/излезни операции во единица време, број на извршени инструкции, брзина на работна меморија и др. Тестовите во конкретниот случај се наменети исклучиво за облакот и не е поддржана ниту една платформа за безсерверско пресметување на работ од мрежата, но позитивно е што како и претходно, изворниот код е јавно достапен под лиценца која дозволува адаптација [132].

Една широко-присутна дилема во денешната литература на полето тестирање и евалуација на перформанси е дали да се користат микро тестови или комплетни апликации врз чиишто севкупни перформанси би имале влијание повеќе аспекти. Користењето на целосни апликациски сценарија подобро ги отсликува реалните задачи кои би се извршувале над платформите, но, пак, го прави потешко идентификувањето тесни грла. Од друга страна, микро тестовите овозможуваат директен увид врз карактеристиките на користениот хардвер и нудат можност за полесно откривање на ограничувањата, иако добиените резултати не секогаш би биле еквивалентни и со оние од конкретните кориснички сценарија.

Авторите на [128] нудат група на тестови составена од 5 конкретни апликации со вистинска примена во реалниот свет со цел евалуација на перформансите на безсерверските платформи. Апликациите се однесуваат на различни категории, како: системи за плаќање, е-трговија, социјални мрежи и координација на голем број беспилотни летала.

Горлатова и др. [52] исто така ја образложуваат потребата од дефинирање стандардизирани, повторливи свити од тестови кои би можеле да се извршуваат на различни платформи, како денес, така и во иднина. Нивните мерења опфаќаат 6 различни локации на работ и во облакот. Конкретниот пристап е поразличен од досега дискутираното, бидејќи мерењата ги вршат од различни физички локации, а не се потпираат на симулација на географска оддалеченост. Заклучоците се насочени кон проблемот на ладен старт и покажуваат дека во најекстремните случаи со недоволно оптимизирани платформи, овој ладен старт може да придонесе до дури 40 пати зголемување на времето на извршување на функцијата.

Најдобри практики за миграција и придобивки

Анализата на литературата откри голем број на трудови кои покрај тоа што го опишуваат начинот за имплементација на одредена нова функционалност или подобрување на постоечка, нудат и резултати од тестирања со цел поткрепување на тврдењата. Ова е секако корисно и дополнително придонесува за севкупното знаење од оваа област и можностите на поединечните решенија. Сепак, очигледна е огромната потреба за користење стандардизирана рамка за изведување на тестовите, наместо постојано дефинирање целосно нови тестови, со што и се отежнува меѓусебното споредување на резултатите. Ова не е проблем кој може да биде самостојно надминат од истражувачката заедница, туку потребна е и соработка со давателите на услуги. Од исклучително значење за понатамошниот развој на безсерверскиот екосистем е дефинирањето на стандардизиран интерфејс за комуникација компатибилен со сите безсерверски платформи. На овој начин би се овозможило еднаш дефинирана свита од тестови да биде

инстанцирана на сите постојни и идни безсерверски платформи, без потреба од рачно прилагодување и несакано воведување ентропија.

2.2.7. Безбедност, интегритет и регулативи кај безсерверското пресметување

Придобивките од користењето на безсерверската парадигма се неспорни кога станува збор за леснотијата на додавање нова функционалност и инфраструктурното скалирање на апликацијата како што расте нејзината популарност. Сепак, иако драстично се олеснува работата на програмерите, мора да се обрне внимание и на безбедносните прашања. Ова е од особена важност кога станува збор за споделени околини од различни корисници кои не може да си веруваат меѓу себе, типично за јавните облаци и за работ на мрежата. Моменталното истражување на ова поле се обидува не само да ги минимизира заканите од малициозни функции поставени од злонамерни корисници, туку и да ја ограничи штетата предизвикана од потенцијален злонамерен администратор на платформата. Слично како во ситуацијата со недостатокот од унифициран интерфејс за комуникација со различните безсерверски инфраструктури, така и во оваа област сè уште не е постигнат консензус во однос на најдобрите безбедносни практики кои би требало да бидат имплементирани како од програмерите, така и од платформите кога станува збор за безсерверското пресметување.

Во табела 2.9 е даден преглед на релевантните трудови од областа на безсерверската безбедност, интегритет на податоците и регулативите.

Табела 2.9
Трудови поврзани со темата „Безбедност, интегритет и регулативи кај безсерверското пресметување“

Категорија	Трудови	Вкупно
Vendor Lock-in	[36], [46], [83], [97]	4
Trustworthiness	[46], [96], [99], [133]	4
Data Repair	[95]	1

Проблем со ограничената компатибилност меѓу давателите на услуги

Еден од најголемите проблеми со кои се соочува безсерверското пресметување денес е ограничената компатибилност меѓу различните даватели на услуги и неможноста за лесно комбинирање на производи меѓу нив. Ова доведува до сценарија каде што корисникот е заклучен во екосистемот на неговиот давател на услуги (англ. vendor lock-in) и какво била миграција кон алтернативно решение бара големо време и човечки ресурси. Иако унифициран и сеопфатен интерфејс за комуникација кој би ги обединил различните решенија сè уште не постои, веќе се вложени иницијални напори и придобивките од нив се видливи на ова поле. Serverless.com [134], [135] овозможува превод на безсерверски функции со цел тие да се извршуваат на различни инфраструктури. Нуди корисничка алатка преку која функциите може да се поставуваат на која било од поддржаните платформи преку унифицирани наредби. Се разбира, ова не е замена за официјална поддршка на унифициран интерфејс од страна на сите даватели на услуги, но е чекор во вистинската насока. Очигледниот проблем е тоа што преку ваков превод не се искористуваат специфичните функционалности на платформата, туку само оние најчестите, со свои еквиваленти и кај останатите платформи.

Најновиот тренд во оваа област е да се отиде чекор понатаму и не само да се понуди унифициран интерфејс за комуникација, туку и тој да се користи за интелигентна распределба на задачи на различни платформи во исто време. Пеле и др. во [97] опишуваат еден таков слој за апстракција што може да се користи за дефинирање

безсерверски функции кои потоа би се поставувале на различни комерцијални безсерверски платформи. Авторите на [83] дефинираат систем за прилагодување на постојни функции кои би можеле потоа да се поставуваат на различни инфраструктури во облакот и на работ од мрежата, што ги прави овие средишни слоеви за компатибилност само уште една алка во ланецот за овозможување непрекинат облак-раб екосистем.

Не чекајќи на конечен слој за компатибилност помеѓу различните инфраструктури, денес постојат и такви платформи кои едноставно ги реимплементираат програмските интерфејси користени од страна на популарните комерцијални решенија. На овој начин се овозможува директна пренамена на постојни функции со цел нивно извршување на новата платформа. Пример за ова е CSPOT [87], [112] кој директно го поддржува AWS Lambda форматот.

Каталози за функции

Дополнителна придобивка од имплементирањето заеднички и унифициран слој за инстанцирање функции низ повеќе даватели на услуги истовремено е можноста за нудење на каталози на функции. Овие каталози може да содржат безсерверски функции поставувани од корисниците кои со неколку клика може да бидат инстанцирани на инфраструктура по избор, користејќи ги предностите на заедничкиот интерфејс за комуникација. Комбинирањето повеќе различни мали функции кои вршат една задача, но се специјализирани за неа, во една голема целина би го олеснила процесот на развој на нови решенија. Секако, ваквите каталози за функции не значи дека ќе нудат исклучиво бесплатни решенија и останува на самите програмери кои би ја поставиле сопствената функција да одредат модел за наплата. Ова може да доведе и до дополнителни иновации во однос на користените бизнис модели, каде што на пример оригиналниот автор би добивал средства за секое извршување на функцијата, на која било платформа.

Првите каталози за безсерверски функции кои може да служат како места за размена и влечење инспирација за идни решенија се веќе достапни [136], но за жал, во моментот тие се ограничени на конкретна платформа, без можност за поставување на функциите содржани во нив на различни инфраструктури.

Изолација и безбедност

Во секоја ситуација каде што се јавува реискористување на туѓ код веднаш се поставува прашањето на безбедност, па и каталозите со функции не се исклучок од ова правило. Не е тешко да се замислат малициозни функции кои би биле поставени на ваквите јавни каталози, со цел да се пробие платформата на која тие би се покренале или, пак, да се украдат податоците со кои би дошле во допир. Дата и др. во [96] нудат решение на овој проблем со претставување на Valve решението. Станува збор за безбедносна рамка за безсерверски функции која може да ги одреди очекуваните услови во кои се извршуваат задачите и да применува различни регулативи со цел одржување на околината во непроменета состојба. Концептот на воспоставување очекувана состојба и нејзиното понатамошно одржување со спречување каква било активност која би довела до нејзино нарушување е познат и од други области, како на пример SELinux рамката кај GNU/Linux оперативните системи.

Треба да се земе предвид дека иако нивото на безбедност се зголемува на овој начин, примената на комплексни правила би можело да доведе до намалување на перформансите при извршување на функциите, а и да предизвика зголемување на нивната целокупна големина, како резултат на дополнителните компоненти кои мора да се дефинираат покрај функцијата. Во денешната литература се спомнува и пристап со

осигурување на безбедноста при извршување на безсерверските функции, во овој случај многу поблиску до хардверот, со користењето на Intel SGX [99], [133]. Ова решение може да помогне во заштитата и изолацијата меѓу различни извршни околини кои би се користеле за функции поднесени од непознати корисници. Intel SGX работи на принцип на шифрирање и дешифрирање делови од меморискиот простор по потреба за време на извршувањето на дадена задача.

Авторите на [46] вршат преглед на безбедносните аспекти кои влијаат врз безсерверското пресметување и истражуваат начини како да се постигне повисоко ниво на безбедност. Нивниот придонес е преставувањето на изменета JavaScript извршна околина која може да гарантира изолација меѓу функциите извршувани во неа.

Интегритет на податоци

Веќе прифатениот начин за извршување на повеќе поврзани безсерверски задачи преку надоврзување, каде што излезот од една би бил влез во друга функција, го поставува прашањето околу интегритетот на податоците. Потребно е осигурување на нивниот интегритет со цел елиминација на сомнежот за каква било неовластена промена или корупција на податоците додека тие транзитираат од една функција кон друга. Ова е од големо значење за IoT сценарија, каде што за неповторливите пресметки во реално време е од особена важност да не се случи грешка како резултат на лоши податоци, бидејќи во тој случај би дошло до губење на сите меѓурезултати и неможност за повторување на пресметките. SANS-SOUCE [95] е додаток за претходно споменатото CSPOT решение [87] и осигурува интегритет на податоците преку бележење на сите информации разменети од функциите на начин кој подразбира само додавање нови записи, а не и бришење (англ. append-only log). На овој начин, дури и да се појават грешки при обработка на податоците длабоко во ланецот на извршување, сите пресметки може да бидат повторени само со наново изминување на белешките. Дополнителна предност е тоа што истите записи може во иднина да се користат и со изменети податоци, задржувајќи ја истата постапка.

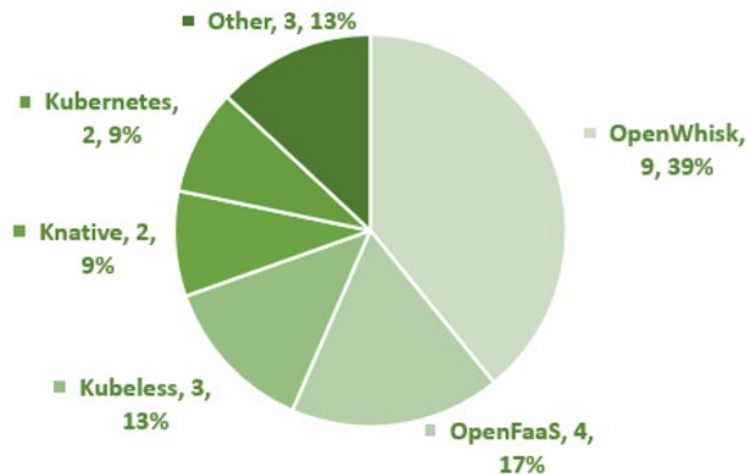
2.2.8. Решенија со отворен код

Како што беше напоменато претходно, голем дел од истражувачките трудови кои се фокусираат на развој на нови алгоритми за распределба на задачи или дизајнирање нови безсерверски решенија, во основата користат некоја од постојните безсерверски платформи со отворен код.

Со цел добивање појасна слика за ситуацијата на ова поле, во табела 2.10 е дадена класификација на избраните трудови во однос на тоа дали се користи постојна безсерверска платформа или, пак, се дизајнира целосно ново решение кое потоа се објавува под отворена лиценца.

Табела 2.10
Трудови поврзани со темата „Решенија со отворен код“

Категорија	Трудови	Вкупно
Based on an OSS Platform	[36], [42], [46], [55], [56], [63], [65], [68], [71], [72], [74], [76], [77], [78], [79], [80], [81], [86], [89], [92], [95], [96], [99], [133]	24
Published as OSS	[17], [42], [56], [61], [65], [73], [75], [77], [82], [84], [85], [87], [89], [90], [95], [128]	16



Слика 2.2 Најпопуларни безсерверски платформи со отворен код

За првата категорија на трудови, оние кои реискористуваат постојни безсерверски платформи при сопствената имплементација, извршена е подетална анализа, со цел да се открие кое е најприсутното решение со отворен код. На слика 2.2 е даден приказ на застапеноста на петте најпопуларни безсерверски платформи со отворен код, според тоа во колкав дел од трудовите тие се јавуваат. Дополнително, табела 2.11 ги содржи сите трудови кои се однесуваат на дадена платформа со отворен код, што може да послужи за пронаоѓање сродни решенија.

Анализата покажува дека OpenWhisk е најпопуларната безсерверска платформа врз која потоа се темелат девет дополнителни решенија сретнати во литературата. Ова не е воопшто изненадувачки, бидејќи дури и комерцијални платформи нудат решенија компатибилни со OpenWhisk, како што е примерот со IBM Cloud Functions [8]. Следна отворена платформа по популарност е OpenFaaS, која исто така претставува флексибилно решение со поддршка за голем број на програмски јазици за пишување на функциите. Kubeless и Knative се трети и четврти по популарноста, соодветно, а заедничкото за нив е тоа што се исклучиво Kubernetes решенија и побаруваат Kubernetes кластер за подигнување. Ова е спротивно на алтернативите кои се пофлексибилни кога станува збор за инфраструктурата каде што се извршуваат и нудат поддршка за различни архитектури. За крај, Calvin, FogFlow и TinyFaaS се сите искористени само по еднаш во корпусот на трудови и станува збор за решенија дефинирани од академската заедница, а потоа подобрени од други истражувачки групи со цел да понудат извршување на безсерверски функции на работ од мрежата.

Анализирајќи колку вкупно труда имаат некаква поврзаност со отворен код и пошироката заедница, 40 од 67 имаат директно или индиректно придонесено за понатамошниот развој во ова поле. 24 труда вршат подобрување на постојни безсерверски платформи со отворен код, додека 16 труда вршат нова имплементација на безсерверско решение кое потоа го објавуваат под слободна лиценца.

Табела 2.11

Анализа на користените безсерверски платформи со отворен код

Име на платформа	**Користена во**	**Вкупно**
OpenWhisk	[55], [63], [65], [68], [74], [76], [77], [78], [99]	9
OpenFaaS	[56], [74], [84], [96]	4
Kubeless	[71], [74], [81]	3
Knative	[72], [74]	2
Kubernetes	[42], [86]	2
Calvin	[92]	1
FogFlow	[80]	1
TinyFaaS	[90]	1

2.3. Отворени прашања и насока на истражување

Со направениот преглед на литературата се докажува огромниот интерес кој го уживаат безсерверските платформи денес, како и тенденцијата за нивно проширување и кон работ од мрежата. Сепак, постојат голем број на отворени прашања на кои ќе треба да се даде одговор со цел издигнување на безсерверската парадигма на едно сосема ново ниво и овозможување транспарентни пресметки во облак-раб екосистемот.

Воспоставувањето робусен облак-раб екосистем во пракса побарува оптимизации во поглед на ефикасноста на различни безсерверски извршни околини, нивните перформанси и побарувања при преместување кон работ, како и можностите за префрлување на податоците од една кон друга локација. На овој начин, меѓу другото, би се надминало и големото доцнење при ладен старт кое е особено значајна пречка за уредите со скромни пресметковни карактеристики од работ на мрежата. Дополнително, какви било подобрувања во однос на перформансите мора да се разгледуваат со унифицирани свити од тестови со поддршка за различни платформи со цел релевантна споредба.

Актуелната литература се фокусира на создавање облак-раб екосистеми од хомогени решенија, понудени од страна на еден давател на услуги. За избегнување на ефектот на заклучување, сè поприсутен кај давателите на услуги, потребно е повеќе различни инфраструктури, без разлика на нивната локација или потекло, да учествуваат во една иста, унифицирана целина. Ваквиот пристап побарува и сеопфатна анализа на можните решенија за вмрежување помеѓу хетерогените инфраструктури, со што би се овозможила вистинска географска транспарентност, безбедност, но и скалабилност преку можноста за лесно проширување и додавање дополнителен пресметковен капацитет во унифицираниот екосистем.

Овие отворени прашања се во тесна врска со претходно поставените хипотези и поглавјата во продолжение ќе бидат посветени на нивно решавање, а сè со една единствена цел, анализа и дизајн на повеќенаменска архитектура на безсерверски платформи за транспарентни пресметки во облак-раб екосистемот.

3. АНАЛИЗА НА ПОСТОЈНИ БЕЗСЕРВЕРСКИ ПЛАТФОРМИ

Направениот преглед на литературата дискутиран во претходното поглавје покажа дека постојат голем број на достапни безсерверски платформи со различна намена. Сепак, кога станува збор за нивно покренување на работ од мрежата, неопходно е да се направи формална поделба во однос на начинот на поставување, перформансите и функционалностите кои ги нудат. Работ од мрежата значително се разликува од облакот во однос на достапниот пресметковен капацитет, мрежното поврзување и начинот на користење. Се поставува прашањето дали општоприфатените начела кои важат при извршувањето безсерверски функции на облакот исто така важат и на работ или, пак, потребно е одредено прилагодување во поглед на користените технологии. Ова всушност води кон поставување на хипотезата *„Моменталните извршни околини кои се користат за покренување безсерверски функции не ги задоволуваат барањата од аспект на брзина на извршување и доцнење при нивната примена на работ од мрежата. Потребна е оптимизација на постојните или примена на целосно нови алтернативни извршни околини за надминување на овие проблеми, што би довело и до задоволување на барањата кои ги поставуваат IoT уредите со своите пресметковни задачи засновани на настани.“* Одговорите на овие прашања се од суштинско значење за понатамошното истражување во однос на идните платформи на работ од мрежата, но и општо за безсерверското пресметување, со цел апстрахирање на местото на извршување (работ наспрема облакот и обратно).

Со цел емпириско испитување на можностите на постојните безсерверски платформи на работ од мрежата, ќе биде применета следнава дистинкција:

- Еднојазлени платформи – платформи со ниско ниво на комплексност, наменети за поставување на еден единствен јазол, без можност за истовремена оркестрација на задачи низ повеќе локации.
- Повеќејазлени платформи – платформи со повисоко ниво на комплексност, наменети за користење врз повеќе јазли истовремено, со напредна логика за распределба на задачите и ресурсите низ различни локации.

Првиот тип на платформи, оние со пониско ниво на комплексност се всушност решенија кои се поставуваат на еден единствен уред, најчесто со ограничени перформанси и се користат во ситуации кога нема потреба од висок пресметковен капацитет. Во зависност од архитектурата, ваквите решенија може и директно да бидат инсталирани врз крајни уреди кои генерираат податоци, наместо воведување хиерархиска структура каде што изворот на податоците е целосно независен од инфраструктурата за обработка.

Вториот тип на платформи, каде што се воведува и концептот на кластерирање и оркестрација, се наменети за инфраструктури на работ од мрежата кои најчесто имаат за задача за бидат користени од повеќе корисници истовремено, нудејќи можности за поделба на ресурсите, изолација и извршување со повисоки перформанси во споредба со првото решение.

Кога станува збор за безсерверското пресметување на работ од мрежата, двете категории на платформи се важни и имаат своја примена. Дополнително, без разлика на избраниот пристап, воспоставувањето облак-раб екосистем со селективна употреба на облакот за комплексни пресметки сè уште постои како можност.

Целта на ова поглавје е да се направи испитување на постојните решенија наменети за индивидуално поставување на еден уред, но и на најпопуларните безсерверски платформи кои овозможуваат извршување на безсерверски функции низ целосна,

покомплексна инфраструктура. Посебен осврт се дава на начините како секое од решенијата се справува со отворените проблеми идентификувани претходно, на пример со ладниот старт, ефикасноста при извршување, големината на функциите и нивото на изолација која го нуди извршната околина. Изборот на решенијата за анализа е направен внимателно, со цел вклучување како на платформи со отворен код, така и на комерцијални продукти, правејќи споредба меѓу нив. Ваква споредба помеѓу комерцијални и отворени решенија е ретка во постоечката литература, имајќи предвид дека трудовите од оваа област се фокусираат исклучиво на едната или другата категорија [22], [23].

3.1. Избор и адаптација на свита за безсерверски тестови

Главниот предизвик поврзан со прашањето за евалуација на какви било перформанси на даден систем е повторливоста на тестовите и веродостојноста на резултатите во однос на нивната способност за прецизно доловување на реалната слика за решението. За надминување на овие проблеми во областа на безсерверското пресметување, денес постојат неколку свити на тестови кои вклучуваат како микротестови, така и целосни имплементации на конкретна апликација која би можела да се користи за испитување на перформансите, како што беше дискутирано во „2.2.6.1. Тестирање комерцијални и отворени безсерверски платформи“. Сепак, треба да се има предвид дека една голема отежнителна околност при евалуација на перформансите на различни решенија е меѓусебната некомпатибилност која е резултат на користењето специфични комуникациски интерфејси. Ваквата поставеност на работите води кон заклучокот дека е потребно користење отворена свита на тестови која би била комбинација од микро тестови и целосни апликации, а достапноста на изворниот код би овозможила рачно адаптирање и додавање поддршка за безсерверските решенија кои тековно се изоставени.

Со цел да се овозможи стандардизирано тестирање низ целата дисертација, адаптирана е FunctionBench свитата на тестови опишана од Ким и др. во [73] и јавно публикувана со отворена лиценца [132]. Иако иницијално наменета за комерцијални безсерверски платформи во облакот, сепак, нуди можност за лесна адаптација кон трети решенија преку едноставно менување на читливиот изворен код.

3.2. Анализа на еднојазлени безсерверски платформи на работ од мрежата

Еднојазлените безсерверски платформи наменети за поставување на работ од мрежата имаат за цел да ги задоволат пресметковните потреби на едноставни сценарија каде што нема техничка потреба, но ни финансиска оправданост, од користење покомплицирани решенија. Дополнителна предност на еднојазлените платформи е тоа што тие најчесто се редуцирани верзии на веќе постоечки безсерверски решенија наменети за облакот, па нудат компатибилност со готови функции.

3.2.1. Избор на платформи за тестирање

Главниот критериум за избор на решенија кои ќе бидат вклучени во евалуацијата е можноста за нивно независно поставување на инфраструктура, без потреба од користење на повеќе јазли или кластерирање со пресметковни ресурси во облакот. Ваквиот пристап води до елиминација на познатите OpenWhisk [10], [137] и OpenFaaS [11] отворени безсерверски решенија, бидејќи нивното покренување подразбира употреба на комплексни платформи за оркестрација на контејнери како што се Docker Swarm или Kubernetes.

Еднојазлено решение кое ги задоволува изнесените критериуми е отворениот FaasD [138], [139, p.] што е всушност поедноставена верзија на OpenFaaS наменета за поставување на работ од мрежата. Станува збор за релативно ново решение кое подлежи на активен развој.

Како претставници на комерцијални решенија за безсерверско пресметување на работ од мрежата, избрани се AWS Greengrass [20] и Azure IoT Hub [21], првенствено како резултат на нивната голема популарност што директно има придонесено и за големиот интерес од страна на академската заедница и нивното вклучување во различни трудови. На овој начин се овозможува лесна споредба на добиените резултати од спроведеното тестирање со оние добиени од страна на други автори. Дополнителна придобивка е тоа што користената свита од тестови, FunctionBench, првично е развиена токму за AWS Lambda и Azure Functions, што се соодветните алтернативи во облак на овие две решенија. Имајќи го ова предвид, користењето на редуцираните верзии овозможува и проучување два дополнителни аспекта:

- нивото на компатибилност помеѓу верзијата на понуденото решение во облакот и онаа на работ;
- комплексноста за правење адаптации врз постојната свита од тестови, со цел евентуална поддршка на слични безсерверски околини како оние кои се веќе поддржани.

Во продолжение се дадени основни информации за избраните решенија поврзани со нивната архитектура, кои потенцијално би имале влијание врз добиените резултати.

FaasD

FaasD [138], [139] го поддржува истиот формат на функции како и оригиналниот OpenFaaS. Поедноставена верзија со помали ресурсни побарувања е добиена преку елиминирање на одредени функционалности. FaasD нема вградена можност за автоматско скалирање на функциите на повеќе од една инстанца или, пак, за автоматско скалирање до нула инстанци при неактивност. Изоставањето на каква било компонента која би понудила оркестрација на контејнери спречува паралелно покренување на повеќе инстанци од истата функција, додека пак скалирање до нула инстанци може да се изведе со експлицитни повици до FaasD програмскиот интерфејс.

Сите функции се инстанцирани со покренување Docker слики кои е потребно да бидат веќе достапни во моментот на дефинирање на функцијата. За олеснување на процесот на градење на сликата, FaasD заедницата има изготвено низа на шаблони [140], [141], [142], кои може да се користат за пакување готова функција напишана во некој од поддржаните програмски јазици. Од кориснички аспект, понудени се два начина за повикување на функциите, синхрон и асинхрон. При синхроното извршување, HTTP одговорот се враќа откако функцијата ќе заврши со процесирање, а барањето, од перспектива на корисникот, се блокира додека траат пресметките. Асинхроното повикување претставува комплетна спротивност и се темели на повратен HTTP повик (англ. callback) кој би го извршувала FaasD платформата кон надворешна дестинација наведена од повикувачот на функцијата, по завршување на процесирањето.

Greengrass

AWS Greengrass [20] е продукт на Amazon Web Services кој овозможува извршување AWS Lambda функции на работ од мрежата врз уреди во сопственост на самите корисници. Главната придобивка, на која се должи и големата популарност, е можноста за директна пренамена на постојни Lambda функции извршувани во облакот и нивно

преместување на работ, без никакви дополнителни промени во изворниот код. Задачите може да се извршуваат или како контејнери или директно нативно врз уредот, без воведување дополнителни нивоа на апстракција. Слично како и Lambda решението во облакот, возможно е стриктно ограничување на пресметковните ресурси доделени на дадена функција, како процесорско време и работна меморија. Поддржано е и автоматско скалирање на бројот на инстанци на дадена функција и нивно намалување сè до нула, во зависност од тековното оптоварување, без потреба од каква било рачна интервенција од страна на администраторот. Иако ваквото однесување е корисно и ја олеснува работата, треба да се има предвид дека последователни функции може да реискористат постоечка извршна околина. Придружено со малата контрола на администраторот од аспект на изолација и паралелизам, ова може да доведе до нарушување на безбедноста на податоците обработувани од функциите.

Препорачаниот начин за комуникација меѓу Lambda функции е преку предавање пораки, со користење на самиот Greengrass уред на работ од мрежата како брокер на пораки или, пак, на соодветна брокер инстанца во облакот. За разлика од FaasD, Greengrass не побарува користење Docker слики за инстанцирање на функциите и употребува стандардни архивски датотеки, со строго ограничување на нивната максимална големина, што може да претставува проблем за покомплексни сценарија со голем број на зависности во форма на програмски библиотеки.

Azure IoT

Azure IoT Hub [21] е Microsoft продукт за извршување на безсерверски функции на работ од мрежата врз уреди во сопственост на корисниците. Слично на FaasD, се потпира на користење контејнери како извршна околина, употребувајќи Docker слики за размена и подигнување на функциите. Не постојат никакви ограничувања при пишувањето на безсерверските функции и нема потреба од вклучување дополнителни библиотеки за интеграција со самата платформа. Како резултат на ова, размената на пораки е само еден од поддржаните комуникациски механизми кој може да се реализира со користење локален или, алтернативно, централизиран брокер во облакот. Платформата нема директна поддршка за хоризонтално скалирање на функциите со покренување повеќе инстанци истовремено, ниту за паузирање на оние инстанци кои не се активно користени. Како последица на ова, останува на самите програмери да управуваат со паралелизмот преку воведување поддршка за нишки и/или процеси во функцијата која ја пишуваат.

Директна компатибилност со соодветното безсеверско решение на Microsoft во облакот е возможно само за функции напишани во C# програмскиот јазик. Во останатите случаи, потребно е програмерот да изгради сопствена Docker слика која потоа ќе се покрене во форма на контејнер на Azure IoT Hub уредот. Интересно, со цел да се намали доцнењето, постои можност за локално покренување на Azure Blob услугата за складирање како и на Azure релациона база на податоци, продукти кои вообичаено би требало да се користат како BaaS решенија во облакот.

Azure IoT Hub ги намалува последиците од проблемот на ладен старт, бидејќи не поддржува скалирање до 0 инстанци и неактивните функции продолжуваат да се извршуваат и кога нема дојдовни барања.

3.2.2. Методологија на извршување тестови

За тестирање на перформансите се користат претходно опишаните 14 безсерверски функции, плус уште една дополнителна за директно мерење на мрежниот пропусен

опсег со помош на iperf3 алатката. Оригиналните верзии на тестовите се адаптирани за извршување на секоја од вклучените платформи. Сите BaaS решенија се заменети со алтернативи покренати во локалната мрежа за да се елиминира негативното влијание од фактори врз кои се нема директна контрола, како оптоварување на самите услуги во облакот или мрежното доцнење во одредени сегменти.

Потребата од поединечна адаптација на секоја од функциите за трите платформи, со индивидуално пакување и дистрибуција на кодот резултира со креирање вкупно 45 различни софтверски артефакти. Ваквиот комплициран процес на споредба само дополнително ја потврдува горливата потреба од унифициран интерфејс за креирање, покренување и распределба на безсерверски задачи низ хетерогени платформи.

Во табела 3.1 се дадени повеќе детали околу конкретните влезни параметри на секоја од функциите во свитата.

Табела 3.1

Тест параметри и нивен опис

Категорија	Име на тест	Параметри	Цел
Процесор, меморија	Float Operation	Случајно генериран број	Чести операции
	Matmul	Број (големина на матрица)	Множење матрици
	Chameleon	500x500 (редици и колони)	Креирање HTML табели
	Image Processing	Линк кон онлајн папка	Трансформација на слика
	Linpack	Број (големина на матрица)	Линеарни равенки
	PyAES	1000, 100 (должина, итерации)	AES операции
	Model Training	Линк кон онлајн папка	Тренирање модели
	Video Processing	Линк кон онлајн папка	Енкодирање видео
Мрежа	iperf3	IP адреса & времетраење	Пропусен опсег
	JSONDumpsLoads	URL кон JSON датотека	Манипулација врз JSON
	Object Download	Линк кон онлајн папка	Преземање онлајн датотеки
Складиште	Dd	100M, 1 (блок & број блокови)	Dd брзина на диск
	Random I/O	100, 1024 (големина датотеки)	Случаен В/И со Python
	Sequential I/O	100, 1024 (големина датотеки)	Секвенцијален В/И со Python
	Gzip	50MB (големина датотека)	Gzip компресија

Со цел да се осигура конзистентноста на резултатите, сите функции за кои е потребно да се внесе број како влезен параметар се повикуваат со истото множество на случајно избрани броеви, низ сите платформи. Вредностите на останатите параметри се избрани во насока да се постигне компромис помеѓу пресметковната комплексност на задачата и вкупното време на извршување, симулирајќи чести активности кои би имало потреба да се извршуваат на работ од мрежата.

Сите три избрани платформи се покренати во x86 виртуелни машини, секоја со по 2 виртуелни процесора и 4 гигабајти работна меморија, отсликувајќи уреди со ограничени перформанси вообичаено достапни на работ од мрежата. Секој тест беше извршен на три различни начини:

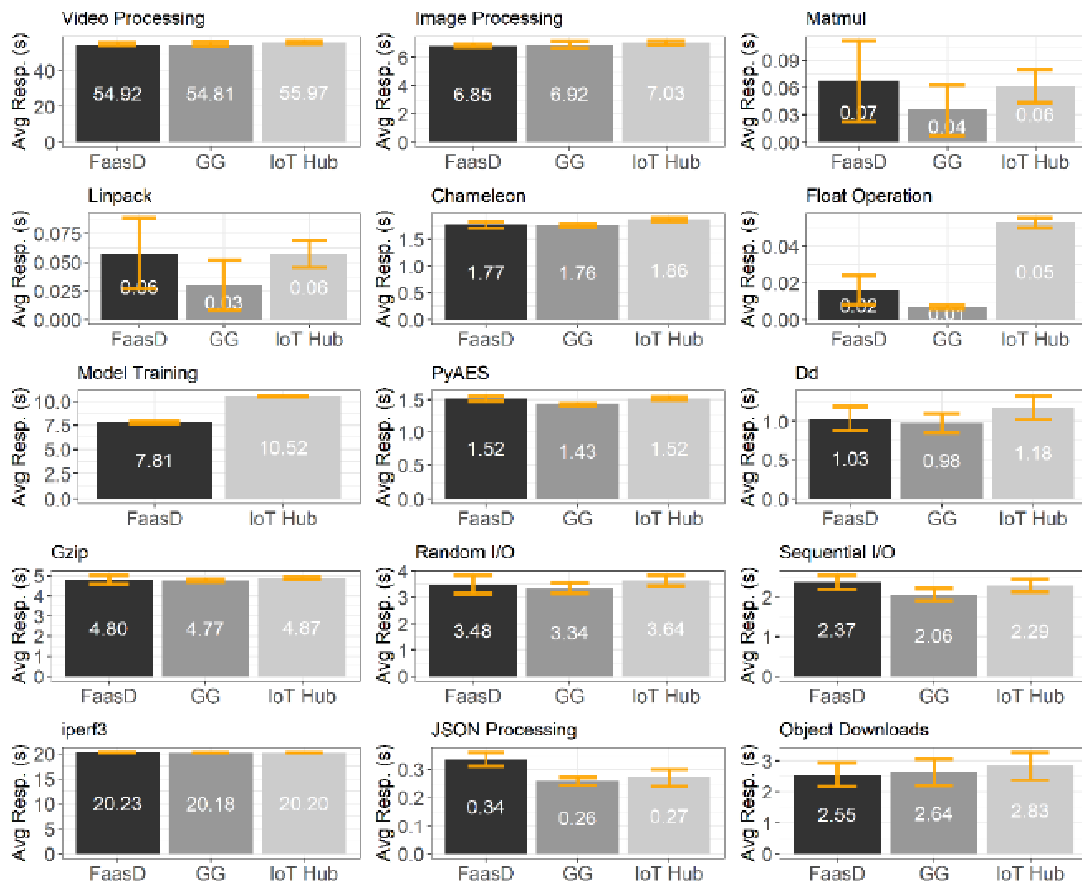
1. синхроно сериско извршување, симулација на едно барање во даден момент;
2. паралелно извршување, симулација на до 100 истовремени барања;
3. сериско извршување, но со функцијата скалирана до 0 инстанци, со цел експлицитно тестирање на доцнењето при ладен старт.

Сите тестови, низ сите начини на извршување беа повикани: 1, 5, 10, 25, 50, 75 и 100 пати. Во сите случаи беше мерено вкупното доцнење, а не само доцнењето при извршување. На овој начин, со земање предвид на вкупното доцнење, исто така беше овозможена споредба меѓу ефикасноста на различните комуникациски протоколи поддржани од секоја од платформите.

3.2.3. Споредбена анализа

Иако секоја од трите платформи ја задоволува примарната цел – покренување безсерверски функции на работ од мрежата врз сопствена инфраструктура, начинот на имплементација и крајните перформанси драстично се разликуваат.

Синхроно сериско извршување

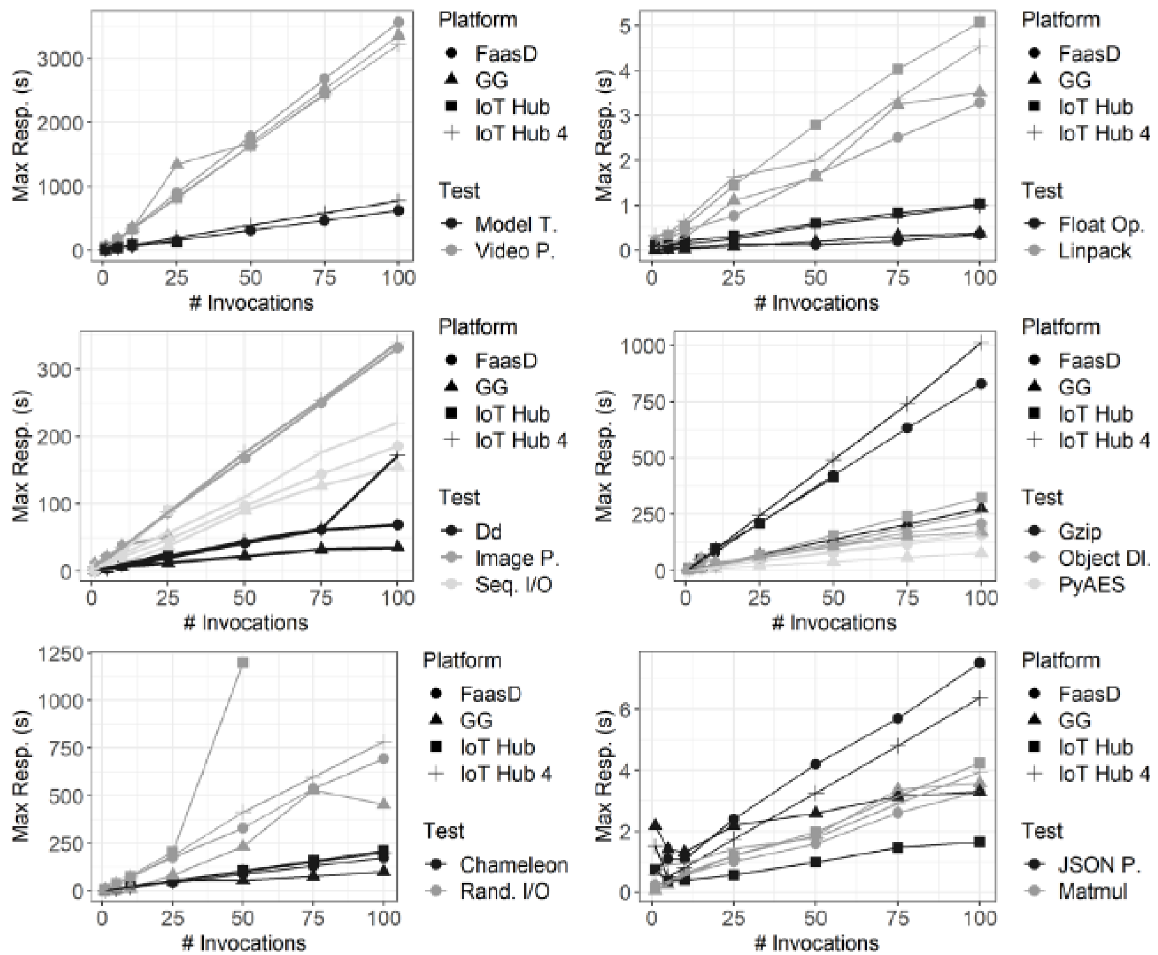


Слика 3.1 Споредба на перформансите при сериско извршување, едно барање во даден момент

Просечното време потребно за синхроно сериско извршување на сите 15 теста низ трите тестирани платформи е прикажано на слика 3.1. Во случајот со Greengrass, недостасуваат резултати за тестот за тренирање модели, бидејќи при пакувањето на сите потребни датотеки за инстанцирање на функцијата, вкупната големина ја надминува максимално дозволената од страна на платформата, со што се спречува извршувањето. Анализирајќи ги резултатите, Greengrass покажува најдобри перформанси во тестовите зависни од брзината на влезно/излезните операции, како на пример dd, тестирање секвенцијален пристап, тестирање случаен пристап и Gzip компресија. Интересно е да се забележи големата разлика во времето потребно за извршување на тестовите за тренирање модели и операции со подвижна запирка, каде што најдобрата платформа е 25.76%, т.е. 80%, соодветно, побрза од Azure IoT Hub. Во сите останати случаи, избраните решенија

покажуваат споредливи резултати, со Greengass на прво место во 12 теста, FaasD во 3 и Azure IoT Hub во 0.

Паралелно извршување



Слика 3.2 Споредба на перформансите при паралелно извршување, со променлив број на истовремени извршувања

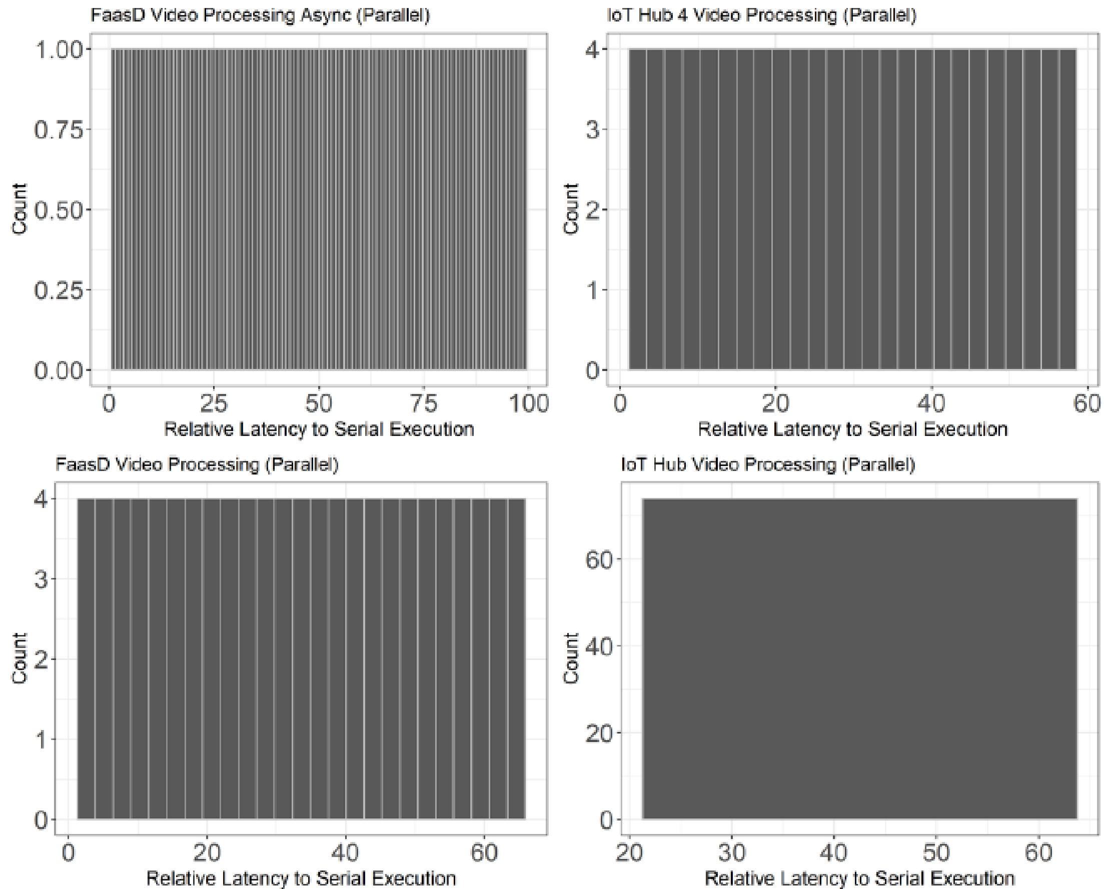
Различни платформи на различен начин имплементираат поддршка за паралелизам, па така ниту FaasD ниту Azure IoT Hub немаат можност за автоматско скалирање на функциите, додека пак оваа функционалност е достапна во Greengrass. Дополнително, сите тестови за FaasD беа имплементирани користејќи го официјалниот и препорачан шаблон за Python функции од страна на развивачите, кој во себе вклучува и веб сервер. Во овој случај, максималниот број на барања кои може симултано да се обработуваат во даден момент зависи од бројот на нишки со кои е стартуван веб серверот. Од друга страна, кога станува збор за Azure IoT Hub, паралелизмот е целосно препуштен на самите развивачи на функциите, без дополнителни олеснувачки аспекти понудени од решението.

На слика 3.2 се прикажани резултатите од паралелното извршување на 14 од 15 функции за сите тестирани платформи, визуелизирајќи го максималното време на одговор при 5, 10, 25, 50, 75 и 100 паралелни повици. iPerf3 тестот е намерно изоставен бидејќи не е релевантен во овој случај.

Анализирајќи ги резултатите, забележлива е значителна разлика во времето на извршување, што и не е изненадувачки, имајќи ги предвид целосно различните стратегии за воведување паралелизам кои ги нудат платформите. Предноста на Greengrass кога станува збор за влезно/излезните операции е присутна и при паралелно извршување. Ограничената и едноставна стратегија за паралелизам кај FaasD и Azure IoT Hub им овозможува да ги завршуваат задачите без поголем проблеми, но со задоцнување во одредени случаи. Вреди да се напомене дека максималното ниво на паралелизам треба да се дефинира на ниво на функција, во зависност од вкупните ресурсни побарувања, со цел избегнување на ситуацијата каде што голем број на паралелни инстанци би се натпреварувале меѓусебно за потребните ресурси, водејќи до изгладнување. Ваквото изгладнување може во екстремни случаи да доведе и до предвремено насилно исклучување на инстанцата, што е и случај при тестот за процесирање слики на Greengrass и Azure IoT Hub платформите. 25 и 50 паралелни извршувања едноставно не беа возможни поради недостаток од пресметковни ресурси. Дополнителен релевантен аспект е и целокупното време на одговор, а не само тоа дали функцијата завршила успешно или не, бидејќи за задачи кои побаруваат голема активност врз влезно/излезни уреди, вкупното доцнење може драстично да порасне за кратко време, како што е и видливо во случајот со Azure IoT Hub и тестот за В/И операции со случаен пристап.

За и емпириски да бидат потврдени можностите за паралелизација понудени од секоја платформа, времињата на одговор се нормализирани и прикажани во форма на хистограм на слика 3.3. Нормализацијата се одвива на тој начин што секое време на одговор при паралелно извршување е поделено со просечното време на одговор при серискoто, секвенцијално, извршување, а големината на кофичките во хистограмот е поставена на количникот помеѓу просечното време на извршување при паралелно извршување и просечното време на извршување при сериско, секвенцијално извршување. Ваквиот количник може да се разгледува и како коефициент на забавување на паралелните наспроти секвенцијални извршувања, како резултат на повеќето барања истовремено. Разликата помеѓу термините време на одговор (англ. response time) и време на извршување (англ. execution time) е тоа што при пресметување време на одговор се гледа вкупното доцнење, додека, пак, за време на извршување се зема предвид исклучиво времето потребно за комплетирање на задачата штом барањето за неа е добиено, занемарувајќи ги потенцијалните доцнења од мрежен аспект.

Користејќи ја опишаната стратегија анализата потврдува дека извршната околина на FaasD во основната конфигурација навистина стартува 4 нишки при испраќање синхрони HTTP барања, но и дека е возможно извршување само 1 паралелно барање при користење на асинхрониот метод за повикување функции. Како резултат на ова, а и со цел да се испита однесувањето на Azure IoT Hub при самостојна имплементација на паралелизам од страна на програмерот, сите тестови на оваа платформа се извршени со два степена на конкурентност – 100 и 4. Мерењата кои се однесуваат на повисоката конкурентност може да се споредуваат со Greengrass, додека пак оние со помало ниво на паралелизам со FaasD. Како што беше случајот и со Greengrass, некои тестови не можат да бидат успешно завршени поради истоштување на достапните ресурси, што се манифестира со нивното отсуство на слика 3.2. Со помош на хистограмот на слика 3.3, навистина се потврдува дека Azure Ion Hub не воведува никакво ограничување во однос на паралелизмот, сè додека има соодветна логика од аспект на дополнителни нишки и процеси во самиот код за функцијата. Последниот график во долниот десен агол на слика 3.3 го покажува паралелното извршување на 75 инстанци од тестот за процесирање видео, по добивањето на 75 повици истовремено.

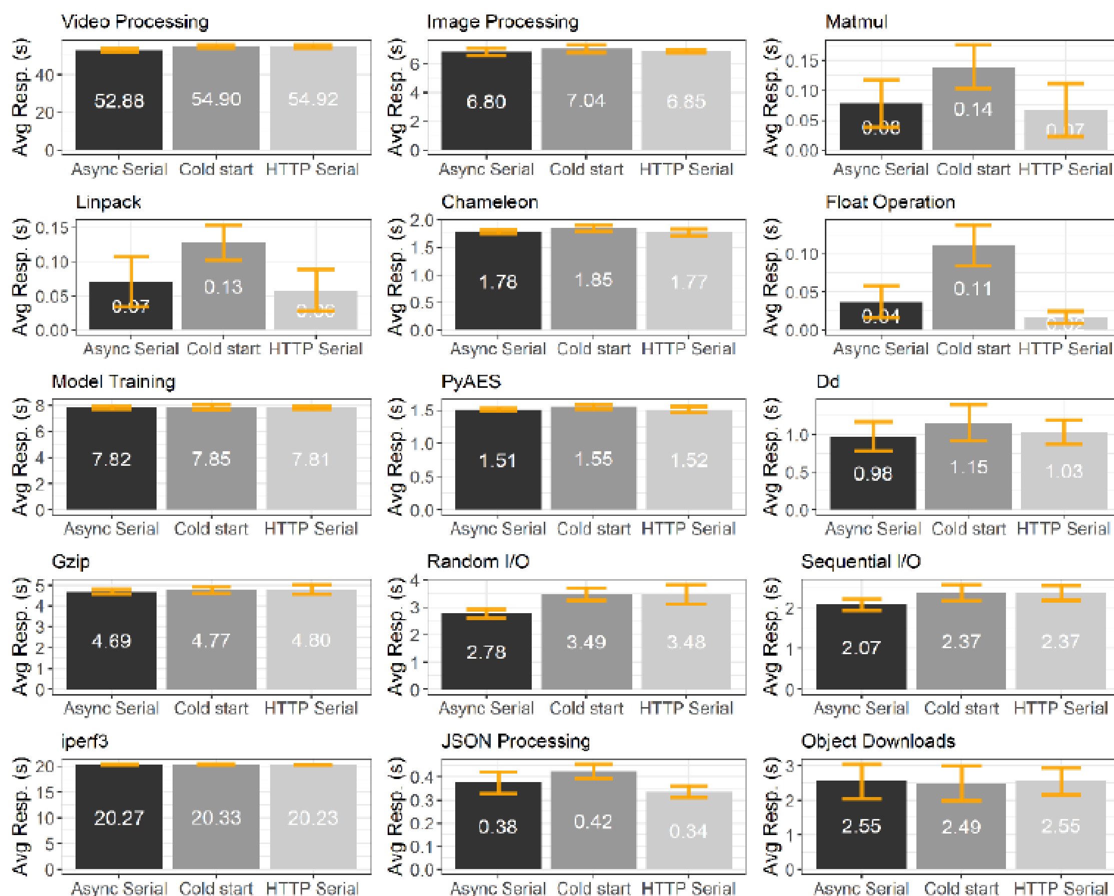


Слика 3.3 Број на истовремени процеси за опслужување барања при паралелно извршување

Ладен старт

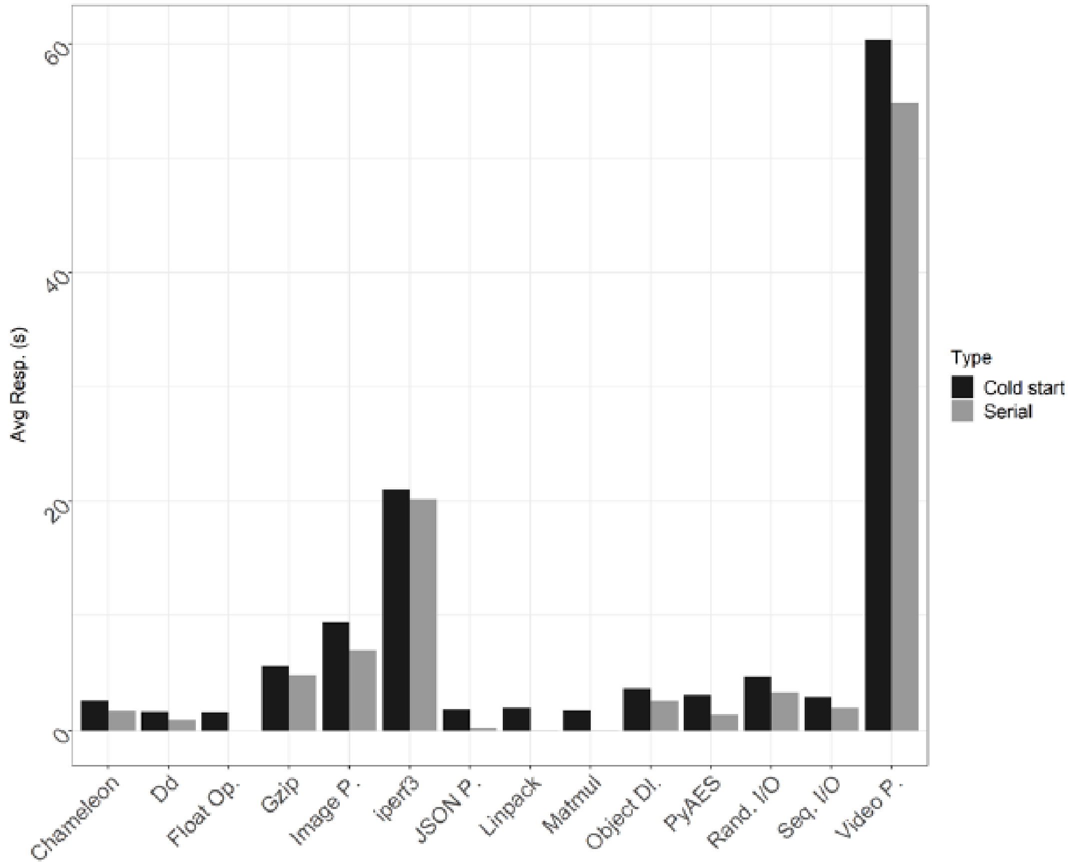
Доцнењето при ладен старт е анализирано за две од трите разгледувани решенија – FaasD и Greengrass. Azure IoT Hub е изоставен во ова сценарио бидејќи не поддржува ниту автоматско, ниту рачно скалирање до 0 инстанци и се очекува еднаш покрената функција да остане активна сѐ до нејзиното бришење.

Слика 3.4 го прикажува просечното време на одговор при 100 сериски повикувања на функција покрената врз FaasD, користејќи синхрон HTTP повик, асинхрон повик и при ладен старт. Мерењето во однос на ладниот старт е извршено со прво експлицитно скалирање до 0 инстанци и потоа извршување синхрон HTTP повик кон функцијата. За одредени функции, особено кај оние со кратко време за извршување, ладниот старт има значително влијание врз вкупното доцнење, што може да се увиди преку тестовите за множење матрици (matmul), решавање системи равенки (linpack) и операции со подвижна запирка (float operation).



Слика 3.4 Споредба на перформансите за извршување кај различните режими на работа на FaasD

Преминувајќи на страната на комерцијалните решенија, на слика 3.5 е дадена споредба на вкупното доцнење за извршување на дадена функција со и без ладен старт кај AWS Greengrass платформата. Забележително е присуство на зголемено доцнење при ладен старт во сите случаи, а најизразено е кај тестовите за обработка на JSON, решавање системи равенки, множење матрици, операции со подвижна запирка и обработка на видео, слично на ситуацијата со FaasD.



Слика 3.5 Доцнење при ладен старт и сериско извршување кај AWS Greengrass

3.3. Анализа на повеќејазлени безсерверски платформи на работ од мрежата

Кога станува збор за комплексни сценарија на работ од мрежата кои побаруваат поголем пресметковен капацитет кој потенцијално би бил и дистрибуиран низ различни географски локации, еднојазлените платформи не се соодветен избор. На нивно место потребно е да се разгледуваат особеностите на повеќејазлените безсерверски платформи кои со помош на меѓусебно кластерирање и дополнителни алатки за оркестрација би можеле да ги задоволат потребите. Во оваа насока, се поставува прашањето со кои предизвици се соочуваат овие повеќејазлени платформи денес и дали тие се поклопуваат со она што беше прикажано при анализата на еднојазлените алтернативи.

3.3.1. Kubernetes како сеприсутен оркестратор и неговата врска со безсерверското пресметување

Неспорно е дека контејнерите се најпопуларната извршна околина за покренување безсерверски функции во моментов, како на работ од мрежата така и во облакот. Како резултат на ова, направени се напори за адаптирање постоечки оркестратори на контејнери кон работ, каде што пресметковните ресурси се поограничени. Kubernetes, како најпопуларниот оркестратор на контејнери денес служи како основа на голем број популарни безсерверски платформи со отворен код [143]. Во моментов постои значителен интерес како од академската заедница, така и од индустријата за дополнителни оптимизации на овој оркестратор, првично дизајниран за облакот, со цел негово прилагодување кон работ од мрежата и употреба во целосно нови кориснички сценарија [42].

Препознавајќи дека една единствена имплементација на Kubernetes оркестраторот не може да ги задоволи сите пресметковни потреби идентификувани од огромниот број на сценарија во кои тој би бил корисен, се појави потребата од дефинирање алтернативни Kubernetes дистрибуции со специфични оптимизации за конкретни случаи [144]. Ваквиот развој може да се сфати и како природен тек на работите, бидејќи првично се одеше кон додавање сè повеќе функционалности кон оригиналното решение, овозможувајќи негово поставување како на инфраструктури со илјадници јазли во облакот, така и на лични пресметковни машини, па и на еднојадрени компактни уреди. Вклучувањето на сите овие дополнителни додатоци влијае врз големината, врз неопходните пресметковни ресурси, но и врз комплексноста на инсталацијата, занемарувајќи го фактот што најголем дел од тие додатоци би останале неискористени во околини со ограничени перформанси, како работ од мрежата. Фокусирајќи се на безсерверското пресметување, каде што еден од примарните аспекти е брзината на покренување на функциите како предуслов за ефикасно скалирање, ваквата дополнителна комплексност може негативно да влијае врз времињата на ладен старт, нарушувајќи го притоа искуството на крајните корисници.

За надминување на овие проблеми се појавија неколку проекти со отворен код, како на пример K3s [124] и MicroK8s [125] чија цел е елиминирање на неопходните Kubernetes компоненти, па во некои случаи и замена на постоечки компоненти со алтернативни имплементации кои би се покажале како подобар избор за поедноставни средини. Како резултат на постојаното настојување за сè попроста инсталација на Kubernetes оркестраторот, алтернативните Kubernetes дистрибуции стануваат погодни за инсталација врз физичка инфраструктура поставена на работ од мрежата со цел воспоставување на безсерверски платформи или вклучување во MEC сценарија. Сепак, не смее да се заборави дека на крајот, без разлика за која дистрибуција на Kubernetes станува збор, сите тие се компатибилни со оригиналниот софтвер и користењето на истиот програмски интерфејс овозможува реискористување како на конфигурација, така и на постоечки апликации [145].

Неизбежно е прашањето дали и до кој степен овие поедноставени Kubernetes дистрибуции влијаат врз извршувањето безсерверски функции и дали некој од прифатените компромиси во име на помала комплексност негативно се рефлектира при нивно поставување на работ од мрежата. Во оваа насока, Ли и др. [146] ги анализираат перформансите на четири различни безсерверски решенија покренати врз Kubernetes оркестраторот. Тие даваат дополнителни информации за архитектурата на решенијата и за тоа како донесените одлуки при дизајнирањето на системите би можеле да влијаат врз крајните резултати.

За популарноста на Kuberetes во полето на безсерверското пресметување сведочат и трудовите кои вршат оптимизации врз постоечки решенија со отворен код. Еден таков пример е [72], каде што авторите ја подобруваат стратегијата за распределба на безсерверски задачи кај Knative платформата со цел намалување на ефектот на ладниот старт. Слични подобрувања се претставени и за Kubeless [71] и OpenWhisk [65]. Потребата за пооптимизирани Kubernetes дистрибуции дополнително ја поткрепуваат резултатите преставени од страна на Ајерман и др. [147] кои покажуваат дека оригиналниот Kubernetes троши повеќе ресурси при неактивност во споредба со алтернативни, но и попрости оркестратори на контејнери. Во оваа насока, Кајал и др. разгледуваат Kubernetes дистрибуции кои потенцијално би можеле да се користат на работ од мрежата [144].

3.3.2. Избор на безсерверски платформи за тестирање

Повеќејазлените безсерверски платформи кои ги користат контејнерите како извршна околина имаат потреба од оркестратор на контејнери за распоредување на функциите низ пресметковните јазли, придружени во пресметковен кластер.

Во продолжение се користи OpenFaaS безсерверската рамка за евалуација на Kubernetes дистрибуциите кои ќе бидат предмет на разгледување. Причината за избор токму на ова решение е поради тоа што OpenFaaS е најпопуларната безсерверска платформа меѓу оние кои имаат директна поддршка за Kubernetes оркестраторот [12], [102], [148], судејќи според бројот на ѕвездички добиени од страна на корисниците на GitHub [149]. Дополнително, OpenFaaS поддржува два различни механизма за скалирање на функциите [150]:

- интерно скалирање, имплементирано од страна на OpenFaaS, врз основа на избрани метрики од страна на администраторот (пр. барања во секунда, HTTP кодови на одговори, искористеност на ресурси);
- скалирање со помош на основното решение на Kubernetes за хоризонтално скалирање на подови (англ. horizontal pod autoscaler, HPA) кое може да носи одлуки во однос на бројот на инстанци врз основа на тековната зафатеност на процесорот или работната меморија.

Со евалуација на двата начина за скалирање, се овозможува проценка на нивната ефикасност при примена во различни дистрибуции.

Дополнителна предност на OpenFaaS е и тоа што поддржува голем број на шаблони за пишување безсерверски функции во популарни програмски јазици, а и компатибилноста со FaasD платформата што го олеснува адаптирањето на FunctionBench тестовите.

3.3.3. Избор на Kubernetes дистрибуции за тестирање

Постојаниот тренд на модуларизација на Kubernetes архитектурата преку воведување стандардизиран програмски интерфејс за интеракција со системи за складирање [151] и мрежно поврзување [152] го поедностави процесот на креирање посебни дистрибуции кои би ги вклучиле само најнеопходните компоненти за исполнување на дадено сценарио.

За изведување на тестовите беа избрани три различни начини за поставување на целосно функционални Kubernetes кластери: Kubespray [153], K3s [124] и MicroK8s [125]. Kubespray е проект со отворен код чија задача е да го поедностави поставувањето на комплексни Kubernetes кластери погодни за извршување продукциски задачи. Со Kubespray може да се покрене кластер во најразлични околини, вклучувајќи и во облакот или на локално достапни физички сервери. Спротивно на ова, K3s и MicroK8s се поедноставени дистрибуции експлицитно наменети за поставување на работ од мрежата. И двете решенија се карактеризираат со едноставна и брза инсталација, со можност за поставување и во повеќе-јазлена инфраструктура.

Сите три избрани решенија доаѓаат со опсежна и добро структурирана документација која помага при нивното користење. Kubespray проектот е всушност збирка од Ansible [154] скрипти кои го автоматизираат поставувањето на генеричен Kubernetes кластер. Од друга страна, пак, K3s претставува целосно алтернативна дистрибуција каде што сите неопходни Kubernetes компоненти, наместо да се извршуваат како посебни програми, се комбинирани во една единствена извршна датотека. За крај, MicroK8s е snap апликација

која може да биде покрената на која било GNU/Linux дистрибуција што го поддржува snap форматот преку snapd демонот.

Покрај начинот на инсталација и извршување, постојат и други значајни разлики помеѓу решенијата кои би можеле да влијаат врз нивната погодност за извршување безсерверски функции. Како резултат на желбата за користење на една единствена извршна датотека, K3s ги има колоцирано и серверот и агентот во ист процесен контекст [155]. Дополнително, за редуцирање на оптоварувањето врз складиштето, типичното решение за зачувување на состојбата, etcd клуч-вредност базата е заменета со поедноставна алтернатива во форма на SQLite база на податоци. Интересно, истиот адаптер кој всушност преведува од etcd во SQLite може да се реискористи и при употреба на други релациони бази на податоци како складиште за информациите од Kubernetes интерфејсот за апликативни програми. Како резултат на овие промени и поедноставувања, K3s има помали минимални хардверски побарувања во споредба со традиционална Kubernetes инсталација. Потребни се 512 мегабајти работна меморија за master јазли и 256 мегабајти за worker јазли [156], споредено со повеќе од 2 гигабајти по јазол во случајот кога се користи Kubeadm [157] (што се употребува и од Kubespray). Разлики се присутни и од аспект на потребни процесорски јадра, па K3s побарува барем 1, а стандардна инсталација на Kubernetes 2.

Навраќајќи се на MicroK8s, слично како и K3s, и ова решение го има заменето etcd со алтернативна имплементација, овојпат именувана Dqlite [158], што е многу слична со SQLite, но со дополнителна можност да работи во дистрибуиран режим, на повеќе јазли истовремено. Минималните хардверски побарувања за MicroK8s се поголеми од оние за K3s, па потребни се барем 540 мегабајти меморија [159]. Во официјалната документација на MicroK8s изостанува препорака за минимален број на потребни процесорски јадра.

3.3.4. Методологија на извршување тестови

Табела 3.2

Информации за околината за тестирање

Компонента	**Вредност**
Оперативен систем	Ubuntu 20.04
Број на јазли	6 (1 master, 5 worker)
Процесор	Intel Xeon X5647
Меморија	8 GB
Складиште	320 GB
Мрежно поврзување	1 Gbps
Kubernetes верзија	1.20.7 (Kubespray и K3s), 1.23.0 (MicroK8s)
CNI додаток	Calico 3.21.2
OpenFaaS верзија	0.21.1

Во табела 3.2 е даден преглед на целокупната извршна околина користена за спроведување на тестовите, вклучувајќи детални хардверски карактеристики на физичките машини и конкретни верзии на софтверските апликации. Особено внимание е обрнато сите физички машини да имаат комплетно иста хардверска конфигурација, елиминирајќи потенцијални разлики во нивните перформанси.

Вкупно се користени 6 физички машини за независно покренување на трите избрани Kubernetes дистрибуции. Секој кластер е сочинет од 1 master јазол и 5 worker јазли. Master јазолот не се користи за извршување на безсерверските тестови, туку исклучиво за опслужување на контролната рамнина од Kubernetes кластерот. По завршување на сите тестови за дадена Kubernetes дистрибуција, машините се репровизионираат од почеток

и се инсталира нов Kubernetes кластер со следниот метод предмет на тестирање. Верзијата 1.20.7 на Kubernetes оркестраторот се користи и во случајот со Kubespray и во оној со K3s. За MicroK8s неопходно е да се користи верзијата 1.23.0 поради тоа што тоа е првата верзија од која MicroK8s поддржува додавање повеќе worker јазли во еден кластер без автоматско овозможување на висок степен на достапност (англ. high-availability) [157]. Проширување на кластерот е поддржано и во постари верзии на MicroK8s, но тоа подразбира дека првите три јазли автоматски би се користеле како дел од контролната рамнина, што во тест сценариото би имало негативни импликации врз крајните резултати. Сите јазли кои учествуваат во контролната рамнина е потребно да ја имаат целата база на податоци реплицирана меѓу нив, што воведува дополнителен товар како врз складиштето, така и врз процесорот. Ова е и причината зошто беше инсистирано на користење единствен master јазол при тестирањето на сите три решенија.

Табела 3.3

Параметри за извршување на секој од тестовите

Категорија	Име на тест	Параметри	Намена	Вредност
Процесор, меморија	Float Operation	n	Број	n: 10 000 000
	Image Processing	URL	Датотека	–
	Linpack	n	Димензија	n: 5000
	Matmul	n	Димензија	n: 5000
	Model Training	URL	Податоци	–
	PyAES	n, m	Должина, итерации	n: 1000 m: 100
	Chameleon	n, m	Редици, колони	n, m: 2000
	Video Processing	URL	Датотека	–
Складиште	Gzip	file_size	Големина во MB	file_size: 50
	Dd	bs, count	Големина на блок, број датотеки	bs: 100M, count: 1
	Random I/O	file_size, byte_size	Големина на датотека и блок	file_size: 100, byte_size: 1024
	Sequential I/O	file_size, byte_size	Големина на датотека и блок	file_size: 100, byte_size: 1024
Мрежа	Object Download	{Input, ouptut} bucket, key	Информации за објектот	100 MB бинарна датотека
	JSONDumpsLoads	URL	Датотека	–

Од мрежен аспект, постојано се користи истиот додаток за мрежно поврзување (англ. container networking interface plugin, CNI) – Calico. Некои од тестираните дистрибуции имаат можност за вклучување и автоматско нагодување на Calico уште при иницијалната инсталација, но практикувана е рачна инсталација, со цел обезбедување конзистентна конфигурација низ сите различни дистрибуции,

Инсталацијата на OpenFaaS платформата се изведува со користење на официјалната Helm карта (англ. Helm chart) [160], а Longhorn [161] проектот се користи за овозможување постојано складиште за оние контејнери кои имаат потреба од тоа.

Имајќи предвид дека на дел од безсерверските функции од FunctionBench свитата треба да им бидат предадени дополнителни параметри при нивното повикување, во табела 3.3 е даден детален опис на конкретната употребена стратегија при тестирање.

Направени се по 5 извршувања на комплетната свита тестови за секоја Kubernetes дистрибуција, испитувајќи различни аспекти:

- Ладен старт – секоја функција се извршува 100 пати за да се анализира доцнењето при ладен старт. По завршувањето на секое извршување, функцијата рачно се

скалира до 0 инстанци. На овој начин, платформата е приморана да креира целосно нов контејнер при нејзиното наредно повикување.

- Сериско извршување – секоја функција беше непрекинато повикувана во период од 5 минути, користејќи една процесорска нишка. По добивањето одговор, веднаш се испраќа ново барање. Какво било автоматско скалирање е експлицитно оневозможено, со цел да не се наруши валидноста на резултатите.
- Паралелно извршување со една инстанца – секоја функција се повикува точно 20 пати користејќи 20 процесорски нишки, истовремено. На овој начин се симулира повикување функција со непредвидливо оптоварување, каде што бројот на дојдовни барања може драстично да се зголеми за краток временски период. И во овој случај, како и во претходниот, автоматското скалирање е експлицитно оневозможено.
- Паралелно извршување со автоматско скалирање преку OpenFaaS – секоја функција е непрекинато повикувана во фиксен временски период со променливо ниво на паралелизам, со цел да се тестира однесувањето на автоматското скалирање вградено во OpenFaaS.
- Паралелно извршување со автоматско скалирање преку Kubernetes – секоја функција е непрекинато повикувана во фиксен временски период со променливо ниво на паралелизам, со цел да се тестира однесувањето на автоматското скалирање понудено од Kubernetes оркестраторот.

За извршување на HTTP барањата, а притоа и да се овозможат прецизни мерења во поглед на вкупното време на одговор се користи hey алатката [162]. Сите мерења се направени од физичка машина која не е дел од Kubernetes кластерот, поставена во истата локална мрежа како и Kubernetes јазлите. Мрежниот сегмент кој се користи за меѓусебно поврзување на машините е изолиран и посветен исклучиво на спроведување на тестовите, без дополнително оптоварување од трети уреди со цел да се намали влијанието од неконтролирани надворешни фактори. Физичките машини не извршуваат никакви дополнителни задачи и немаат друга намена освен спроведување на тестовите или опслужување на функциите, со цел добивање што е можно поверодостојни резултати [163]. Сите екстерни ресурси потребни за спроведување на дел од тестовите се хостирани на независен сервер во локалната мрежа, слично како и при тестирањето на еднојазлените платформи.

3.3.5. Споредбена анализа

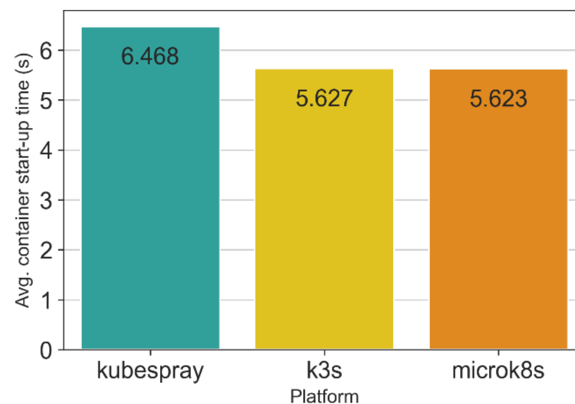
Сите три дистрибуции избрани за тестирање (Kubespray, K3s и MicroK8s) поддржуваат целосно управување со животниот циклус на кластерите, вклучувајќи додавање јазли, бришење јазли и ажурирање кон понови верзии, но комплексноста за изведување на овие операции се разликува. На Kubespray му е потребно најдолго време како за покренување нов кластер, така и за ажурирање постоечки, што се должи и на фактот што управувањето е направено на конвенционалниот начин, користејќи ја kubeadm алатката и вклучувајќи ги сите додатоци. Од друга страна, пак, на K3s и MicroK8s им треба значително помалку време за извршување на истите операции. Разликите се впечатливи и во однос на бројот на дополнителни помошни апликации кои би можеле да бидат инсталирани во кластерот додека трае неговото поставување. Ова може да биде корисна функционалност, со цел избегнување рачна инсталација на компоненти за интеграција со надворешни системи за складирање, софтверски дефинирани мрежи и мониторирање од страна на администраторот штом кластерот е поставен. Kubespray го поддржува ваквото однесување преку користење стандардни Kubernetes манифести претворени во шаблони, овозможувајќи промена на клучни параметри при процесот на инсталација или

ажурирање на кластерот. Во случајот со K3s, може да се овозможат дополнителни функционалности со предавање аргументи на инсталациската скрипта задолжена за целосното управување со животниот циклус на кластерот. Кај MicroK8s овозможувањето и оневозможувањето на додатоците се прави со извршување на наредби на командна линија, користејќи го соодветниот MicroK8s клиент. Во позадина овие наредби всушност повикуваат соодветни инсталациски скрипти развиени од страна на MicroK8s проектот.

Ладен старт

Применетата методологија за тестирање, каде што се користат целосно исти хардверски конфигурации на физичките машини врз кои е инсталирана безсерверската платформа овозможува тестирање на влијанието на различните дистрибуции врз ладниот старт на контејнерите. Имајќи предвид дека единствената променлива која се воведува низ тестовите е Kubernetes дистрибуцијата, какви било забележани разлики би се должеле исклучиво на нејзината ефикасност.

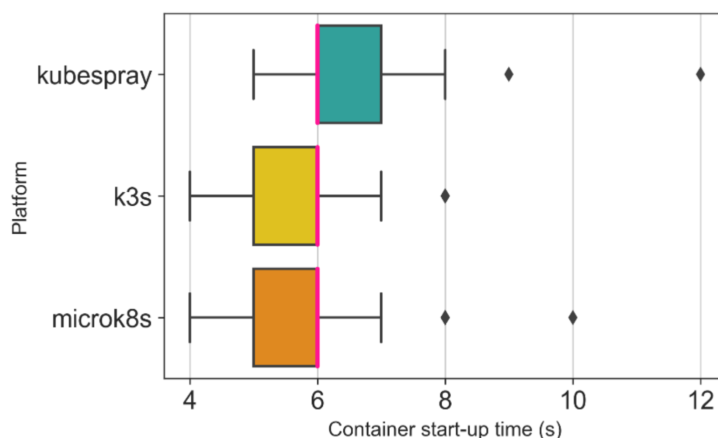
Слика 3.6 го прикажува просечното доцнење при ладен старт за сите тестирани дистрибуции, низ сите 14 функции од свитата. Секоја функција се извршува 100 пати, што резултира со 1400 мерења по дистрибуција (низ сите функции), т.е. 4200 мерења вкупно, земајќи ги предвид трите дистрибуции. Може да се забележи дека Kubespray, кој покренува целосен Kubernetes кластер без специфични оптимизации, покажува 15% зголемување во доцнењето при ладен старт во споредба и со K3s и со MicroK8s. Фокусирајќи се само на дистрибуциите оптимизирани за поставување во околини со ограничени ресурси, и двете нудат многу слични резултати при стартувањето нови контејнер.



Слика 3.6 Просечно доцнење во секунди при ладен старт на контејнери за секоја од трите тестирани платформи

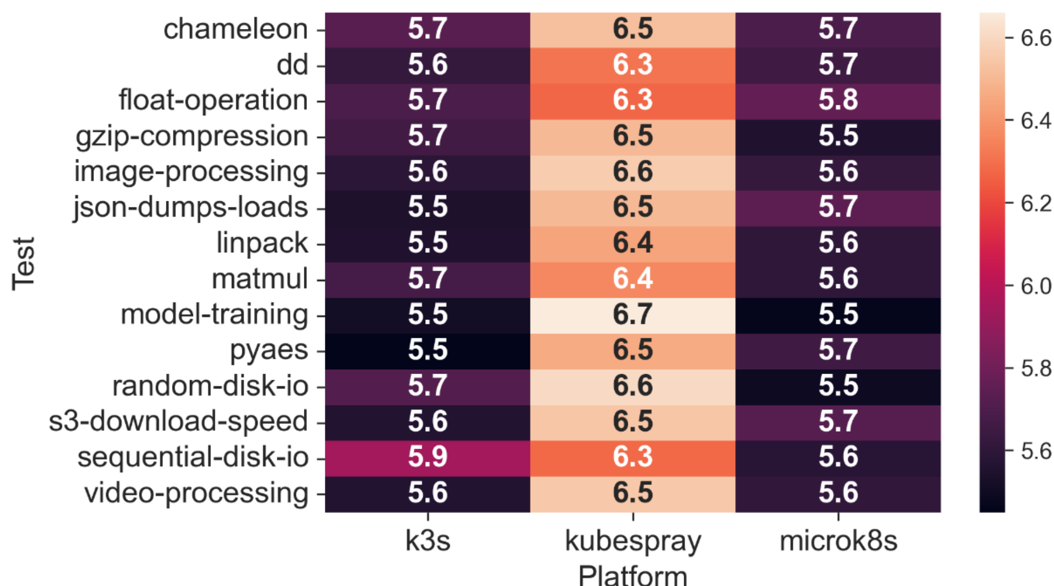
За подобра визуелизација на разликите во перформансите, на слика 3.7 е прикажана алтернативна визуелизација, отсликувајќи ги медијаната, минимумот, максимумот и нетипичните отстапувања[13] за мерењата низ сите испитувани платформи. Иако медијаните се конзистентни, поскромните перформанси на Kubespray се видливи и тука.

[13] Екстремните вредности за K3s платформата (оние поголеми од 17 секунди) беа изоставени во име на читливост на графикот. Сепак, и овие екстремни вредности беа земени предвид при сите понатамошни анализи и пресметки.



Слика 3.7 Алтернативен приказ на доцнењето при ладен старт за секоја од платформите

Продолжувајќи ја анализата на перформансите при ладен старт, слика 3.8 нуди поопширен преглед на измерените времиња. Слично како и претходно, веднаш е забележлива значителна разлика помеѓу Kubespray од една страна и K3s и MicroK8s од друга, како пооптимизирани решенија.



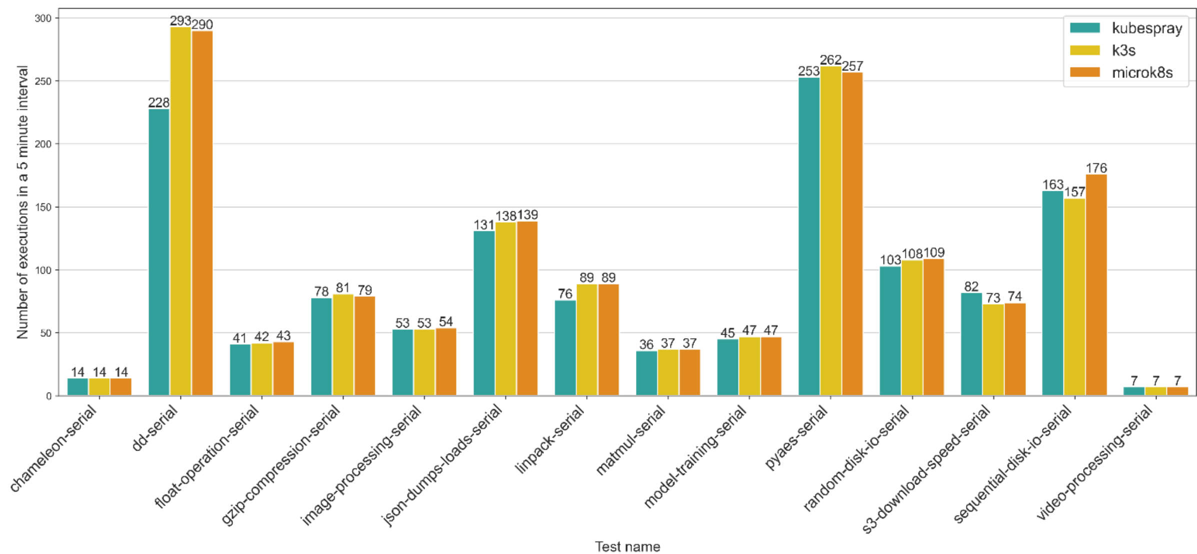
Слика 3.8 Просечно доцнење при ладен старт за секоја од функциите при различните платформи

Сериско извршување

Најголем дел од безсерверските платформи кои користат контејнери како извршни околини за покренување функции имплементираат методи за намалување на доцнењето при ладен старт. Овие стратегии може да бидат со различно ниво на софистицираност и комплексност, а во некои случаи подразбираат и реискористување постоечка контејнерска инстанца за извршување на наредна функција која би била повикана за кратко време. Мотивацијата во овој случај е дека еден повик никогаш не се извршува индивидуално, туку по него следат дополнителни повици, феномен познат и како временска централност во дистрибуираните системи.

За тестирање на суровите перформанси на секоја од Kubernetes дистрибуциите, во идеален случај, без земање предвид на ладниот старт, секоја функција се извршува непрекинато во рок од 5 минути. За време на дадената временска рамка, како што се

добива одговор на испратеното барање, така се испраќа и следно барање, внимавајќи да нема повеќе од 1 издадено барање во единица време. Сите механизми за скалирање, како од OpenFaaS така и вградените во Kubernetes, се експлицитно оневозможени со цел да се избегне несакано зголемување во бројот на функциски инстанци што негативно влијае врз веродостојноста на резултатите. Сериско тестирање се извршува поединечно за секоја од функциите во свитата.

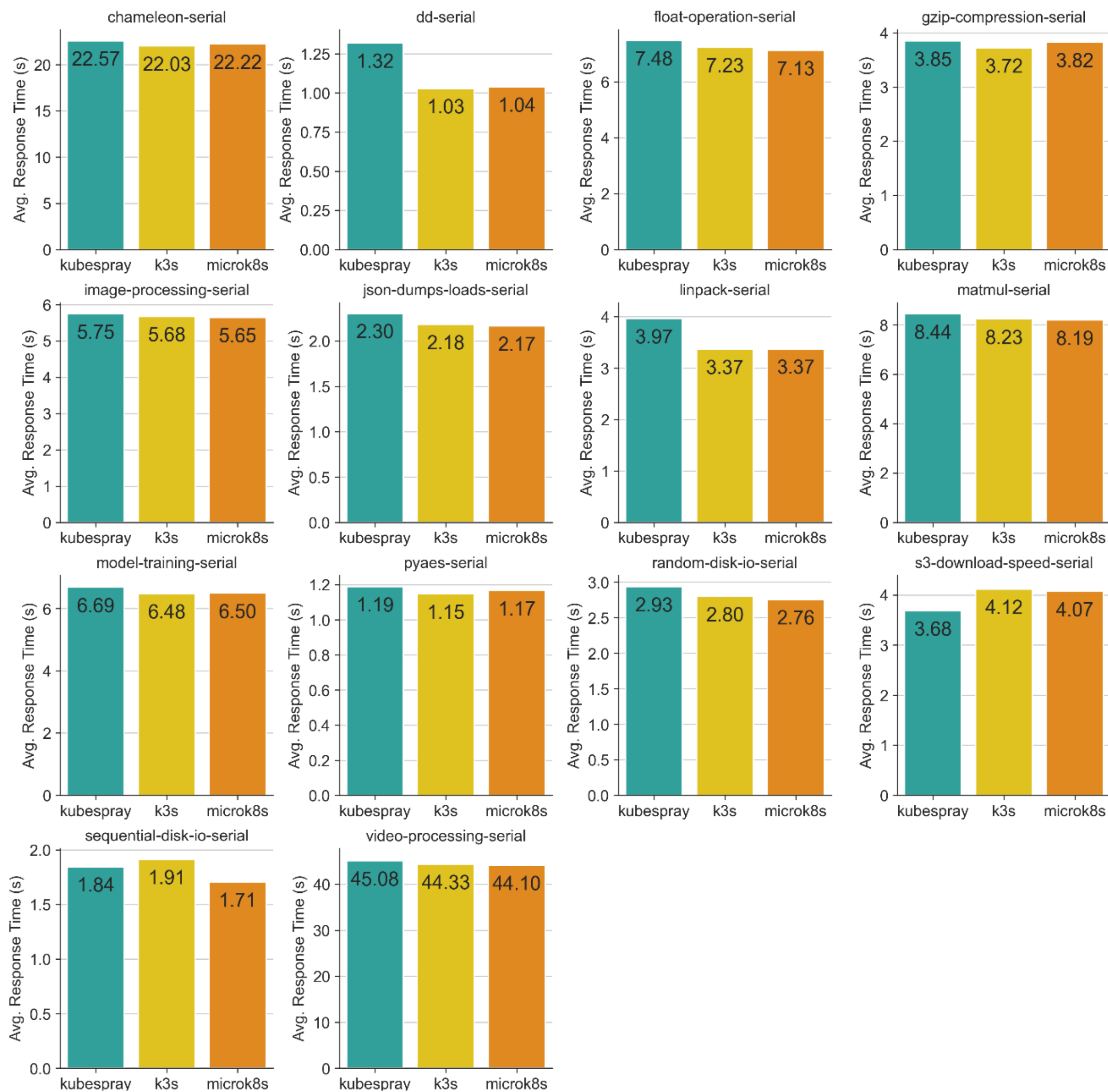


Слика 3.9 Вкупен број на извршувања за секоја функција во период од 5 минути

На слика 3.9 е прикажан вкупниот број извршувања на секоја од функциите постигнати во 5-минутниот тест интервал. Како и кај тестовите за мерење на ладниот старт, K3s и MicroK8s покажуваат многу слични перформанси и се изедначени или се разликуваат за едно единствено извршување во 10 од 14 тестови. Kubespray има најголем број на извршувања во временската рамка во еден тест, оној за преземање на голема датотека од складиште за објекти (s3-download-speed). Дополнително, Kubespray покажува поскромни резултати при тестови кои бараат високи процесорски перформанси, како на пример AES шифрирање/дешифрирање или решавање линеарни равенки.

Слика 3.10 нуди дополнителна перспектива на резултатите, покажувајќи го просечното време на одговор за секоја функција за време на нејзиното 5-минутно извршување. Како што може и да се очекува досега, и во овој случај, резултатите помеѓу K3s и MicroK8s се споредливи.

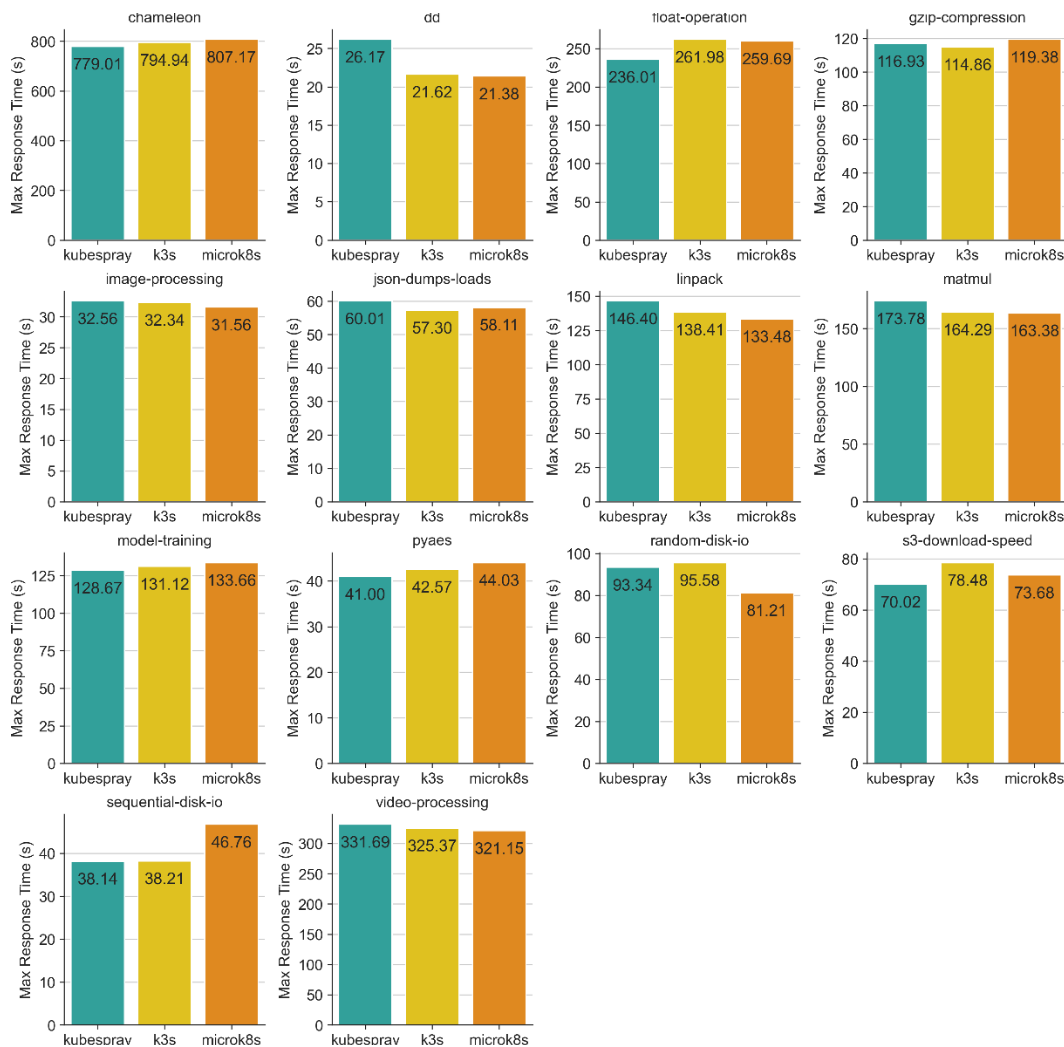
Согласно добиените резултати очигледно е дека разликите помеѓу K3s и MicroK8s се многу помали споредено со оние со Kubespray, во најголемиот дел од случаите.



Слика 3.10 Просечно време на одговор за секоја функција при сериско извршување

Паралелно извршување без автоматско скалирање

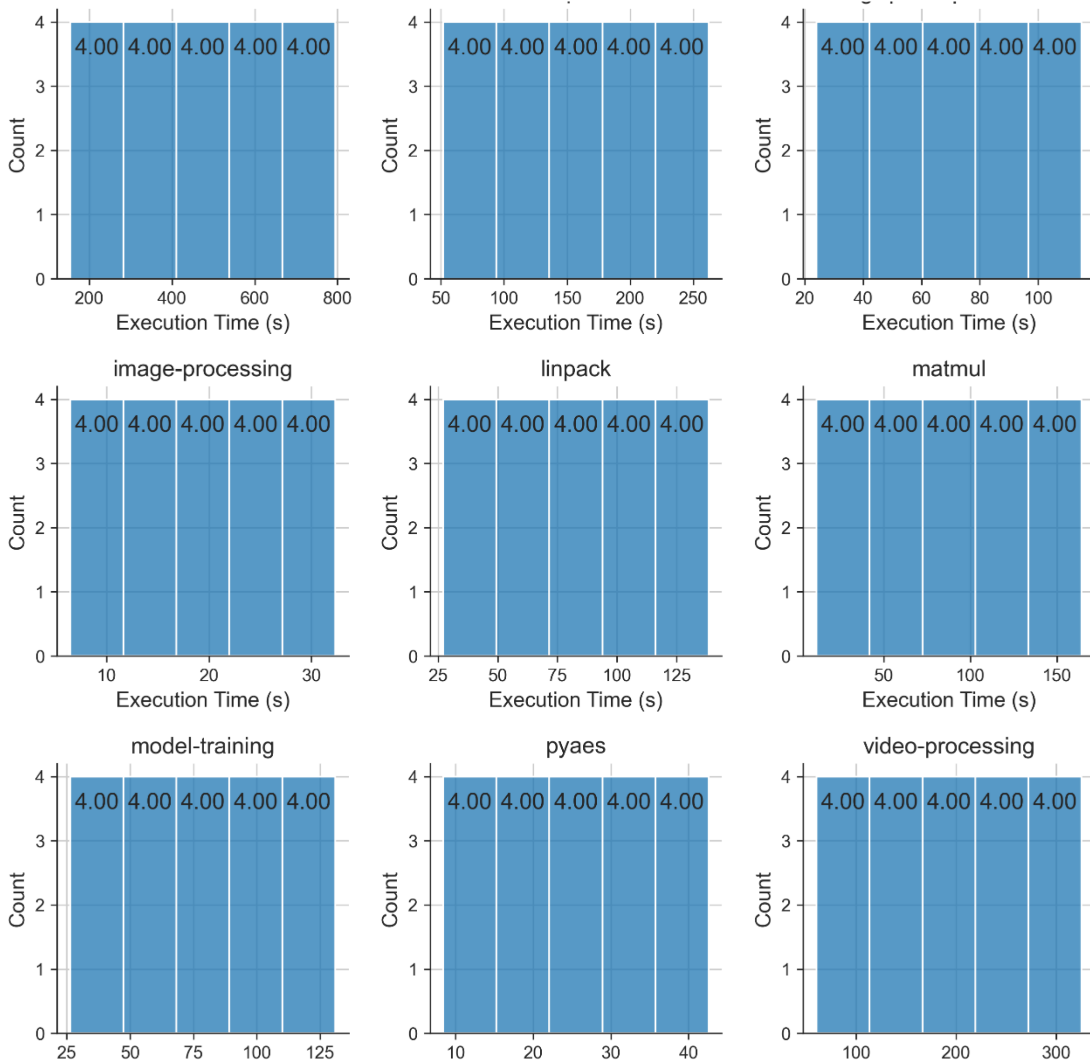
Честа практика е да се реискористи истата извршна околина за повеќе последователни повици на функцијата, заштедувајќи на времето потребно за нејзино иницијално креирање и нагодување. Во конкретниот случај за OpenFaaS, веднаш е очигледно дека еден контејнер може да се реискористи за повеќе последователни извршувања на дадена функција, бидејќи контејнерите не се уништени по враќањето одговор. За испитување колку всушност паралелни повици може да бидат опслужени од една контејнерска инстанца, испраќани се 20 паралелни барања истовремено, мерејќи го времето на одговор. Ваквата стратегија за тестирање дополнително овозможува и анализа на тоа како се справуваат различни Kubernetes дистрибуции при непредвидливо оптоварување и неочекувано, нагло зголемување на бројот на барања. Како и претходно, сите механизми за автоматско скалирање се оневозможени за да има само една инстанца по функција.



Слика 3.11 Вкупно време потребно за опслужување 20 барања користејќи 1 инстанца од функцијата

Слика 3.11 го илустрира вкупното време потребно на секоја од дистрибуциите за завршување со обработката на 20 паралелни барања. Спротивно на добиените резултати досега, Kubespray се карактеризира со подобри резултати и од K3s и од MicroK8s во 6 од 14 тестови: chameleon, float-operation, model-training, pyaes, s3-download-speed, and sequential-disk-io. K3s покажува најдобри резултати во 2 теста: gzip-compression и json-dumps-loads; MicroK8s има најкратко време на извршување во 6 (исто како и Kubespray): dd, image-processing, linpack, matmul, random-disk-io, video-processing.

Имајќи ја предвид извлечената поука од паралелното тестирање на FaasD на почетокот од поглавјето и максимум четирите паралелни извршувања кои е способен да ги спроведе апликацискиот сервер во основната конфигурација, направено е аналогно испитување и за OpenFaaS. На слика 3.12 се претставени времињата на одговор за секоја од функциите. Очигледно е дека во сите прикажани примери OpenFaaS е способен да опслужи максимум 4 паралелни повици кон дадена функција со еден единствен контејнер. Оттука може да се заклучи дека однесувањето во поглед на вградениот паралелизам е потполно исто како кај OpenFaaS, така и кај FaasD, но вреди да се повтори дека ова се однесува на основната конфигурација и може да подлежи на промени од страна на развивачот на функцијата или администраторот на платформата.



Слика 3.12 Број на паралелни барања опслужени од една инстанца на контејнер

Паралелно извршување со автоматско скалирање

Иако сериските тестови се соодветен начин за тестирање на суровите перформанси на безсерверските платформи и инфраструктурите над кои тие се поставени, не треба да се заборави дека една од главните придобивки на безсерверската парадигма е всушност лесното скалирање на бројот на функции како што расте оптоварувањето. Една од причините која оди во прилог на изборот токму на OpenFaaS како платформа за спроведување на тестовите врз различните Kubernetes дистрибуции е тоа што поддржува два сосема различни начини за автоматско скалирање. Со ваквиот пристап е возможно да се тестираат секој механизам и неговата ефикасност во случај кога се оптимизираат различни перформансни аспекти.

Вградено автоматско скалирање во OpenFaaS

Основната стратегија за скалирање кај OpenFaaS се потпира на мерења кои се добиваат со користење на популарната Prometheus мониторинг алатка со отворен код [164] и правила и прагови нагодени преку Alertmanager [165]. Штом метриката од интерес го достигне однапред дефинираниот праг, се креира известување кое всушност претставува повик кон програмскиот интерфејс на OpenFaaS со цел зголемување на бројот на реплики од засегнатата функција.

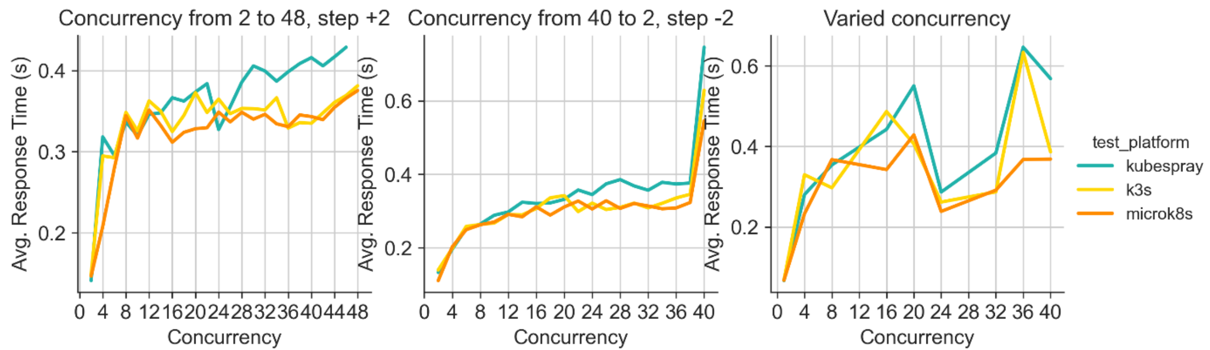
Ваквиот пристап е навистина флексибилен, овозможувајќи им на администраторите да дефинираат соодветни правила кои би се однесувале на која било од стотиците метрики следени од Prometheus во даден момент. Сепак, за жал, правилото кое е однапред овозможено на нови инсталации е непотребно ограничувачко и бинарно по природа. Во основната верзија се врши мерење на бројот на успешни повици во изминатите 10 секунди и оваа евалуација се одвива на секоја секунда, добивајќи практично формат на временски базиран лизгачки прозорец. Ако бројката на повици е поголема од 5, ја скалира функцијата, додавајќи ѝ нова инстанца, сè до достигнување на максималниот број дозволени инстанци. Ваквото однесување значи дека дури и при извршување константен број на барања во секунда, на пример 6, по одредено време ќе се достигне максималниот број на инстанци од функцијата, исто како што тој број би бил оправдано достигнат ако бројот на барања е драстично поголем, на пример 30 или 40. Ваквото однесување е тестирано и во пракса, со тоа што се извршува по 1 барање во секунда од страна на 6 паралели јазли во времетраење од над 200 секунди, со што успешно се достигнува максималниот број на инстанци и покрај скромниот број на барања во секунда.

Ова однесување каде што во правилото за скалирање не се зема предвид моменталниот број достапни инстанци од функцијата во кластерот и наместо тоа се работи се потпира исклучиво со апсолутниот број на барања во дадена секунда, води до неоптимални одлуки при скалирањето. При изложување на неваријабилен и постојан товар, или ќе се достигне за кратко време максималниот број на дозволени реплики или, пак, воопшто нема да се изврши скалирање. И покрај ова, како резултат на отвореноста на решението, секој има можност да го измени ваквото однесување, со што би се зеле предвид сите потребни аспекти, подобрувајќи ги целосните перформанси на системот.

Автоматско скалирање

Алтернативниот метод за автоматско скалирање на функции исто така поддржан од OpenFaaS е со користење на хоризонталното скалирање на подови Kubernetes. HPA е вграден механизам за скалирање во Kubernetes и може да се користи и за традиционални контејнери, не само за безсерверски функции. Во основната конфигурација се потпира на метрики (оптоварувања на процесор и меморија) извлечени од страна на серверот за метрики (англ. metrics server) како опционална, но најчесто присутна компонента во сите Kubernetes кластери.

На различни функции може да бидат придружени различни профили за автоматско скалирање, во зависност од нивната природа и од хардверскиот аспект кој треба да се оптимизира. При тестирањето рачно се конфигурира специфичен HPA профил кој врши скалирање секој пат кога float-operation функцијата користи повеќе од 350 делчиња од процесорот (0.35, т.е. 35% од едно јадро). На овој начин се потврдува дека при повик на float-operation функцијата со влезен параметар 100 000 се искористуваат отприлика околу 350 делчиња. Овие 350 делчиња соодветствуваат на четири паралелни повици во даден момент кон поединечен контејнер, што беше предмет на дискусија претходно во делот за паралелизмот на OpenFaaS.



Слика 3.13 Просечно време на одговор при паралелни повици

На слика 3.13 се прикажани резултатите од паралелното повикување на float-operation функцијата повеќе пати, со различни степени на паралелизам и тоа:

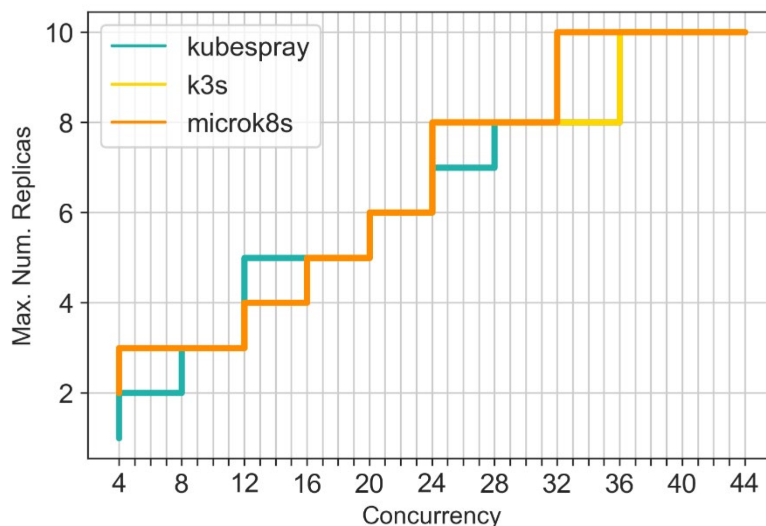
1. Почнувајќи од 1 реплика, се извршуваат 2 паралелни барања во секунда и потоа се пристапува кон зголемување на бројот на барања за 2 на секои 5 минути, сè додека не се достигне извршување од 48 барања во секунда.
2. Почнувајќи од 1 реплика, се извршуваат 40 паралелни барања во секунда, намалувајќи го бројот на барања за 2 на секои 5 минути, сè додека не се достигне извршување на 2 паралелни барања во секунда.
3. Почнувајќи од 1 реплика, се менува бројот на паралелни повици кон функцијата во даден момент, користејќи ја следнава стратегија на извршување: 8, 1, 20, 4, 40, 24, 1, 4, 16, 1, 36, 22 паралелни повика.

Мотивацијата зад овие различни сценарија на извршување е да се тестира како HPA се справува под различни услови. И (1) и (2) постепено го зголемуваат, т.е. намалуваат товарот, додека, пак, во случајот со (3) има значајно варирање и непредвидливост на секои 5 минути, со што се очекуваат подраматични промени во поглед на бројот на активни инстанци.

Според истата слика 3.13, Kubespray се одликува со поголеми времиња на одговор во сите три прикажани сценарија, додека, пак, резултатите од K3s и MicroK8s се слични помеѓу себе, со таа разлика што MicroK8s има забележителна предност во третиот, најнепредвидлив случај.

На слика 3.14 е прикажан соодносот помеѓу бројот на паралелни барања во единица време и бројот на инстанци покренати за опслужување на тие барања. Визуелизацијата е направена врз резултатите добиени кога функцијата е инстанцирана со 1 реплика и 4 паралелни барања во секунда, чијшто број потоа се зголемува за 4 на секои 5 минути. Токму оваа стратегија е применета бидејќи таа теоретски би требало да резултира со зголемување на бројот на реплики на секои 5 минути, т.е. при секое нагорно зголемување на бројот на паралелни барања.

Како и кај сите претходни тестови извршени досега, однесувањето при автоматското скалирање помеѓу K3s и MicroK8s е многу слично, со разлика во само еден случај. MicroK8s го зголемува бројот на реплики побрзо кога е соочен со 32 барања во секунда, додека, пак, K3s го прави соодветното прилагодување во бројот на инстанци при 36 паралелни барања. Kubespray заостанува зад останатите дистрибуции во два случаја, додека пак издвојува предност во еден.



Слика 3.14 Промена во бројот на инстанци во зависност од бројот на барања во секунда

3.4. Ретроспекција кон резултатите

Ова поглавје спроведе анализа на две различни фамилии безсерверски платформи – еднојазлени и повеќејазлени. Еднојазлените, како поедноставни имплементации, се погодни само за специфични кориснички сценарија каде што функциите се поставувани од страна на само еден ентитет – лице или компанија. Од друга страна, повеќејазлените платформи може да бидат предмет на споделување меѓу различни корисници, кои потенцијално не си ни веруваат меѓусебно. Разликите помеѓу двете фамилии на безсерверски платформи продолжуваат и од аспект на техничките карактеристики и побарувања, па така еднојазлените се оптимизирани за поставување врз уреди со ограничени пресметковни ресурси, притоа користејќи што е можно полесни извршни околини, врз база на контејнери, за покренување на функциите. Од друга страна, пак, повеќејазлените платформи имаат повисоки побарувања, а дел од тоа се должи и на фактот што е неопходна и дополнителна, трета, компонента при нивното поставување – оркестратор на контејнери. Оваа компонента овозможува стандардизиран начин за покренување на функциите низ повеќето јазли, без притоа да има потреба ваквата логика да се имплементира како дел од самата безсерверска платформа.

Со цел да се пополни идентификуваната празнина во постоечката стручна литература, при анализата на еднојазлените платформи се вклучени како комерцијални решенија, така и отворени. Резултатите се разнолики и разоткриваат неколку горливи проблеми. Поради меѓусебната некомпатибилност на решенијата, дури и при тестирање на перформансите потребни се специфични измени во изворниот код на функциите, сè со цел тие функции да може да се покренат на секоја од платформите. Со вакви предизвици се соочуваат и корисниците на безсерверската парадигма во реалниот свет, секогаш кога се обидуваат да преминат од еден екосистем во друг. Дополнително, постојат и значајни разлики во стратегиите за скалирање, до тој степен што базичниот пристап на некои решенија се коси со основното начело на безсерверското пресметување – можноста за скалирање до 0 инстанци. Обединувачки фактор низ сите решенија беше присуството на дополнително доцнење при иницијално стартување на функциите (ладен старт), со делумни варијации во однос на конкретната извршна околина.

Од аспект на повеќејазлените платформи, земајќи го предвид опсежниот корпус на трудови со фокус врз споредба на резултатите помеѓу вакви постоечки решенија, беше

одлучено да се посвети внимание на понизок аспект, т.е. како избраниот оркестратор влијае врз перформансите при извршување на безсерверските функции. За оваа намена се тестирани три различни дистрибуции на моментално најпопуларниот, а воедно и во областа на безсерверското пресметување најискористениот контејнер оркестратор – Kubernetes. Резултатите покажуваат дека оние дистрибуции специфично оптимизирани да бидат покренати врз инфраструктура со поскромни перформанси, како што е и типично за работ на мрежата, нудат подобри услови за извршување безсерверски функции. Сепак, мора да се разгледа и негативниот аспект на оваа „редукција во комплексност", а тоа е ограничената можност за интеграција со трети системи, како резултат на отсуство на потребните компоненти врз кои се потпираат таквите поврзувања. Од аспект на заемна споредба на резултатите, проблемот со ладниот старт повторно доминира и кај повеќејазлените платформи, особено што во тој случај извршната околина е униформна – секогаш станува збор за контејнери.

Ова поглавје го дозволува заклучокот дека со последните напредоци на полето на повеќејазлените безсерверски платформи, драстично се поедноставува поставувањето на потребната инфраструктура и управувањето со нејзиниот животен циклус. Реискористувањето на истите конфигурациски парадигми и задржувањето на постојниот формат на функции на работ од мрежата како и во облакот, во случајот со комерцијалните еднојазлени платформи, е позитивен аспект. Сепак, проблемот се јавува при покренување на прашањето за компатибилност меѓу хетерогените решенија претставени од различни компании. Тековно веќе се постигнува максимумот кој може да го понудат контејнерите како извршна околина за безсерверски функции. Нивното досегашно користење беше оправдано со можноста за брзо креирање прототипи и лесна употреба, како резултат на широката распространетост во останати сфери на пресметувањето. Сепак, истражувањата покажуваат дека контејнерите не се универзално решение за сите типови на функции и во одредени случаи големото доцнење при ладен старт ги прави несоодветен избор. Оттука се потврдува потребата за примена на нови и алтернативни извршни околини за надминување на идентификуваните проблеми, а со тоа и задоволување на строгите барања поставени од работ на мрежата и IoT уредите. Важно е при понатамошните истражувања во оваа насока да не се заборави крајната цел, а тоа е елиминацијата на ладниот старт за функциите на кои тоа им е потребно и чии севкупни перформанси најмногу страдаат како резултат на бавното покренување. Целта во никој случај не е да се направи целосна замена на контејнерите, туку дополнување и подобрување на постојните безсерверски платформи со можност за користење и алтернативни извршни околини, а изборот на крајот да се сведе на тоа која е најдобра при дадено конкретно сценарио. Со презентираните резултати и изведените заклучоци и формално се потврдува поставената хипотеза дека *„Моменталните извршни околини кои се користат за покренување безсерверски функции не ги задоволуваат барањата од аспект на брзина на извршување и доцнење при нивната примена на работ од мрежата. Потребна е оптимизација на постојните или примена на целосно нови алтернативни извршни околини за надминување на овие проблеми, што би довело и до задоволување на барањата кои ги поставуваат IoT уредите со своите пресметковни задачи засновани на настани."*

4. АЛТЕРНАТИВНИ ИЗВРШНИ ОКОЛИНИ ЗА БЕЗСЕРВЕРСКИ ФУНКЦИИ

Најраспространетите извршни околини за безсерверски функции, без разлика дали станува збор за работ на мрежата или облакот, денес, се контејнерите и микро виртуелните машини. Еден од првите проблеми со кои се соочува секоја нова извршна технологија претставена во контекст на безсерверската парадигма е прашањето околу вкупното време потребно за нејзина целосна подготовка при ладен старт. Иако и контејнерите и микро виртуелните машини се далеку поефикасни (во поглед на потребни пресметковни ресурси, а и време на покренување) од традиционалните виртуелни машини, сепак, нивните времиња на ладен старт се мерат во рангот на стотици милисекунди, па и неколку секунди, во зависност од инфраструктурата и типот на функцијата која е предмет на разгледување. Ваквото нивно однесување ги прави несоодветни за справување со временски чувствителни податоци [25]. Овој заклучок впрочем го потврдуваат и испитувањата спроведени како дел од претходното поглавје. Проблемот на ладен старт важи и за еден од најголемите предизвици при поставување на безсерверски платформи за транспарентни пресметки во облак-раб екосистемот, бидејќи има особено влијае врз перформансите кои може да се постигнат на работ од мрежата. Оттука произлегува потребата за дополнително истражување во насока на решавање на овој проблем, чиешто надминување, меѓу другото, би требало да овозможи и секој повик на дадена функција да биде извршен во посебна извршна околина, без споделени ресурси. Ваквиот пристап не само што би ја гарантирал безбедноста, туку и би дозволил директно стартување на соодветната функција штом е добиен повик за неа, наместо да се користат алтернативни техники како подгревање, кои отстапуваат од духот на безсерверската парадигма.

Решението на проблемот на ладен старт не лежи во понатамошна оптимизација на постојните извршни околини иницијално наменети за извршување во облакот, туку, напротив, во сосема нова технологија дизајнирана за извршување врз уреди со ограничени перформанси. Неодамнешниот развој на полето на WebAssembly [166] овозможи негова примена и на серверската страна. WebAssembly модулите немаат ограничување во однос на тоа на какви уреди може да се извршуваат (во поглед на перформанси, архитектура или извршна локација) и нудат брзо време на воспоставување на извршната околина, а со тоа и стартување на дадениот модул. Имајќи ги предвид ваквите сценарија, како и компатибилноста со широк спектар на уреди, не може да се занемари големиот потенцијал на WebAssembly за надминување на проблемот на ладен старт при извршување безсерверски функции.

Крајната цел не е да се најде целосна замена за контејнерите како најраспространета извршна околина за безсерверски функции, туку, напротив, да се понуди алтернатива која би можела да работи рамо-до-рамо со контејнерите. Ваквиот пристап би овозможил избор на најпогодната извршна околина за извршување на дадена функција, во однос на нејзините побарувања. Оттука следува и хипотезата *„Виртуелните машини и контејнерите како извршни околини може да се комбинираат заедно со нови и поефикасни алтернативи кои се подобро прилагодени за работ на мрежата со цел креирање една унифицирана платформа што ќе понуди флексибилност преку избор на најсоодветната технологија за извршувањето на дадена задача.“*

Целта на ова поглавје е да го истражи најновиот развој на WebAssembly и неговото можно влијание врз безсерверската парадигма во однос на овозможување транспаренти пресметки во облак-раб екосистемот. Посебен осврт се дава на начините како WebAssembly може да се интегрира со постојни решенија со цел да се избегне

ситуацијата на воведување уште една дополнителна технологија некомпатибилна со она што веќе постои и е во широка употреба.

4.1. Подемот на WebAssembly и преминувањето на серверската страна

WebAssembly е стандард под закрила на конзорциумот на светскиот веб простор (англ. World Wide Web Consortium, W3C) и неговата прва верзија е публикувана во 2017 со цел да се подобрат перформансите на веб апликациите извршувани на клиентска страна. Станува збор за стек базирана виртуелна машина која може да извршува специфично компајлиран код. Денес најпопуларните јазици чиишто компајлери го поддржуваат WebAssembly се C, C++, Rust и Go. Се очекува дека со текот на времето со созревањето на WebAssembly ќе бидат поддржани сè повеќе програмски јазици. Оние програмски јазици кои се интерпретирани исто така може да бидат извршувани во WebAssembly околина, доколку претходно е задоволен еден предуслов – самиот интерпретер е потребно да биде искомпајлиран како WebAssembly модул [167]. Иако станува збор за релативно нова технологија, голем број од веб прелистувачите веќе имаат поддршка за WASM. На овој начин се овозможува стар софтвер и софтвер кој никогаш не бил планиран за извршување во веб околина да биде покренат на едноставен начин на клиентската страна, во посебна изолирана средина, притоа нудејќи и напредни интеграции со останати актуелни технологии во самиот веб прелистувач [168].

Покрај трендот за воведување WASM поддршка во прелистувачите, од неодамна се појавија и независни WebAssembly извршни околини, без директна потреба за покренување на целосен веб прелистувач за извршување WASM модули. На овој начин, WebAssembly се трансформираше од веб технологија на клиентската страна во технологија која може независно и успешно да се користи и на серверската страна [169]. Главната придобивка од ваквиот развој е можноста за независно извршување задачи во повеќе-кориснички околини. Притоа се нудат задоволителни перформанси во споредба со алтернативните методи на извршување: екстремно брзо стартување, високо ниво на изолација и лесно користење преку директната поддршка за голем број популарни програмски јазици. Користењето на WebAssembly овозможува истиот бинарен модул да биде извршуван на најразлични архитектури и платформи, без потреба од негово рекомпајлирање или дополнителни специфични оптимизации [170].

Сепак, треба да се напомене дека во пракса можни се одредени предизвици при компајлирање постоечки апликации. Како резултат на високото ниво на изолација гарантирано од стек-базираната виртуелна машина некои од системските функционалности кои се земаат здраво-за-готово во вишите програмски јазици не се целосно достапни. Влезно/излезни операции врз датотеки [171], пристап до мрежата [172] и поддршка за повеќе нишки [173] се функционалности кои, пред да може да бидат користени, мора експлицитно да бидат имлементирани од соодветната WASM извршна околина. Со воведувањето на WebAssembly системскиот интерфејс (англ. WebAssembly Systems Interface, WASI) [174] оние извршни околини кои ја имплементираат спецификацијата можат да вршат интеракција со самиот оперативен систем во чија надлежност се овие функционалности. Ваквата интеракција се одвива преку системски повици кои се слични на стандардните можности на преносливиот интерфејс за оперативни системи (англ. Portable Operating System Interface, POSIX). Всушност, WASI е последното делче кое овозможува WebAssembly непречено да се користи за извршување задачи на серверската страна.

Имајќи ги предвид сите придобивки кои ги нуди WebAssembly, особено оние од аспект на реискористливост на постојни програмски кодови и изолација, станува збор за

технологија која би можела да биде соодветен избор за извршување на безсерверски функции на различни локации од мрежата, како на работ, така и во облакот [17], [175]. Постојаниот развој и имплементација на сè повеќе проширувања под капата на WASI е дополнително охрабрување и води кон изедначување на функционалностите со оние на стандардните извршни околини. Значајна е и појавата на WebAssembly стратегијата за справување со различни компоненти (англ. WebAssembly component model) [176], која овозможува креирање на споделени компоненти кои може да се реискористат од повеќе WASM модули, наместо многу пати да се имплементира истата функционалност. Ова е од особено значење за безсерверското пресметување, бидејќи овозможува поедноставување на функциите и од аспект на комплексност на кодот и од аспект на физичка големина во бајти.

Денес постојат повеќе WebAssembly извршни околини, а некои може да бидат покренати и на серверската страна. Дел од овие решенија се развиени од академската заедница, а други, пак, од индустријата, но заедничкото за нив е тоа што се применливи во најразлични ситуации.

4.2. WebAssembly како нова безсерверска извршна околина

Паралелно со порастот во бројот на функционалности кај новите WebAssembly извршни околини расте и истражувачкиот интерес за оваа технологија. Препознавајќи го големиот потенцијал за безсерверски апликации, Лонг и др. [177] имаат спроведено истражување за перформансните разлики помеѓу традиционалните Docker контејнери и WebAssembly модулите. Нивната методологија се заснова на споредба на времињата за ладен старт при извршување WASM модули во три различни околини (V8, Lacet, SSVM) наспроти извршување во Docker средина. Резултатите покажуваат дека WebAssembly модулите имаат барем 10 пати побрзо време на стартување во споредба со контејнерите, придружено со побрзо време на извршување и подобри перформанси при влезно/излезни операции. При тоа, WebAssembly околините специфично оптимизирани за извршување модули на серверска страна, како Lucet и SSVM покажуваат уште подобри резултати. Иако придобивките од аспект на брзина на стартување се импресивни, во одредена литература се известува за полоши сурови перформанси при изведување пресметки со WASM наспроти нативното извршување. Трудовите кои се занимаваат со оваа тема [170], [178], [179], [180] посочуваат забавувања помеѓу 1,5 до 10 пати во најекстремните случаи. Вреди да се забележи дека одредено ниво на забавување е секако очекувано, бидејќи во дадениот случај станува збор за присуство на дополнителен апстракциски слој лоциран помеѓу крајниот бинарен код кој се извршува и хардверот, во форма на WebAssembly виртуелната машина. Од друга страна, оваа покомплексна архитектура овозможува значајни безбедносни придобивки.

Одредени автори имаат направено обид да прошират некои од постојните безсерверски платформи за да поддржуваат и извршување функции во WebAssembly околини. Марфи и др. [181] опишуваат пристап за додавање поддршка за WASM модули кај OpenWhisk и AWS Lambda. Нивниот пристап подразбира повикување WebAssembly извршна околина преку Node.js помошен код, со цел да се избегне потребата од правење големи промени во логиката на самите безсерверски платформи кои може да бидат и од затворен карактер. Хал и др. [62] го демонстрираат користењето на WebAssembly за безсерверско пресметување на работ од мрежата, додека, пак, Мекитало и др. [182] го идентификуваат како идна најпогодна технологија за извршување пресметки поврзани со IoT.

Со примената на WebAssembly во контекст на безсерверското пресметување, може да се надминат постојните проблеми, но и да отворат целосно нови кориснички сценарија кои

претходно не биле возможни. За остварување на идејата за целосен облак-раб екосистем од безсерверски инфраструктури потребна е меѓусебна интеграција на голем број различни технологии, меѓу кои клучни се WebAssembly и контејнерите. Во моментов, во литературата, не постои истражување за можната интеграција на WebAssembly модулите со веќе достапен контејнеризациски софтвер и за начините како таквата потенцијална интеграција би влијаела врз времињата за ладен старт и перформансите. Целта на ова поглавје е да го надмине токму овој недостаток и да истражи како контејнерите, во улога на извршна околина, може да се комбинираат заедно со новите и поефикасни алтернативи подобро прилагодени за работ на мрежата.

4.3. Интеграција помеѓу традиционални контејнерски околини и WebAssembly

За остварување на зацртаната цел околу интеграција на WebAssembly околини со постојни контејнеризациски решенија, но и за евалуација на перформансите од ваквиот систем, избрани се 3 софтверски решенија кои овозможуваат извршување на WASM модули на серверската страна. Со цел да се добијат релевантни и споредливи резултати, искористена е адаптирана свита од вкупно 13 теста, комбинација од реални сценарија и микро тестови.

4.3.1. Избор на WebAssembly извршни околини

Постојат десетици WebAssembly извршни околини кои во моментов се во фаза на активен развој или повремено одржување во форма на закрпи за идентификуваните пропусти [183]. Некои од нив се наменети исклучиво за користење на клиентската страна, додека други се оптимизирани за самостојно извршување WebAssembly задачи на серверска страна. Има и примери каде одредена околина имплементира функционалности надвор од тековната WebAssembly спецификација, со цел добивање предност во споредба со конкуренцијата. Имајќи го предвид ваквиот разнолик WebAssembly екосистем, при процесот на одлучување кои околини да бидат предмет на испитување, употребени се следните критериуми:

- Извршната околина мора да поддржува покренување WebAssembly модули самостојно, без потреба од користење веб прелистувач.
- Мора да има целосна поддршка за WASI.
- Изворниот код мора да е јавно достапен со слободна лиценца.
- Решението мора да е активно одржувано.
- Постои техничка можност за интеграција со контејнерска извршна околина преку користење софтверски посредници (англ. shims), овозможувајќи инстанцирање на WebAssembly модули преку истиот програмски или команден интерфејс.

Со примената на овие критериуми врз достапните WebAssembly решенија, се издвојуваат 3 кандидати: WasmEdge [184], Wasmtime [185] и Wasmer [186]. Табела 4.1 ја споредува нивната функционалност, вклучувајќи ги и оние решенија што не го поминаа процесот на селекција.

Сите избрани извршни околини може да бидат интегрирани преку посредници со Containerd извршувачот на контејнери [187]. Еден од начините за реализација на поврзувањето е со помош на Crun [188], извршна околина наменета за иницијативата на отворени контејнери (англ. open container initiative, OCI) и програмска библиотека, поддржана од Containerd. Crun може да извршува стандардни OCI контејнери со помош на контејнеризациските механизми понудени од Linux јадрото, како и WebAssembly модули со користење на некоја од трите избрани WASM извршни околини.

Табела 4.1

Споредба меѓу разгледуваните WebAssembly извршни околини

Околина	**Јазик**	**Заедница**[14]	**Ѕвезди**[15]	**Независност**[16]	**WASI**	**Инт.**[17]
WasmEdge	C++	124	5,2	✓	✓	✓
Wasmtime	Rust	341	11,2	✓	✓	✓
Wasmer	Rust	131	14,2	✓	✓	✓
Wasm3	C	54	5,8	✓	✓	X
Awsm	C	6	210	✓	X	X
Wamr	C	106	3,4	✓	✓	X
Wasmi	Rust	40	1	✓	~[18]	X
Spiderm.	C++	/[19]	/[20]	X	X	X
V8	C++	461	20,7	X	X	X

За поопсежна анализа на потенцијалните разлики во перформансите на извршување во поглед на ладниот старт, вклучени се уште две решенија. Овие решенија користат неизменета верзија на Containerd за извршување на апсолутно истите функции како и WASM околините, но овојпат извршните датотеки се поставени во стандардни контејнери, без WebAssembly. За добивање на стандардните извршни верзии се користи основниот компајлер за јазикот во кој е напишана дадената функција и потоа се прават две посебни контејнерски слики. Првата е базирана на distroless основната слика [189], а втората на често користената debian:bullseye основа. Distroless сликите се помали од аспект на големина во споредба со алтернативите, бидејќи се вклучени само оние програмски библиотеки апсолутно неопходни за извршување на задачата. Сепак, користењето на ваквите distroless слики не е популарно колку користењето на постандардна основа како на пример debian:bullseye, со што се оправдува вклучувањето и на навидум понеефикасна алтернатива во тестирањето. Дополнително, досегашното искуство покажува дека голем број на безсерверски платформи со отворен код ги користат стандардните debian или ubuntu слики како основа за сите функции.

4.3.2. Свита за тестирање WebAssembly безсерверски функции

Табела 4.2

Опис на функциите дел од свитата за тестирање

Име	**Јазик**[21]	**Опис**	**Тип**[22]
audio-sine-wave	Rust	440Hz синусоиден бран како .wav датотека	M
fuzzysearch	Rust	Пребарување текст во датотека	P
n-body	Rust	Моделирање на движењето на планети	M
prime-numbers	Rust	Барање на прости броеви во првите n броеви	M
whatlang	Rust	Одредување на природен јазик за пишан стринг	P
zip-compression	Rust	Компресија на датотеки во ZIP архива	P
aes	Go	Шифрирање текст со AES алгоритамот	P

[14] Број на луѓе кои учествуваат во развојот на решението.

[15] Број на луѓе кои имаат искажано интерес за решението преку означување на GitHub платформата.

[16] Можност за независно извршување модули (пр. надвор од веб прелистувач).

[17] Можност за интеграција со Containerd.

[18] Делумна поддршка.

[19] Непознато бидејќи проектот не е поставен на GitHub или GitHub не е основната локација за развој.

[20] Непознато бидејќи проектот не е поставен на GitHub или GitHub не е основната локација за развој.

[21] Програмски јазик во кој е напишана функцијата.

[22] M – Микро тест; P – Реално сценарио.

checksum	Go	Калкулирање на MD5, SH256, and SHA512 хешови	P
diskio	Go	Пишување и читање 100000 линии во датотека	M
float-operation	Go	Калкулација sin, cos, и квадратен корен од даден број	M
imageprocessing	Go	Ротација на слика и промена на бои	P
linear-equations	Go	Решавање системи линеарни равенки	M
matmul	Go	Множење квадратни матрици	M

Поради тековните ограничувања на WebAssembly не е возможно едноставно користење на некоја од постојните свити на безсерверски тестови. За надминување на проблемот, креирана е нова свита на тестови сочинета од 13 различни функции, инспирирани од претходно користениот FunctionBench. Повеќе информации за сите тестови се дадени во табела 4.2. Целата програмска логика за тестовите е адаптирана од постојни програмски репозиториуми со отворен код [73], [132] или со користење примери од документацијата на популарни програмски библиотеки (исто така со отворен код) [190], [191], [192], [193], [194], [195], [196]. За оние примери каде што е неопходно користење трети програмски библиотеки, особено внимание се обрнува за да се овозможи сериско извршување без нишки при компајлирањето или да се најде алтернативна библиотека која самата по себе овозможува сериско извршување.

4.3.3. Процес на извршување на тестовите

Стратегијата за извршување на секој од вкупно 13 тестови вклучува:

1. За секој тест (функција) се креираат 8 различни OCI слики, тестирајќи специфични аспекти за секоја извршна околина:
   i. навремено (англ. just in time, JIT) компајлирање при извршување или, алтернативно, интерпретирање, за секоја од трите WebAssembly извршни околини (WasmEdge, Wasmtime, Wasmer);
   ii. предвремено (англ. ahead of time, AOT) оптимизирано компајлирање користејќи го препорачаниот WASM компајлер за секоја од трите WebAssembly извршни околини (WasmEdge, Wasmtime, Wasmer);
   iii. контејнерска слика користејќи distroless како основа, која ја содржи исклучиво извршната датотека за функцијата, компајлирана за x86 архитектурата со помош на основните параметри за оптимизација на користениот компајлер, во зависност од изворниот програмски јазик;
   iv. контејнерска слика користејќи debian:bullseye како основа, која ја содржи исклучиво извршната датотека за функцијата, компајлирана за x86 архитектурата со помош на основните параметри за оптимизација на користениот компајлер, во зависност од изворниот програмски јазик.
2. Прикачување на контејнерската слика на регистар за контејнери (англ. container registry).
3. Предвремено преземање на сите контејнерски слики на машините на кои се врши тестирањето, со цел да се елиминира времето потребно за пренос како фактор во резултатите.
4. Извршување на секоја OCI слика 100 пати врз посебна физичка машина која не се користи за ниту една друга активност или опслужување мрежна услуга.
5. Мерење на севкупното потребно време за извршување користејќи ја time алатката достапна кај UNIX базираните оперативни системи.

Се користат OCI слики за дистрибуција на WebAssembly модулите и нивно покренување од страна на трите WASM извршни околини. Овие OCI слики се создаваат користејќи празна scratch основа. Над неа се поставува само еден единствен дополнителен слој кој ја содржи бинарната WebAssembly датотека за покренување во соодветната извршна

околина. При покренување на модулот, бинарната датотека се екстрахира од OCI сликата и се предава директно на извршната околина. Како што е и пракса, AOT компајлирањето се изведува на истиот систем каде е извршувањето, со цел да се осигура користењето на најсоодветните оптимизации за дадената компјутерска архитектура.

Од технички аспект, Crun компонентата потребна за Containerd да извршува WebAssembly модули поддржува и интерпретација и AOT компајлирано извршување за WasmEdge, додека пак за останатите две околини – Wasmtime и Wasmer достапно е само JIT компајлирано извршување. За добивање порепрезентативни резултати рачно е додадена поддршка за AOT компајлирано извршување и за Wasmtime и за Wasmer.

За гарантирање на веродостојноста и повторливоста на креираните OCI слики се користи повеќефазно градење (англ. multi-staged builds), со што последно добиениот софтверски артефакт не вклучува непотребни алатки користени само при процесот на компајлирање на изворниот код. За извршување на сите експерименти се користи група од 5 работни станици со комплетно идентична хардверска конфигурација како онаа опишана во минатото поглавје. Три од петте машини служат за извршување WebAssembly модули со различните WASM извршни околини, една е посветена на извршување стандардни OCI контејнери со неизменета верзија на Containerd и последната се користи за градење на секоја од сликите, вклучително и оние со AOT компајлирани WASM модули. При извршувањето на сликите, иако тие веќе се однапред преземени и достапни, се води сметка секогаш да станува збор за чиста инсталација на Containerd, за да се избегне какво било кеширање на некој од поединечните слоеви, а со тоа и негативно да се влијае врз релевантноста на крајните резултати и мерењата околу ладниот старт.

Сите дискутирани промени може веднаш да се применат врз која било безсерверска платформа што под себе го користи Containerd како извршувач за контејнерите [11], [12], [148]. Тестирање врз комерцијални безсерверски платформи во моментов не е возможно, бидејќи тие сè уште не поддржуваат WebAssembly, ниту, пак, нудат опции како крајните корисници да направат измена на извршната околина.

4.4. Испитување на перформансите во пракса

Резултати од сите 100 извршувања на секоја од 13 функции користени за тестирање на 8 различни околини се дискутирани во продолжение. 6 од овие околини се базираат на WebAssembly, а последните 2 околини се традиционални контејнеризациски средини кои користат различни контејнерски слики.

4.4.1. WasmEdge

WasmEdge е повеќенаменска WebAssembly извршна околина фокусирана примарно на IoT сценарија. Официјално е дел од портфолиото алатки на Фондацијата за одомаќинето пресметување во облакот (англ. Cloud Native Computing Foundation, CNCF) и тоа во фаза на тестирање (англ. sandbox). Нуди поддршка како за WASI, така и за дополнителни WebAssembly функционалности кои сè уште се во фазата на дефинирање и не се официјално дел од спецификацијата [197].

WasmEdge има можност или да ги интерпретира WebAssembly модулите директно, како што тие се извршуваат или, пак, да покренува модули за кои е извршено AOT компајлирање. Користејќи го wasmedgec компајлерот возможно е да се изврши прекомпајлирање на постоечки WASM модули со цел тие да станат AOT, што би требало да доведе и до поефикасно и побрзо нивно извршување. Сепак, основниот режим на работа на WasmEdge е режимот на интерпретирање, а причината за тоа е полесно дебагирање, но по цена на намалени перформанси.

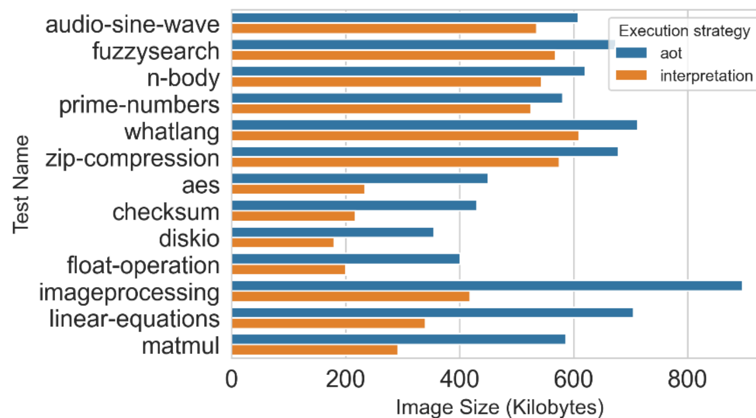
Вреди да се напомене дека при компајлирањето предодредените нагодувања користат оптимизациско ниво 2. Wasmedgec ја употребува истата терминологија за нивоа на оптимизација како и познатиот clang, а ниво 2 се дефинира како средно ниво, каде што е овозможен најголемиот број оптимизации [198]. Во табела 4.3 е покажана споредба помеѓу времињата за извршување на 13 различни функции кога тие се извршувани во режим на интерпретирање и при нивно AOT компајлирање.

Табела 4.3

Споредба помеѓу времето на извршување при AOT компајлирање и при интерпретација кај WasmEdge

Име	**AOT време**	**Интерпретација време**	**Забрзување**[23]
audio-sine-wave	0,8011s	0,9675s	x1,21
fuzzysearch	0,8162s	3,3380s	x4,09
n-body	0,7696s	0,7155s	x0.93
prime-numbers	0,7637s	0,8491s	x1,11
whatlang	0,7929s	0,7369s	x0,93
zip-compression	3,0367s	36,9394s	x12,16
aes	0,8178s	3,1143s	x3,81
checksum	2,8217s	420,7572s	x149,11
diskio	0,9131s	13,2456s	x14,51
float-operation	0,8006s	0,6979s	x0,87
imageprocessing	5,5801s	957,1059s	x171,52
linear-equations	0,9700s	31,2551s	x32,22
matmul	3,6132s	709,4923s	x196,36

Анализирајќи ги резултатите, се забележува дека средното време на AOT извршување е помало од она за интерпретација во 10 од 13 теста. Очекувано, коефициентот на забрзување е правопропорционален со комплексноста на функцијата, т.е. функции со поголеми пресметковни побарувања како checksum, imageprocessing и matmul се повеќе од 100 пати побрзи при AOT компајлирано извршување отколку при интерпретација.



Слика 4.1 Големина (во килобајти) на секоја од контејнерските слики за тестирање на WasmEdge (интерпретација наспрема AOT)

Интересно, при 3 теста (n-body, whatlang, float-operation) интерпретираното извршување покажува маргинално подобри резултати споредено со AOT. За сите 3 случаи важи дека станува збор за едноставни функции кои не побаруваат големи пресметковни ресурси. Дополнително, како што може да се види и од слика 4.1, големините на OCI сликите

[23] Претставува колку пати AOT извршување е побрзо од интепретацијата во конкретниот случај. Забрзување помало од 1 означува дека интепретацијата е побрза од AOT.

користени за дистрибуција на модули кои се интерпретираат се помали во споредба со оние за AOT компајлираните верзии. Ваквата разлика во големини исто така може да служи како објаснување зошто во споменатите три случаи интерпретацијата е побрза од AOT компајлирањето. На помалите слики им е потребно помало време да бидат вчитани во работната меморија, што заедно со малата пресметковна комплексност води до дополнително подобрување на времето на извршување.

4.4.2. Wasmtime

Wasmtime е најстарата WebAssembly извршна околина меѓу трите кои ги задоволуваат критериумите за евалуација. Во 2022 беше постигнато важно достигнување со објавата на првата стабилна верзија подготвена за продукциско работење – 1.0.0. Wasmtime поддржува два режима на извршување, JIT или AOT компајлирање. Од перспектива на перформансите, JIT би требало да биде побрзо од претходно дискутираното интерпретирање, но побавно од веќе однапред извршено компајлирање, како што е случајот со AOT. Кога станува збор за JIT, оптимизациите се прават штом WebAssembly модулот е покренат, но пред соодветниот код да започне со извршување. Еднаш направени, оптимизациите се кешираат за идна употреба, со цел да не се повторува истиот макотрпен процес (од аспект на пресметковни ресурси). Имајќи предвид дека Crun библиотеката не поддржува покренување WASM модули користејќи го Wasmtime со AOT стратегија, направени се рачно измени за додавање на потребната функционалност. При предвременото компајлирање се користи предодреденото ниво на оптимизации 2, аналогно на WasmEdge. Во табела 4.4 е прикажана споредбата помеѓу времињата потребни за AOT и JIT извршување на секоја од функциите.

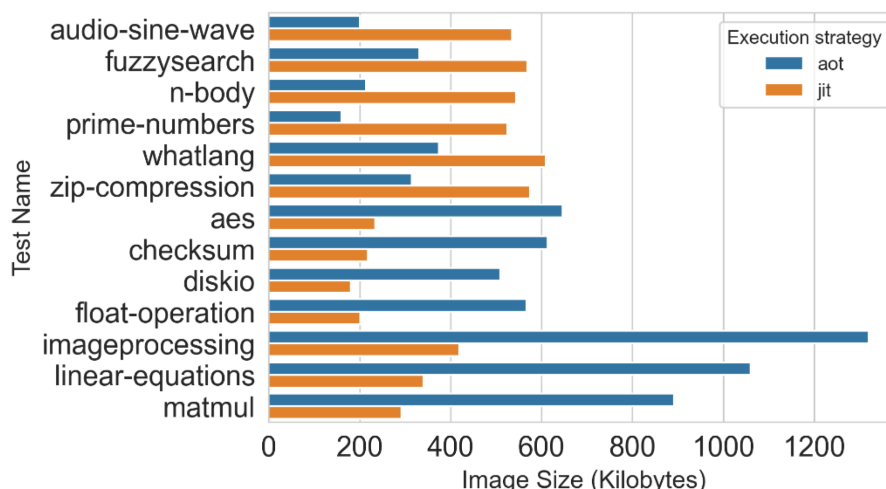
Табела 4.4

Споредба помеѓу времето на извршување при AOT компајлирање и JIT кај Wasmtime

Име	**AOT време**	**JIT време**	**Забрзување**[24]
audio-sine-wave	0,6232s	0,8178s	x1,31
fuzzysearch	0,7046s	0,8600s	x1,22
n-body	0,7210s	0,8373s	x1,16
prime-numbers	0,8438s	0,8021s	x0.95
whatlang	0,6722s	0,8547s	x1,27
zip-compression	3,0558s	3,2597s	x1,07
aes	0,7370s	1,2215s	x1,66
checksum	3,6143s	4,1259s	x1,14
diskio	0,8756s	1,3369s	x1,53
float-operation	0,7048s	1,1013s	x1,56
imageprocessing	8,6605s	9,8001s	x1,13
linear-equations	0,9241s	1,5558s	x1,68
matmul	6,3049s	6,9273s	x1,10

Резултатите покажуваат дека AOT компајлираното извршување е побрзо од JIT во 12 од 13 теста, со забрзувања кои се движат помеѓу 1,07 (zip-compression) и 1,68 пати (linear-equations). Пресметката на прости броеви (prime-numbers) е единствениот тест каде што JIT компајлирањето е благо побрзо од AOT. Станува збор за задача која не бара преголеми пресметковни перформанси, бидејќи врши барање на прости броеви во опсегот од 1 до 100, па најголемото време е всушност потрошено на инстанцирање на околината.

[24] Претставува колку пати AOT извршување е побрзо од JIT во конкретниот случај. Забрзување помало од 1 означува дека JIT е побрз од AOT.



Слика 4.2 Големина (во килобајти) на секоја од контејнерските слики за тестирање на Wasmtime (JIT наспрема AOT)

Анализирајќи ги големините на соодветните OCI слики за секој од тестовите прикажани на слика 4.2, може да се забележи делумна изедначеност помеѓу AOT и JIT стратегиите. Во 7 од 13 теста (aes, checksum, diskio, float-operation, imageprocessing, linear-equations, matmul) сликите за AOT WASM модулите се поголеми од алтернативите компајлирани со JIT стратегијата. Но, во преостанатите 6 теста (audio-sinewave, fuzzysearch, n-body, prime-numbers, whatlang, zip-compression) ситуацијата е свртена наопаку и всушност JIT сликите се поголеми од оние со AOT компајлирање на модулите.

4.4.3. Wasmer

Wasmer WebAssembly извршната околина има најголем број на GitHub ѕвездички меѓу тестираните решенија, а и најрано ја има публикувано својата прва стабилна верзија, 1.0.0, во јануари 2021 година. Wasmer самиот поддржува и JIT и AOT компајлирано извршување. Слично како и за Wasmtime, направени се рачни измени во изворниот код на Crun, со цел имплементирање на AOT извршување. При AOT компајлирање, стратегијата е иста како и за WasmEdge и Wasmtime, користејќи ги основните нагодувања на Wasmer компајлерот. Во табела 4.5 се прикажани резултатите за 6 од вкупно 13 тестни функции. Недостасуваат сите функции кои извршуваат влезно/излезни операции врз податочниот систем, бидејќи иако Wasmer поддржува WASI, не е возможен датотечен пристап при интеграцијата со Crun.

Резултатите за оние 6 функции кои успешно ги завршија своите 100 извршувања покажуваат дека сите имаат подобри AOT перформанси во споредба со JIT. Коефициентите на забрзување се протегаат од скромни 1,08 пати (matmul) до позначајни 1,65 пати (linear-equations).

Табела 4.5

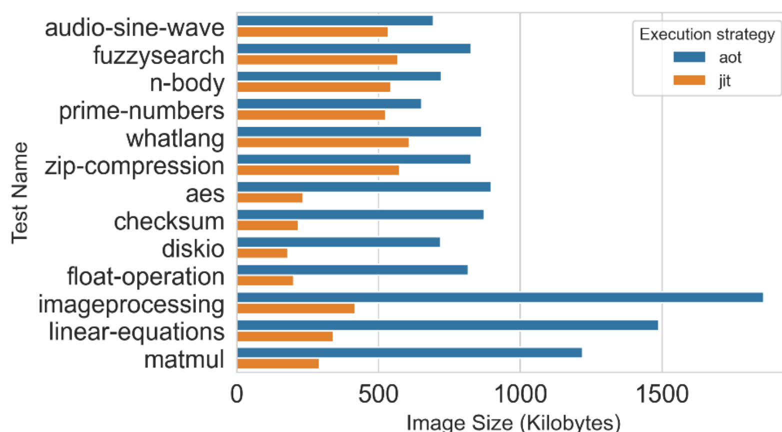
Споредба помеѓу времето на извршување при AOT компајлирање и при JIT кај Wasmer

Име	**AOT време**	**JIT време**	**Забрзување**[25]
n-body	0,7248s	0,8320s	x1,15
prime-numbers	0,7017s	0,8298s	x1,18
whatlang	0,7346s	0,8404s	x1,14

[25] Претставува колку пати AOT извршување е побрзо од JIT во конкретниот случај. Забрзување помало од 1 означува дека JIT е побрз од AOT.

float-operation	0,7508s	1,2034s	x1,60
linear-equations	1,0072s	1,6638s	x1,65
matmul	6,5564s	7,0870s	x1,08

Со цел да се добие сеопфатна слика за големините на OCI сликите при користењето на Wasmer, анализата ги вклучува и оние функции кои не се извршуваат успешно. Според слика 4.3, забележливо е дека во сите ситуации сликите за AOT компајлираните модули се поголеми во споредба со JIT алтернативата. Разликата е најголема во случајот со imageprocessing тестот, карактеристичен по тоа што користи дополнителни софтверски библиотеки за обработка на фотографиите.



Слика 4.3 Големина (во килобајти) на секоја од контејнерските слики за тестирање на Wasner (JIT наспрема AOT)

4.4.4. Стандарден пристап со контејнери

Заокружувајќи го испитувањето на перформансите на различните околини во пракса, истите 13 функции се извршуваат и во стандардни OCI контејнери, како што е типично за најголемиот дел од безсерверските платформи денес. На овој начин може да се направи директна споредба на карактеристиките помеѓу WebAssembly како извршна технологија од една страна и контејнеризацијата, од друга.

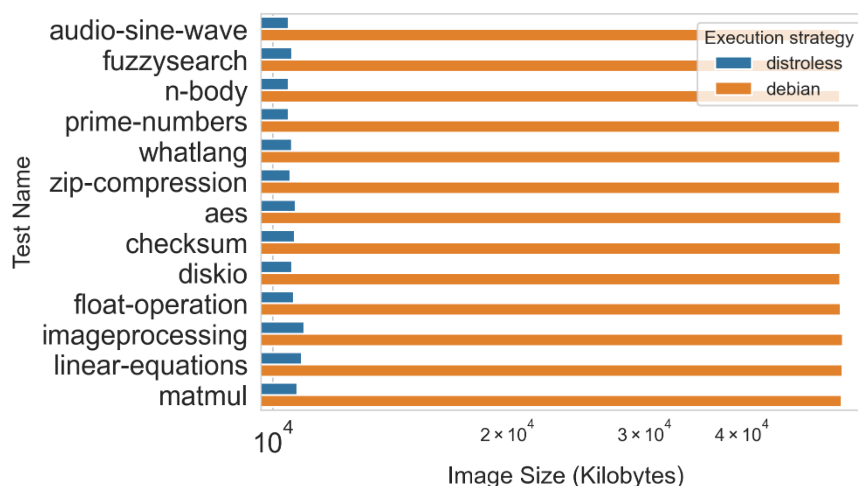
Во табела 4.6 се прикажани разликите во времињата за извршување на контејнерите кога се користи distroless основна слика, наспроти сценариото во кое се користи традиционална основна слика, со повеќе зависности, т.е. debian:bullseye.

Во сите 13 случаи сликите базирани на distroless покажуваат подобри резултати, со забрзувања во опсегот од 1,51 (zipcompression) до 2,42 пати (fuzzysearch). Ваквите резултати и не се толку изненадувачки, ако се земе предвид дека debian:bullseye сликите се околу 5 пати поголеми во споредба со distroless алтернативите, што резултира со поголемо време на вчитување на функцијата во меморија, пред да започне нејзиното извршување. Ако се занемари иницијалното доцнење за покренување на сликата и се разгледуваат исклучиво пресметковните перформанси по стартувањето, не се очекуваат разлики, бидејќи станува збор за апсолутно идентична бинарна датотека. На слика 4.4 се прикажани разликите во големините на сликите кога станува збор за distroless и debian:bullseye основите.

Табела 4.6

Споредба помеѓу времето на извршување на функциите при користење на distroless и debian:bullseye контејнерските слики како основа

Име	**Distroless време**	**Debian време**	**Забрзување**[26]
audio-sine-wave	1,5910s	3,5241s	x2,22
fuzzysearch	1,4540s	3,5214s	x2,42
n-body	1,7005s	3,5800s	x2,11
prime-numbers	1,7005s	3,5919s	x2,11
whatlang	1,4946s	3,5440s	x2,37
zip-compression	3,9159s	5,9046s	x1,51
aes	1,8785s	4,1197s	x2,19
checksum	2,7728s	4,8078s	x1,73
diskio	1,6408s	3,7154s	x2,26
float-operation	1,5234s	3,5300s	x2,32
imageprocessing	2,4391s	4,5162s	x1,85
linear-equations	1,5887s	3,5529s	x2,24
matmul	2,6960s	4,7505s	x1,76



Слика 4.4 Големина (во килобајти) на секоја од контејнерските слики за тестирање на distroless и Debian:bullseye

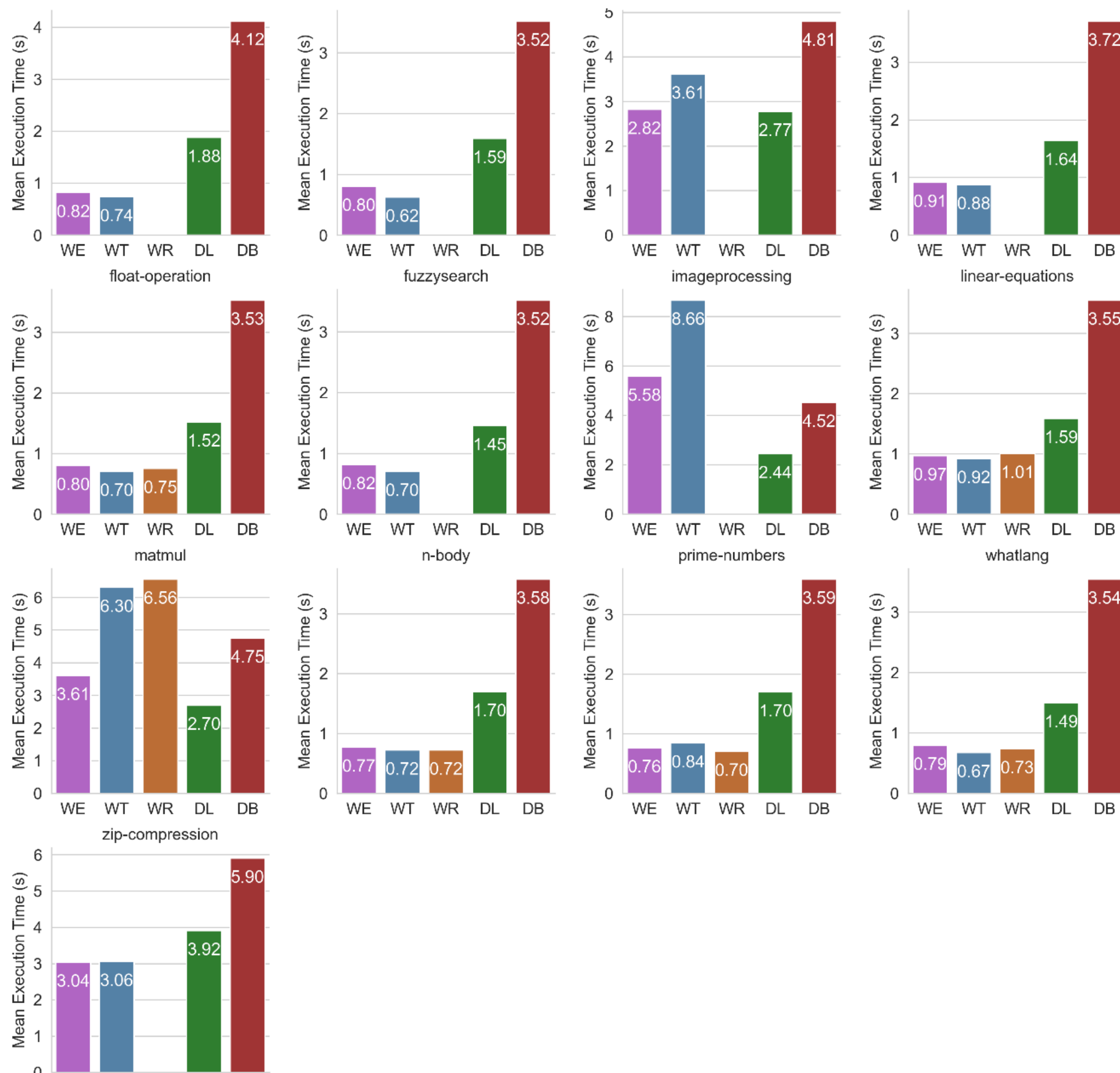
4.5. Избор на најсоодветна WebAssembly извршна околина за безсерверски функции

Анализирајќи ги заедно сите резултати добиени од тестирањето различни извршни околини, се забележува дека низ сите околини AOT компајлираните WebAssembly функции имаат подобри перформанси споредено и со интерпретацијата и со JIT компајлирањето во најголемиот број случаи. Дополнително, може да се изведе и заклучокот дека OCI сликите креирани со scratch слој како основа се за барем една величина помали од алтернативните слики кои користат слика со целосна GNU/Linux дистрибуција како основа.

Со цел вршење на поопсежна споредбена анализа и избирање најсоодветна WebAssembly околина за интегрирано извршување безсерверски функции заедно со контејнери во насока на овозможување транспарентни пресметки во облак-раб екосистемот, на слика 4.5 се прикажани средните времиња на извршување при AOT компајлирање за сите

[26] Претставува колку пати во конкретниот случај користењето на distroless основна слика при извршувањето е побрзо од debian:bullseye.

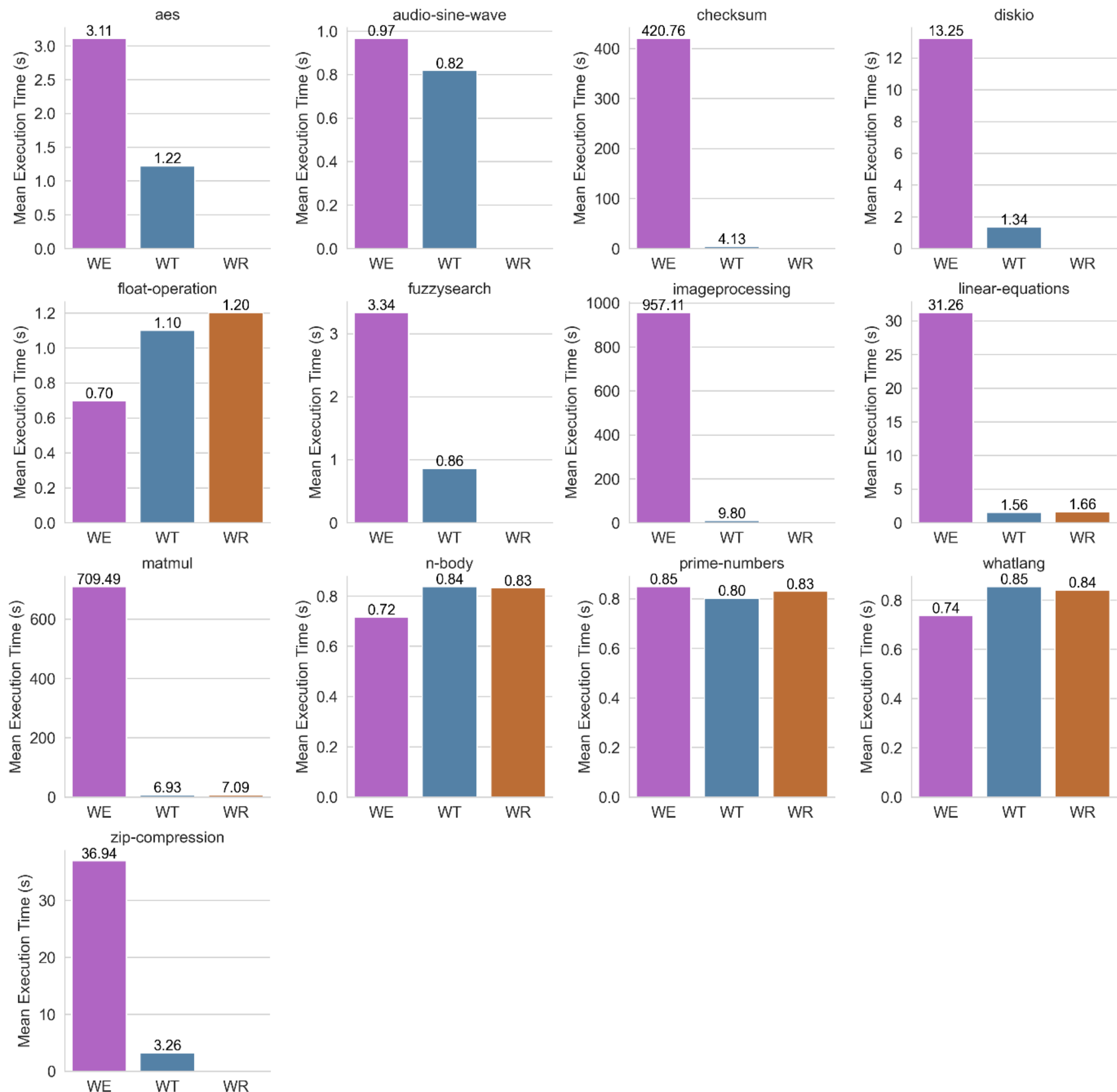
платформи. Исклучително значајно е што во 10 од 13 теста сите 3 WebAssembly извршни околини постигнуваат подобри резултати во споредба со традиционалните контејнери (aes, audio-sine-wave, diskio, float-operation, fuzzysearch, linear-equations, n-body, prime-numbers, whatlang, zip-compression). Во случајот со checksum тестот, контејнерското извршување во distroless слика е побрзо во споредба со двете WebAssembly околини кои успешно го завршија тестот (WasmEdge и Wasmtime), а, пак, извршувањето со debian:bullseye сликата како основа е најбавно. При тестирање на перформансите за манипулација на слика (imageprocessing), што е пресметковно интензивен процес, традиционалните контејнерски извршни околини покажуваат подобри резултати. За крај, при множење матрици (matmul) резултатите се разнолики, т.е. WasmEdge е побавен од distroless, но побрз од debian:bullseye, додека, пак, преостанатите две WebAssembly извршни околини заостануваат зад debian:bullseye.



Слика 4.5 Споредба на средното време на AOT извршување меѓу петте поддржани извршни околини (WE-WasmEdge; WT-Wasmtime; WR-Wasmer; DL-distroless; DB-debian:bullseye)

Ако за момент вниманието се префрли исклучиво на резултатите постигнати од страна на WebAssembly извршните околини при AOT компајлирано извршување, Wasmtime нуди најдобри перформанси во 8 од 13 теста (aes, audio-sine-wave, diskio, float-operation, fuzzysearch, linear-equations, n-body, whatlang), WasmEdge во 4 (checksum, imageprocessing, matmul, zip-compression), а Wasmer само во еден (whatlang).

На слика 4.6 е дадена споредба помеѓу алтернативните режими на извршување кои не се AOT компајлирани кај трите извршни околини. Во случајот со WasmEdge, прикажани се резултати од интерпретирањето, имајќи предвид дека JIT компајлирање не е поддржано. Во останатите случаи прикажани се времињата при JIT компајлирано извршување. Резултатите за distroless и debian:bullseye контејнеризираните извршувања се намерно изоставени.



Слика 4.6 Споредба на средното време на интерпретација (WasmEdge) и JIT (Wasmtime, Wasmer) меѓу трите поддржани извршни околини (WE-WasmEdge; WT-Wasmtime; WR-Wasmer)

Продолжувајќи со анализата, може да се заклучи дека методот на интерпретација претставен со WasmEdge, е најбавен во сите случаи, со исклучок на 3 (floatoperation, n-body, whatlang). Интересно, во тие три случаи, интерпретацијата е побрза од JIT компајлираното извршување и на Wasmtime и на Wasmer. Објаснувањето за ова однесување е што тестовите кои се предмет на разгледување не се пресметковно интензивни. Од тука се изведува заклучокот дека самиот процес на JIT комапјлирање е побавен отколку само да се интерпретираат едноставните инструкции, како што ќе се наиде на нив. Сепак, предностите на JIT во однос на интерпретацијата се драстично

видливи кај комплексни тестови (checksum, diskio, imageprocessing, lilnearequations, matmul, zip-compression). Тука JIT компајлираното извршување е побрзо за цела величина. Wasmtime е најбрзата WebAssembly извршна околина при JIT извршување во 10 од 13 функции, но, од друга страна, Wasmer не учествуваше во сите тестови поради проблемите со влезно/излезните операции.

По направената анализа, неоспорно е дека WebAssembly нуди јасни придобивки од аспект на намалување на времето за ладен старт споредено со традиционалното извршување во контејнеризирани околини. WebAssembly успева да постигне вкупни времиња за извршување на целата функција, заедно со нејзиното подигнување и спуштање, кои се помали од една секунда при низа тестови, што е значајно достигнување. Сепак, кога станува збор за функции кои извршуваат покомплексни пресметки, како на пример процесирање слики, компресија или голем број нумерички операции, времето заштедено со побрзиот ладен старт се анулира поради побавното вршење на пресметките во WASM средина. Ова побавно пресметување е резултат на поголемото ниво на апстракција и изолација во WebAssembly извршните околини, за разлика од нативното или контејнеризираното извршување.

Со ова дополнително се поткрепува тезата дека е неопходно безсерверските платформи од следната генерација да поддржуваат повеќе од една извршна околина. WebAssembly има јасно изразени предности кога станува збор за извршување пресметковно едноставни функции, каде што ладниот старт има голем удел во севкупното време на извршување. Во спротивност на ова, кога станува збор за пресметковно комплексни функции, контејнерите нудат подобри перформанси, но со времиња на ладен старт кои се и до цела величина поголеми од оние на WebAssembly.

Заклучокот за потреба од повеќе извршни околини во рамките на една безсерверска платформа е значаен чекор напред кон дополнително подобрување и унапредување на безсерверската парадигма. Пред да може придобивките од WebAssembly интеграцијата да бидат применети во пракса, постојат сè уште отворени прашања на кои недостасуваат одговори.

4.6. Отворени прашања

Денес сме во фаза каде што безсерверското пресметување во облакот ужива широка популарност со тенденција и за негово префрлање на работ од мрежата со што би се овозможило и транспарентно поврзување меѓу овие две локации и оформување на целосен облак-раб екосистем. Со ваквата поставеност на работите, веќе на површина испливуваат сите оние проблеми кои не беа решени во пораните фази од развојот, со цел избегнување прерана оптимизација. Досега изнесените резултати ја потврдуваат потребата од воведување нова извршна околина, која би служела како дополнување, а не како целосна замена на контејнерите. Со претставената стратегија за интегрирање на контејнерите со WebAssembly извршни околини се потврдува и првиот дел од поставената хипотеза, а тоа е *„Виртуелните машини и контејнерите како извршни околини може да се комбинираат заедно со новите и поефикасни алтернативи подобро прилагодени за работ на мрежата.“* Вториот дел од хипотезата кој се осврнува на крајната цел, т.е. *„креирање една унифицирана платформа која ќе понуди флексибилност преку избор на најсоодветната технологија за извршувањето на дадена задача“* е сè уште отворено прашање и побарува решение за управување и оркестрација на функции.

Оркестрацијата на WebAssembly безсерверски функции претставува сосема ново поле на истражување и е еден од најгорливите проблеми во моментов за кој актуелната научна

литература сѐ уште нема решение. Замислата за оваа оркестрација е да биде компатибилна со постојните алатки на контејнерите, со цел да се постигне транспарентност како од аспект на администрација на решението, така и од корисничка гледна точка. Следното поглавје, поглавјето 5 се занимава токму со изнаоѓање решение и давање одговор на ова екстремно важно прашање.

5. ОРКЕСТРАЦИЈА НА WEBASSEMBLY БЕЗСЕРВЕРСКИ ФУНКЦИИ

При изнаоѓањето сличности меѓу безсерверските платформи денес, особено оние со отворен код, еден елемент се издвојува и е навидум сеприсутен: изборот на решение за оркестрација. Најголемиот дел од безсерверските платформи денес, барем оние кои ги извршуваат функциите во контејнеризирана извршна околина, во заднина го користат Kubernetes како оркестратор за контејнери [199]. Оттука природно следува дека првичната интеграција помеѓу WebAssembly и постојните безсерверски инфраструктури треба да се случи на ниво на оркестраторот. Ваквото тврдење е дополнително поткрепено со фактот што веќе постојат начини како Kubernetes може да се прошири за оркестрирање на други извршни околини, како на пример стандардни виртуелни машини и микро виртуелни машини [200].

Во постоечката литература, не постои робусна интеграција помеѓу Kubernetes и WebAssembly која овозможува непречено оркестрирање безсерверски функции претставени како контејнери и WebAssembly. Целта на ова поглавје е да ја пополни идентификуваната празнина, темелејќи се на искуството и поуките извлечени од претходните поглавја во однос на начините за поврзување на контејнерски извршни околини со WebAssembly, но и перформансните карактеристики на најпопуларните WebAssembly извршни околини денес. Истражувањето во продолжение се фокусира на *„креирање една унифицирана платформа која ќе им даде флексибилност на корисниците да ја изберат најсоодветната технологија за извршувањето на дадена задача“*, што е и всушност вториот дел од хипотезата делумно потврдена со минатото поглавје, *„Виртуелните машини и контејнерите како извршни околини може да се комбинираат заедно со нови и поефикасни алтернативи подобро прилагодени за работ на мрежата со цел креирање една унифицирана платформа која ќе понуди флексибилност преку избор на најсоодветната технологија за извршувањето на дадена задача.“* Претставувањето успешно решение за оркестрација значи дефинирање на последното делче кое недостасува за WebAssembly да може да се користи како соодветна извршна околина за безсерверски функции во продукциски околини, рамо-до-рамо со останатите. Со надминувањето на проблемот на ладниот старт работ на мрежата се издигнува како соодветна околина за извршување безсерверски пресметки, со што се врши и приближување кон крајната цел, а тоа е овозможување транспарентни пресметки во облак-раб екосистемот.

5.1. Осврт кон постојните напори за WebAssembly оркестрација

Воведувањето на двата фундаментални WebAssembly концепти кои се предмет на широка експлоатација денес – WebAssembly моделот со компоненти и WebAssembly системскиот интерфејс, донесе значајни придобивки за безсерверските функции. Како што беше споменато претходно, WASI е искрата која овозможи WebAssembly да стане корисен за извршување задачи на серверска страна, додека, пак, со моделот со компоненти се овозможи честата функционалност да нема потреба да се имплементира од почеток во секој WASM модул. Ваквата можност за користење споделени компоненти е од особена важност токму за безсерверската парадигма, бидејќи би овозможило логиката за секоја функција да биде имплементирана еднаш и потоа споделена. На овој начин, програмерите треба да се грижат исклучиво за функционалноста на сопствената функција, додека сите други аспекти, како на пример избор и имплементација на стратегија за повикување, се препуштаат на посебна компонента која може да биде развиена и понудена од самата платформа. При интеграцијата на WebAssembly извршни околини со постоечки контејнерски извршувачи и оркестратори, од особена важност е да

се изберат меѓусебно компатибилни технологии, со цел да биде поддржано споделувањето на WebAssembly компоненти.

Kubernetes како најпопуларното оркестрациско решение денес, не само што може да служи за распоредување и извршување на безсерверски WebAssembly функции покренати од крајни корисници, туку може да вгради и WebAssembly компоненти во сопствената архитектура. Во актуелната литература сè поприсутен е трендот за намалување на хардверските побарувања за платформите кои целат да бидат поставени на работ од мрежата. Во оваа насока, Шебрехтс и др. [201] нудат решение како некои од Kubernetes составните компоненти би можеле да бидат имплементирани во форма на WebAssembly модули. На овој начин се тврди дека би се постигнало намалено користење ресурси, но и поголема преносливост, без потреба од рекомпајлирање за секоја посебна архитектура. Ова значи дека постои потенцијал WebAssembly да најде уште една дополнителна примена, како неразделен дел од лесни Kubernetes дистрибуции, аналогно на претходно евалуираните MicroK8s и K3s.

Од перспектива на оркестрациски можности за WebAssembly модули кои не се стриктно имплементирани врз Kubernetes, но и немаат поддршка за коегзистенција со останати извршни околини, денес се актуелни две решенија – WasmCloud и Spin.

5.1.1. WasmCloud

WasmCloud [202] е WebAssembly развојна рамка и платформа која го промовира концептот на користење т.н. актери и даватели на можности (англ. actors and capability providers). Главната цел е да се надмине проблемот на вклучување комплексна функционалност во секој WebAssembly модул поединечно, без експлицитно да се поддржи WebAssembly спецификацијата за моделот со компоненти. WasmCloud актерите се најмалото парче код чијашто задача е обработка и справување со индивидуални пораки предадени преку давателите на можности. Актерите дополнително може да искористат одредена функционалност преку поврзување со овие даватели на можности. Од друга страна, пак, давателите на можности се независни од крајните актери и ставаат на располагање често користена функционалност, како на пример справување со HTTP барања и одговори или примање и испраќање пораки од брокер на пораки. За актерите постои строго ограничување дека мора да бидат компајлирани во WebAssembly, но давателите на можности може да бидат претставени и од стандарден софтвер напишан во посакуван програмски јазик. WasmCloud поддржува оркестрација на актерите низ различни пресметковни јазли и нивно управување преку команден интерфејс или веб портал. Од аспект на дистрибуција, како актерите, така и давателите на можности се спакувани во форма на OCI слики. Во моментов, актерите може да бидат напишани во Rust или Go програмските јазици, а потоа компајлирани во WebAssembly модули. Се користи автоматско компајлирање и стартување со секоја промена на кодот при развивање на нов актер што го олеснува дебагирањето и отстранувањето грешки при програмирањето.

5.1.2. Spin

Spin [203], слично како и WasmCloud, претставува развојна рамка за WebAssembly апликации заснована врз Wasmtime извршната околина. Проблемот на вклучување комплексни зависности се решава на еден од два начина, во зависност од тоа кој изворен програмски јазик е користен. За оние јазици кои го поддржуваат моделот со компоненти, функциите напишани од крајните корисници може динамички да бидат поврзани со постоечки WASM модули што овозможуваат различни механизми за повикување. Во овој момент, поддржан е и HTTP протоколот, но и Redis редица на пораки за интеракција со

безсерверските функции. За останатите програмски јазици кои сè уште не го поддржуваат моделот со компоненти, развиена е посредничка компонента наречена Wagi. Начинот на функционирање во овој случај е многу сличен на архаичните скрипти од минатото кои го користеа општиот приклучувачки интерфејс (англ. common gateway interface, CGI). При повик на дадена функција, сите параметри внесени од корисникот се пресретнати од Spin и предадени на WASM модулот како параметри на стандарден влез. При враќањето одговор, функцијата се очекува да го испечати резултатот на стандарден излез. Потоа Spin ќе го спакува излезот во порака која ќе ја форматира во соодветен HTTP одговор.

Од аспект на оркестрација, направени се иницијални напори да се користи Hashicorp Nomad [204] оркестраторот за поставување на т.н. Fermyon Platform [205] . Но, во овој случај не постои директна интеграција со останати извршни околини, а и методот на дистрибуција на WASM артефактите не е во форма на OCI слики, туку со помош на трета алатка со засебен формат, Bindle [206].

5.1.3. Можности за интеграција

Тековно достапните оркестрациски решенија за WebAssembly модули не ги задоволуваат побарувањата кои би им овозможиле непречено користење во безсерверски средини како на работ, така и во облакот, ниту пак овозможуваат интеграција со останати извршни околини. Идеалното решение треба да биде засновано на добро позната WebAssembly извршна околина, за која е гарантирано дека ќе продолжи да се развива. Дополнително, потребно е користење на WebAssembly моделот со компоненти за да се обезбеди повисоко ниво на апстракција што би го олеснило дефинирањето нови функции, како за искусни корисници, така и за апсолутни почетници. За крај, за интеграцијата со останатите извршни околини да биде успешна, мора да се изнајде заеднички метод за дистрибуција на софтверските артефакти, без разлика дали станува збор за виртуелни машини, контејнери или, пак, WASM модули.

5.2. Патот до унифицирано оркестрирање на безсерверски функции

За остварување на крајната цел за обединета средина каде што контејнерите и WebAssembly модулите може да се извршуваат едни до други, оркестрирани од единствена инстанца на Kubernetes оркестраторот, потребни се низа прилагодувања низ различни софтверски компоненти. Прва тема на дискусија е начинот на проширување на Containerd за да извршува WebAssembly модули, користејќи софтверски спојки со WASM извршна околина. Следно е Kubernetes интеграцијата која овозможува оркестрација на WASM модули. Секако, овие аспекти треба да се тестираат со цел да се добијат и релевантни резултати за споредба со останатите техники дискутирани досега.

Иако во ова поглавје главен предмет на интерес е интеграцијата помеѓу WebAssembly и контејнери, ова може да се прошири и во триумвират, со вклучување виртуелни машини или микро виртуелни машини.

5.2.1. Проширување на постојните извршни околини за контејнери

Користејќи ги резултатите и сознанијата од претходните поглавја, заклучокот е дека најоптимално би било интеграцијата на Kubernetes со WebAssembly да се темели врз најефикасната WASM извршна околина – Wasmtime. Оттука, изборот се сведува на тоа дали да се користи нативната имплементација на Wasmtime или, пак, некое решение над него, кое би вовело дополнителни можности и слоеви на апстракција. Spin од една страна ја користи токму оваа извршна околина во позадина, а и дополнително има значајни додатни функционалности кои го олеснуваат развивањето нови безсерверски функции.

Задачата оттука па натаму се трансформира во прилагодување на Spin околината со цел интеграција со Kubernetes, но и воведување можност за дистрибуција на Spin WASM артефактите како OCI слики, наместо со официјално поддржаното Bindle решение.

Благодарение на модуларноста на Containerd, возможно е да се развие софтверска спојка помеѓу Containerd контејнер извршувачот од една страна и изменета верзија на Spin околината, од друга. OCI сликите за дистрибуција на артефактите би биле креирани слично на начинот опишан во минатото поглавје, од празен слој, кој во овој случај би содржел само две датотеки, бинарниот WASM модул и соодветната конфигурациска датотека потребна за негово извршување од страна на Spin околината.

Од аспект на функционалноста, широко е прифатена тезата дека за да се искористат до максимум предностите на безсерверската парадигма, потребна е непречена мрежна комуникација помеѓу функцијата и трети системи. Всушност, ваквото однесување е вградено уште во самата дефиниција за безсерверското пресметување: симбиоза помеѓу функција како услуга и заднина како услуга. Комуникацијата со сите заднински услуги се изведува преку мрежа. Во сегашната верзија на WASI спецификацијата, на WebAssembly му се достапни ограничени можности за работа со мрежни сокети. Но, со помош на Spin и моделот со компоненти, која било безсерверска функција може да оствари контакт со надворешниот свет или со стандардни HTTP барања, како што е обичај и кај други безсерверски платформи или, пак, преку испраќање пораки на Redis редица. На овој начин се постигнува еквивалентност во понудените функционалности, без разлика дали функцијата се извршува како контејнер или како WebAssembly. Стандардните BaaS услуги за складирање, извештаи, автентикација, авторизација и многу други, стануваат достапни за користење и во овој случај.

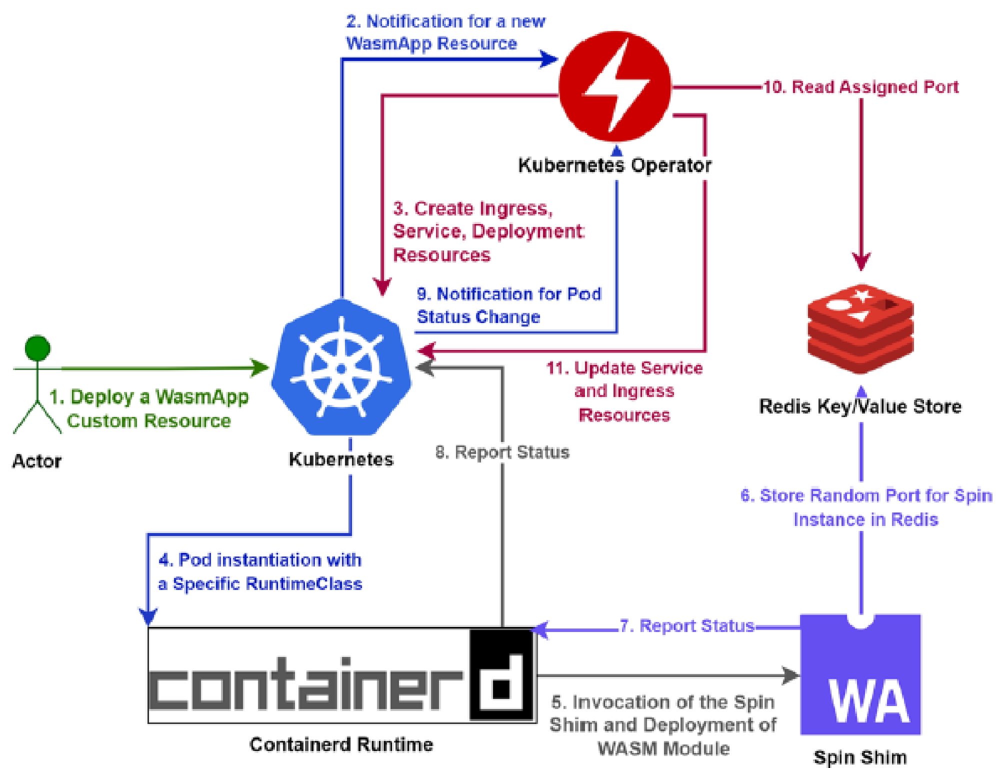
Со цел зголемување на безбедноста, Spin дозволува појдовна мрежна комуникација исклучиво кон локации експлицитно наведени во споменатата конфигурациска датотека, која се наоѓа во OCI сликата, заедно со бинарната верзија на модулот. Интеграцијата со остатокот од мрежата може да се направи на еден од два начина. Наједноставно е да се користи истиот мрежен логички сегмент (англ. network namespace) како и оној на домаќинот каде што е поставена функцијата. Иако овој пристап теоретски би нудел помало ниво на изолација, треба да се земе предвид дека ризикот се минимизира со ограничувањата поставени во конфигурациската датотека за далечни дестинации. Алтернативно, можна е и интеграција со самата контејнерска надмрежа преку интеграција со контејнерскиот мрежен интерфејс (англ. container network interface, CNI). Без разлика која опција ќе се избере, возможно е публикување на WebAssembly функциите користејќи ги стандардните Kubernetes конструкти како услуги (англ. services) и влезни контролери (англ. ingress controllers).

5.2.2. WebAssembly поддршка во Kubernetes

Со цел да се овозможи лесно поставување нови WASM модули преку процес кој би наликувал на оној за покренување контејнери, развиен е Kubernetes WASM оператор. Kubernetes операторите се трети компоненти кои имаат пристап до Kubernetes API серверот и може да слушаат за одредени настани за кои претходно би се регистрирале. При таков настан, контролерот може да реагира преку примена на специфична логика. Главниот концепт кај контролерите се заснова на т.н. очекувана и тековна состојба, каде што тековната состојба е отсликана од актуелните случувања во кластерот, додека, пак, очекуваната состојба е она кон што треба да се стреми, со помош на операторот. При секој настан од интерес, контролерот ја проверува тековната состојба и ако има отстапувања од онаа очекуваната, ги прави потребните промени со цел нивно

изедначување. Во конкретниот случај со WASM операторот, поставена е и нова дефиниција на посебен ресурс (англ. custom resource definition, CRD) за претставување на WASM апликации, наречена WasmApp. На овој начин се воведува дополнителен апстракциски слој, кој ги консолидира сите неопходни ресурси за подигнување WASM апликации (Kubernetes Deployment, Ingress, Service...) во еден единствен. Со ова се постигнува поедноставување, бидејќи наместо непотребно да се повторуваат статички елементи, дефиницијата на посебниот ресурс е дизајнирана да ги содржи само неопходните информации релевантни за дадениот контекст.

Користењето CRD кој изворно не е дел од Kubernetes спецификацијата е целосно поддржано од постоечки алатки. Всушност, ова е и препорачаниот начин за проширување на функционалноста на Kubernetes. Како за пример, WasmApp ресурсите би можеле да бидат инстанцирани користејќи го кој било постоечки управувач на апликации, како на пример Helm [207], во рамките на постоечка Helm карта, заедно со останати ресурси или целосно независно.



Слика 5.1 Процес на поставување нова WASM безсерверска функција

За непречена работа на претставениот Kubernetes оператор, неопходно е присуството на изменета верзија на софтверската спојка за Spin и Containerd. Редоследот на сите чекори извршени при покренување нов WasmApp ресурс се дадени на слика 5.1, а може да бидат сумирани на следниов начин:

1. Се креира нов ресурс од тип WasmApp во кластерот.
2. Се генерира нов настан, со кој се покренува логиката на операторот за изедначување на тековната состојба со посакуваната состојба. Операторот ги проверува параметрите наведени во WaspApp ресурсот и доколку се во ред, врши инстанцирање на основните Kubernetes ресурси неопходни за покренување на апликацијата. За секоја WasmApp, во најмала рака, потребно е дефинирање на:

2.1. Deployment, каде што се наведува и OCI контејнерската слика која ја содржи бинарната верзија на WASM модулот, посакуваната извршна околина за извршување и мрежните порти кои би требало да бидат отворени.
2.2. Service со кои се прават достапни наведените порти од Deployment ресурсот и се дозволува пристап до нив во рамките на кластерот.
2.3. Ingress со чија помош се врши јавно публикување на WasmApp со цел пристап и надвор од кластерот.

3. Со креирање на Deployment ресурсот се инстанцира нов Pod распределен за извршување на соодветен пресметковен јазол во кластерот. Локалната Kubelet компонента на избраниот јазол, задолжена за интеракција со Kubernetes API серверот, ќе ја повика изменетата верзија на софтверската спојка за Spin. Со ова HTTP слушачот ќе се стартува на произволна порта, споделена со Kubernetes операторот користејќи го Redis за зачувување на клуч-вредност парови.

4–9. Подигнување на WasmApp.

10. Во моментот кога дадениот Pod кој го претставува WASM модулот ќе влезе во состојбата „Ready“, Kubernetes операторот се обраќа до Redis инстанцата, ја чита случајно избраната порта и ги прави неопходните промени врз Service и Ingress објектите, со цел WASM модулот да биде јавно достапен и надвор од кластерот.

Истиот процес се повторува при каква било промена на спецификацијата на постоен WasmApp ресурс или при негово бришење. За извршување на развиениот оператор потребна е Kubernetes верзија поголема од 1.20, бидејќи тогаш е воведена неопходната поддршка за RuntimeClass која овозможува Kubernetes да биде свесен која софтверска спојка да ја искористи при подигнувањето на даден Pod. Со ова всушност се воведува поддршка за повеќе извршни околини [208].

Имајќи предвид дека во објаснувањето за тоа како се покренуваат WASM модули во Kubernetes беше користен терминот „Pod“, што честопати се асоцира со контејнери, вреди да се потенцира дека при користењето на Spin софтверската спојка, постои изолација на ниво на барање. Извршната околина не е споделена меѓу последователни барања, па веднаш при враќањето одговор до клиентот таа се уништува и при следното барање се креира од почеток. Ваквото однесување не само што ја подобрува безбедноста и меѓусебната изолација меѓу различни модули, туку исто така и ги намалува и побарувањата за ресурси и овозможува поддршка за вистинско скалирање до 0 инстанци. Брзото поставување на извршната околина веднаш по добивање на барањето може да биде особено корисно при пресметување на работ од мрежата, затоа што овозможува надпровизионирање од аспект на бројот на функции кои реално би можеле да бидат извршувани со дадениот хардвер, под претпоставка дека тие не треба да се извршуваат паралелно.

5.2.3. Стратегија за валидација на решението

Ниту едно инфраструктурно решение не е целосно без негово практично тестирање и валидација на потенцијалните подобрувања кои би требало да ги овозможи. За оваа цел, оформена е специјализирана инфраструктура за евалуација на предложената интеграција на Kubernetes со WebAssembly извршна околина.

Безсерверски функции и безсерверски платформи

За споредба на перформансите на предложеното решение со оние на традиционална контејнеризирана безсерверска платформа, реискористена е истата основа од минатите поглавја [28], [73]. Табела 5.1 ја рекапитулира намената на секој од тестовите.

Табела 5.1

Информации за користената свита на тестови

Име	**Јазик**[27]	**Опис**	**Тип**[28]
fuzzysearch	Rust	Пребарување текст во онлајн датотека	P
n-body	Rust	Моделирање на движењето на планети	M
prime-numbers	Rust	Барање на прости броеви во првите n броеви	M
whatlang	Rust	Одредување на природен јазик за пишан стринг	P
aes	Go	Шифрирање текст со AES алгоритамот	P
float-operation	Go	Калкулација sin, cos, и квадратен корен од даден број	M
linear-equations	Go	Решавање системи линеарни равенки	M
matmul	Go	Множење квадратни матрици	M
user-manager	Rust	Комуникација со BaaS база на податоци	P

User-manager функцијата е експлицитно додадена во свитата со цел да се тестира можноста на Spin за извршување појдовни HTTP барања. Преку неа може да се изведе заклучок дали навистина е спроведлива непречена комуникација и со надворешни BaaS услуги при извршување на безсерверската функција во WASM околина. User-manager функцијата при секој повик испраќа HTTP POST барање до BaaS база на податоци, додавајќи нови записи.

OpenFaaS е избран како референтна имплементација на контејнерска безсерверска платформа, во чекор со применетата евалуациска стратегија во останатите поглавја. OpenFaaS тука се користи за да се добијат референтни вредности кои би служеле за проценка на перформансите во однос на ладниот старт и вкупното време на извршување карактеристични за контејнерско извршување.

Извршување тестови

Четири различни сценарија за извршување на тестовите се дефинирани за двете околини (WebAssembly и контејнери). Овие сценарија се тесно поврзани со оние дискутирани во поглавје 4:

- Сериско извршување – секоја функција е непрекинато повикувана во временска рамка од 5 минути, користејќи една процесорска нишка.
- Време за инстанцирање функција и време за прво целосно извршување – секоја функција е инстанцирана 100 пати, мерејќи го времето потребно за да стане спремна за опслужување барања. Потоа, функцијата е повикана, мерејќи го времето за прво извршување.
- Паралелно извршување – секоја функција е непрекинато повикувана 1 минута со 5 паралелни нишки, секоја ограничена на максимум 5 барања во секунда. Во најоптималното сценарио, ова би резултирало со 25 барања во секунда во рамките на 1 минута, т.е. 1500 барања за целиот период на извршување.
- Големина на OCI слики – споредба на големините на OCI сликите помеѓу WASM и контејнеризираните верзии на функциите.

Сите тестови се извршени во истата тестна околина користена и низ минатите поглавја. OCI сликите за сите 18 функции (9 за WebAssembly и 9 за OpenFaaS) се преземаат однапред на јазлите, со цел да се елиминира непредвидливо доцнење како резултат на оптоварување на трети системи кои не би биле под директна контрола.

[27] Програмски јазик во кој е напишана функцијата.

[28] M – Микро тест; P – Реално сценарио.

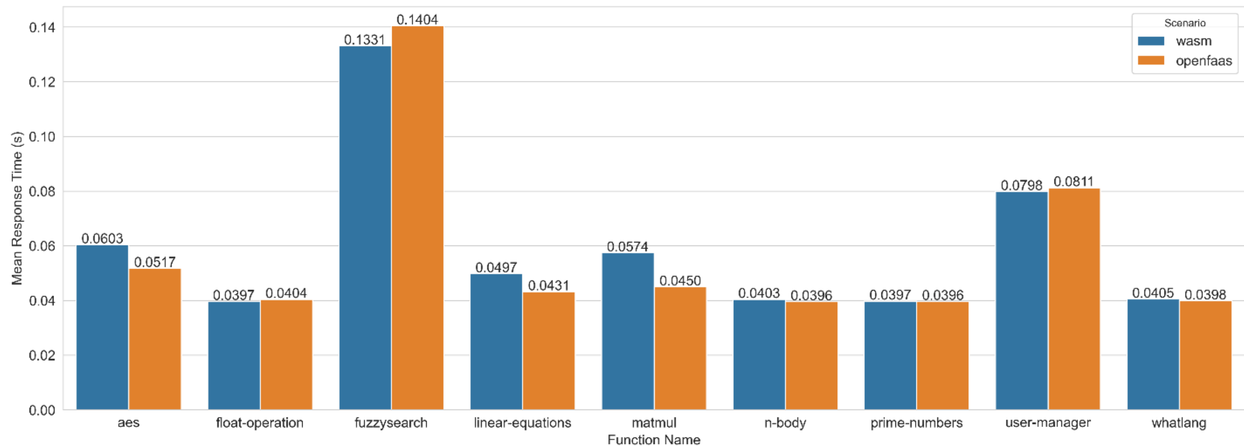
Од аспект на Kubernetes кластерот, потпирајќи се на резултатите од тестирањето различни Kubernetes дистрибуции, се користи K3s како најефикасно решение. Водејќи се од желбата да се ограничи колку што е можно повеќе сообраќајот кон надворешни дестинации, со цел да се елиминираат непредвидливи влијанија врз финалните резултати, неопходните BaaS услуги како бази на податоци и складирање податочни објекти се покренати во истата локална мрежа врз посебна Docker машина резервирана исклучиво за таа намена. Складирањето на објекти кое е неопходно за извршување на whatlang функцијата се симулира со Nginx веб сервер нагоден да сервира статички датотеки. REST програмски интерфејс за базата на податоци, неопходен за user-manager апликацијата е овозможен со користење на pRest проектот со отворен код [209].

5.3. Валидација на решението за оркестрација

Резултатите добиени со примена на претходно опишаната методологија за тестирање се предмет на разгледување во продолжение. Дискусијата започнува со разгледување на перформансите при сериско извршување, постигнати од страна на WebAssembly и контејнеризираните функции од свитата. По ова, фокусот преминува на времето потребно за инстанцирање на нова функција и анализа на перформансите кои Spin ги постигнува од аспект на ладниот старт, споредено со традиционалното решение. На крајот од оваа потпоглавје следи осврт на перформансите при паралелно извршување од страна на повеќе корисници на една иста функција, како и мерење на вкупната големина на OCI сликите за секоја од функциите низ различните стратегии на извршување.

5.3.1. Сериско извршување

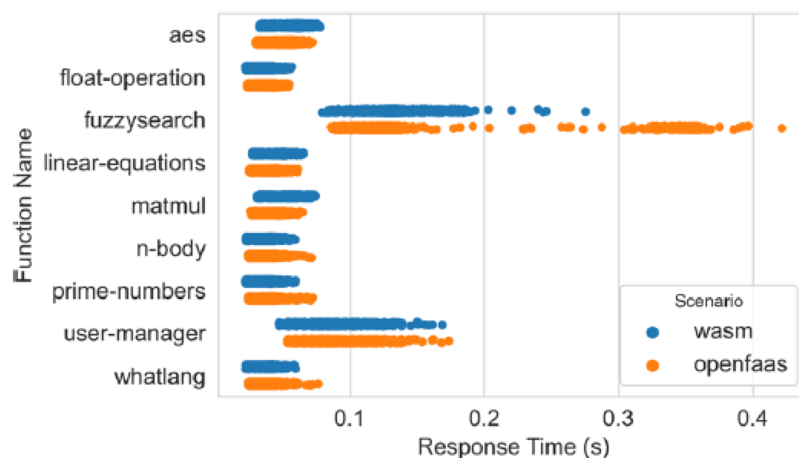
Перформансите при сериско извршување на функции од страна на двете решенија се евалуираат преку непрекинато извршување на секоја од функциите во времетраење од 5 минути, користејќи една нишка и ограничувајќи го бројот на паралелни барања на 1. Експлицитно е оневозможено користење постоечки TCP конекции, со цел да не се влијае врз веродостојноста на резултатите. Имајќи предвид како работи Spin WebAssembly извршната околина, ова сценарио и не е најоптималното за неа, па се очекува контејнеризираното извршување да покаже подобри резултати соодветно на подобрите пресметковни перформанси. Дополнително, при овој тест се елиминира главната слабост на контејнерите, големиот ладен старт, бидејќи контејнерот е веќе стартуван при испраќање на повикот за функцијата. Од друга страна, пак, Spin врши покренување на целата околина штом се добие барањето, без никакво претходно подгревање. Целта на ваквата стратегија на извршување е да се дообјаснат разликите во перформанси кои се јавуваат помеѓу извршување безсерверски функции во веќе стартуван контејнер и WebAssembly околина која е ад-хок покрената при добивање на барањето. Овој тест е одлична можност да се анализира дали побрзиот ладен старт кој веќе беше докажан за WebAssembly е анулиран од страна на поголемиот пресметковен потенцијал кој го покажуваат контејнерите.



Слика 5.2 Споредба на средното време на одговор (во секунди) кај WASM (Spin) и OpenFaaS

Врз основа на резултатите прикажани на слика 5.2, OpenFaaS нуди побрзо извршување во 6 од 9 теста. WebAssembly модулите се побрзи при извршување на: float-operation, fuzzysearch и user-manager тестовите. Особено е интересно што двете функции кои имаат потреба од комуникација со надворешни системи преку HTTP, fuzzysearch и user-manager, покажуваат подобри перформанси во WebAssembly средина. Во најголемиот дел од случаите, средните времиња на извршување не отстапуваат драстично едни од други. Исклучок од ова се aes и matmul тестовите каде разликата е најголема. OpenFaaS нуди 16,63% побрзо извршување во случајот со aes и 27,56% во случајот со matmul. Ваквото однесување е и очекувано, имајќи предвид дека станува збор за тестови кои бараат високи пресметковни перформанси.

Разгледувајќи ја слика 5.3, може да се забележи дека OpenFaaS покажува зголемена неконзистентност во резултатите, кои се и понепредвидливи. Продолжувајќи ја оваа анализа, во табела 5.2 се дадени стандардната девијација и коефициентот на варијација за тестовите.



Слика 5.3 Време на одговор (во секунди) при различни сериски извршувања

Табела 5.2

Споредба на стандардната девијација и коефициентот на варијација при сериско извршување

Име	СД[29] WASM	СД OF[30]	КВ[31] WASM	КВ OF
fuzzysearch	0,0137	0,0606	0,1029	0,4317
n-body	0,0044	0,0057	0,1093	0,1451
prime-numbers	0,0044	0,0058	0,1099	0,1463
whatlang	0,0044	0,0059	0,1083	0,1469
aes	0,0055	0,0068	0,0907	0,1312
float-operation	0,0045	0,0052	0,1124	0,1286
linear-equations	0,0047	0,0054	0,0935	0,1253
matmul	0,0051	0,0057	0,0885	0,1260
user-manager	0,0121	0,0131	0,1522	0,1608

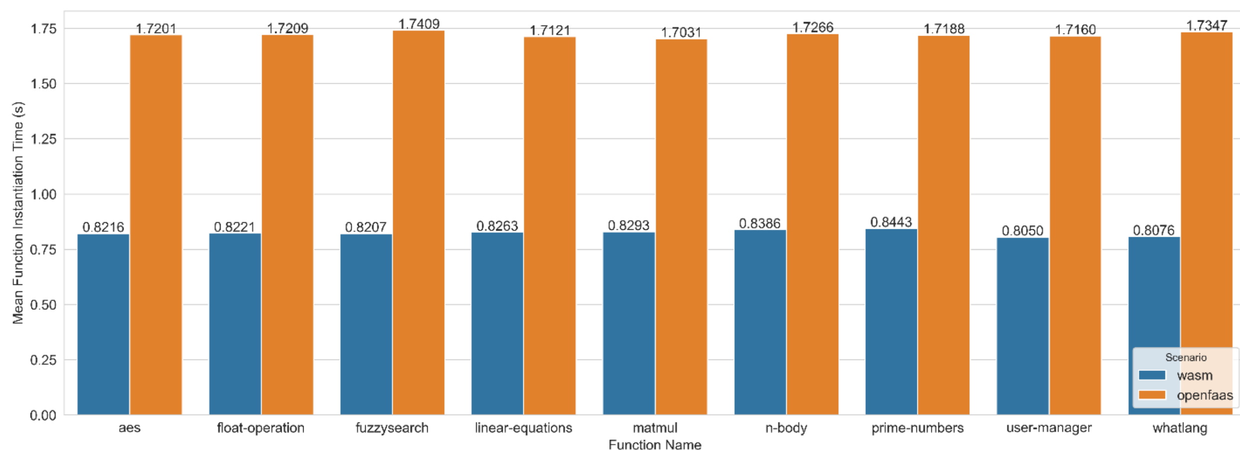
Низ сите 9 теста, WebAsembly се карактеризира со помали вредности за стандардната девијација и коефициентот на варијација споредено со контејнеризираниот пристап. Иако бројките зборуваат сами за себе, во одбрана на OpenFaaS, треба да се земе предвид дека портфолиото на технологии користени од негова страна е пообемно и покомплексно во споредба со WebAssembly [210]. Меѓу другото, неопходно е поставување на посредничка компонента низ која мора да поминат сите барања со цел нивно крајно упатување до соодветна инстанца од повиканата функција, како и агенти за мониторирање кои помагаат при скалирањето на функциите. Извршувањето на овие дополнителни компоненти воведува недетерминистички товар кој влијае врз времињата на извршување, објаснувајќи ја варијацијата. Од друга страна, пак, во случајот со WebAssembly, единствената неопходна компонента е само софтверската спојка за интеграција со Containerd.

5.3.2. Подигнување функции

Времето потребно за подигнување, т.е. инстанцирање нова функција е многу важна карактеристика кај безсерверското пресметување, особено при сценарија каде што е потребна агресивна миграција од еден пресметковен јазол на друг (пр. MEC) или, пак, нагло зголемување на бројот на активни инстанци како резултат на зголемен број дојдовни барања. Со цел анализа на времето за подигнување на функциите во двете разгледувани околини, секој тест е инстанциран 100 пати и мерено е потребното време за Kubernetes оркестраторот да го означи соодветниот Pod дека е спремен за опслужување барања. На слика 5.4 се дадени средните времиња за подигнување на деветте функции во двете околини.

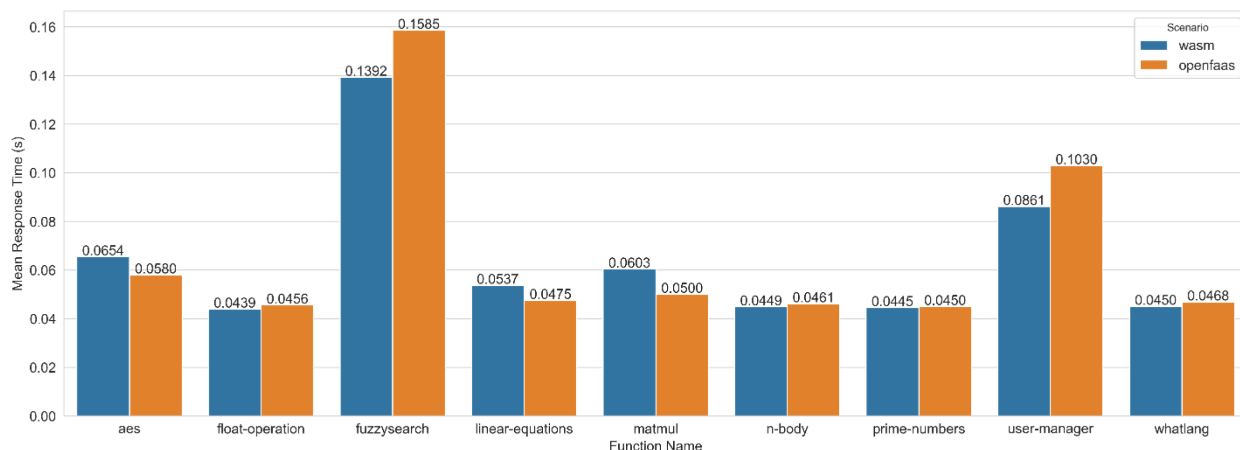
Брзото време на инстанцирање е една од главните предности на WebAssembly, што го потврдуваат и добиените резултати. WASM модулите во сите случаи имаат околу 2 пати побрзо време на инстанцирање споредено со контејнерите на OpenFaaS. Ваквите резултати се должат на тоа што WASM артефактите се помали споредено со класичните контејнерски слики, феномен дискутиран подетално во 5.3.4. Малата големина, заедно со помал број на вкупни слоеви при WASM модулите не само што заштедува на складишен простор, туку и овозможува побрзо покренување, бидејќи е намалено времето потребно за отпакување на архивата. За појаснување, средните времиња прикажани на слика 5.4 не го вклучуваат времето потребно за првото извршување.

[29] Стандардна девијација.
[30] OpenFaaS.
[31] Коефициент на варијација.



Слика 5.4 Споредба на средното време за подигнување функции помеѓу WASM (Spin) и OpenFaaS

Со цел да се анализира дали првото извршување по ладен старт нуди полоши перформанси споредено со серискoто извршување, секоја функција се извршува по нејзиното стартување и потоа повторно се брише. На слика 5.5 се дадени средните времиња на извршување измерени со директно извршување по комплетирање на процесот на инстанцирање.



Слика 5.5 Споредба на средното време на одговор при првото повикување на функциите по нивното подигнување

Средните времиња на одговор за извршување по ладен старт, до одредена мера, се поголеми споредено со оние при сериско извршување. Ова не е изненадувачки, а еден од факторите кои играат улога е и механизмот за кеширање вграден во оперативниот систем над кој се извршуваат безсерверските платформи. Временските разлики за двете функции кои имаат потреба од појдовни HTTP барања (fuzzysearch и user-manager) се уште поголеми помеѓу WASM и OpenFaaS. Дополнително, WASM покажува подобри резултати уште во два теста, n-body и prime-numbers, што не беше случај претходно. Двете функции постојано се извршуваат со истите влезни параметри, што потенцијално му овозможува на OpenFaaS да изврши кеширање на меѓурезултатите, имајќи предвид дека контејнерите не се терминираат по нивното извршување при сериските повици. Во сегашното сценарио, по секое извршување контејнерот се уништува, елиминирајќи ја оваа предност.

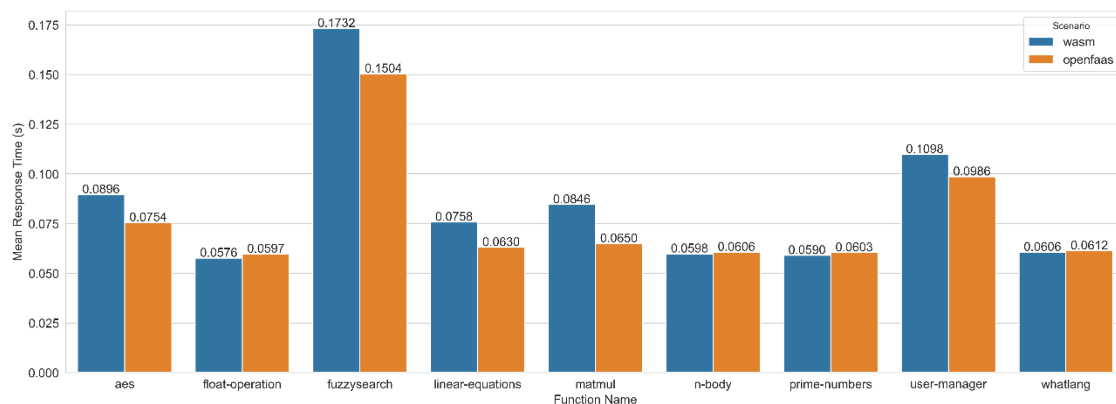
Може да се заклучи дека WASM нуди два пати побрзо покренување на функции во сите случаи, а и нуди соодветни перформанси при извршувањето, кои не отстапуваат од она

што го постигнуваат традиционалните контејнеризирани околини, барем кога станува збор за поедноставни задачи без потреба од комплексни пресметки.

5.3.3. Паралелно извршување

За тестирање на перформансите на двете околини при постојано оптоварување како резултат на паралелно извршување, секоја функција се извршува во времетраење од 1 минута, користејќи 5 нишки, секоја ограничена на максимум 5 барања во секунда. Во најоптималниот случај, кога честоста на дојдовни повици е помала од времето потребно за опслужување на еден повик (па нема редица на чекање), ваквата стратегија би резултирала со вкупно 1500 барања во рок од 60 секунди. Сите механизми за автоматско скалирање се експлицитно оневозможени за OpenFaaS, додека, пак, за Spin се користи предодреденото однесување секој повик да се извршува во посебна, изолирана, извршна средина. Во случајот со OpenFaaS, сите барања се опслужуваат од еден контејнер со вграден HTTP сервер, додека, пак, Spin покренува вкупно 1500 инстанци при тестирањето.

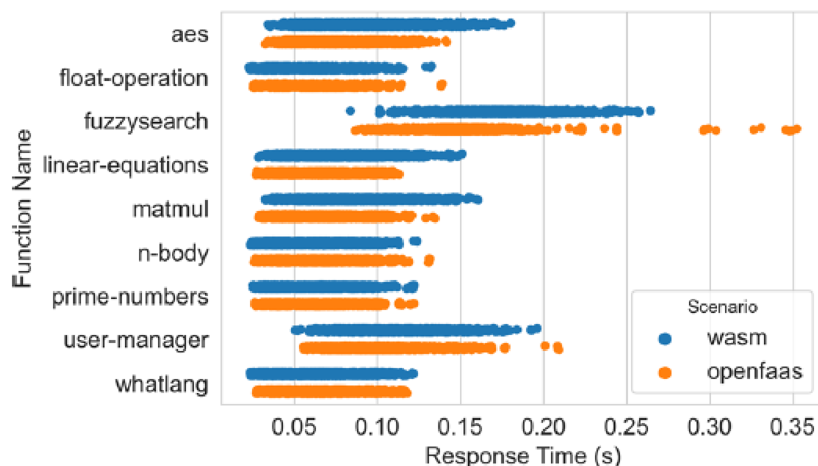
Слика 5.6 ги прикажува резултатите добиени со паралелно извршување на деветте функции.



Слика 5.6 Средно време на одговор при паралелно извршување на функциите

Може да се забележи дека WASM покажува помало средно време на одговор за 4 функции: float-operation, n-body, prime-numbers и whatlang. За двете функции кои испраќаат HTTP барања, fuzzysearch и user-manager, OpenFaaS покажува подобри резултати, што не е случај при нивното сериско извршување. Едно објаснување за ваквото однесување е тоа што програмската HTTP библиотека може да има напредна стратегија за управување со мрежните врски (англ. connection polling) [211], што би резултирало со подобри перформанси кога еден контејнер врши поголем број на барања до одредена дестинација во краток временски период.

На слика 5.7 се прикажани времињата за одговор постигнати од сите извршувања, со цел да се изврши проценка на степенот на конзистентност кој би можел да се очекува.



Слика 5.7 Време на одговор (во секунди) при различни паралелни извршувања

Слично како и кај сериското извршување, fuzzysearch покажува понепредвидливи резултати при извршување кај OpenFaaS наспрема WASM. Фокусирајќи се исклучиво на стандардната девијација и коефициентот на варијација низ сите паралелно извршени функции, може да се забележи дека вредностите за двете околини се блиски помеѓу себе, како што е видливо во табела 5.3.

Табела 5.3

Споредба на стандардната девијација и коефициентот на варијација при паралелно извршување

Име	**СД[32] WASM**	**СД OF[33]**	**КВ[34] WASM**	**КВ OF**
fuzzysearch	0,0231	0,0227	0,1336	0,1506
n-body	0,0200	0,0199	0,3347	0,3277
prime-numbers	0,0193	0,0187	0,3279	0,3102
whatlang	0,0203	0,0193	0,3353	0,3159
aes	0,0253	0,0205	0,2828	0,2711
float-operation	0,0201	0,0200	0,3481	0,3356
linear-equations	0,0221	0,0195	0,2923	0,3097
matmul	0,0235	0,0195	0,2777	0,3004
user-manager	0,0233	0,0230	0,2120	0,2334

Анализата на резултатите покажува дека OpenFaaS нуди подобри перформанси при паралелно извршување на функции, под претпоставка дека извршната околина е веќе покрената, т.е. контејнерот е подгреан. Од друга страна, пак, WASM нуди скалирање до 0 во вистинска смисла на изразот, споредливи перформанси при паралелно извршување на задачи кои не се пресметковно комплексни и високо ниво на изолација, а со тоа и безбедност.

5.3.4. Големина на софтверски артефакти

Големината на софтверските артефакти кои ги претставуваат функциите се важен фактор кога станува збор за безсерверски платформи и нивните извршни околни. Земајќи предвид дека безсерверските платформи, барем оние во продукциски средини, може да се протегаат низ стотици пресметковни јазли, како и дека дадена функција би се нашла

[32] Стандардна девијација.
[33] OpenFaaS.
[34] Коефициент на варијација.

на различни јазли во својот животен век, големи артефакти може да имаат значително влијание врз оптоварувањето. Нивното пренесување може да воведе дополнителен товар врз мрежата, складиштето, па дури и пресметковните перформанси во случај кога станува збор за архивски формат кој би требало да се отпакува, како OCI сликите. Заедничко за OpenFaaS и претставената интеграција на Spin со Kubernetes е тоа што двете решенија го поддржуваат OCI форматот на слики, овозможувајќи прецизна и релевантна споредба за тоа како варираат големините при употреба на различна средина.

Од архитектурна гледна точка, Spin побарува присуство исклучиво на WASM бинарната датотека и текстуалниот манифест за инструкции како да се подигне сликата, додека, пак, OpenFaaS шаблоните вклучуваат во себе и дополнителен софтвер. Најголемиот дел од официјално препорачаните OpenFaaS функциски шаблони се темелат на оптимизираните alpine или debian-slim контејнерски слики. Во табела 5.4 се дадени разликите во големина на сликите за сите 9 функции помеѓу WASM и OpenFaaS.

Табела 5.4

Споредба на големината на OCI сликите за WASM и OpenFaaS (во килобајти)

Функција	**WASM големина (KB)**	**OF[35] големина (KB)**	**Зголемување[36]**
fuzzysearch	712	52620	x73,90
n-body	562	50970	x90,69
Prime-numbers	675	50950	x75,48
whatlang	753	51040	x67,78
aes	132	8780	x66,52
float-operation	122	8790	x72,05
linear-equations	161	8920	x55,40
matmul	134	8850	x66,04
user-manager	733	52500	x71,62

Значајна разлика е забележлива во случајот со Rust базирани функции, бидејќи и по бројните оптимизации направени врз официјалниот шаблон и воведувањето повеќефазно градење на сликите, сè уште е неопходно користењето на „debian:bullseye-slim“ како основна слика. И покрај сите подобрувања, OpenFaaS артефактите се за цела величина поголеми од еквивалентните WASM слики. Ваквата ситуација директно влијае со поголеми трошоци за нивно складирање, зголемена оптовареност на мрежните врски, зголемени надоместоци за интернет сообраќај, побавен ладен старт и зголемено оптоварување на процесорот, придружено со поголема потрошувачка на електрична енергија при отпакување на архивските слики.

5.4. Придобивки од примена на решението

Безсерверско повеќејазлено оркестрациско решение кое може да биде поставено и на работ од мрежата и во облакот е полезно во голем број домени особено кога може да понуди исклучително брзо стартување на нови функции. Без сомнеж, едно такво сценарио е и повеќе-пристапното пресметување на работ од мрежата.

Иако првичниот фокус на MEC иницијативата беше виртуелизацијата и користењето виртуелни машини како извршни околини за MEC апликациите, неодамна соодветната група за спецификации во индустријата (англ. industry specification group, ISG) објави документ кој го отвора патот за воведување и алтернативи [212]. Примерите вклучуваат користење уникернели [5] или Kata контејнери како поефикасни методи од

[35] OpenFaaS.

[36] Зголемување на OpenFaaS OCI сликата споредено со WASM модулот.

традиционалните виртуелни машини [213], [214], [215]. Во актуелната литература се споменува и користењето на безсерверската парадигма во рамките на MEC [22], а одредени практични имплементации се и веќе опишани [78].

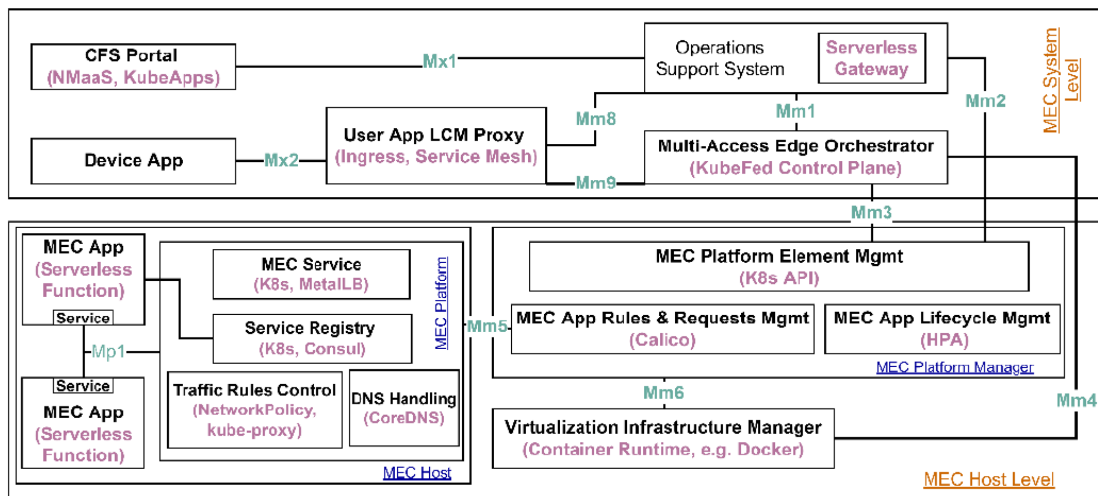
Имајќи ја предвид моменталната поставеност на MEC екосистемот, лесно е да се увиди придобивката од воведувањето на WebAssembly како извршна околина која би овозможила екстремно брз старт за функции кои не побаруваат големи пресметковни перформанси, како на пример оние за справување со одредени настани генерирани од паметни уреди.

Во продолжение, опишано е сценарио кое се темели на официјално предложената MEC архитектура од страна на ETSI, а чија цел е користење на Kubernetes оркестраторот за извршување MEC безсерверски апликации како во контејнери, така и како WebAssembly модули.

5.4.1. WebAssembly како дел од MEC

Постојната MEC архитектура опишана од ETSI ISG е изградена како хиерархиски систем, сочинет од две нивоа – системско ниво и ниво на домаќин (во продолжение домаќинско ниво). За комплетност, вреди да се напомене дека во референтната рамка се спомнува и трето ниво, мрежното, но тоа не е дел од архитектурата.

За интеграцијата со Kubernetes и вклучувањето различни извршни околини за MEC апликации, од особено значење е домаќинското ниво.



Слика 5.8 Предлог мапирање на софтверски алатки во различни компоненти од MEC архитектурата

На слика 5.8 е прикажано комплетното мапирање помеѓу елементите опишани во MEC референтната архитектура и конкретни системи кои би овозможиле безсерверско извршување со помош на различни извршни околини. Во продолжение се опишани компонентите од домаќинското ниво. Елементите релевантни за системското ниво, вклучувајќи можности за напредно вмрежување, оркестрација низ околини и кориснички портали се предмет на следните потпоглавја.

Домаќинското ниво е претставено од Kubernetes кластери покренати на работ од мрежата, на инфраструктура овозможена од мобилните оператори. Секој Kubernetes кластер е сочинет од посебна контролна рамнина под надлежност на Kubernetes мастер

јазли, кои, пак, управуваат со Kubernetes јазлите задолжени за извршување на апликациите.

5.4.2. MEC управувач со платформи

MEC управувачот со платформи (англ. MEC Platform Manager) е претставен од контролната рамнина на различните Kubernetes кластери. Контролната рамнина ја управуваат мастер јазли кои служат за координација на стандардните Kubernetes јазли (англ. worker nodes) кои всушност извршуваат конкретни задачи и апликации. Водејќи се од потенцијално ограничените пресметковни ресурси на работ од мрежата, се препорачува користење оптимизирана Kubernetes дистрибуција, како на пример K3s, која е и основата на оркестрациското решение дискутирано во рамките на ова поглавје.

5.4.3. MEC управувач со инфраструктура за виртуелизација

MEC управувачот со инфраструктура за виртуелизација (англ. MEC Virtualization Infrastructure Manager, VIM) е претставен од извршната околина интегрирана со Kubernetes кластерот на работ од мрежата. Во опишаното сценарио овој управувач всушност се темели на Containerd кој може истовремено да извршува контејнери и WASM модули, благодарение на развиената Spin софтверска спојка. Мрежен пристап во рамките на истиот кластер или до оддалечени дестинации е возможен или преку соодветната контејнерска надмрежа или со директно користење на мрежата на домаќинот (англ. host network) кој ја извршува безсерверската функција.

5.4.4. MEC платформа

MEC платформата (англ. MEC Platform) е претставена од стандардните Kubernetes јазли кои ги извршуваат доделените задачи и безсерверски функции распределени од страна на MEC управувачот со платформи (Kubernetes контролната рамнина). Со користење на основните Kubernetes функционалности проширени со неколку дополнителни додатоци се овозможува имплементација на потребните правила за мрежен сообраќај, опслужување DNS барања, како и регистрирање нови и откривање постоечки мрежни услуги.

Препорачаниот DNS разрешувач за Kubernetes околини, CoreDNS, може да се користи не само за интерно разрешување во рамките на кластерот, туку и како рекурзивен разрешувач за корисниците на мобилната мрежа. DNS зоните и останатите нагодувања во ова сценарио се под надлежност на MEC управувачот со платформи. Ваквиот пристап овозможува распределба на барањата испратени од страна на корисниците низ повеќе инстанци од истата функција, како и рутирање до најблиската инстанца во случај на подвижен корисник, исклучиво преку едноставна ротација на адресите вратени како одговор на соодветните DNS прашања.

5.4.5. MEC апликации

Благодарение на проширената функционалност на управувачот со инфраструктура за виртуелизација, MEC апликациите во предложеното решение може да бидат извршувани во две извршни околини, како контејнери или како WASM модули. Имајќи ја предвид проширливоста на Containerd и можноста за развивање дополнителни софтверски спојки, во никој случај не е исклучена можноста за воведување и дополнителни можности, како на пример микро виртуелни машини или стандардни виртуелни машини [216], [217].

5.5. Значењето на WASM оркестрацијата за облак-раб екосистемот

Ова поглавје претстави и валидира оркестрациско решение кое покрај поддршка за традиционалните контејнери како извршна околина за безсерверски функции, овозможува и интеграција со алтернативна средина со уникатни придобивки, WebAssembly. Резултатот е унифицирана платформа која има поддршка за различни извршни околини и може да извршува безсерверски функции како на работ, така и во облакот. Со ова се потврдува и вториот дел од поставената хипотеза, а со тоа и нејзино целосно потврдување, т.е. *„Виртуелните машини и контејнерите како извршни околини може да се комбинираат заедно со нови и поефикасни алтернативи подобро прилагодени за работ на мрежата со цел креирање една унифицирана платформа која ќе понуди флексибилност преку избор на најсоодветната технологија за извршувањето на дадена задача.“* Претставеното решение може да се користи како во нови, така и во постоечки средини, вклучително и во MEC контекст, со што се потврдува неговата флексибилност. Ваквиот пристап доведе до одговори за најгорливите отворени прашања во постоечката литература денес од областа на WebAssembly и безсерверската парадигма – оркестрацијата на функциите.

Сепак, имајќи ја предвид крајната цел, а тоа е воспоставување на транспарентен безсерверски облак-раб екосистем, истражувањето оттука па натаму треба да води кон поврзување повеќе различни вакви оркестрациски средини, без разлика на инфраструктурата во која се поставени. Следните поглавја на дисертацијата ќе обрнат внимание на мрежните предизвици кои е потребно да се надминат при воспоставување на облак-раб екосистемот, како и при дизајнирањето слоевита архитектура која би вклучувала унифицирана контролна рамнина за различните унифицирани средини.

6. ТРАНСПАРЕНТНО МРЕЖНО ПОВРЗУВАЊЕ НИЗ ОБЛАК-РАБ ЕКОСИСТЕМОТ

Со воведувањето на работ од мрежата како нова пресметковна локација и со раѓањето на идејата за облак-раб екосистемите, се поставува прашањето дали ваквите виртуелни приватни мрежи може да се користат како соодветен начин за взаемно, транспарентно и безбедно поврзување меѓу сите инфраструктури во рамките на даден екосистем. Еден чест и широко прифатен начин за задоволување на ова барање е користењето виртуелни приватни мрежи (англ. Virtual Private Network, VPN), технологија со чија помош може да се воспостават доверливи шифрирани мрежни тунели низ недоверлива мрежна инфраструктура. Овие виртуелни приватни мрежи може да се користат како од страна на крајни корисници, со цел поврзување со оддалечена инфраструктура (клиентски пристап, англ. client access VPN), така и за меѓусебно поврзување на две или повеќе оддалечени филијали (локација-локација поврзување, англ. site-to-site VPN).

При избор на VPN решение за транспарентно поврзување меѓу инфраструктурите кои сочинуваат еден облак-раб екосистем, потребно е да се земат предвид голем број аспекти, како на пример: поддржаните алгоритми за шифрирање заедно со дополнителни безбедносни функционалности, леснотијата на користење, изборот на транспортен протокол (TCP или UDP), поддржани компјутерски архитектури каде може да биде покренато решението и, се разбира, поддржаните мрежни топологии кои може да се оформат со учесниците во виртуелната приватна мрежа. Најголемиот дел од VPN решенијата денес имаат поддршка за најразлични платформи, вклучително и сервери, лични компјутери, мобилни телефони, па и во некои IoT уреди. Од аспект на транспортните протоколи, UDP е најчестиот избор кога е потребна оптимизација во однос на перформансите, поради малата брзина која се постигнува при тунелирање на TCP сообраќај генериран од погорен слој низ TCP VPN тунел [218]. Исклучок од ова се мрежи каде што има рестриктивни политики за мрежна комуникација и строго филтрирање на влезните и излезните мрежни порти дозволени за користење. Во такви ситуации, најчесто само мал дел од често користените TCP порти се дозволени од страна на корпоративниот заштитен ѕид, правејќи го користењето на VPN решение засновано на UDP невозможно. Кога станува збор за севкупните перформансни карактеристики, не треба да се занемари и фактот дека улога игра и самата архитектура на софтверското решение. Оние VPN софтвери кои поддржуваат паралелизам на ниво на оперативниот систем покажуваат подобри резултати од оние кои работат исклучиво со една нишка [219], [220]. Со цел овозможување подобри перформанси, одредени VPN решенија дури имаат и сопствени јадрени модули [221] (англ. kernel modules). Други, пак, ги заменуваат подобрите перформанси кои може да се остварат преку ваквата интеграција со јадрото на ниско ниво, за подобро корисничко искуство, имплементирајќи ги соодветните функционалности во корисничкиот домен од оперативниот систем [222].

Најпопуларните VPN протоколи во моментов се OpenVPN [223] и Wireguard [221]. Во двата случаја станува збор за софтвер со отворен код, достапен под слободна лиценца. Како резултат на ова, дури и голем број од комерцијалните VPN решенија, всушност, во позадина, се темелат или на OpenVPN или на Wireguard (во некои ситуации и на двата). И OpenVPN и Wireguard имаат поддршка за огромен број на оперативни системи и компјутерски архитектури, а може самостојно и бесплатно да се користат и врз сопствена, приватна инфраструктура. Сепак, рачното нагодување не е скалабилно при поставување голем број на уреди на различни локации со цел креирање облак-рак екосистем, па се јавува потребата од алатки за автоматизација или дополнителен помошен софтвер. Во случајот со OpenVPN, автентикацијата на уредите може да се направи со лозинки или сертификати, но за имплементација на вториот пристап

потребно е воспоставување целосна инфраструктура за управување со приватни клучеви (англ. private key infrastructure, PKI) [224]. Првичната конфигурација на Wireguard е поедноставна и се темели на генерирање парови од приватни и јавни клучеви и размена на јавните клучеви меѓу уредите кои треба да воспостават взаемна врска, процедура слична како онаа при користење најава со клучеви кај SSH протоколот [225].

Благодарение на поедноставената размена на клучеви кај Wireguard, неодамна се појавија поголем број на како-услуга решенија, кои целат кон поедноставување на вмрежувањето на голем број уреди и на неопходната меѓусебна размена на клучеви, дискутирани во продолжение. Ваквите развитоци одат директно во прилог на воспоставување облак-раб екосистем составен од разновидни уреди, дистрибуирани низ различни локации. Стратегијата на работа најчесто се заснова на замена на децентрализираниот пристап со централизирана алтернатива и воведување координациски сервер кој ја игра улогата на посредник во размената на клучевите меѓу учесниците. Секако, доверливоста на податоците е загарантирана. Централниот сервер во никој случај не може да изврши дешифрирање на сообраќајот, бидејќи приватните клучеви никогаш не ги напуштаат јазлите каде што тие биле генерирани. Односно, серверот за координација има пристап исклучиво до јавните клучеви. Користејќи ја оваа стратегија, кој било јазол може да биде поврзан со произволен број на останати јазли брзо, едноставно и безбедно. Додавањето нов јазол откако е формирана мрежата исто така не побарува никаква дополнителна рачна активност. Ваквата архитектура овозможува напуштање на традиционалната ѕвезда топологија често асоцирана со VPN мрежите, каде што целиот сообраќај минува низ VPN серверот, кој претставува тесно грло. Наместо тоа, со помош на координациските сервери сега е возможно дефинирање нова топологија, така што секој јазол е поврзан со сите останати јазли и се оформува сестрано поврзување (англ. full-mesh). Со сестраното поврзување, се елиминираат централизираните тесни грла, се зголемува безбедноста преку отстранувањето непотребни посредници и се воведува робусност и можност за користење алтернативни патеки до одредена дестинација.

Подобрувањата дискутирани досега одат директно во прилог на воспоставувањето облак-раб екосистем, овозможувајќи ѝ на инфраструктурата на работ од мрежата безбедно и едноставно да се вмрежи како меѓу себе, со останати елементи на работ, така и со податочните центри во облакот. Топологијата на сестрано поврзување гарантира минимално доцнење и елиминирање непотребни скокови. Модерните техники за заобиколување на NAT (англ. Network Address Translation), вклучително и целосно препраќање на сообраќајот низ алтернативен јазол (во екстремни случаи), осигуруваат VPN поврзување дури и во најрестриктивните мрежи.

Црпејќи инспирација од новите можности за подобрување на облак-раб пресметувањето овозможени од воведувањето дополнителни извршни околини, како и од неодамнешните напредоци на полето на виртуелните приватни мрежи, целта на ова поглавје е да дефинира начини за транспарентно мрежно поврзување низ облак-раб екосистемот. Ваквото поврзување би било основа за дефинирање безсерверски платформи кои, пак, би овозможиле транспарентни пресметки низ поврзаните локации во рамките на целиот облак-раб екосистем. Посебен осврт се врши на начините на кои VPN, со топологија на сестрано поврзување, може да се користат како подмрежи за Kubernetes надмрежите од страна на додатоците кои го дефинираат контејнерскиот мрежен интерфејс [152]. На овој начин се постигнува транспарентно заемно поврзување на Kubernetes јазли без разлика на физичката мрежа во која тие се наоѓаат или географската локација.

6.1. Преглед на постојни технологии за виртуелни приватни мрежи и нивното место во облак-раб екосистемот

Протоколите за тунелирање мрежен сообраќај, а со тоа и начините за овозможување безбедна врска меѓу оддалечени јазли се важна инфраструктура тема која е од голем интерес за имплементација на ефикасни и безбедни облак-раб екосистеми. Робусни и брзи VPN решенија кои нудат едноставно поврзување и целосно управување со животниот циклус на јазлите (додавање, ажурирање, бришење) имаат огромен потенцијал да влијаат врз идниот развиток на облак-раб екосистемите, овозможувајќи податоците прво да бидат првично обработени на работ, пред да бидат испратени преку безбедна врска до облакот за долгорочно складирање и дополнителна анализа [44]. Во литературата постојат голем број трудови кои дефинираат сосема нови VPN протоколи или вршат споредба на постоечките. Еден таков пример е [226], каде што се споредуваат перформансите на OpenVPN со оние на IPSec, извлекувајќи го заклучокот дека IPSec има предност во најголемиот дел од случаите. Иако OpenVPN е сеприсутен во литературата, како резултат на неговата голема популарност и докажаност во пракса, понови трудови исто така се фокусираат и на Wireguard. Декер и др. во [227] ги опишуваат разликите меѓу три VPN протоколи: StrongSwan, OpenVPN и Wireguard при брзини од 1Gbit/s. Нивното истражување покажува дека StrongSwan и OpenVPN имаат најдобри резултати при тунелирањето UDP сообраќај (UDP во UDP), додека, пак, Wireguard јадрената имплементација е победникот во однос на времето потребно за воспоставување врска. Wireguard имплементацијата која работи во корисничкиот домен на оперативниот систем има покажано најголемо оптоварување на процесорот низ сите тестови. И резултатите презентирани од Геталс и др. во [228] ги потврдуваат придобивките од користењето Wireguard, тврдејќи дека тоа е најбрзото решение споредено со останатите кои биле предмет на разгледување (OpenVPN, Wireguard, ZeroTier, Tinc, SoftEther). Истите автори опишуваат и тестови кои имаат за задача да ја покажат скалабилноста на секое од решенијата, воведувајќи различен број на истовремени клиенти и можност за справување со испади.

Инспирирани од добрите перформанси на Wireguard од една страна, но и фрустрирани од макотрпната рачна конфигурација која подразбира размена на клучеви помеѓу јазли, од друга страна, Паилис и др. опишуваат централизирана контролна рамнина за Wireguard која може да понуди автоматизирана размена на клучеви и воспоставување сестрано поврзување меѓу јазлите [229].

Можноста за избегнување на ѕвезда топологијата и нејзина замена со сестрано поврзување е од особена важност кога станува збор за географски дистрибуирани инфраструктури. Авторите на [230] вршат тестирање на перформансите на Wireguard во сценарио кога над него е покренат Kubernetes кластер, а мерењата се темелат на вкупното време на одговор кое го нуди веб апликација инстанцирана во самиот кластер. Иако претставеното сценарио е имплементирано и валидирано, сепак, изостанува дискусија за алтернативни VPN решенија како и дополнителни мерења (пр. време на одговор на Kubernetes API серверот), покрај оние врз инстанцираната апликација. Слично на ова и Ванг и др. се фокусираат на воспоставување Kubernetes кластер над Wireguard мрежа, но нивниот главен интерес не е самото VPN поврзување, туку генералната архитектура на една таква платформа [231]. Во [232] опишано е како Kubernetes кластер може да се прошири сѐ до работ на мрежата, преку вклучување уреди со висока енергетска ефикасност и нивно вмрежување со Netmaker, VPN решение базирано на Wireguard кое нуди централизирана контролна рамнина за лесна размена на клучеви. Фалконе и др. во [233] го проучуваат Wireguard од аспект на алатка која може да помогне во

воспоставувањето федерации од Kubernetes кластери, преку користењето на т.н. виртуелни kubelets [234], кои би служеле како интерфејс за управување со оддалечените кластери.

Постојат и случаи каде што се претставени целосно нови VPN протоколи, како што е случајот со EdgeVPN [235]. Релевантно за него е тоа што нуди поддршка за заобиколување на NAT со користење на стандардните STUN и TURN протоколи, карактеристични за сферата на мултимедијата. За жал, авторите не опишуваат дали во позадина TURN серверот поддржува препраќање исклучиво преку UDP или, пак, и преку TCP. Валидација на EdgeVPN е направена со негово користење како подмрежа за Kubernetes кластер каде што е покренат Flannel додатокот за мрежно поврзување.

Ваквите VPN решенија може да се сфатат како почетна точка преку која ќе се реализира идејата за транспарентно мрежно поврзување низ облак-раб екосистемот. Поддршката за надминување на NAT пречките [236] би овозможила и поставување на јазлите на работ во рестриктивни мрежи каде што во спротивно комуникацијата со остатокот од работ или облакот не би била можна. Со гаранцијата за високо ниво на интегритет и доверливост на информациите кои патуваат низ шифрираните тунели, може да се имплементираат и техники за динамичка миграција на задачи кои се во процес на извршување од работ кон облакот и обратно, со помош на некои од новите техники за миграција на контејнери [237], потпирајќи се на вмрежувањето овозможено од VPN.

Денес постои опсежна литература која нуди споредба меѓу перформансите на најразлични VPN решенија, земајќи предвид и посебни топологии на поврзување. Сепак, нема сеопфатна анализа од аспект на VPN решенија за сестрано поврзување и како тие може да бидат користени во функција на подмрежа за дистрибуирани Kubernetes кластери во рамките на облак-раб екосистемите. Во продолжение, фокусот е токму на овие аспекти, со посебен осврт на техниките за надминување NAT пречки и можноста за поставување Kubernetes јазли дури и во мрежи со високо ниво на изолација, овозможувајќи разноликост на инфраструктурите дел од еден таков екосистем.

6.2. Анализа на VPN решенија за воспоставување екосистеми меѓу облакот и работ

Имајќи предвид дека постојат голем број на потенцијални VPN решенија кои може да се користат за воспоставување екосистем меѓу облакот и работ од мрежата, потребно е дефинирање робусна, добро структурирана методологија за евалуација на кандидатите. Потпирајќи се на резултатите од претходните поглавја и фактот дека Kubernetes како оркестрациско решение е денес широко прифатено во рамките на безсерверската парадигма, дефинираната методологија треба да обрне посебно внимание на овој аспект, со цел осигурување компатибилност. Мрежната комуникација во рамките на Kubernetes кластерите е покомплицирана од едноставно поврзување меѓу класични пресметковни јазли и во продолжение се дискутираат специјалните побарувања кои треба да бидат земени предвид.

6.2.1. Критериуми за избор на решенија

Критериумите за избор на VPN решенија кои би биле подложни на понатамошна анализа се сочинети од два аспекта:

- неопходни функционалности кои се од суштинско значење;
- дополнителни, помошни, функционалности кои иако не се задолжителни, сепак придонесуваат за полесно работење и подобро корисничко искуство.

Неопходните функционалности кои мора да бидат задоволени од кое било VPN решение со цел негово вклучување во понатамошното истражување се:

- станува збор за софтвер со отворен код, со слободна лиценца, без никакви затворени додатоци. Постои можност за поставување на сопствена, приватна инфраструктура, без комуникација со трета услуга во облакот која не е под контрола на корисникот. Со овој критериум се елиминира потенцијалот за заклучување во портфолиото на услуги понудени од единствен давател на услуги.
- подлежи на активен развој, доказ за што е континуираното публикување нови верзии. На овој начин се осигурува навремено отстранување на потенцијални багови, како и ажурност во однос на имплементирање безбедносни закрпи.
- мора да поддржува топологија на сестрано поврзување, каде што секој јазол во мрежата се поврзува со сите останати, елиминирајќи тесни грла и гарантирајќи ниско доцнење при меѓусебната комуникација.
- можност за автоматска регистрација на нови јазли, без рачна интервенција, со што се осигурува еластичноста како на самата инфраструктура, така и на Kubernetes јазлите кои би можеле да бидат додавани или одземани по потреба.
- поддршка за листи за контрола на пристап (англ. access control lists, ACL) со цел ограничување кои јазли може да имаат непречена комуникација меѓу себе. Со оваа функционалност станува возможно истото VPN решение да се користи за поврзување повеќе, независни, глобално дистрибуирани Kubernetes кластери, со јасно дефинирана сегрегација меѓу нив. Со помош на ACL се прави логичка поделба на мрежата на повеќе паралелни сестрано поврзани подмрежи без допирна точка (од тополошки аспект) меѓу себе.
- можност за инсталација на различен хардвер и поддршка за повеќе процесорски архитектури, вклучително и x86 и ARM. Поддршката за ARM е денес веќе неопходна, поради сè поголемата присутност на ваквите уреди на работ од мрежата, а и нивната помала енергетска потрошувачка.
- напредни техники за надминување на пречките поставени во рестриктивни мрежи, како заобиколување NAT или целосно маскирање на VPN сообраќајот и препраќање низ посреднички јазли преку UDP и TCP. Препраќањето на сообраќајот треба да биде имплементирано на начин на кој посредникот не би имал директен пристап до нешифриран сообраќај.

Множеството функционалности кои, пак, би биле полезни, но не и неопходни се:

- напредна, грануларна поддршка за ACL. Иако поддршката за сè или ништо е неопходна функционалност, полезна би била и можноста за дефинирање поспецифични правила за комуникација помеѓу јазлите, како на пример дозволениот правец на комуникација, овластените протоколи од погорните слоеви и експлицитно наведување на отворените порти.
- можност за дефинирање подмрежни упатувачи (англ. subnet routers) кои би се користеле како упатувачки уред од страна на останати уреди во локалната мрежа со цел добивање пристап до VPN. На овој начин се овозможува учество во изолираната мрежа дури и на уреди кои самите не би можеле директно да станат дел од сестраното поврзување, како резултат на егзотичен оперативен систем или ниски перформанси. Негативниот аспект е тоа што уредот во улога на подмрежен упатувач ќе има пристап до целиот сообраќај кој е разменуван, без шифрирање.

Врз основа на овие барања, во табела 6.1 е дадена класификација на функционалностите на најпопуларните VPN решенија денес. Изборот за тоа кој VPN софтвер да биде

разгледуван е направен со помош на веб пребарување и анализа на мета GitHub репозиториуми кои агрегираат моментално достапни решенија [238], [239]. Исполнувањето на зададените критериуми е проценето преку преглед на официјалните документации, читање на изворниот код, како и рачно тестирање на решенијата во пракса.

Од вкупно 11 решенија, 4 ги задоволуваат сите задолжителни критериуми: Headscale, Netbird, Zerotier и Netmaker. Innernet и Nebula не задоволуваат само едно, а тоа е тунелирање на VPN сообраќајот низ TCP, во случаи кога не може да се оствари UDP поврзување. Ваквото TCP тунелирање, иако би довело до полоши перформанси, сепак, е значајно за рестриктивни мрежи каде што барем некакво поврзување е подобро од никакво. Сите евалуирани решенија ги поддржуваат и x86 и ARM архитектурите. Во оваа насока, вреди да се истакне и дека Headscale и популарниот Tailscale всушност ги користат истите клиентски апликации, со таа разлика што Headscale е само реимплементација со отворен код на Tailscale контролната рамнина, која самата по себе е затворена.

Разгледувајќи ги незадолжителните функционалности, сите 4 решенија кои ја поминуваат првичната селекција поддржуваат дефинирање подмрежни упатувачи. Headscale поддржува и грануларни ACL напишани со синтакса налик онаа на JSON и со помош на контролната рамнина дистрибуирани до сите учесници [240], [241]. ACL правилата претставуваат дополнување на постојните правила на локалниот огнен ѕид поставен на уредот и се спроведуваат со помош на Tailscale клиентот инсталиран на машината во координација со Wireguard имплементацијата која работи во корисничкиот домен на оперативниот систем.

Табела 6.1

Споредба на VPN решенија и нивна класификација во однос на поставените критериуми

Име	Основни информации			Критериуми						
	Git записи[37]	**Протокол**	**Ѕвездички**[38] **(во илјади)**	**ОК**[39]	**Активно одржувано**	**Сестрано поврзување**	**СР**[40]	**ACL**	**Арх.**[41]	**НМП**[42]
Headscale [242]	2941	Wireguard (КИ[43])	13.6	✓	✓	✓	✓	✓	x86 и ARM	✓
Netbird [243]	903	Wireguard (КИ и ЈИ[44])	4.6	✓	✓	✓	✓	✓	x86 и ARM	✓
ZeroTier [244]	6108	Сопствен	11.4	✓	✓	✓	✓	✓	x86 и ARM	✓
Netmaker [245]	5292	Wireguard (КИ и ЛИ)	6.9	✓	✓	✓	✓	✓	x86 и ARM	✓
Tailscale [246]	5802	Wireguard (КИ)	12.4	X	✓	✓	✓	✓	x86 и ARM	✓
Firezone [247]	2212	Wireguard (КИ и ЛИ)	4.3	✓	✓	X	X	✓	x86 и ARM	✓
WgEasy [248]	141	Wireguard (КИ и ЛИ)	7.5	✓	✓	X	X	X	x86 и ARM	X
Mistborn [249]	220	Wireguard (КИ и ЛИ)	0.5	✓	X	X	✓	✓	x86 и ARM	X
Wesher [250]	186	Wireguard (ЈИ)	0.8	✓	X	✓	✓	X	x86 и ARM	X
Innernet [251]	322	Wireguard (ЈИ)	4.3	✓	✓	✓	✓	✓	x86 и ARM	X
Nebula [252]	383	Сопствен	11.9	✓	✓	✓	✓	✓	x86 и ARM	X

[37] записи, англ. commits.
[38] Број на ѕвездички или додавања во листа на омилени софтвери на соодветниот Git репозиториум. Се користи како мерка за популарност.
[39] ОК – отворен код.
[40] СР – самостојна регистрација.
[41] Арх. – архитектура.
[42] НМП – напредни можности за поврзување.
[43] КИ – корисничка имплементација.
[44] ЈИ – јадрена имплементација.

6.2.2. Стратегија за евалуација на кандидатите

Сите тестирања дискутирани во продолжение се извршени во стандардната специјализирана околина опишана претходно, во која секоја машина има посебна, точно дефинирана улога:

- 1 Kubernetes master јазол, одговорен за Kubernetes контролната рамнина и системските компоненти на Kubernetes додатокот за воспоставување надмрежа.
- 2 Kubernetes worker јазли, врз кои не се извршуваат никакви дополнителни задачи, освен контејнерите за тестирање и неопходните апликативни процеси, дел од надмрежата. Целта на ваквиот пристап е да се осигура прецизноста на резултатите.
- 1 јазол како посредник кој учествува при препраќањето и тунелирањето на VPN сообраќајот со дополнително ниво на UDP или TCP енкапсулација.
- 1 јазол со улога на огнен ѕид и насочувач претставен со PfSense[45] инсталација, овозможувајќи целосна контрола врз комуникацијата меѓу различните јазли, како и симулација на различно ниво на рестрикции во мрежата.
- 1 помошен јазол во улога на хипервизор, со цел креирање виртуелни машини за дополнителни компоненти неопходни за некои од VPN решенијата. Беше одлучено во овој случај да се користат виртуелни машини, бидејќи инстанцираните компоненти не беа дел од податочната патека, така што низ нив не поминуваше големо количество сообраќај, а и самите немаа побарувања за високи перформанси.

Сите физички уреди се поврзани на истиот 1Gbit/s мрежен преклопник и секоја машина е поставена во сопствен изолиран VLAN, користејќи го PfSense огнениот ѕид како упатувач како кон надворешниот свет, така и меѓу VLAN-овите. Ваквото поставување на Kubernetes јазлите во посебни сегменти на ниво 3 овозможува симулација на дистрибуирана околина, каде што различните учесници не се во истата локална мрежа и немаат директно поврзување на второ ниво. Од аспект на екосистемот, на овој начин се симулира комуникација меѓу независни и оддалечени инфраструктури кои би биле поставени на облакот или на работ од мрежата. Користејќи ги сознанијата од претходните поглавја, K3s е користената дистрибуција за подигнување на Kubernetes кластерот поради нејзината ефикасност, високи перформанси и леснотија на користење [25].

Calico е мрежниот додаток за воспоставување на Kubernetes надмрежата. Енкапсулацискиот метод за целиот сообраќај меѓу Kubernetes јазлите, имплементиран од Calico, беше VXLAN, кој за разлика од IPIP, поддржува IPv6 сообраќај [27]. При конфигурирањето на максималната единица за пренос (англ. maximum transmission unit, MTU), особено внимание се обрнува на имплементирањето на најдобрите практики и земањето предвид на дополнителното VXLAN заглавје.

Секое од решенијата подложни на евалуација има посебна архитектура и процедура за поставување. Сепак, во сите случаи е направен напор да се задржи предодредената конфигурација и да се одбегнат рачни промени во изворниот код на софтверот, но, за жал, ова не е секогаш возможно.

[45] https://www.pfsense.org/

Поставување референтна точка

За попрецизна анализа и споредба на резултатите, потребно е воспоставување референтна точка, односно основа, која всушност би служела како еден вид најдобро можно сценарио, отсликувајќи ги перформансите на мрежата без користење VPN. Ваквата референтна точка е дефинирана со извршување на истите тестови, но без VPN подмрежа и со MTU од 1450 и 1230 бајти. Изборот на MTU од 1450 бајти се објаснува со предодредената големина на пакетите која при користење на Calico во VXLAN режим преку етернет е токму 1450 [253]. Од друга страна, пак, MTU од 1230 бајти е најсоодветната вредност при користење софтвери за VPN. Сите решенија, со исклучок на ZeroTier имаат предодредена MTU вредност од 1280 бајти на Wireguard виртуелните мрежни интерфејси. Земајќи го предвид дополнителното VXLAN заглавје, конечната MTU вредност треба да биде поставена на 1230 бајти за најоптимални перформанси. Во случајот со ZeroTier, при креирањето нова мрежа MTU вредноста се поставува на 2800 бајти, но ова подоцна лесно може да се промени од страна на мрежниот администратор.

Headscale

Headscale е името на имплементацијата со отворен код која е отворена замена за Tailscale SaaS контролната рамнина. Станува збор за решение развиено од заедницата кое е компатибилно со Tailscale клиентските алатки и овозможува воспоставување VPN сестрано поврзување, без потпирање на комерцијални, затворени решенија во облакот. Headscale нуди само серверска компонента, бидејќи Tailscale клиентите се самите по себе лиценцирани со отворена лиценца и нивниот код е јавно достапен. Во продолжение, во име на конзистентноста, ќе се употребува исклучиво називот Headscale, иако изнесениот опис е точен и за Tailscale.

Headscale во позадина го користи Wireguard протоколот, имплементиран да работи во корисничкиот домен на оперативниот систем и поддржува грануларни ACL кои би можеле да работат покрај постојните, рачно дефинирани правила на заштитниот ѕид. Комуникација е дозволена исклучиво меѓу уреди за кои постои експлицитно правило за непречен проток на сообраќај или барем сообраќај кон одредена порта. Основната вредност за MTU е 1280 бајти, а може да се интегрира и со надворешни даватели на идентитет (англ. identity providers, IdP), со цел имплементација на централизирана најава.

Заобиколувањето на заштитите во рестриктивни мрежи може да се изведе со користење на предодредени посредници за шифрирани пакети (англ. Designated Encrypted Relay for Packet, DERP) [254]. На овој начин се овозможува препраќање VPN сообраќај со помош на посреднички јаозл со користење TCP порта 443. Со ова се лажира дека станува збор за HTTPS сообраќај дозволен во најголемиот дел од мрежите. Во случај да е возможна непречена и директна врска помеѓу јазлите, посредничкото решение не се користи. Во моментов има десетици DERP сервери овозможени од Tailscale кои се поставени на различни географски локации и се бесплатни за користење. Изворниот код за DERP серверот е јавно достапен, овозможувајќи му секому да покрене сопствена инстанца. Тоа и е направено, водејќи сметка да не се воведат зависности врз кои се нема директна контрола. Една од физичките машини е намената да игра улога на DERP сервер преку компајлирање на неговиот изворен код [255]. Со измена на конфигурациските датотеки на контролната рамнина (Headscale), VPN клиентите се приморани да го користат токму тој сервер, а не некој од останатите под надлежност на Tailscale.

Користењето посреднички DERP сервер за препраќање на сообраќајот е транспарентно за крајниот корисник, бидејќи клиентските апликации имаат можност автономно да

откријат пречки во поврзувањето и да донесат соодветна одлука дали да се користи DERP или не. Се разбира, користењето на DERP дополнително би го намалило пропусниот опсег и би довело до зголемено доцнење, но овие недостатоци можат да бидат надминати до одредена мера со поставување на сопствена инстанца на добро одбрана географска локација со соодветно мрежно поврзување. Префрлувањето од директна конекција кон DERP и обратно е скоро инстантно и не повлекува долги прекини во мрежната комуникација која се одвива низ VPN.

Netbird

Netbird е софтверска алатка со отворен код која се потпира на Wireguard протоколот за воспоставување сестрано поврзување меѓу поголем број уреди во VPN. Од технички аспект, има поддршка за двете имплементации на Wireguard, и онаа која работи во корисничкиот домен на оперативниот систем, но и јадрената. Кој од двата пристапа ќе се користи во даден момент зависи од уредот на кој е инсталиран Netbird клиентот и од тоа дали оперативниот систем и хардвер имаат поддршка за пооптималната јадрена имплементација или не. Со цел поедноставно управување со контролната рамнина, достапен е и веб интерфејс, кој слично како кај Headscale може да биде интегриран со услуги за единствена најава, користејќи сопствен IdP. Предодредената вредност за MTU е 1280 бајти. Имајќи ги предвид понапредните можности кои ги поддржува, потребно е инсталирање проширена имплементација на Wireguard клиентот, наречена Netbird клиент.

Една од позначајните функционалности наведени од страна на Netbird е можноста за воведување препраќање преку посреднички јазол, во случај кога не е возможна директна врска помеѓу две точки. Препраќањето се изведува со постојниот и докажан TURN протокол [256]. Препорачаниот TURN сервер од страна на Netbird заедницата е Coturn [257]. Во VPN на оваа дискусија, вреди да се напомене дека при процесот на избор на ВПМ алатки, беше извршена анализа на документацијата на Netbird, но не беше најдена информација дали препраќањето со TURN е поддржано и со помош на TCP и со помош на UDP протоколите. Потпирајќи се на фактот што Coturn, препорачаниот TURN сервер, ги поддржува двата транспортни протокола, а и дополнителните пронајдени индикатори во изворниот код на Netbird [258], беше претпоставено дека и UDP и TCP може да се користат при препраќањето пакети од VPN врските низ TURN серверот. Во фазата на тестирање, и емпириски се потврди дека функционира TURN препраќање со UDP, но тоа не е случај со TCP, каде што по сè изгледа дека TCP препраќањето досега не е целосно имплементирано. Сепак, беше донесе одлука да се вклучи Netbird, бидејќи и покрај сè, користи јадрена имплементација на Wireguard, нешто што не е случај ниту со Headscale ниту со ZeroTier.

Во околината за тестирање покрената за потребите на истражувањето, префрлувањето помеѓу UDP TURN препраќање и директна врска не е надежно и не секогаш функционира според очекувањата. Ова сценарио е важно, бидејќи во случај да е отстранета пречката за директна врска, секако дека е посакувано да се прекине со препраќањето преку посредник и наместо тоа засегнатите уреди директно да комуницираат меѓусебно. Со цел да се надминат проблемите при префрлање, изменет е изворниот код на Netbird и додадена е предлог закрпа која е јавно достапна на интернет, но сè уште не е дел од актуелната верзија на Netbird [259]. Сите уреди кои учествуваат во Netbird тестирањата потоа продолжуваат со користење на оваа изменета верзија на netclient алатката (Netbird client).

За непречено функционирање на Netbird сестраното поврзување и негово користење како подмрежа за Calico контејнерска надмрежа, беше потребно да се изврши дополнителна промена во нагодувањата. За да се овозможи непречена работа на Calico, сите Calico интерфејси инстанцирани на Kubernetes јазлите мора експлицитно да се стават на црна листа. На овој начин Netbird нема да ги користи како можност за меѓусебно поврзување. Netbird поврзувањето секогаш мора да оди низ физичкиот етернет интерфејс, а не низ виртуелни. На овој начин се решава кружната зависност, каде што Calico мрежата е покрената над VPN виртуелните интерфејси, а, пак, потоа, Netbird ги гледа новите виртуелни Calico интерфејси и ги користи за комуникација со Netbird јазлите. Во случај да не се изврши забрана за користење на Calico виртуелните интерфејси, кога тие ќе бидат одбрани за VPN комуникација меѓу јазлите, мрежата паѓа и поврзувањето меѓу Kubernetes кластер јазлите е прекинато. Од аспект на Coturn TURN серверот, како со случајот со Headscale, оваа компонента беше поставена на посебна физичка машина со гарантирани пресметковни и мрежни перформанси.

Тековно Netbird има поддршка само за „груби“ полиси за контрола на пристап, дозволувајќи или блокирајќи сообраќај без прецизна контрола за тоа кои порти се дозволени или блокирани. Сепак, ова е доволно за фрагментација на една голема мрежа во повеќе помали, независни, сестрано поврзани топологии, управувани од една единствена контролна рамнина.

ZeroTier

ZeroTier е уште еден SaaS продукт и аналогно на Tailscale изворниот код е до одредена мера слободно достапен. Со ZeroTier е возможно да се покрене сопствена контролна рамнина, но без официјалниот веб интерфејс кој е дел од SaaS решението во облак. За самостојни инсталации, достапно е само REST API кое овозможува управување со мрежите и уредите. Заедницата околу ZeroTier има изградено и алтернативни веб интерфејси, но овие не се официјално поддржани и спаѓаат во групата на додатоци развиени од трети страни.

ZeroTier не користи постоечки VPN протокол како на пример OpenVPN или Wireguard, туку има целосно сопствена имплементација [260]. Архитектурата на прв поглед е покомплицирана од онаа на алтернативите, но може лесно да се преслика во соодветните слоеви од ISO/OSI моделот. ZeroTier мрежата за сестрано поврзување е сочинета од крајни уреди (англ. end-devices), мрежни контролери (англ. network controllers) и корени (roots). На крајните уреди е инсталиран zerotier-one клиентскиот софтвер со чија помош може да се изврши придружување или напуштање постоечки ZeroTier мрежи. Контролерите се задолжени за технички аспекти како одобрување нови членови во мрежата, управување со сертификати и синхронизација на конфигурацијата на VPN низ сите јазли учесници. Задачата на корените е да учествуваат во процесот на откривање јазли и помагање во воспоставувањето меѓусебна врска. Во случај кога не е возможна директна врска помеѓу два јазла, може да помогнат корените, препраќајќи ги податоците и играјќи улога на посредници. И контролерите и корените се хостирани на инфраструктура во сопственост на ZeroTier, но постои и можност за нивно поставување врз сопствена инфраструктура. Покренувањето сопствен контролер е добро документирано и објаснето [261], но, за жал, тоа не е случај и за корените. Нагодувањето сопствен корен подразбира креирање на нова „world“ датотека, опишувајќи ја приватната инфраструктура. Дополнително, поставувањето на сопствен корен побарува и рачни промени како во изворниот код на ZeroTier, така и во конфигурацијата на секој од клиентите [262].

Препраќање на сообраќајот преку TCP е возможно и постојат поголем број на глобално дистрибуирани посреднички јазли управувани од страна на ZeroTier кои се достапни преку anycast. За да не се потпира врз трети услуги во туѓа надлежност, покрената е сопствена инстанца од посредникот со користење на официјалната имплементација за ZeroTier TCP посредникот чиј изворен код е слободен [263]. Како и во останатите случаи, користењето на посредникот подразбира рачни промени во конфигурациските датотеки на клиентите. Притоа, посредничката компонента е поставена на посебна физичка машина.

За гарантирање надежно поврзување во рамките на Calico надмрежата, потребно е и во овој случај експлицитно блокирање на Calico виртуелните интерфејси, со цел тие да не бидат користени од страна на ZeroTier, избегнувајќи кружна зависност.

Кога станува збор за ACL, ZeroTier наликува на традиционалните, физички мрежи. Еден уред може да биде дел од повеќе ZeroTier виртуелни приватни мрежи истовремено, а возможно е и паралелно конфигурирање на повеќе топологии на сестрано поврзување меѓу независни уреди, користејќи една единствена контролна рамнина.

Netmaker

Netmaker е едно од VPN решенијата разгледувано за вклучување во тестирањата, имајќи предвид дека од неодамна е воведена можност за препраќање на сообраќајот низ посреднички јазли, во ситуации кога не е можна директна врска. Се темели врз Wireguard протоколот и слично како и Netbird, ги поддржува и корисничката и јадрената имплементација, во зависност од уредот каде е извршена инсталацијата. Друга сличност со Netbird е тоа што и во овој случај препраќањето се врши со помош на TURN сервер. Во фазата на евалуација беше испробана првата верзија на Netbird која има поддршка за препраќање (верзија v0.20.0), но се наиде на проблеми кои го направија вклучувањето на Netmaker во понатамошното истражување невозможно. Механизмот за избор дали да се користи директна врска или препраќање меѓу јазлите не работи соодветно и предизвикува Netmaker секогаш да користи TURN препраќање, дури и во случаи кога нема никакви рестрикции во мрежата. Дополнително, дури и при препраќање, мрежното поврзување е ненадежно и се случуваат прекини на секои десетина секунди. Инаку, во тековната имплементација, Netmaker поддржува само TURN препраќање со помош на UDP.

Како резултат на претходно опишаните проблеми, Netmaker не е вклучен во финалните резултати, бидејќи и нивната веродостојност не може да се гарантира поради честите прекини. Сепак, Netmaker архитектурата е слична како онаа на Netbird, а и двете решенија ги поддржуваат двете Wireguard имплементации, па и слични крајни резултати би биле очекувани.

6.2.3. Дефинирање тестови за симулација на облак-раб екосистем низ различни мрежи

Сите претходно претставени решенија имаат потенцијал да овозможат сестрано поврзување меѓу различни пресметковни инфраструктури, а со тоа и да учествуваат во воспоставувањето на облак-раб екосистем кој би се протегал низ различни мрежи. Сепак, за формално потврдување на нивните карактеристики, вклучително и можноста за справување со различни мрежни аномалии како зголемено доцнење, загуба на пакети и попречено поврзување, потребно е спроведување детални практични тестови.

Тестовите за евалуација на VPN решенијата се дефинирани земајќи предвид 5 различни конфигурации, од кои две се посветени на оформување референтна вредност во случаи

кога Calico надмрежата не е поставена врз VPN, користејќи 1450 бајти и 1230 бајти MTU на Calico VXLAN интерфејсите. Целта на преостанатите три теста е евалуација на перформансите на Headscale, Netbird и ZeroTier решенијата. За трите решенија се дефинирани и повеќе потсценарија, симулирајќи мрежи кои го ограничуваат сообраќајот и вршат негово агресивно филтрирање и/или NAT преведување. Потсценаријата служат за опсежно тестирање на функционалностите за препраќање податоци преку посреднички јазли. И за Headscale и за ZeroTier ова се постигнува со блокирање на целосниот UDP сообраќај меѓу учесниците во виртуелната приватна мрежа, користејќи го PfSense огнениот ѕид. Ваквата активност резултира со TCP тунелирање преку посредник за VPN сообраќајот, немајќи друга алтернатива за осигурување на поврзувањето. Тестирањето на Netbird открива дека не е возможно TCP препраќање со помош на TURN протоколот, па стратегијата се менува така што се блокира директен UDP сообраќај меѓу јазлите освен кон предодредената UDP порта на TURN серверот.

Вкупно се дефинирани 8 различни сценарија за тестирање, оперирајќи со 5 конфигурации, при што во секоја конфигурација се користи Calico како надмрежа:

- MTU од 1450 бајти без VPN подмрежа.
- MTU од 1230 бајти без VPN подмрежа.
- Headscale со директно поврзување меѓу јазлите.
- Headscale со препраќање преку TCP низ DERP сервер поставен во изолиран VLAN, но поврзан за истиот мрежен преклопник како и останатите јазли.
- Netbird со директно поврзување меѓу јазлите.
- Netbird со препраќање преку UDP со помош на Coturn TURN сервер, поставен во изолиран VLAN, но поврзан за истиот мрежен преклопник како и останатите јазли.
- ZeroTier со директно поврзување меѓу јазлите.
- ZeroTier со препраќање преку TCP, користејќи ја официјалната имплементација за TCP посредник поставена во изолиран VLAN, но поврзана за истиот мрежен преклопник како и останатите јазли.

За секое сценарио, спроведени се следниве тестови:

- Kubernetes Pod-Pod TCP и UDP тестови за пропусен опсег, соодветно, со комуникација меѓу подовите која се одвива преку Calico надмрежата. Секој тест е извршен 60 секунди и повторен 100 пати.
- Kubernetes Pod-Pod TCP и UDP тестови за пропусен опсег, со индуцирана загуба на пакети од 1%, 5% и 10%, соодветно, на Calico интерфејсот. Секој тест е извршен 60 секунди и повторен 10 пати.
- Kubernetes Pod-Pod TCP и UDP тестови за пропусен опсег, со индуцирано доцнење на пакети од 50, 250 и 350 милисекунди соодветно, на Calico интерфејсот. Секој тест е извршен 60 секунди и повторен 10 пати.
- Време на одговор на Kuberentes API-то, со 10 итерации по сценарио, каде што една итерација се состои од вкупно 5000 барања упатени кон Kubernetes, со максимален степен на паралелизам од 10 барања истовремено во даден момент.

Дефинираните сценарија не го разгледуваат само случајот каде што контролната рамнина на Kubernetes кластерите е достапна преку VPN мрежата, туку и случајот кога самата комуникација меѓу Kubernetes јазлите се одвива низ воспоставените тунели. Ова во пракса овозможува еден ист Kubernetes кластер да се протега низ повеќе локации од екосистемот, со јазли и во облакот и на работ од мрежата.

При спроведување на тестовите се следи и оптоварувањето врз процесорот, со цел извлекување заклучоци во однос на ефикасноста на секое од решенијата. Постои разлика меѓу бројот на извршувања кај тестовите каде што не се индуцирани мрежни проблеми и кај оние каде што тоа е сторено, бидејќи првите служат за евалуација на клучните карактеристики и однесувањето на VPN решенијата. Мрежни проблеми се индуцираат само за потребите на евалуацијата, т.е. дали е воопшто возможна каква било мрежна комуникација во таква непријателска средина. Вредностите за времето на доцнење се избрани за да се моделираат глобално дистрибуирани кластери и екстремно оптоварени мрежи, каде што нема дефинирано соодветни правила за квалитетот на услугата, давајќи приоритет на временски зависниот сообраќај. Слично на ова, степените на загуба на пакети од 1%, 5% и 10% претставуваат ненадежни мрежни врски кои може да се појават поради различни причини, како на пример несоодветна опрема или огромни количества сообраќај што ги преплавуваат баферите на некои од мрежните уреди лоцирани на податочната патека. На овој начин се симулираат реални сценарија можни при воспоставување облак-раб екосистем, земајќи предвид дека станува збор за дистрибуирана архитектура каде што се нема целосна контрола врз мрежното поврзување, затоа што се одвива низ јавни врски кои може да бидат преоптоварени (зголемено доцнење) или да воведуваат загуба на пакети. За воведување на доцнењето и загубата на пакети при извршувањето на тестовите се користи алатката tc (traffic control) [264].

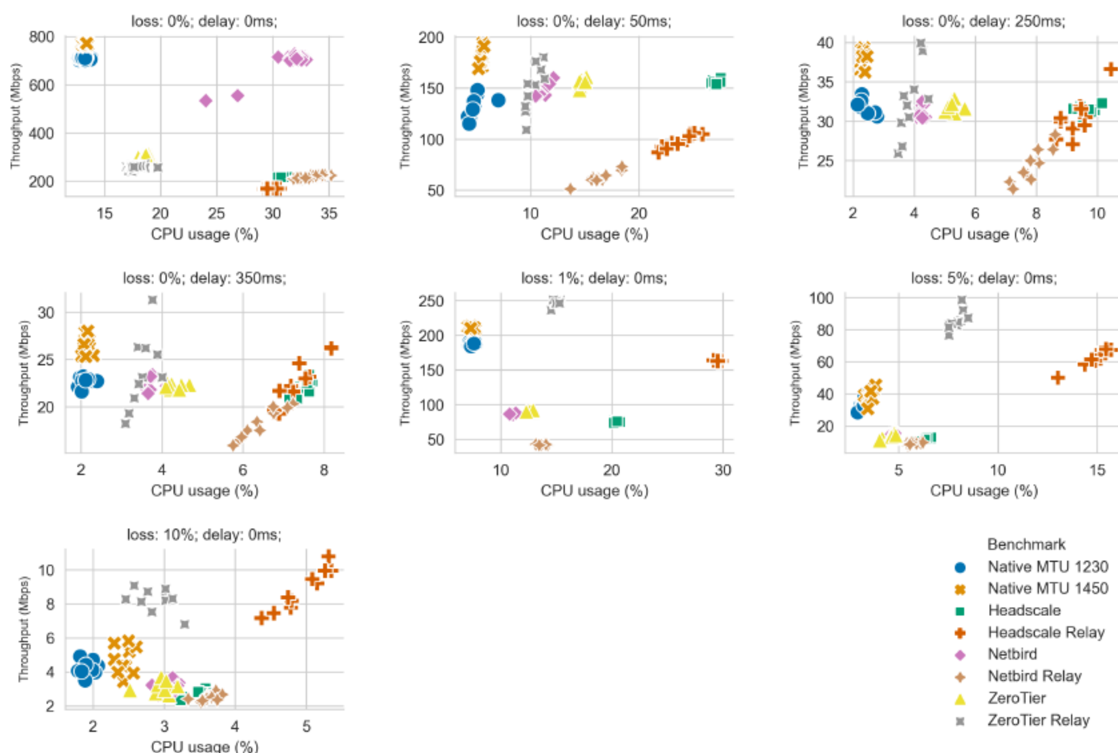
За практична имплементација на тестовите, се користи адаптирана верзија на Kubernetes Network Benchmark (KNB) свитата тестови [265], [266] за сите Pod-Pod сценарија. KNB во позадина ја користи познатата ipef3 алатка за спроведување тестови меѓу различни подови, каде што еден од нив игра улога на сервер, а другиот на клиент. И покрај тоа што сите Kubernetes јазли имаат идентична хардверска конфигурација, поставувањето на клиент и сервер улогите е фиксирано низ сите тестови, со цел уште поголема сигурност во веродостојноста на резултатите. На овој начин, практично, еден јазол постојано ја извршува серверската улога, а друг клиентската.

Тестирањето на времето на одговор на Kubernetes API серверот е направено во рамките на Kubernetes кластерот, користејќи ја Hey алатката [162]. Операцијата која постојано се извршува е добивање листа на активни подови во кластерот. Времето на одговор на Kubernetes API серверот е од особено значење, бидејќи целата комуникација со контролната рамнина од страна на јазлите се извршува токму на овој начин. Неконзистентни перформанси или подолг период на недостапност на API серверот може да ја загрозат севкупната функционалност на Kubernetes кластерот, елиминирајќи го механизмот за координација меѓу јазлите кои учествуваат во извршување на пресметковните задачи.

6.3. Споредба на ефикасноста на VPN решенијата при обезбедување транспарентно поврзување во облак-раб екосистемот

Во продолжение следи осврт на резултатите добиени со изведување на тестовите опишани во претходната секција.

6.3.1. TCP пропусен опсег



Слика 6.1 Споредба помеѓу TCP пропусниот опсег и искористеноста на процесорот за секое од сценаријата

За секое од седумте сценарија беше тестиран pod-pod пропусниот опсег за TCP сообраќај, над Calico надмрежата. Имајќи предвид дека најголемиот број на безсерверски функции користат TCP на транспортно ниво како за нивно повикување, така и за комуникација со BaaS алатките, ваквиот тип на сообраќај се очекува да биде најзастапен во облак-раб екосистемот. На слика 6.1 се претставени добиените резултати за брзината на мрежата, како и процесорското оптоварување за времетраење на тестот.

Започнувајќи ја дискусијата со она сценарио каде што нема несакани мрежни пречки, најдобрите резултати се постигнати со MTU од 1450 бајти, без користење на VPN поврзување. Ваквото сценарио во реалноста ретко би се практикувало при воспоставувањето облак-раб екосистеми, поради недостатокот од дополнително шифрирање на мрежниот сообраќај. Двете референтни вредности, MTU од 1450 бајти и MTU од 1230 бајти покажуваат слично процесорско оптоварување, со таа разлика што во случајот на MTU од 1450 бајти, пропусниот опсег е поголем, што може да се препише на поголемите пакети. Осврнувајќи се кон резултатите постигнати од решенијата за виртуелни приватни мрежи кои овозможуваат дополнително шифрирање, Netbird при работа во режимот со директно поврзување меѓу учесниците има јасна предност пред останатите, ако се суди според пропусниот опсег. Средната вредност постигната од Netbird изнесува 707.45 Mbps, што е споредливо со референтната вредност при користење MTU од 1230 бајти, каде што брзината е 709.94 Mbps. Ваквите одлични перформанси се должат на ефикасната имплементација на Wireguard која работи во јадрениот простор и е поддржана од Netbird. Второто место припаѓа на ZeroTier, додека, пак, Wireguard имплементацијата во корисничкиот простор на оперативниот систем користена од Headscale е најбавна. Осврнувајќи се на процесорското оптоварување, што е од особено значење кога станува збор за уреди со потенцијално ограничени

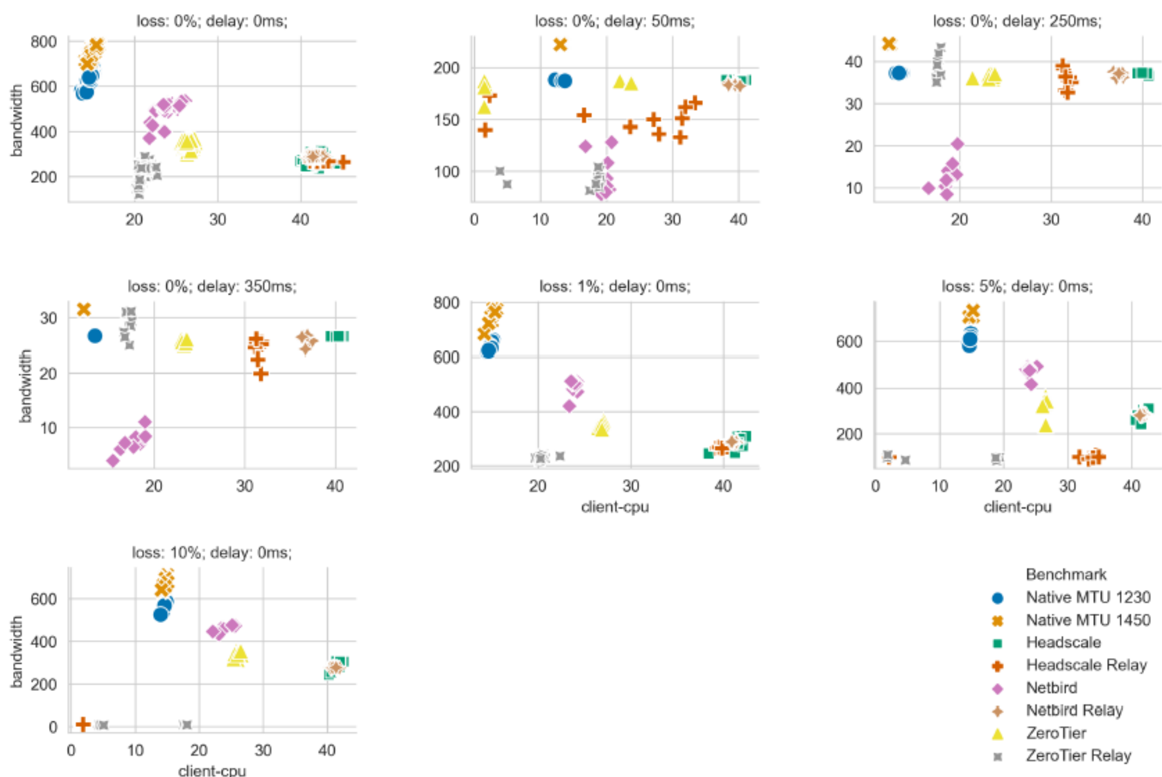
пресметковни капацитети, како оние од работ од мрежата, ZeroTier е најефикасен, како при директно поврзување меѓу јазлите, така и при препраќање низ TCP посредник. Сценариото каде што Netbird користи UDP посредник покажува најголемо оптоварување на процесорот, нудејќи полоши перформанси од ZeroTier препраќањето, но подобри од TCP препраќањето на Headscale.

Вреди да се напомене дека во сите случаи, очекуваните перформанси при употреба на поврзување во VPN е очекувано да бидат полоши од оние без VPN. Кај VPN се додаваат дополнителни нивоа на апстракција и потребна е додатна енкапсулација на сообраќајот; почнувајќи од енкапсулацијата за VPN тунелите, преку опционалното препраќање низ посредник, продолжувајќи со Calico надмрежата која го енкапсулира сообраќајот што потекнува од подовите со VXLAN и на крајот, на врвот, сообраќајот од апликацискиот слој генериран во рамките во самите подови, во овој случај со користење на iperf3 алатката.

Напуштајќи ги идеалните услови во мрежата и фокусирајќи се на сценариото во кое постепено се воведува дополнително доцнење со цел симулација на реални проблеми при воспоставување поврзување низ облак-раб екосистемот, забележливо е дека MTU од 1450 бајти референтната вредност без VPN повторно нуди најдобри резултати во сите три случаи, за доцнење од 50, 250 и 350 милисекунди. Како што се зголемува доцнењето, така перформансите на Netbird при UDP препраќање опаѓаат, покажувајќи најлоши резултати во сите три тестирања, што не беше случај претходно. Од друга страна, пак, Headscale сега покажува споредливи резултати со оние останатите и има најдобра средна брзина од 156.3 Mbps при тестот со доцнење од 50 милисекунди, каде што ZeroTier освојува споредливи 155.6 Mbps. Очекувано, како што се зголемува доцнењето, така опаѓаат и перформансите на сите VPN решенија. Занемарувајќи ги резултатите при препраќање и фокусирајќи се само на сценаријата со директно поврзување, сите решенија постигнуваат средна брзина од 22 Mbps при најекстремното доцнење од 350 милисекунди. Забележливо е тоа што ZeroTier нуди резултати карактеризирани со висока стандардна девијација низ сите три сценарија со доцнење. Тоа резултира со непредвидливост на доцнењето што негативно би влијаело при комуникацијата во рамките на облак-раб екосистемот. Од аспект на препраќање, Headscale и ZeroTier нудат подобри перформанси од UDP препраќањето на Netbird, како резултат на користењето на TCP протоколот кој нуди надежен пренос на податоците при сè полоши мрежни услови. Zerotier заедно со својот сопствен протокол сè уште користи најмалку пресметковни ресурси низ сите три сценарија. Headscale користи најмногу што може да се препише на Wireguard имплементацијата во корисничкиот домен.

Второто сценарио кое разгледува пречки во мрежата е додавање на сè поголема загуба на пакети, со цел симулација на прекини во мрежното поврзување меѓу различните инфраструктури од облак-раб екосистемот. Анализирајќи ги резултатите, Headscale и ZeroTier препраќањето нудат конзистентно драстично подобри перформанси од алтернативите, вклучувајќи ги и референтните вредности без VPN при користење MTU од 1230 и 1450 бајти. Единствената сличност помеѓу овие две решенија која може да го објасни ваквото однесување е користењето на TCP протоколот при препраќањето на сообраќајот. Напредните можности за контрола на застојот имплементирани од TCP резултираат со скоро два пати подобри брзини споредено со алтернативите. Како и досега, анализирајќи го оптоварувањето на процесорот, Headscale имплементацијата е помалку ефикасна споредено со ZeroTier.

6.3.2. UDP пропусен опсег



Слика 6.2 Споредба помеѓу UDP пропусниот опсег и искористеноста на процесорот за секое од сценаријата

Аналогно на анализата при користење TCP сообраќај дискутирана претходно, на слика 6.2 се дадени резултатите при користење UDP, заедно со оптоварувањето врз процесорот. Евалуацијата на перформансите при користење UDP сообраќај е од големо значење за облак-раб екосистемот, особено со сè поголемата популаризација на HTTP/3 протоколот кој работи над UDP на транспортното ниво [267].

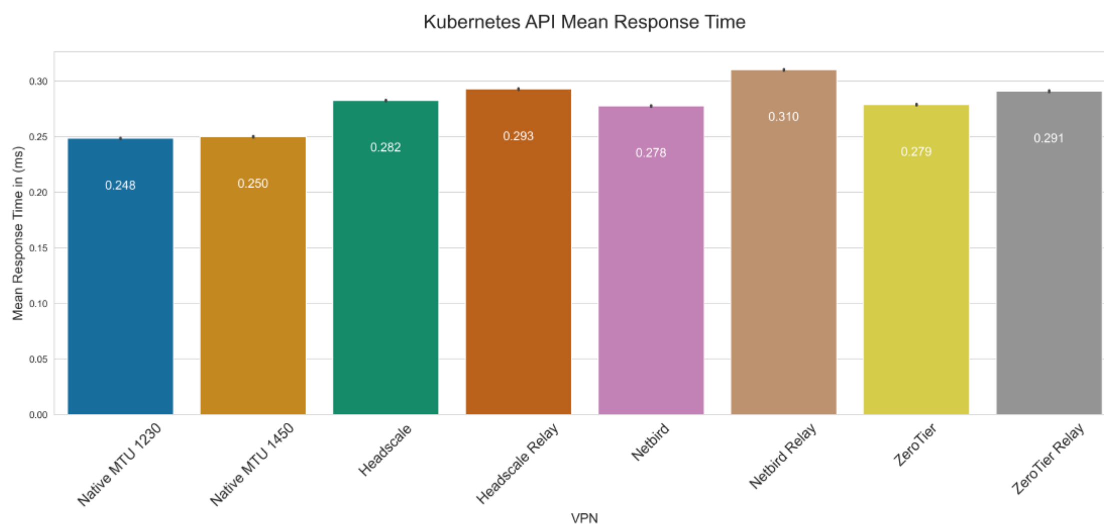
Во случајот кога нема несакани пречки во мрежата, повторно референтните вредности на MTU од 1230 и 1450 бајти, без користење VPN, покажуваат најдобри резултати, што не е изненадувачки и е идентично со она што беше видено претходно. Друга сличност со TCP сценаријата е тоа што Netbird и во овој случај нуди најдобри перформанси, ZeroTier е на второ место, а Headscale на трето. При препраќање, Netbird постигнува најголема средна брзина од 289,51 Mbps, при UDP низ UDP тунелирање, а потоа следат Headscale и ZeroTier со своите TCP имплементации за препраќање, со 256,7 Mbps и 227,09 Mbps, соодветно. Headscale покажува најголемо оптоварување на процесорот достигнувајќи и над 40%, а Zerotier најмало.

Во однос на сценариото каде што постепено се воведува доцнење во мрежата, гледајќи ја визуелизацијата за 50 милисекунди, се забележува драстично зголемување во стандардната девијација за сите VPN решенија, со исклучок на Headscale од една страна и Netbird препраќањето со UDP, од друга. Headscale исто така постигнува најдобра брзина со средна вредност идентична со онаа на референтното мерење извршено при MTU од 1230 бајти (187,5 Mbps). Netbird и Zerotier се следни со средни вредности од 183,7 Mbps и 193,11 Mbps, соодветно. ZeroTier препраќањето има најмала средна вредност од 92,76 Mbps, карактеризирајќи се со неконзистентни резултати и висока

стандардна девијација. Сепак, како што се зголемува доцнењето, така се зголемуваат и перформансите на ZeroTier препраќањето, постигнувајќи најдобри резултати при 250 и 350 милисекунди доцнења, поголеми дури и од референтната вредност со MTU од 1230 бајти. Останатите резултати се сродни меѓусебе, но забележливо е тоа што Netbird не се справува на соодветен начин кога сообраќајот од апликациско ниво користи UDP транспортен протокол и има доцнења во мрежата, покажувајќи најлоши резултати во сите три сценарија. Од друга страна, пак, Netbird покажува подобрување кога станува збор за загуба на пакети, заземајќи го првото место при 1%, 5% и 10% загуба. Интересно, спротивно на она што беше видено во минатата потсекција при анализа на TCP перформансите, сега, при UDP апликациски сообраќај, Headscale и ZeroTier тунелирањата низ TCP нудат најлоши резултати. Ова води до заклучокот дека тунелирање на UDP апликациски сообраќај низ TCP врска не е оптимална стратегија.

Од аспект на оптоварување на процесорот, забележливо е негово зголемено користење при UDP сценариото, споредено со TCP. ZeroTier и Netbird конзистентно нудат најмал степен на оптоварување на процесорот, сè разбира, не земајќи ги предвид референтните вредности со 1230 и 1450 бајти MTE. Ова е така дури и во случајот со 250 милисекунди и 350 милисекунди доцнења, каде што ZeroTier не само што има најмало оптоварување на процесорот, туку и покажува најдобри перформанси. Headscale, со Wireguard имплементацијата во корисничкиот домен предизвикува најголемо процесорско оптоварување. Во одредени случаи, поефикасно од аспект на процесорот е да се врши препраќање наместо директно поврзување, но треба да се има предвид дека во таквите ситуации и брзината е помала.

6.3.3. Време на одговор при користење на програмскиот интерфејс на Kubernetes



Слика 6.3 Средно време на одговор за Kubernetes API серверот
при користење на различните VPN решенија

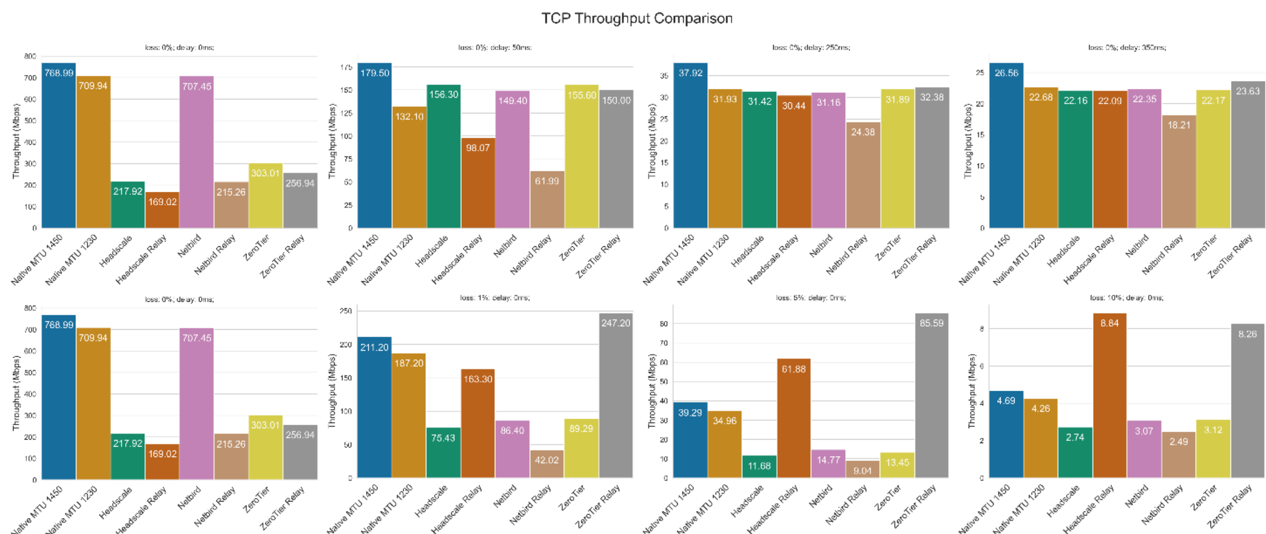
Последниот спроведен тест е мерење на перформансите на Kubernetes интерфејсот за апликативни програми, анализирајќи го неговото однесување при користење различни VPN решенија за воспоставување врска меѓу јазлите. Ова е од особено значење во рамките на облак-раб екосистемите, бидејќи ја отсликува робусноста и брзината на комуникацијата во сценарија каде што екосистемот е сочинет од различни Kubernetes кластери обединети меѓусебно. Во ваквите случаи координацијата меѓу различните околини би се одвивала токму преку Kubernetes API серверите.

На слика 6.3 се дадени средните времиња на одговор на Kubernetes интерфејсот, при извршување на максимум 10 барања во секунда. Како и претходно, двете референтни вредности повторно покажуваат најдобри резултати, што веќе е и очекувано. Осврнувајќи се на VPN решенијата, Netbird нуди најдобри перформанси, по што следат ZeroTier и Headscale. Ваквата поставеност е еквивалентна на виденото при TCP и UDP тестовите, при отсуство на несакани мрежни пречки. При препраќање, ZeroTier и Headscale покажуваат слично однесување, а, пак Netbird посредништвото со UDP протоколот е најмалку ефикасно.

6.4. Импликации врз воспоставувањето транспарентен облак-раб екосистем

Разгледувајќи ги добиените резултати, станува јасно дека крајната одлука зависи од предностите и недостатоците на секое од решенијата и нивното влијание врз можноста за исполнување на крајната цел – воспоставување на транспарентно поврзување како предуслов за градење унифициран облак-раб екосистем. Суровите перформанси не треба да бидат единствениот критериум кој се зема предвид. Од особена важност се и леснотијата за придружување нови уреди во сестраното поврзување, надежноста на воспоставените конекции, како и севкупната комплексност на поставување и одржување на VPN инфраструктурата.

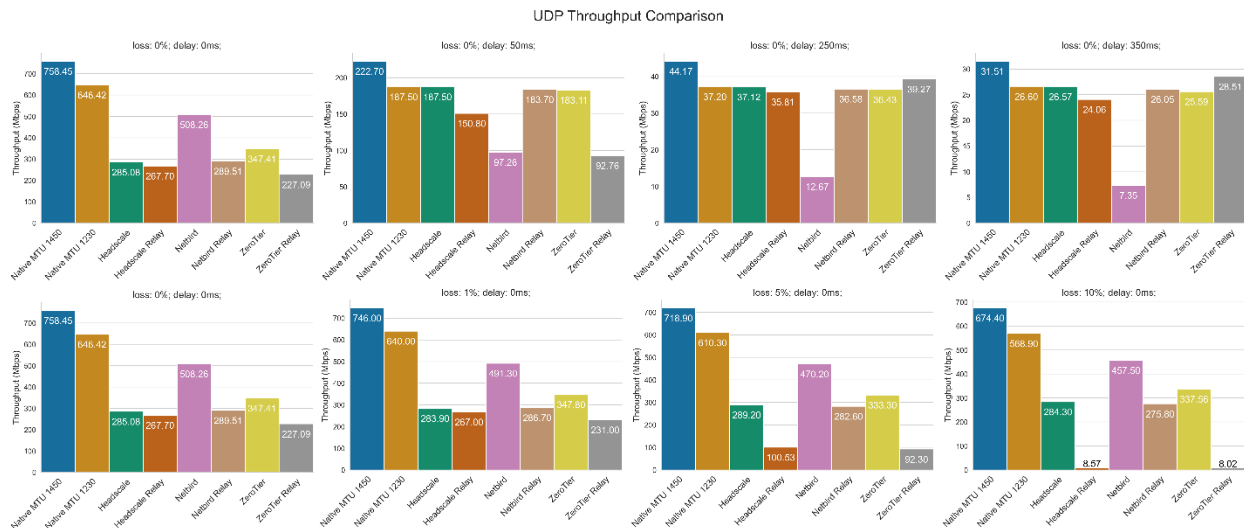
Како поткрепа на дискусијата околу предностите и слабостите на секое од решенијата, на слика 6.4 и слика 6.5 се претставени средните TCP и UDP пропусни опсези, во Mbps, измерени при тестирањето.



Слика 6.4 Средни вредности за пропусниот опсег при користење TCP

Анализирајќи ја најочигледната карактеристика прва, пропусниот опсег, Netbird покажува најдобри резултати кога мрежното поврзување меѓу инволвираните јазли е робусно и не се присутни значителни доцнења или загуба на пакети. Netbird Wireguard имплементацијата интегрирана во јадрото на оперативниот систем е ефикасна и не води до драстичен пад во перформансите при користењето VPN. Од друга страна, пак, кога станува збор за ситуации каде што постои зголемено доцнење меѓу одредени јазли, и Headscale и ZeroTier се подобар избор. Двете решенија имаат многу слични перформанси при справувањето со TCP и UDP сообраќај, па во ситуации кога доцнењето е предизвик, изборот би зависел од останатите карактеристики на решенијата, како на пример леснотијата на додавање нови уреди, дискутирана подолу. При справување со еден друг тип на мрежен предизвик, загубата на пакети, ZeroTier издвојува најдобри резултати во

сите случаи, со исклучок на еден, оној при 10% загуба, каде што Headscale има предност, како што може да се види од слика 6.4. Притоа, поефикасно е да се користи функционалноста на TCP препраќање преку посреднички јазол, понудена од Headscale или ZeroTier, за тунелирање на сообраќајот при загуба на пакети, наместо воспоставување директна VPN врска меѓу јазлите, која би користела UDP. Сепак, ваквиот заклучок е релевантен само во ситуации кога апликацискиот протокол кој се тунелира низ ВПМ работи врз TCP. Кога апликацискиот протокол користи UDP, Netbird повторно покажува најдобри резултати.



Слика 6.5 Средни вредности за пропусниот опсег при користење UDP

Постигнатата брзина е многу важен и сериозен, но не и единствен фактор кој има импликации врз воспоставувањето облак-раб екосистем. Од аспект на трудот кој треба да се вложи за подигнување на сите потребни компоненти, Headscale и Netbird имаат очигледна предност пред ZeroTier. ZeroTier го обесхрабрува самостојното управување со јазли кои ја играат улогата на корени и се дел од контролната рамнина и за оваа цел неопходни се рачни промени. Официјалната документација содржи информации за бараните промени, но, сепак, станува збор за активност која би одзела поголемо време, особено ако се земе предвид дека е потребно воспоставување целосна развојна околина со цел изменување на изворниот код. Дополнително, ZeroTier мобилната апликација не поддржува користење сопствени корени во овој момент [268], додека десктоп апликациите поддржуваат, но исто така со претходно рачно спроведени измени. ZeroTier има и дополнителен недостаток, а тоа е огромното време на конвергенција кога е потребно да се вклучи препраќање преку посреднички јазол. При тестирањето, мерењата покажаа дека ова време е во рангот од 3 до 4 минути. Вакви проблеми кои се манифестираат со губење на VPN врската при преминување од директно поврзување кон препраќање и обратно не беа забележани ниту со Netbird, ниту со Headscale.

Кога станува збор за Netbird, во моментов, поддржано е исклучиво TURN препраќање преку UDP, но има иницијативи за поддржување и на TCP во иднина. При тестирањето со најновата верзија на софтверот во дадениот момент, механизмот за детекција дали да се врши препраќање не беше комплетно надежен, па мораше да бидат имплементирани измени во изворниот код, со цел да се форсира постојано TURN препраќање и да се комплетираат соодветните тестови без проблеми и без закани по релевантноста на резултатите.

Headscale, како целосно слободна алтернатива на Tailscale контролната рамнина, не манифестираше поголеми проблеми при тестирањето. Дополнително, Headscale има и поддршка за прецизни ACL кои се имплементирани од страна на Tailscale клиентот инсталиран на секој јазол од сестраното поврзување. Еден сè уште релевантен негативен аспект е потребата за пишување на сите ACL правила во JSON датотека која е неопходно да биде поставена на самиот сервер каде што се наоѓа Headscale контролната рамнина. Не само што е потребен директен, привилегиран, пристап до серверот за ажурирање на ACL правилата, туку неопходно е и рестартирање на соодветниот процес во оперативниот систем. Функционалноста за повторно вчитување на датотеката со ACL правила без потреба од целосно рестартирање е достапна само во затворената верзија на официјалната Tailscale контролна рамнина.

Резимирајќи ги импресиите од корисничкото искуство добиено со секое од решенијата, исклучиво Netbird има официјален, вграден, веб интерфејс. Останатите решенија, Headscale и ZeroTier, нудат само REST интерфејси за апликативни програми, но постојат трети проекти со отворен код кои може да изградат визуелен приказ врз REST.

6.5. Избор на најсоодветно решение

Резултатите покажаа дека изборот на најсоодветното решение на ова поле зависи од контекстот во кој тоа би се употребувало и од придружените карактеристики на самата мрежа. Врз основа на ова, Netbird, со својата Wireguard јадрена имплементација нуди перформанси кои се блиски до оние нативните, видливи кога не се користи VPN, но само во случаи кога постоечката мрежна е робусна, без доцнења или загуба на пакети. Netbird е исто така препорачаното решение кога постои загуба на пакети, но само под услов апликацискиот протокол кој се тунелира, да користи UDP. Ова не е случај при тунелирање TCP сообраќај, бидејќи и Headscale и ZeroTier постигнуваат драстично подобри перформанси во тие сценарија. Имајќи предвид дека сообраќајот кој се тунелира при извршување безсерверски функции во најголем дел од случаите е TCP базиран, оваа предност не игра улога при воспоставувањето на облак-раб екосистемот. Во случај во мрежата да се забележува поголемо доцнење, Headscale или ZeroTier се подобриот избор.

Сите тестирани решенија успешно воспоставуваат сестрано поврзување меѓу инволвираните јазли и возможно е подигнување на Kubernetes кластери над нив кои би го сочинувале облак-раб екосистемот. Напредните функционалности за заобиколување NAT и препраќање преку посреднички јазли овозможуваат проширување на постоечки или креирање сосема нови Kubernetes кластери дури и во рестриктивни мрежи, со ограничено поврзување кон надворешниот свет.

Земајќи ги предвид осознаените карактеристики на секое од решенијата при нивното тестирање во пракса, заклучоците кои може да се изнесат во контекст на нивната примена за воспоставување облак-раб екосистем се следни:

- Употреба на Netbird во ситуации кога може да се гарантира дека мрежите во кои се наоѓаат јазлите од екосистемот се отворени, без дополнителни ограничувања во поглед на мрежни порти или транспортни протоколи. Во прилог на ова одат високите перформанси како резултат на јадрената имплементација и вградениот веб интерфејс за управување.
- Употреба на ZeroTier во ситуации кога во екосистемот учествуваат јазли кои постојано имаат ненадежна мрежа и висок степен на загубени пакети. Ова решение покажува најдобри резултати при вакви сценарија, но тешкото прилагодување за целосно автономна работа, без потпирање на SaaS компонентите, го прави несоодветно за употреба во општ случај. Огромното

време на конвергенција при промена на условите во мрежата дополнително ја отежнува примената во екосистем каде што пресметковните јазли би се наоѓале во високо рестриктивни мрежи.

- Употреба на Headscale како универзално решение кое може ефикасно да се прилагоди на променливи услови во мрежата, притоа нудејќи лесна адаптација за целосно независна употреба од соодветното SaaS решение.

Headscale го нуди најдоброто корисничко искуство, без потреба од какви било рачни промени врз изворниот код за постигнување целосна функционалност и има ефикасен механизам за транзиција помеѓу директно поврзување и препраќање. Дополнително, во случај да има потреба, препраќањето се извршува преку порта 443 и се маскира да изгледа како HTTPS сообраќај, зголемувајќи ги шансите за сообраќајот да биде успешно пропуштен. Поддршката за прецизни ACL правила е исто така позитивен аспект со кој Headscale се издвојува од останатите решенија. Како резултат на ова, во овој момент Headscale претставува најсоодветниот избор за креирање подмрежи мрежи над кои би се поставиле Kubernetes надмрежи за дистрибуирана комуникација меѓу јазлите во екосистемот.

Овие сознанија водат кон потврдување на третата хипотеза, т.е. *„Новите решенија за воспоставување на надмрежи може да се користат за создавање на ефикасно меѓуповрзување на целосно различни пресметковни околини, како од аспект на нивните перформанси, така и во однос на нивната географска поставеност. На овој начин се овозможува креирање целосно ново ниво на апстракција над физичките мрежи, што е неопходен чекор за воспоставување на посакуваниот облак-раб екосистем.“* Поглавјето кое следи ја продолжува дискусијата за воспоставување на унифициран облак-раб екосистем со претставување архитектура втемелена на сестрано поврзување меѓу дистрибуирани пресметковни инфраструктури.

7. ПРЕДЛОГ АРХИТЕКТУРА ЗА ТРАНСПАРЕНТНИ ПРЕСМЕТКИ ВО ОБЛАК-РАБ ЕКОСИСТЕМОТ

Со дискусијата и резултатите прикажани во претходните поглавја беше потврдено дека безсерверската парадигма е соодветна за широка примена како во облакот, така и на работ од мрежата. Сепак, за овозможување транспарентни пресметки и воспоставување целосно функционален облак-раб екосистем, потребно е да се овозможи тесна меѓусебна интеграција. На овој начин ќе се искористат предностите и на двете околини до нивниот максимум. Безсерверските функции кои се извршуваат на облакот се карактеризираат со високи перформанси и лесна скалабилност, но бавен ладен старт, големо мрежно доцнење и намалена контрола врз податоците. Од друга страна, пак, безсерверски функции на работ од мрежата се отсликуваат со ниско мрежно доцнење и можност за извршување блиску до локацијата на самите податоци, но поограничени перформанси како резултат на скромниот пресметковен капацитет на уредите на работ од мрежата. Дополнителна оптимизација на ладниот старт може да се направи со користење специфични извршни околини како WebAssembly. Со проширување на пресметковните инфраструктури така што тие би овозможиле транспарентно извршување на задачите и на облак, и на работ од мрежата, може да се надминат овие типични недостатоци. Ваквите недостатоци се карактеристични не само за безсерверското пресметување, туку и за останатите моментално актуелни парадигми, со што дополнително се потенцира големината и важноста на проблемот. Резултатите од претходните поглавја како од аспект на мрежно поврзување, така и од аспект на извршни околини и оркестрација може да служат како темел за дизајн на повеќенаменска архитектура на безсерверски платформи кои транспарентно би извршувале пресметки низ унифицираниот облак-раб екосистем. На овој начин, одредена пресметковна задача би можела динамички да биде распределена на најпогодната инфраструктура за нејзино извршување, притоа земајќи ги предвид побарувањата во однос на перформанси, доцнење и цена при носењето на одлуката, нудејќи решение за актуелните проблеми на ова поле.

Најголемиот недостаток на постојните иницијативи за воспоставување облак-раб пресметковни екосистеми е тоа што тие се во голема мера затворени, ограничувајќи ги корисниците исклучиво на продукти од еден конкретен давател на услуги, како во облакот, така и на работ. Една од главните цели на оваа докторска дисертација е дизајн на архитектура на унифициран екосистем кој ќе ги обедини различните екосистеми во едно заедничко транспарентно решение. Во насока на ова, терминот кој ќе се користи за постојните затворени решенија ограничени на еден давател на услуги во ова поглавје е „пресметковна заедница“ со цел да се направи разлика во однос на воведувањето на новиот обединувачки унифициран екосистем.

Трендот корисниците да се ограничуваат да може да користат ресурси од само еден давател на услуги до одреден степен е и парадоксален, бидејќи еден од главните пориви за прифаќање на пресметувањето на работ од мрежата, покрај намаленото доцнење, беше токму можноста за избегнување на централизацијата карактеристична за облакот. Иако централизацијата во општ случај не значи и ограничување на слободата на избор, кога станува збор за услуги од ист давател на услуги, во практиката тоа е така. Големите компании на ова поле нудат целосни портфолија кои вклучуваат десетици, па и стотици различни услуги. Овие услуги лесно се интегрираат меѓу себе, но многу тешко, ако не и невозможно со слични алтернативи од други компании. Дополнително, дури и да станува збор за решение кое користи отворени протоколи, интеграцијата на различни услуги понудени од различни даватели на услуги би значела огромен трошок за мрежен сообраќај. Тенденцијата скапо да се наплаќа за појдовниот и дојдовниот мрежен

сообраќај кон/од трети мрежи, служи како дополнително охрабрување да се остане во рамките на дадената околина. Ваквото заклучување на корисниците и нивното ограничување само на користење затворени услуги, без можност за интероперабилност со трети инфраструктури, води кон рекурентна врска која само го зајакнува ефектот на централизација на услугите од една страна, без притоа давателите на услуги да имаат мотив да осигураат компатибилност меѓу своите портфолија.

Соодветно решение за разбивање на ваквиот негативен тренд е развојот на повеќенаменска архитектура која има унифицирачки карактер во однос на различните даватели на услуги. Со примената на стратегијата за слоевит дизајн, комплексноста на архитектурата се намалува, овозможувајќи точно дефинирање на надлежностите на секоја од главните компоненти. Непреченото меѓусебно поврзување на различни облак-раб пресметковни заедници воспоставени меѓу инфраструктурите на конкретни даватели на услуги, без разлика каде или под чија надлежност тие се наоѓаат, би било задача токму на еден од предвидените слоеви. Ваквиот слој би играл улога на обединувачки комуникациски интерфејс за корисниците на кои им е потребно брзо, ефикасно, но и финансиски пристапно решение за извршување на сопствените задачи. Обединувањето на повеќе пресметковни заедница во една целина води кон воспоставување на посакуваниот унифициран облак-раб екосистем.

Целта на ова поглавје е да се направи дизајн и валидација на повеќенаменска архитектура на безсерверски платформи за транспарентни пресметки во облак-раб екосистемот. Темелејќи се на резултатите од претходните поглавја, воспоставена е референтна архитектура на обединети пресметковни заедници претставени од широко дистрибуирани, повеќекориснички безсерверски инфраструктури кои се протегаат од облакот до работ на мрежата и обратно, овозможувајќи лесна проширливост. Дополнително, со воведување отворен каталог од апликации и алатки се овозможува олеснет пристап до целокупниот екосистем за крајните корисници. Главните придобивки од понуденото решение се:

- транспарентно извршување (од кориснички аспект) на безсерверски функции на работ од мрежата и на облакот, овозможувајќи високи перформанси, ниско доцнење, контрола врз податоците и оптимални инфраструктурни трошоци;
- поддршка за различни извршни околини со што се овозможува уште поголема слобода при избор на технологијата за имплементација на одредена функционалност; олеснета можност за прифаќање на безсерверската технологија, а со тоа и сите нејзини предности, од страна на постоечки (застарени) апликации;
- намалување на трошоците за мрежен сообраќај и оптоварувањето на мрежните врски со испраќање на кодот за иницијално процесирање до самите податоци на работ од мрежата, наместо податоците до кодот (во облак);
- елиминација на ефектот на заклучување, со примена исклучиво на отворени протоколи и софтвер со отворен код, гарантирајќи интероперабилност со различни безсерверски системи, дури и оние кои се надвор од претставениот унифициран екосистем (класични безсерверски инфраструктури во облак).

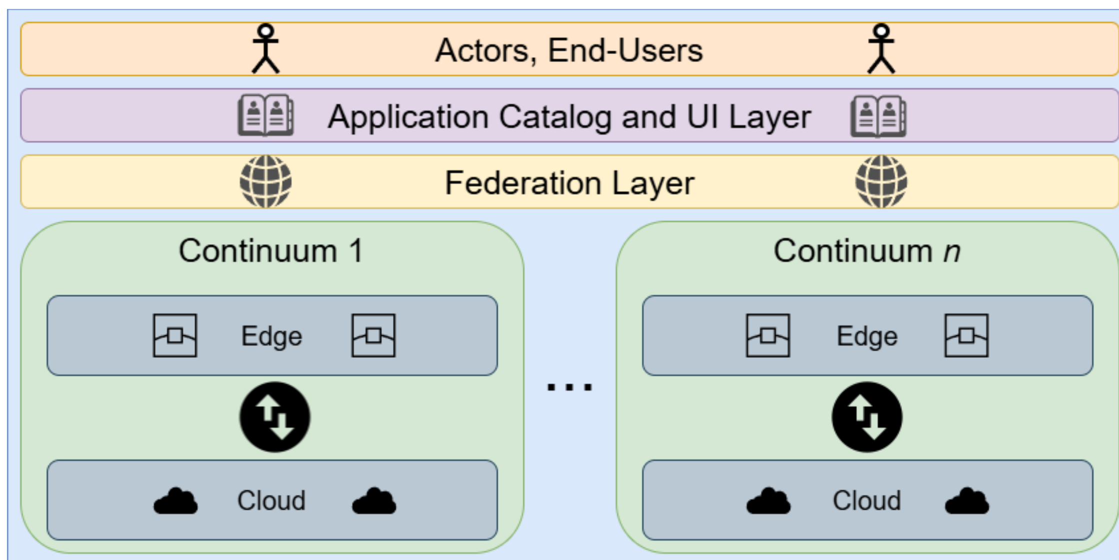
Досегашното истражување покажа потенцијал за потврдување на првата хипотеза на дисертацијата – *Со помош на последните научни достигнувања во релевантните области и преку нивно комбинирање во заедничка целина се овозможува имплементација на базична функционална платформа за пресметување која би ги надминала ограничувањата на досегашните алтернативни решенија.“* Сепак, конечното потврдување на хипотезата би било возможно единствено преку дефинирање на повеќенаменската архитектура на безсерверски платформи за

транспарентни пресметки во облак-раб екосистемот. Дополнително, валидацијата на предложената архитектура преку референтна имплементација во пракса би водело кон потврдување и на петтата хипотеза, т.е. *„Со комбинација на нови и постојни софтверски решенија со отворен код може да се изгради отворена платформа за пресметување која не само што ќе овозможи пресметување на различни локации користејќи различни извршни околини, туку и ќе промовира реискористување на постојното знаење преку отворен каталог на веќе дефинирани алатки и апликации.“*

7.1. Врската меѓу пресметковните заедници и облак-раб екосистемот

Во моментов, модерната пресметковна парадигма се заснова на централизирани даватели на услуги со широко портфолио на сервиси кои може лесно да се интегрираат меѓу себе на работ, во облакот или, пак, на двете локации. Иако поврзувањето сродни услуги од еден ист давател на услуги е едноставно, спарување услуги од посебни даватели на услуги е исклучително тешко. Користењето услуги од единствен давател на услуги, им овозможува на корисниците лесно да започнат со работа и да ја поставаат својата инфраструктура, но по цена на проблеми кои би можеле да се јават во иднина, во зависност од промената на барањата, потребите и типот на извршувани задачи. Во одредени случаи, фокусирањето на единствен централизиран извор на услугите може да ја доведе во прашање и одржливоста на сопственото решение, доколку давателот на услуги одлучи да ја ограничи достапноста на одредени пресметковни региони или целосно да ги укине.

Со дизајнот на обединувачка архитектура над пресметковните заедници составени од инфраструктури кои се протегаат како во облакот, така и на работ од мрежата се овозможува унифициран поглед и можности за управување со глобално дистрибуирани ресурси, без разлика каде тие моментално се наоѓаат. На овој начин би им се дозволило на сите заинтересирани страни, како: организации, отворени здруженија, па и индивидуи да покренат инфраструктура со високи перформанси која би ги задоволила побарувањата дури и на најкомплексните задачи. Остварувањето на оваа идеја побарува распоредување безсерверски функции низ работ од мрежата и низ облакот, врз основа на паметен систем за распоредување кој ги зема предвид индивидуалните потреби во однос на доцнење и пресметковна моќ на секоја задача. Овие потреби потоа се разгледуваат од аспект на сите пресметковни заедници кои се дел од унифицираниот поглед и би можеле да одговорат на нив. Земајќи ја предвид хиерархиската природа на системот, самите функции се транспарентно (од корисничка перспектива) дистрибуирани низ облак-раб заедниците воспоставени над приватни или јавни инфраструктури. Со користење брзо и редундантно мрежно поврзување меѓу различните заедници се гарантира висока достапност и можност за брзо префрлање на задачите на алтернативна инфраструктура во случај на испад. За подобра визуелизација, на слика 7.1 е даден графички приказ на хиерархиските врски меѓу различните компоненти кои ја градат предложената обединета инфраструктура.



Слика 7.1 Приказ на хиерархиските врски меѓу различните компоненти од архитектурата

7.2. Функционални барања за имплементација на облак-раб екосистемот

Поврзувањето на високо специфичните пресметковни заедници кои се протегаат низ работ и облакот е корисно за кое било корисничко сценарио со голем број на географски дистрибуирани корисници, на кои им е потребно како мало доцнење, така и голема пресметковна моќ. Безсерверската парадигма има потенцијал да ги задоволи ваквите побарувања преку своите можности за брзо и ефикасно скалирање и за работа без локална состојба, а наместо тоа да се користат BaaS услуги. Овие карактеристики се главниот предуслов за овозможување миграција на задачите во рамките на целиот облак-раб екосистем.

За остварување на оваа визија, во продолжение се претставени конкретните функционални побарувања за воспоставување унифициран пресметковен безсерверски екосистем кој се протега низ различни пресметковни заедници. Побарувањата се темелат врз сознанијата и резултатите од претходните поглавја, практичните ограничувања на модерните пресметковни решенија, како и претходно искуство со работа со сродни технологии. Иако секое побарување пред себе има бројка, ова подредување не е базирано на важност или каков било друг критериум – едноставно претставува начин за поедноставно навраќање на секое од барањата во дискусијата која следи во остатокот од поглавјето.

- Ф1: Географска транспарентност – Унифицираниот облак-раб екосистем може да биде проширен со дополнителни пресметковни ресурси без разлика на нивната географска локација, мрежа или матична инфраструктура. Овие ресурси може да бидат во форма на постоечки пресметковни заедници кои би биле вклучени во екосистемот како целина или, пак, поединечни уреди. На овој начин се овозможува поголема пристапност и отвореност на екосистемот, овозможувајќи им и на помали даватели на услуги (без постоечка пресметковна заедница), па дури и индивидуи да станат дел од него преку додавање на сопствените ресурси. За поголема флексибилност, треба да биде возможно и вклучување инфраструктури кои се наоѓаат во рестриктивни мрежи, каде што сообраќајот е агресивно филтриран. На овој начин се овозможува сегрегирање помеѓу процесот на провизионирање на инфраструктурата и воспоставување мрежно поврзување, дозволувајќи да се користат постојните мрежни врски без промени. Во зависност

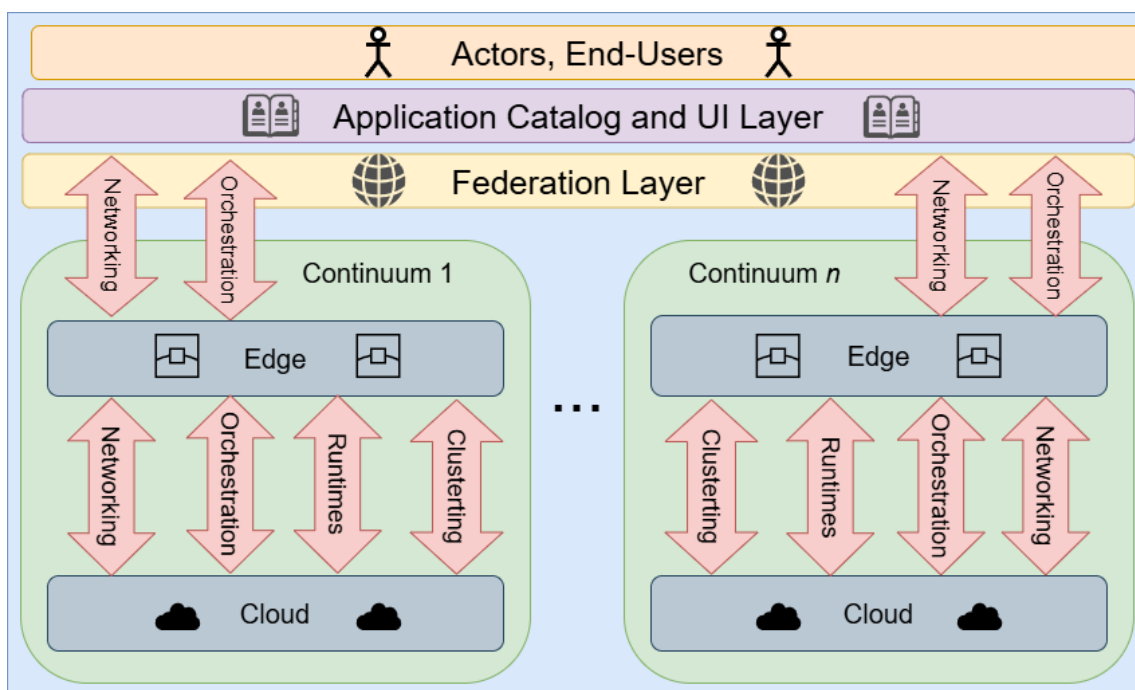
од перформансните побарувања и мрежните услови, треба да бидат поддржани повеќе топологии на поврзување кои ќе се протегаат низ облакот и работ од мрежата.

- Ф2: Споделување меѓу посебни чинители – Целосниот потенцијал на унифицираниот екосистем може да биде остварен исклучиво ако тој може да поддржи истовремена работа на повеќе независни крајни корисници (чинители) во даден момент. Треба да се има предвид дека функциите кои потекнуваат од нив може да не се проверени, а во некои случаи дури може и да се злонамерни, па потребни се мерки за остварување високо ниво на изолација. Ваквите мерки треба да бидат применети како врз мрежата, така и врз извршните околини каде што се извршуваат самите безсерверски функции.
- Ф3: Еластичност – Додавањето нов, но и елиминацијата на вишок пресметковен капацитет треба да биде едноставна задача, дозволувајќи ѝ на платформата во секој момент да одговори на тековните побарувања. Еластичност е потребна и при справување со непредвидени мрежни или пресметковни испади, гарантирајќи висока достапност преку распределување на активните задачи низ преостанатите инфраструктури дел од унифицираниот екосистем.
- Ф4: Транспарентност на извршна околина – унифицираниот екосистем треба да има поддршка за широка лепеза на извршни околини за покренување безсерверски функции, со можност за нејзино проширување со нови решенија во иднина. Како што покажаа претходните поглавја, ниту една од актуелните извршни околини не може да ги задоволи сите кориснички сценарија самостојно, соочувајќи се со проблеми како ладен старт или намалени пресметковни перформанси. Треба да се земат предвид повеќе извршни околини, стремејќи се да се овозможи меѓусебна компатибилност, со што би се дозволило реискористување дури и на постоечки безсерверски функции. Исто така, треба да биде поддржана миграција од една кон друга извршна околина, секако, под претпоставка дека биле направени неопходните промени врз изворниот код на самата функција.
- Ф5: Финансиска исплатливост – И покрај тоа што унифицираниот облак-раб екосистем не мора да се користи исклучиво за комерцијални цели, финансиската исплатливост се однесува и на трошоците за негово одржување, а не само на оние наметнати врз корисниците. Оттука, извршните околини за функциите треба да бидат ефикасни, елиминирајќи ја потребата од какви било ад-хок техники за одржување на пресметковните перформанси на задоволително ниво.
- Ф6: Леснотија на користење – Покренувањето безсерверски функции во рамките на унифицираниот екосистем треба да биде што е можно поедноставно од перспектива на крајните корисници, затскривајќи ги комплексните (и неважни) детали за инфраструктурата. Ова значи дека носењето одлуки во однос на скалирање, иницијално поставување и изолација на функциите треба да биде целосно препуштено на самата платформа. Севкупната архитектура на унифицираниот екосистем исто така треба да биде транспарентна за крајните корисници кои директно ги повикуваат функциите, без какви било разлики во корисничкото искуство споредено со традиционалните инфраструктури, освен подобрени перформанси. Интеракцијата со услуги понудени од унифицираниот екосистем не треба да побарува какви било дополнителни алатки или знаење.
- Ф7: Перформанси – Со примената на Ф4, се очекува да биде овозможена поддршка за повеќе различни извршни околини за покренување безсерверски функции. Сепак, различни околини имаат различни предности и недостатоци, па потребно е да се нагодат соодветни предодредени вредности кои автоматски би

биле применувани. Неопходно е точниот профил кој ќе ги моделира перформансите да биде грануларно применлив, на ниво на функција со што се гарантира дека перформансите ќе бидат одржувани на задоволително ниво.

7.3. Дизајн на архитектура за облак-раб екосистемот

Процесот на дизајнирање архитектура за обединување повеќе облак-раб заедници опфаќа аспекти од различни области и потребна е прецизна анализа на факторите неопходни за остварување на зацртаната цел. За подобра визуелизација на основните столбови кои би ги поддржувале седумте функционални барања дискутирани претходно, на слика 7.2 се прикажани врските и зависностите меѓу инволвираните компоненти. Идентификувани се четири фундаментални столба, каде што вмрежувањето и оркестрацијата се протегаат сè до слојот на обединувачки екосистем, додека, пак, извршните околини и деталите околу начинот на кластерирањето влегуваат во доменот на поединечните даватели на услуги.



Слика 7.2 Главни столбови во архитектурата за унифициран облак-раб екосистем

7.3.1. Вмрежување меѓу заедниците во екосистемот

Вмрежувањето е круцијален дел од севкупната архитектура, овозможувајќи меѓусебна комуникација помеѓу облакот и работ од мрежата како во рамките на една заедница, така и меѓу различните заедници дел од унифицираниот екосистем за транспарентни пресметки. Мрежното решение кое би се користело на ниво на унифициран екосистем дозволено е да биде различно од она што би се користело од страна на инфраструктурниот давател на услуги во рамките на сопствената облак-раб заедница. Мрежната имплементација задолжена за обединување на различностите во поврзувањето при поединечните заедници треба да нуди унифициран, географски транспарентен поглед врз севкупната инфраструктура, правејќи какви било разлики невидливи за крајните корисници. Решенијата применувани во рамките на овој столб неопходно е и да имаат поддршка за повеќе чинители, овозможувајќи различни корисници истовремено да ја користат истата инфраструктура за различни намени. При имплементирање на поддршката за повеќе чинители, треба да се земат предвид два

важни аспекта – овозможување контрола на пристап како на инфраструктурно, така и на апликациско ниво.

Поддршка за ACL на инфраструктурно ниво би овозможила попрецизна контрола во однос на тоа кои корисници може да ги користат кои заедници, како и да ја регулира комуникацијата меѓу самите заедници. Дополнително, би било неопходно дефинирање мрежни правила и на апликацискиот слој со моделот на најмали привилегии, ограничувајќи ја комуникацијата меѓу функции од различни корисници кои меѓусебно не си веруваат. На овој начин не само што се зголемува безбедноста на поединечните задачи извршувани врз инфраструктурата, туку се подобрува и самата нејзина заштита, ограничувајќи ја штетата од потенцијално компромитирање или извршување злонамерни задачи.

Не треба да се занемари фактот што целокупната еластичност на екосистемот овозможена со додавање или одземање заедници, до одредена мера зависи и од направените избори во стратегијата за вмрежување. Провизионирањето и депровизионирањето уреди во рамките на мрежата треба да биде што е можно полесно и побрзо, овозможувајќи мрежната топологија динамички да се адаптира во зависност од бројот на заедници што е потребно да бидат меѓусебно поврзани. Ваквата прилагодливост игра клучна улога и кога станува збор за високо ниво на достапност за секоја функција, овозможувајќи ѝ да премине од една заедница во друга при јавување на испад или недостаток на пресметковен капацитет, под претпоставка дека мрежното решение е доволно робусно да се прилагоди на новонастанатата ситуација.

Столбот за вмрежување ја овозможува имплементацијата на следниве функционални барања: Ф1 (географска транспарентност), Ф2 (споделување меѓу повеќе чинители), Ф3 (еластичност).

7.3.2. Оркестрација меѓу заедниците и во рамките на екосистемот

Поддршката за интелигентна оркестрација е значаен аспект кој наоѓа примена и во делот на унифицираниот екосистем и во делот на поединечните заедници. Како резултат на хиерархиската природа на предложеното решение, при инстанцирање на нова безсерверска функција, потребно е да се изврши нејзино распоредување во два чекора. Напредниот алгоритам за распределба на задачи прво би ја одбрал најсоодветната заедница со можност да ги задоволи барањата на функцијата од аспект на доцнење, перформанси и цена. Потоа оркестрациската компонента во рамките на самата таа заедница, свесна за целокупната топологија и достапноста на ресурсите во рамките на облакот и работ од мрежата, ќе ја одбере најоптималната локација.

Оркестраторите во рамките на заедниците треба да бидат свесни за различните извршни околини поддржани на секоја локација, споделувајќи ја оваа информација во редуцирана форма со оркестрациската компонента од екосистем слојот. Вака би се постигнала транспарентност во однос на конкретните извршни околини и би се олеснил процесот на подигнување нови функции од корисничка перспектива. Ова би подразбирало автоматски избор на најсоодветната извршна околина, земајќи ги предвид карактеристиките на функцијата и нејзината компатибилност со понудените можности. Вреди да се напомене дека функционалното барање за транспарентност во однос на извршните околини се однесува и на поддршка на повеќе компјутерски архитектури, овозможувајќи безсерверските платформи не само да користат традиционални x86 сервери, туку исто така и алтернативи како ARM или RISC-V [269], [270].

Кога станува збор за оркестрација, честопати се поставуваат и прашања во однос на оптимизација на цената, па ова може да биде и дополнителен фактор при носењето на крајната одлука каде ќе биде распределена одредена функција. Очекувано, не сите заедници во рамките на унифицираниот екосистем би нуделе исти услови за нивно користење. Во одредени случаи може да се јават разлики дури и при извршување на дадена функција во друга извршна околина во рамките на истата заедница. Одредени извршни околини имаат специфично однесување кога станува збор за: користењето пресметковни ресурси додека се неактивни, брзина на ладен старт, скалабилност, итн. Сите овие аспекти треба да бидат земени предвид при носењето на финалната одлука за распределба на функцијата. Ефикасноста на компјутерската архитектура, а со тоа и поддршката за конкретни инструкциски множества исто така може да игра улога.

Оркестрацискиот слој го поткрепува имплементирањето на следниве функционални барања: Ф4 (транспарентност на извршна околина), Ф5 (финансиска исплатливост), Ф6 (леснотија на користење), Ф7 (перформанси).

7.3.3. Кластерирање во рамките на заедниците

Инфраструктурата која се наоѓа низ различните локации од заедниците функционира како дистрибуиран систем составен од посебни јазли кои треба непречено да комуницираат меѓусебе. Во предложената архитектура, одговорноста за овозможување географска транспарентност за пресметковните ресурси е препуштена на столбот за кластерирање. Покрај иницијалното распоредување на задачите, во негова надлежност е и скалирањето како надолу, така и нагоре, грижејќи се секогаш да бидат задоволени перформансните побарувања на инстанцираните функции.

Имајќи предвид дека столбот за кластерирање е ограничен на ниво на различните заедници, може да се очекува дека интерните топологии во нив значајно би се разликувале. Одредени заедници може да одлучат поставување единствен кластер кој би имал јазли низ различните локации на работ и на мрежата. Ваквиот пристап овозможува поедноставено управување на целокупната заедница, без притоа да воведе дополнителни нивоа на апстракција. Достапен е еден единствен интерфејс за интеракција со инфраструктурата што го прави користењето поедноставно за крајните корисници. Карактеристично за ваквото поврзување е тоа што комуникацијата меѓу различните јазли (вклучително и онаа за координација на самиот кластер) се одвива низ јавни мрежни врски. Овој тип на топологија, каде што целата заедница е претставена како еден единствен кластер, во продолжение ќе се нарекува „топологија на широк кластер“. Широките кластери овозможуваат лесно скалирање на ресурсите, бидејќи мрежата е одделена од пресметковниот хардвер, овозможувајќи секој од овие два елемента да се гледа (и управува) како посебен ентитет. Сепак, внимание заслужува мрежното доцнење, бидејќи високата латентност може да влијае негативно врз интегритетот на целата инфраструктура и нејзините перформанси. При поставувањето широки кластери, групирањето на јазлите не треба да се заснова на нивната географска локација, туку треба да се земат предвид и врските меѓу автономните системи во кои тие се наоѓаат.

Од друга страна, пак, во рамките на заедницата може да се покренат поединечни кластери на секоја засебна локација. На овој начин уште во рамките на самата заедница би станувало збор за множество од повеќе кластери. Во ова сценарио различните кластери се независни едни од други и секој кластер има сопствена интерна управувачка рамнина. Сообраќајот во поглед на координација на поединечните кластери се задржува само во рамките на локалната мрежа, каде што и се сите јазли. Оваа топологија во продолжение ќе се именува како „топологија на тесен кластер“. Воспоставувањето тесни

кластери е препорачано во оние околини каде што има доволно пресметковни ресурси концентрирани на една иста локација и постоечката мрежна топологија овозможува поврзување на ниво 2 меѓу сите јазли.

Без разлика која топологија е одбрана, заедничко за двете е дека треба да биде понудено лесно додавање нови и бришење постоечки јазли, без потреба од следење комплексни процедури за провизионирање или реконфигурација од страна на техничко лице. Ваквата можност за брзо и автоматизирано скалирање би имала и позитивно влијание во ситуации на зголемено оптоварување, како што е случај при неочекуван пораст на добиени барања од страна на крајни корисници. Се разбира, сите детали поврзани со реорганизација на топологијата, направени на работ или во облакот, треба да се интерни во рамките на дадената пресметковна заедница, а транспарентни кон унифицираниот екосистем.

Столбот за кластерирање учествува во задоволување на следните функционални барања: Ф1 (географска транспарентност), Ф3 (еластичност), Ф7 (перформанси).

7.3.4. Поддршка за различни извршни околини во рамките на заедниците

Додавање поддршка за повеќе извршни околини на различни локации на работ од мрежата и во облакот, би осигурало дека унифицираниот екосистем е погоден за употреба за голем број различни кориснички сценарија. Како што беше покажано во минатите поглавја, различни извршни околини имаат различни перформансни карактеристики, но и влијаат врз тоа кои безсерверски функции на која локација може да бидат извршени. Одредени извршни околини може да бидат оптимизирани во насока на зголемена безбедност, брзина на извршување, компатибилност или финансиска економичност.

Безбедносните аспекти при извршување дадена функција, исто така зависни од нивото на изолација кое го пружа извршната околина, играат важна улога во реализација на можноста за користење на унифицираниот екосистем од повеќе чинители истовремено. Во оваа насока, столбот за извршни околини овозможува изолација на ниво на барање за оние функции означени како високо доверливи, гарантирајќи дека секоја функциска инстанца ќе биде извршена во сопствена околина, без споделување датотеки или процеси. На овој начин се спречуваат и несакани протекувања на тајни информации запишани во привремени датотеки или работна меморија. Ваквите барања за работа со повеќе чинители се тесно поврзани со концептот на цепкање (англ. slicing) особено релевантен во MEC литературата. Цепкањето овозможува лесно партиционирање на безсерверските околини понудени од пресметковните заедници, а со тоа и целокупниот унифициран екосистем, давајќи можност за негово користење од страна на комплетно независни корисници.

Свртувајќи го вниманието на финансиската исплатливост за момент, скалирањето до нула инстанци е круцијално за овој аспект, спречувајќи бесцелно користење пресметковни ресурси кога нема активна обработка на податоци. Ваквите стратегии не само што се корисни за крајните корисници (намалени трошоци), туку носат придобивки и за операторите на инфраструктурата, овозможувајќи им да хостираат повеќе функции со истиот хардвер.

Претходните поглавја покажаа дека сè уште не постои една единствена извршна околина која би ги задоволила сите побарувања, низ сите сценарија, самостојно. Нереално е да се очекува изнаоѓање златно решение, имајќи ги предвид техничките компромиси кои мора секогаш да се направат. Затоа, од особена важност е земањето предвид на конкретните

предности и недостатоци на секоја опција при изборот на извршната околина за одредена функција, како што беше потенцирано во делот посветен на оркестрацијата.

Столбот на извршни околини ја потпомага реализацијата не следниве функционални барања: Ф2 (споделување меѓу посебни чинители), Ф4 (транспарентност на извршна околина), Ф5 (финансиска исплатливост), Ф6 (леснотија на користење), Ф7 (перформанси).

7.4. Карактеристики на предложената архитектура

Една од главните карактеристики на предложената архитектура е поддршката за повеќе извршни околини, во зависност од потребите на конкретната функција која се разгледува. Ваквиот пристап овозможува едно унифицирано решение да обединува повеќе различни пресметковни заедници, со потенцијално дијаметрално различни перформансни карактеристики. Поддршката за WebAssembly како извршна околина го надминува проблемот на ладен старт, овозможувајќи напуштање на ад-хок закрпите кои неретко се во спротивност со фундаменталните принципи на безсерверската парадигма. Во текот на истражувањето, во поглавјата 4 и 5, беше покажано дека кога станува збор за комплексни задачи WebAssembly нуди поскромни пресметковни резултати споредено со традиционалните контејнери. Отсуството на совршена извршна околина која самата би ги задоволила сите побарувања оди во прилог на потребата од симбиоза меѓу решенијата денес. Ова е суштинската мотивација за воведување напредни функционалности за поддршка на повеќе од една извршна околина за безсерверски функции низ пресметковните заедници кои, пак, потоа стануваат дел од поголем пресметковен екосистем. WebAssembly е најпогодна стратегија кога станува збор за функции на кои не им е потребна висока пресметковна моќ, но кои се извршуваат многу често од страна на голем број корисници, па лесното скалирање и брзото стартување се од есенцијално значење. Од друга страна, пак, контејнерите се резервирани за задачи каде што најголем дел од времето се поминува во извршување комплексни пресметки и времето на стартување учествува со мал дел во вкупното време на извршување. Решението на оваа загатка не лежи само во изборот на точната локација во рамките на дадена заедница (облак наспрема раб) за дадена задача, бидејќи веќе се покажа дека ниту облакот не е имун на ефектите на ладниот старт. Најголемиот дел од решенијата за ладен старт, кои всушност се косат со парадигмата на безсерверско пресметување, како реискористување на една иста извршна околина за повеќе функции или однапред стартување на контејнерите пред да има потреба за тоа (подгревање), потекнуваат токму од облакот. Напротив, решението побарува осврнување кон природата на самата функција која се извршува и употребата на соодветната извршна околина за нејзино транспарентно извршување низ воспоставениот екосистем. Ова е и главната придобивка од унифицираниот екосистем, преку обединување на разнолики пресметковни заедници се раѓа можност за изнаоѓање конкретен извршувач кој ќе одговори на побарувањата како од аспект на перформанси, така и од аспект на географска локација и мрежно доцнење.

Со изготвување на референтна имплементација во пракса, се дава дополнителна тежина на претставената архитектура и емпириски се потврдува дека воспоставувањето на транспарентен екосистем за безсерверски пресметки низ различни заедница може да се реализира.

7.5. Референтна имплементација на пресметковен екосистем врз различни облак-раб заедници

Со помош на функционалните барања дискутирани во 7.2 и клучните столбови за нивна поткрепа опишани во 7.3, во продолжение се дискутира референтна имплементација на

претставената архитектура. Со ваквиот пристап дополнително се развива визијата за унифициран пресметковен екосистем преку издигнување на концептуалната замисла на сосема ново, поопипливо ниво, преку нудење конкретни решенија за негово воспоставување во пракса.

Во текстот кој следи ќе биде применет пристапот на објаснување од долу-нагоре на конкретни решенија наменети за секој од поединечните столбови, почнувајќи од слојот на пресметковните заедници и движејќи се нагоре кон слојот на унифицирачки екосистем.

7.5.1. Kubernetes како решение за кластерирање

Референтната имплементација користи Kubernetes за задоволување на сите потреби околу кластерирањето во рамките на поединечните пресметковни заедници. Иако во минатото Kubernetes беше првично контејнер оркестратор, со проширувањата и адаптациите дискутирани во претходните поглавја, станува возможно Kubernetes да биде надграден за поддршка на повеќе, различни, извршни околини. Ова е и едно од главните начела на унифицираниот пресметковен екоситем, имајќи предвид дека не може да се задоволат сите барања од една единствена извршна околина.

Во секој случај, треба да се има предвид дека Kubernetes особеностите се релевантни исклучиво за администраторите на различните заедници, па важно е крајните корисници да може да го користат екосистемот за покренување сопствени функции, без да бидат свесни за деталите околу поставеноста на пресметковната или мрежната инфраструктура. Примената на Kubernetes во рамките на заедниците е дополнително поддржана од фактот што најголемиот дел од приватните и јавни даватели на услуги веќе нудат првокласна поддршка за овој оркестратор. Сознанијата од претходните поглавја во однос на покренувањето Kubernetes кластери во облакот, но и на работ од мрежата врз уреди со ограничени пресметковни ресурси, како и на уреди со различни архитектури дополнително ја оправдуваат неговата примена во разнолики околини.

Префрлувајќи се на технички детали, Kubernetes кластери е возможно да се управуваат на најразлични начини, со специфични алатки [25]. Во зависност од желбите, потребите и експертизата, може да се оди од една во друга крајност – од комплетно рачно управување со познавање на сите ситни детали до целосно автоматизиран пристап, каде што решението за автоматизација прави и апстракција на инфраструктурата од аспект на администраторот. Различни даватели на услуги користат различни, најчесто затворени алатки за управување со кластерите покренати на нивната инфраструктура. Доколку се разгледува пристап кој е независен и од поглед на давателот на услугата и од хардверот на кој се поставува кластерот, препорачаниот начин за подигнување нови Kubernetes инстанци е со помош на K3s Kubernetes дистрибуцијата [124]. K3s, според резултатите од поглавјето 3, нуди одлични перформанси споредено со алтернативите, едноставен е за користење и поддржува различни компјутерски архитектури.

Алтернативно решение на ова поле е и Cluster API (CAPI) [271], алатка способна да покренува нови Kubernetes кластери, автоматизирано, на различни приватни и јави инфраструктури, користејќи ги нивните интерфејси за апликативни програми. Сепак, CAPI не е директно наменет за кластери кои би биле поставени на работ од мрежата, што го прави K3s преферираn избор кога станува збор за околини кои се протегаат и во облакот, но и на работ од мрежата.

Едноставното провизионирање како на master, така и на worker јазли со помош на K3s гарантира робусен и флексибилен механизам за скалирање. Worker јазлите кои и

всушност се задолжени за извршување на кориснички поставените функции можат да бидат додавани или елиминирани во кој било момент, без таквите активности да влијаат врз целокупната функционалност на кластерот. Внимание заслужува важното прашање како да се направи одлуката за користење широка или тесна топологија на кластерот. Пристапот со широка топологија, каде worker јазлите се поставени на различна инфраструктура од онаа на која се наоѓа контролната рамнина, треба да се употребува исклучиво доколку мрежните услови, вклучително и доцнењето, може да гарантираат такви дистрибуирани сценарија. Во продолжение не се дадени конкретни гранични вредности за тоа под кои услови е дозволиво да се користат широки кластери, бидејќи потребите, а и толеранциите, варираат од еден до друг случај. Сепак, важно е да биде потенцирано дека преголемото доцнење меѓу worker јазлите и контролната рамнина може да го направи кластерот нестабилен, а со тоа и да има негативно влијание на веќе поставените безсерверски функции. Во зависност од корисничкото сценарио, треба да бидат следени официјалните препораки околу максималното мрежно доцнење. Па така, во случај на MEC инфраструктури и 5G, препораката IMT-2020 ја поставува вредноста од 4ms како максимално доцнење кое треба да го искусат крајните корисници при ултра надежна комуникација со ниско доцнење (англ. Ultra-Reliable Low Latency Communications) [272]. Слични гранични вредности се публикувани од различни работни групи или тела за стандардизација релевантни во соодветните области.

7.5.2. Извршни околини низ унифицираниот екосистем

Задачата на слојот на унифициран екосистем е да понуди единствен интерфејс кој би ги скрил разликите во извршните околини користени од страна на различните пресметковни заедници. Денес, најпопуларната извршна околина за безсерверски функции, па и останати типови апликации во облакот, се контејнерите. Но, контејнерите страдаат од големи доцнења при ладен старт, што беше и докажано во третото поглавје како од аспект на еднојазлени, така и од аспект на повеќејазлени безсерверски платформи. За надминување на овој недостаток најчесто се применува нивно реискористување, намалувајќи ја изолацијата и елиминирајќи некои од најзначајните начела и придобивки од безсерверското пресметување. Со воведувањето повеќе, алтернативни извршни околини, ваквите проблеми би се надминале, овозможувајќи избор на најсоодветното техничко решение во зависност од побарувањата на функцијата. Можноста за извршување во WebAssembly околина е клучно подобрување кое би помогнало за надминување на некои од актуелните проблеми на ова поле. Прикажаната стратегија за интеграција меѓу Kubernetes и WebAssembly, преку проширување на Containerd е соодветна за примена не само во поединечни заедници, туку и на ниво на унифицираниот екосситем. Користењето на OCI форматот на слики за дистрибуција и на WASM артефакти, како што тоа е случај при размена на контејнерски слики, дополнително ги олеснува ваквите интеграциски аспекти, како што беше предложено во четвртото поглавје.

Истата стратегија на проширување на Containerd за поддршка на WASM опишана во петтото поглавје, „Оркестрација на WebAssembly безсерверски функции“, може да се примени за да се воведат дополнителни опции, како виртуелни машини со помош на Kubevirt [217], микро виртуелни машини со Kata околината [213], [216], но и останати WASM решенија, покрај Spin [28].

Од аспект на севкупниот екосистем, потребно е да се понуди рамка за извршни околини лесно проширлива со дополнителни решенија во иднина. Имајќи предвид дека промените во извршните околини би биле на ниво на различните заедници, овој процес треба да е апстрахиран од екосистемот, без да побарува промени во погорните слоеви.

Вакво високо ниво на апстракција сега е возможно со Kubernetes, бидејќи додавањето нови извршни околини не побарува директни измени во оркестрациското решение [29].

7.5.3. Оркестрација низ унифицираниот екосистем

Оркестрацискиот столб треба да врши поткрепа и на слојот на пресметковни заедниците и на слојот на унифицираниот екосистем, овозможувајќи централизирано управување со севкупната пресметковна инфраструктура и распределба на безсерверски функции низ различни локации и околини. До неодамна, во рамките на Kubernetes се развиваше официјална федерациска функционалност под закрила на проектот KubeFed. Но, со престанокот на развој на KubeFed, неговата улога ја преземаа помодерни решенија претставени од поширокаta Kubernetes заедница. Еден таков пример е Karmada [273], кој може да се смета и за наследник на KubeFed, поради сличноста во работењето. Karmada овозможува обединување повеќе Kubernetes кластери во една целина, без притоа да има потреба од специјални стратегии за поставување на управуваните кластери или специфична пресметковна инфраструктура. Единствениот предуслов за даден кластер да се придружи е неговата контролна рамнина да биде сертифицирана Kubernetes дистрибуција која ја имплементира целосната Kubernetes спецификација за интерфејсот за апликативни програми. Ова не претставува пречка, бидејќи сите даватели на услуги во облак, но и независни Kubernetes дистрибуции, го задоволуваат ваквото барање. Неопходните Karmada компоненти кои би ја претставувале централната контролна рамнина на унифицираниот екосистем е препорачано да бидат покренати врз посебен Kubernetes кластер кој ќе се користи исклучиво за таа намена. Ваквиот кластер на кој се наоѓа централизираната контролна рамнина најчесто се покренува во стратешки избран податочен центар во облакот кој нуди оптимално мрежно доцнење до останатите кластери. По неговото првично воспоставување, екосистемот може да се прошири со дополнителни кластери на еден од два начина (опишани подолу од перспектива на централната контролна рамнина):

- Режим на буткање (англ. push mode) – централната контролна рамнина е таа која се поврзува до API серверот на конкретниот кластер.
- Режим на влечење (англ. pull mode) – се инсталира посебен агент во конкретниот кластер со задача повремено да се поврзува до централизираната контролна рамнина за да ги синхронизира најновите информации.

Препорачаниот метод за регистрација нови кластери од страна на Karmada е режимот со буткање, имајќи предвид дека не побарува никакви дополнителни компоненти во самите кластери. Истиот е и поефикасен во поглед на искористување пресметковни ресурси, споредено со алтернативниот режим на влечење. Под претпоставка дека веќе постои мрежно поврзување низ сите локации, како што е опишано во шестото поглавје, регистрација во режим на буткање е соодветно решение за сите кластери кои би учествувале во екосистемот. Дополнително, со тунелирање на Kubernetes API сообраќајот низ VPN мрежа се елиминира и потребата од јавно публикување на Интернет на API портите, зголемувајќи ја целокупната безбедност.

Од аспект на перформансите, предложената архитектура се потпира на изборот на најсоодветната извршна околина со цел оптимизација на вкупното време на извршување и перформансите на функциите. WebAssembly се користи за често извршувани, но не и пресметковно комплексни функции, додека, пак, контејнерите се подобар избор во случај на посложени функции каде што се потребни високи перформанси. Се разбира, имајќи предвид дека во екосистемот би опстојувале поголем број на кластери, а и крајните корисници кои ги повикуваат функциите поставени на нив не би биле географски

статички, туку би можеле да мигрираат од една локација кон друга, треба да се обрне внимание и на прашањето околу распоредувањето на функциите.

Проблемот кој треба да се надмине од аспект на распоредувањето на безсерверските функции е како да се насочат дојдовните барања, испратени од крајните корисници, до најсоодветната инстанца на дадената функција. Најчесто (но не и секогаш) најсоодветната инстанца се претпоставува дека е онаа која се извршува на инфраструктура најблиску до корисникот. Таквото насочување до најблиската инстанца би осигурало оптимално мрежно доцнење, но не го зема предвид оптоварувањето на функциите, со што зголеменото време на извршување би можело да ги поништи заштедите од помалото доцнење. Овој дел ќе се осврне на распределбата на задачите, додека, пак, во 7.5.4. „Унифицирано вмрежување“ ќе се разгледуваат глобалните мрежни аспекти на предложеното решение.

Karmada има поддршка за напредни стратегии за распределба на контејнери, вклучително и толеранција за грешка низ повеќе кластери истовремено. При инстанцирањето на нов контејнер може да се наведат стриктни ограничувања кои би се зеле предвид при носењето одлука на кој јазол и во рамките на кој кластер да се извршува дадената задача. Од друга страна, кога станува збор за WebAssembly, таквите напредни стратегии за распределба на задачите и нивна миграција се непотребни, бидејќи функциите се покренуваат од нула при секое добиено барање и нема трошење непотребни ресурси кога тие се во мирување. Поради ова, соодветен пристап е да се регистрираат истите функции низ сите кластери дел од унифицираниот екосистем, со што тие би се извршувале исклучиво по потреба, при добиено барање. Пресметковните ресурси се употребуваат само во моментот на извршување, па какви било проблеми од аспект на преоптоварување може да се решат едноставно со паметни упатувачки одлуки применети врз дојдовните барања.

Дополнителна предност на безсерверското пресметување која може да се експлоатира при упатување на барањата, а би придонела и за намалување на севкупните трошоци, е фактот што функциите не се очекува да чуваат локална состојба. За која било информација што треба да биде зачувана и достапна од едно извршување до друго предвидено е да се користат BaaS услуги. Ова го прави носењето одлука кое барање каде да биде упатено многу пофлексибилно, бидејќи може да се изврши упатување кон која било достапна инстанца, без да се води грижа за лепливи сесии (англ. sticky sessions) и за тоа каде било извршено претходното барање од истиот корисник. Од аспект на финансиската исплатливост, ова би значело дека секогаш може да се извршуваат функциите кои се наоѓаат на најевтината инфраструктура во моментот, префрлувајќи се од еден екосистем во друг како што се менуваат цените, без притоа да се извршуваат комплексни миграциски процеси.

Анализата на сите компоненти во предложеното решение, почнувајќи од Spin, преку Containerd и завршувајќи со Kubernetes оркестраторот, покажува дека сите тие, без исклучок, имаат целосна поддршка како за x86, така и за ARM компјутерската архитектура. Со ова се овозможува креирање кластери каде што дел од јазлите би биле на една, а дел на друга архитектура. Ваквиот пристап дозволува и имплементација на понапредни стратегии за распределба на функциите кои може да ја земат предвид и архитектурата на достапните пресметковни јазли, преку користење на концептот на Kubernetes лабели, што е воедно и поддржано од Karmada како клучно решение за воспоставување на унифицираниот екосистем [274]. На овој начин се овозможува оркестрацискиот столб да ги задоволи побарувањата во однос на транспарентност на платформата и извршните околини.

7.5.4. Унифицирано вмрежување

Овозможувањето обединет поглед над сите мрежни ресурси достапни како во екосистемот, така и низ различните пресметковни заедници, се заснова на два клучни аспекта: додатоци за контејнерско вмрежување (англ. container network interface plugin, CNI) и виртуелни приватни мрежи. Додатоците за вмрежување контејнери овозможуваат поврзување во рамките на еден кластер меѓу сите јазли кои се дел од него, додека, пак, CNI помага во воспоставување врски меѓу оддалечени локации, воведувајќи географска транспарентност.

Како што беше дискутирано и претходно, CNI креира надмрежа која се користи во рамките на даден Kubernetes кластер [152]. Иако буквата „C" (англ. container) во кратенката стои за зборот „контејнерско", сепак, овие додатоци може да се адаптираат за работа и со други извршни околини, како WebAssembly. Во зависност од постоечката мрежна инфраструктура, ваквите додатоци може да работат во различни режими. Веројатно најпопуларниот и наједноставен режим е со користење енкапсулација [275]. За референтната имплементација, следејќи ги стапките на сите претходни истражувања и тестови, избран е Calico како CNI поради неговата зрелост, популарност, робусност и голем број функционалности. Една од важните можности кои стојат на располагање при употребата на Calico е дефинирањето NetworkPolicy објекти со кои се регулира дојдовниот и појдовниот сообраќај на ниво на функција. Дополнителна предност е можноста за директно користење енкапсулација, што го прави првичното поставување поедноставно и ја елиминира потребата од нагодување BGP сесии со постоечките упатувачи во инфраструктурата [276]. Воведувањето на BGP како предуслов за поставување Kubernetes кластери би било несоодветно барање за голем број инфраструктури на работ од мрежата, па наместо тоа соодветна замена е користењето VXLAN енкапсулација. Севкупна VXLAN енкапсулација на сообраќајот во кластерот исто така го решава проблемот кој се јавува кај голем број даватели на услуги поради престрогите (но оправдани) ограничувања на податочен слој. Имено, најчесто е оневозможено излегување сообраќај со различна MAC адреса од онаа која е доделена на единствениот виртуелен интерфејс на самата виртуелна машина од страна на хипервизорот, со цел да се спречи напад со лажирање MAC адреси. Сепак, без енкапсулација, сообраќајот генериран од pod-овите би имал MAC адреса на интерфејсот на pod-от, па соодветните пакети директно би биле отфрлени од заштитниот механизам. Со енкапсулацијата оригиналниот пакет е обвиткан во нов пакет чија изворна и дестинациска MAC адреса директно се совпаѓа со конкретни адреси на постоечки јазли (виртуелни машини).

Користењето на друг CNI кој не е Calico во рамките на даден екосистем не е никаков проблем и не ја нарушува компатибилноста меѓу различните инволвирани кластери. CNI однесувањето може да се апстрахира и управува преку Kubernetes интерфејсот за апликативни програми, кој со сигурност ќе биде ист низ сите кластери, без разлика дури и на тоа која Kubernetes дистрибуција била користена.

Од безбедносен аспект, треба да се обрне внимание при поставувањето надмрежи врз небезбедни подмрежи. VXLAN, како и слични енкапсулациски технологии, не нуди вграден механизам за шифрирање на енкапсулираните податоци. Сепак, овој проблем може да биде решен на елегантен начин со користење на VPN како подмрежа, па над VPN да се постави Calico. Резултатите од шестото поглавје, „Транспарентно поврзување низ облак-раб екосистемот", покажаа дека VPN решенијата за сестрано поврзување се доволно зрели за користење како Kubernetes подмрежи [26]. Во овој дел, избраното решение е Headscale поради леснотијата на користење, едноставната проширливост и

робусниот механизам за препраќање на сообраќајот во случај на рестриктивни мерки во мрежата [254].

Вреди да се напомене дека со помош на сестраното поврзување и поддршката за напредно дефинирање VPN во комбинација со техники за заобиколување NAT, се овозможува проширување на инфраструктурите на работ од мрежата со дополнителни пресметковни јазли, во приватна сопственост, кои доброволно би биле придружени од страна на заинтересирани корисници.

Од аспект на распределба на барањата до постоечките инстанци, столбот за вмрежување е должен да даде поддршка на оркестрациското решение преку ефикасно насочување на повиците, овозможувајќи географска транспарентност и еластичност. Референтната имплементација нуди две можности за реализација на оваа цел: користење на DNS за распоредување на товарот или anycast.

DNS како механизам за распоредување на товарот од дојдовните барања кон крајните функции беше претставен во поглавје 5, „Оркестрација на WebAssembly безсерверски функции", во контекст на MEC. Главната идеја е да се користи специфична инстанца на CoreDNS која работи во рамките на најблискиот Kubernetes кластер до корисниците како DNS разрешувач [78], [277], разрешувајќи ги DNS барањата во IP адресите на најсоодветните функции.

Алтернативен пристап кој исто така би овозможил распоредување на товарот на ефикасен начин е преку користење anycast [278]. Ваквото однесување може да биде имплементирано на два начина: или на ниво на CoreDNS инстанци или на ниво на безсерверски функции. Во првиот случај, CoreDNS инстанците низ заедниците на екосистемот се сите публикувани со единствена IP адреса споделена помеѓу нив, без разлика што се покренати на различни географски локации. Специфични правила за насочување низ мрежата осигуруваат дека DNS барањата се доставени до најблиската CoreDNS инстанца, процес кој е транспарентен за крајните корисници. Најблиската CoreDNS инстанца, пак, потоа, го разрешува барањето и како одговор ја враќа IP адресата на најсоодветната инстанца од побараната безсерверска функција. Иако ваквиот пристап наликува на оној опишан претходно во контекст на MEC, главната разлика е тоа што не е потребно промена на DNS разрешувачот на крајните уреди при нивно преминување од една зона во друга.

Вториот начин на користење anycast, на ниво на функција, подразбира публикување на безсерверските функции преку споделена load-balancer IP адреса, иста за сите кластери. На овој начин се елиминира DNS од равенката (може да се користи кој било разрешувач), а anycast ќе осигура дека дојдовните барања ќе бидат проследени до онаа load-balancer инстанца извршувана во најблискиот кластер. Потоа, load-balancer инстанцата ќе го предаде барањето на конкретната функција која била повикана.

7.6. Валидација на референтната имплементација

Со цел валидација дали дискутираните решенија во рамките на четирите столба навистина ги задоволуваат сите функционални барања во реални услови, покрената е унифицирана инфраструктура која се протега низ повеќе земји и континенти. Во рамките на сценариото се нагодени вкупно 5 Kubernetes кластери, некои од нив користејќи тесна, а некои широка топологија, дистрибуирани низ различни географски региони. Во табела 7.1 се прикажани повеќе детали за секој од кластерите.

Табела 7.1

Информации за користената инфраструктура

Ред. Број	Локација	Јазли	Топологија	Ограничувања[46]
1	С. Македонија	3	Тесна	Нема
2	САД	3	Тесна	Нема
3	С. Македонија	3	Тесна	NAT
4	Германија	3	Широка	Нема
5	С. Македонија	3	Широка	Без UDP

Кластерите 1, 2 и 3 се поставени користејќи тесна топологија, бидејќи сите јазли се наоѓаат во истиот автономен систем и географска локација, но не секогаш и во истата подмрежа. Од друга страна, пак, кластерите 4 и 5 ја користат широката топологија и јазлите се поставени во различни автономни системи, на различни локации, но во рамките на една иста земја. Со цел симулирање ограничено мрежно поврзување, кластерот 3 е покренат зад упатувач кој врши NAT, со што се добива можност за потврдување на способностите на Headscale за заобиколување на ваквата компликација. Дополнително, во инфраструктурата каде што се наоѓа кластерот 5 блокиран е каков било UDP сообраќај, на ниво на огнен ѕид, со што намерно се оневозможува воспоставување директна ВПМ врска меѓу јазлите. Единствениот начин за надминување на овој проблем е со користење на функционалноста за препраќање на сообраќајот низ Headscale DERP точка.

Во сите случаи Headscale е користен како VPN подмрежа над која се поставува Calico надмрежата, задоволувајќи го Ф1 барањето. За секој кластер се наведени експлицитни ACL, формирајќи сестрано поврзување исклучиво со останатите јазли кои се дел од него и ограничено поврзување до неопходните порти од контролната рамнина на екосистемот, покрената во рамките на кластер 1. Сите кластери се воспоставени користејќи ја K3s Kubernetes дистрибуцијата и се приклучени на екосистемот со помош на Karmada во режимот на буткање. Не сите Kubernetes јазли се додаваат во исто време, поради неопходните подготовки кои мораа да бидат извршени со цел обезбедување на ресурсите за широките кластери. Ваквите околности ја отвораат вратата за тестирање на скалабилноста на кластерите покренати со K3s, бидејќи јазлите се додавани во рамките на веќе функционални Kubernetes кластери. Со оваа операција, и практично се потврдува дека се задоволени барањата на Ф3.

Ф2 и Ф4 барањата се задоволени со имплементирање на поддршката за извршување Spin WebAssembly модули користејќи ја соодветната Containerd софтверска спојка [29]. На овој начин се согледува дека Karmada навистина е способна да распоредува WebAssembly безсерверски функции низ сите кластери, без потреба од каква било промена во нејзината постоечка логика за распределба на задачи. Вниманието е намерно насочено исклучиво на екосистем нивото при распределбата на задачите, без испитување на WebAssembly перформансите споредено со останатите решенија, бидејќи оваа тематика беше детално обработена во поглавјата 4, „Алтернативни извршни околини за безсерверски функции“, и 5, „Оркестрација на WebAssembly безсерверски функции“, [170], [178], [179], [180]. Се користи истото множество на безсерверски функции како и во споменатите поглавја со цел тестирање на функционалноста на Karmada. Повикувањето на функциите се одвива со користење на Spin HTTP методот, а за управување со состојбата се ставени во употреба трети BaaS решенија, со кои интеракцијата се одвива преку Spin HTTP клиентските библиотеки. По компајлирањето

[46] Ограничувања присутни во мрежата, доколку постојат.

на потребните функции во WASM модули, се креираат OCI слики од нула, употребувајќи неизменета верзија на Docker, по што тие веднаш се прикачуваат на јавни регистри за контејнерски слики. На овој начин директно се потврдува исполнетоста на Ф6.

За евалуација и на последните две функционални барања, Ф5 и Ф7, истото множество на WASM безсерверски функции се покренува на сите 4 кластери (кластерот 1, каде што се наоѓа контролната рамнина на екосистемот намерно е изоставен, со цел да се гарантираат неговите перформанси). Потоа, се пристапува кон тестирање на DNS методот за распоредување на товарот, дискутиран во погорните потсекции. За таа цел, секоја виртуелна машина во улога на клиент ја користи CoreDNS инстанцата на најблискиот Kubernetes кластер како рекурзивен разрешувач. Spin ја извршува соодветната WASM безсерверска функција од почеток при секој повик, нудејќи вистинско скалирање до нула инстанци, заедно со гарантирана изолација меѓу различни барања.

Информациите за софтверските верзии на сите компоненти користени при покренувањето на пример унифицираниот екосистем се дадени во табела 7.2.

Табела 7.2
Информации за околината за евалуација

Компонента	**Верзија**
Оперативен систем	Ubuntu 22.04 LTS
Containerd верзија	v1.5.16
Kubernetes верзија	v1.22.17
K3s верзија	v1.22.17+k3s1
Karmada верзија	v1.6.1
Spin верзија	v0.7
Tailscale верзија на клиент	v1.46.1
Headscale верзија	v0.21.0

7.7. Унифицираниот екосистем од аспект на крајните корисници

Користењето стандардизирани апликациски програмски интерфејси низ целокупниот екосистем овозможува интеграција со трети инфраструктурни апликации кои веќе поседуваат поддршка за Kubernetes кластери. Ова има значајни позитивни импликации врз функционалноста на екосистемот, а пример за тоа е и можноста за користење готови решенија за дефинирање каталози од апликации и безсерверски функции. Ваквите каталози промовираат споделување на знаењето и овозможуваат подлабоко ниво на соработка меѓу корисниците. Дополнително, се овозможува и полесно користење на достапната инфраструктура, преку веб базиран интерфејс, наместо тоа да се прави со специјализирани алатки кои најчесто треба да се извршуваат на командната линија.

Вреди да се потенцира дека овие каталози се овозможени благодарение на отвореноста на екосистемот, слоевитата архитектура и користењето стандардизирани интерфејси, како услов за неговата модуларност. Во пракса, фундаменталниот принцип на кои функционираат овие каталози е преку користењето веќе дефинирано API-то кое овозможува покренување апликации врз дадената инфраструктура, без разлика дали во позадина станува збор за еднојазлена, повеќејазлена платформа, пресметковна заедница или, пак, во овој случај и целосен екосистем од повеќе заедници. Имајќи предвид дека Kubernetes интерфејсот е унифицираниот интерфејс за комуникација како во рамките на пресметковните заедници, така и низ екосистемот во целост, овие решенија се директно компатибилни со екосистемот, без потреба од комплексни измени или ад-хок прилагодувања.

За формална потврда на флексибилноста на унифицираниот екосистем во пракса, во продолжение ќе се претстави еден ваков отворен каталог на апликации, nmaas. nmaas е избран за референтно решение во овој домен поради тоа што е целосно со отворен код, нуди лесна можност за додавање нови апликации во каталогот и има едноставна софтверска архитектура која овозможува лесно проширување со дополнителни функционалности по потреба.

nmaas [34], [279] е софтверско решение развиено во рамките на европскиот GÉANT проект [280] кое овозможува лесно покренување апликации на ниво на еден Kubernetes кластер, истиот кластер во кој е инсталиран и самиот nmaas. Нуди каталог на достапни апликации кои може да бидат покренати и нагодени со неколку клика. Во позадина, секоја апликација е претставена со Helm карта која го опишува начинот и контејнерските слики што треба да бидат користени за нејзино инстанцирање.

nmaas е сочинет од три посебни компоненти кои вршат тесна интеракција меѓусебе со цел да ја реализираат целокупната функционалност – nmaas Platform, nmaas Portal и nmaas Janitor. nmaas Platform е задницата на решението и нуди програмски интерфејс за инстанцирање апликации, кој, пак, е користен од страна на предницата, nmaas Portal. nmaas Janitor е одговорен за повеќе типови на позадински задачи, вклучително и бришење на стари и веќе некористени податочни волумени, манифести, но и за синхронизација на какви било промени направени во Git репозиториумите кои ја чуваат конфигурацијата за одредена инстанца. Со помош на GitOps парадигмата која се потпира на поддршката за управување со Git репозиториуми се овозможува олеснето нагодување на апликациите и безсерверските функции по нивното инстанцирање [281]. На овој начин може да се врши директна измена на различни датотеки, вклучително и на конфигурациски датотеки. Секоја направена промена директно се синхронизира во сите моментално активни, но и идни инстанци (при процесот на хоризонтално скалирање). На овој начин се овозможува детална евиденција и историја за сите направени промени.

Интеграцијата на nmaas со унифицираниот екосистем се прави со извршување два релативно едноставни чекора, без потреба од промени во самиот изворен код на решението. Првиот чекор е покренување на nmaas во рамките на централната контролна рамнина, каде што се наоѓа и Karmada. На овој начин се добива можност за унифицирано покренување апликации врз сите пресметковни заедници, а со тоа и екосистемот во целина. Вториот чекор е прилагодувањето на Helm картите за постојните апликации и/или креирање сосема нови карти кои инстанцираат различни безсерверски функции, со цел да се користат Kubernetes шаблоните воведени од страна Karmada. Имајќи предвид дека и WebAssembly интеграцијата всушност е видлива на ниво на Kubernetes шаблони (преку WasmApp дефиницијата за посебен ресурс), на овој начин nmaas би се проширил да работи и со WASM како извршна околина.

Благодарение на слоевитиот пристап, останати аспекти како распределба на функциите, нивно скалирање и миграција при испад се под надлежност на главните столбови од унифицираниот екосистем.

7.8. Дискусија

Систематскиот преглед на досегашните научни публикации од полето на безсерверското пресметување со фокус на IoT и работ од мрежата откри слабости во досегашните имплементации и платформи. Постојните извршни околини, развиени во ерата на пресметувањето на облак, и покрај направените оптимизации, сè уште не може да одговорат на техничките побарувања кога станува збор за безсерверско пресметување на работ од мрежата. Ваквата ситуација ја роди идејата за истражување во насока на

надминување на идентификуваните проблеми. Уште од самиот почеток, целта беше да се претстави решение лесно за користење, како од перспектива на управувачите на инфраструктурата, така и од крајните корисници кои би покренувале функции. Дополнително, од огромна важност е да се избегне создавање затворен екосистем, кој не би бил компатибилен со останати сродни и отворени решенија достапни денес.

Ограничувањата на постојните решенија кога станува збор за извршување функции на работ од мрежата се и формално опишани преку анализата како на еднојазлени, така и на повеќејазлени безсерверски платформи во третото поглавје, „Анализа на постојни безсерверски платформи“. Доцнењето при ладен старт е постојан предизвик, кој не е надминат од ниту едно постојно решение и уште повеќе доаѓа до израз при извршувањето на работ од мрежата, поради поограничените пресметковни ресурси. Евалуацијата на Kubernetes дистрибуции како основа за повеќејазлени безсерверски платформи, утврди дека оние решенија оптимизирани за работ од мрежата навистина имаат помали ресурсни побарувања. K3s како еден од претставниците на ваквите дистрибуции покажува најдобри перформанси, мали ресурсни побарувања, но и леснотија при користење и скалирање. Интерпретацијата на добиените резултати во поглед на ладниот старт оди во насока на тоа дека проблемот не е во оркестрациското решение, туку во извршните околини кои се користат за функциите. Влечејќи инспирација од ваквото сознание, во четвртото поглавје се претставени алтернативни извршни околини за извршување на безсерверски функции, со цел намалување на ефектот од ладен старт. Спроведеното истражување потврдува дека WebAssembly може да се трансформира во технологија на серверска страна и да понуди драстично подобрено време на иницијално стартување. Ова се отсликува со времиња на ладен старт кои се под 1 секунда [28], [29], што е особено значајно постигнување, споредено со времињата кои се движат во рангот на неколку секунди при користењето контејнери [25]. Сепак, имајќи предвид дека станува збор за нова технологија која досега не се има користено за покренување безсерверски функции, нејзината примена во реалниот свет е условена од изнаоѓање скалабилен начин за следење на животниот циклус на голем број вакви функции одеднаш. Дополнително, употребата на WebAssembly не е универзално решение и неговата употреба нема за цел да ги отфрли останатите извршни околини. Напротив, потребна е меѓусебна интеграција, имајќи предвид дека WebAssembly во моментов покажува полоши сурови пресметковни перформанси споредено со алтернативите како виртуелни машини и контејнери. Всушност, потребно е робусно решение за заедничка оркестрација на безсерверски функции кои го користат WebAssembly како извршна околина, но и на контејнери и виртуелни машини. Токму ова е и во фокусот на петтото поглавје, „Оркестрација на WebAssembly безсерверски функции“. Претставеното решение за првпат овозможува оркестрација на WebAssembly модули од страна на Kubernetes оркестраторот, притоа нудејќи драстично намален ладен старт (од редот на неколку величини) и вистинско скалирање до 0 инстанци. Ваквото целосно исклучување на функциите гарантира дека не се трошат никакви ресурси кога тие не се во активна употреба. Според првичната замисла, можноста за извршување на функциите во контејнеризирана околина останува непроменета.

За реализација на крајната визија која подразбира обединување на повеќе безсерверски платформи во унифициран екосистем за транспарентни пресметки, потребно е изнаоѓање начин за едноставно, брзо и сигурно вмрежување на поединечните учесници, безсерверските платформи. Шестото поглавје, „Транспарентно мрежно поврзување низ облак-раб екосистемот“ формално ги претставува потенцијалните начини за реализација на ваквото поврзување. Ефикасноста на достапните решенија е висока и тие нудат брзини споредливи со оние кога не се врши дополнителна енкапсулација во VPN тунели.

Можноста за напредно дефинирање безбедносни правила и ACL овозможуваат истовременото, безбедно користење на инфраструктурата од повеќе чинители. На овој начин се врши целосна апстракција на физичкото поврзување на различните локации, нудејќи унифициран поглед, овозможувајќи лесно вмрежување на различни пресметковни инфраструктури во единствен, унифициран, пресметковен екосистем, без разлика на нивната географска поставеност.

Со овој пристап, формално беа евалуирани главните столбови врз кои може да се дизајнира и имплементира повеќенаменска безсерверска платформа за транспарентни пресметки, а воедно и воспоставувањето на посакуваниот екосистем. Токму во ова поглавје, поглавјето 7, постепено се претставува нејзината архитектура, изведена преку 7 функционални барања како појдовна точка. Различните контролни рамнини заедно со управуваните јазли на секоја географска локација се обединети во една заедничка логичка групација со помош на решенијата за сестрано поврзување. Целосно новиот слој претставен од унифицираниот екосистем, издигнат над индивидуалните пресметковни заедници, го гарантира централизираното управување кое е независно од конкретните инфраструктури или даватели на услуги. Оттука следува дека со комбинација на нови и постојни софтверски решенија со отворен код може да се изготви унифициран и транспарентен екосистем кој не само што ќе овозможи пресметување на различни локации користејќи различни извршни околини, туку и ќе промовира реискористување на постојното знаење преку отворени каталози на веќе дефинирани алатки и апликации, без каков било ефект на заклучување. Со ова и формално се потврдува петтата хипотеза на дисертацијата.

8. ЗАКЛУЧОК

Безсерверското пресметување е нова пресметковна парадигма чиишто придобивки доведоа до вртоглав пораст во нејзината популарност и го привлекоа вниманието како на академската фела, така и на индустријата. Имајќи го предвид големиот број на засегнати чинители на ова поле, целта на дисертацијата е да се даде одговор на отворените прашања преку анализа и дизајн на повеќенаменска безсерверска платформа за транспарентни пресметки во облак-раб екосистемот, со која се отклучуваат нови можности и сценарија преку приближување на безсерверското пресметување до уште поголема група на корисници. Во насока на ова, поставени се пет хипотези кои се осврнуваат на создавање повеќенаменска архитектура на безсерверски платформи за транспарентни пресметки во облак–раб екосистемот. Ваквата архитектура, независна од поединечни даватели на услуги, има обединувачка улога, овозможувајќи унифицирање повеќе различи инфраструктури, преку нудење единствен интерфејс за комуникација, криејќи ги грубите разлики на подолните нивоа.

Почетокот на реализацијата на зацртаната цел е преку систематски преглед на литературата, во кој се анализираат трудови поврзани со безсерверското пресметување во IoT контекст, со примена на работ од мрежата. Со пребарување низ ризници на трудови, по прелиминарната селекција, идентификувани се вкупно 64 релевантни труда на оваа тема. Опсежната анализа на нивната содржина овозможи дефинирање 8 уникатни области од интерес: (i) имплементација на апликации; (ii) ефикасност; (iii) распределба; (iv) тестирање; (v) имплементација на платформи; (vi) континууми; (vii) безбедност, интеграција, правила; (viii) софтвер со отворен код. Користејќи ги овие категории за класификација на трудовите, се забележува голем интерес за примена на безсерверското пресметување не само на традиционалниот начин, во облакот, туку и на работ од мрежата.

Доколку треба да се изведе единечен заклучок од сите анализирани трудови, тоа е дека интернетот на нештата е полето во кое безсерверското пресметување има потенцијал да ја најде својата најширока примена, поради добрата прилагоденост што ја покажува кон пресметковните побарувања на задачите засновани на настани. Сепак, за реализирање на овој потенцијал, потребно е надминување на отворените прашања на ова поле. Едно од нив е сеприсутно и во анализираните трудови, а тоа е (не)ефикасноста на извршните околини за безсерверски функции и големото време потребно за ладно стартување. Ова е особено проблематично при често повикување една иста функција, каде што за одговарање на барањата потребно е да се покренат поголем број инстанци во исто време. Како потенцијални алтернативи на контејнерите, кои се всушност и моментално најпопуларната извршна околина, се спомнуваат микро виртуелните машини, уникернелите, но и WebAssembly. Постоечките истражувања и литература немаат понудено зрели и конкретни имплементации на уникернели. Нивниот развој сѐ уште е во првична фаза, несоодветна за користење во реални сценарија. Од друга страна, пак, иако WebAssembly наоѓа широка примена во доменот на веб апликациите, потребно е дополнително истражување и прилагодување кога станува збор за негово користење на серверска страна, како што е случајот при извршувањето безсерверски функции.

Во иднина, како резултат на очекуваното зголемување на бројот на IoT уреди за домашна и индустриска употреба, се очекува уште поголемо внимание за аспекти поврзани со безбедноста и приватноста. Ваквиот фокус е неопходен со цел да се обезбедат осетливите и потенцијално тајни информации кои се генерирани од крајните уреди, а се предмет на обработка од страна на безсерверските функции. Во оваа насока, идентификувана е и потребата од строга изолација при извршување, потпомогната од

новите технологии како на хардверско, така и на софтверско ниво. Кон оваа цел може да придонесе и самото пресметување на работ од мрежата, бидејќи покрај намалено доцнење, овозможува и првична обработка на осетливите податоци и нивна анонимизација пред нивно понатамошно складирање во облакот.

Кога станува збор за безсерверски платформи, една од можните класификации е во однос на тоа на колку јазли истовремено функциите се распределуваат и извршуваат, па се разликуваат еднојазлени и повеќејазлени решенија. Навраќајќи се на најпопуларната извршна околина – контејнерите, со цел да се утврдат нејзините перформанси како во поскромни средини, со ограничени ресурси, така и при работа на поголеми инфраструктури, направена е анализа врз еднојазлени и повеќејазлени безсерверски платформи. Со користење постоечко множество на безсерверски тестови, адаптирани за работа на работ од мрежата, најпрво е извршена споредба на перформансите на три различни еднојазлени безсерверски решенија. Преку истовременото вклучување на комерцијални и отворени платформи, се пополнува празнината присутна во постоечката литература, каде што трудовите се фокусираат исклучиво или на едниот или на другиот тип платформи, без компаративна анализа меѓу нив. Резултатите од тестирањата покажуваат дека перформансите се слични кога станува збор за сериско извршување на функциите, со еден повик во даден момент, но различни при воведување паралелизам. Разликите при паралелно извршување се јавуваат како резултат на различните стратегии за скалирање кои ги применуваат платформите. Во овој дел, беа забележани три различни стратегии: автоматско скалирање, каде што самата платформа се грижи за сите одлуки; рачно скалирање за кое одговорен е оној кој ја поставил функцијата; без директна можност за скалирање, каде што единствената опција е воведување паралелизам внатре во програмската логика на функцијата, со имплементација на нишки или повеќе процеси.

Ако се земат севкупните резултати добиени од анализата на еднојазлените платформи, комерцијалните решенија како Greengrass и Azure IoT Hub имаат предност во однос на целосниот екосистем од поддржани алатки и лесната меѓусебна интеграција. Конкретно, Greengrass покажува и најдобри резултати во тестовите за брзина на влезно/излезни операции. Од друга страна, пак, отворените платформи како FaasD, се одликуваат со екстремна прилагодливост и можности за нивна адаптација на сценарија кои и не биле иницијално предвидени.

Спроведувајќи ја истата анализа, со истото множество на тестови, но од призмата на повеќејазлените платформи, добиените резултати раскажуваат слична приказна. Како оркестратор кој би ги распределувал функциите на различните јазли, учесници во кластерот, избран е популарниот Kubernetes. Дополнително, со цел да се добие што е можна поголема релевантност на резултатите, како безсерверско решение се користи OpenFaaS, проширена верзија на FaasD кој е вклучен во тестовите на еднојазлзлени платформи. Тестирањето на три различни Kubernetes дистрибуции: K3s, MicroK8s и Kubespray, покажува дека упростувањето спроведено за K3s и MicroK8s се пресликува и во побрзо стартување на функциите и намален бавен старт на соодветните контејнери. Сепак и во овој случај, како и во претходниот, ладниот старт се мери во опсегот од неколку секунди.

Земени во целина, резултатите во сите тест случаи и емпириски ги потврдуваат отворените проблеми присутни во сферата на безсерверското пресметување како: долг ладен старт, нестандардизирани интерфејси за апликативни програми и проблематично справување со големи функции како резултат на многу програмски зависности. Со ова станува јасно дека е потребно разгледување алтернативни извршни околини, покрај дополнителното оптимизирање на контејнерите. Вакви нови решенија, кои уште од

самиот почеток ќе бидат дизајнирани за извршување на уреди со ограничени ресурси и брзо стартување, имаат голем потенцијал за конечно решавање на горливите проблеми. Овој пристап ја елиминира потребата од понатамошни дополнителни оптимизации врз извршни околини иницијално развивани за сосема друга намена и инфраструктура, а не за безсерверско извршување на работ од мрежата.

WebAssembly е ветувачка технологија која со дополнителен развој може да ги реши проблемите околу долгиот ладен старт и изолацијата меѓу различни барања, зголемувајќи ја безбедноста. Со неодамнешното претставување на WebAssembly моделот со компоненти и WebAssembly системскиот интерфејс, се отвора вратата за користење на оваа технологија не само на клиентската страна, како што беше случај досега, туку и на серверската. Во рамките на дисертацијата направена е анализа на две постоечки решенија на ова поле: WasmCloud и Spin. Во двата случаја станува збор за безсерверски рамки кои во позадина користат WebAssembly извршна околина за покренување безсерверските функции. Сепак, нивното функционирање на ниско ниво покажува големи меѓусебни разлики, од архитектонската поставеност, преку начинот на извршување, до формат за дистрибуција на самите функции. Заедничко за двата случаја е тоа што не постои едноставен начин за интеграција со постоечки извршни околини или безсерверски платформи, побарувајќи воспоставување целосно нови инфраструктури, нешто што, сѐ разбира, би било пожелно да се избегне во пракса.

Гледајќи на неможноста за интеграција на WebAssembly со останати извршни околини како понатамошна инспирација, евалуирани се начини за спарување на WebAssembly со интерфејсите за апликативни програми и интерфејсите на командна линија на постоечки контејнеризациски софтвери. За оваа намена е креирано ново множество на безсерверски тестови, адаптации на оние претходните, но овојпат напишани во програмски јазици компатибилни со WASM – Rust и Go. Комбинацијата од микро тестови и реални сценарија овозможува релевантна споредба меѓу доцнењето при ладен старт и севкупното време на извршување за секоја функција при нејзината работа во WebAssembly околина, наспрема контејнерска околина. Резултатите покажуваат дека WebAssembly извршните околини при извршување однапред компајлирани модули, имаат подобри перформанси во 10 од вкупно 13 теста, споредено со контејнеризираните верзии на истите функции. Но, кога станува збор за покомплексни задачи кои побаруваат поголема процесирачка моќ, како на пример множење матрици или обработка на слики, контејнерите издвојуваат предност и покрај подолгиот ладен старт. Во иднина, интересно би било да се види како на сегашните резултати ќе влијае воведувањето поддршка за повеќе програмски нишки во WASM, функционалност која во моментов е во фаза на изработка. Конкретизирајќи ги добиените резултати, Wasmtime извршната околина покажува најдобри времиња во 8 од 13 теста, споредено со останатите две разгледувани WASM решенија: WasmEdge и Wasmer. По него следи WasmEdge со најдобри времиња за 4 функции и WasmEdge со 1.

Со успешната интеграција на WebAssembly со постоечка контејнеризациска извршна околина, се покажува дека овие две решенија може да работат рамо-до-рамо. Барем во моментов, тие треба да се гледаат во светлото на сојузништво, каде што нивната врска е симбиотска, па имајќи ги предвид нивните предности и слабости сѐ уште не може да стане збор за целосно замена на едното со друго решение. Сепак, ваквото меѓусебно поврзување ги отвора вратите за користење постоечки и докажани решенија од полето на контејнерите, за истовремено управување и со WASM модули. Прашањето за оркестрација на WebAssembly безсерверски функции е сѐ уште отворено, а одговорот може да лежи токму во една таква адаптација на постоечки контејнерски оркестрациски

систем, штом интерфејсите за апликативни програми со претставеното решение се унифицираат и за покренување контејнери и за WASM модули.

Во оваа насока, следниот чекор води кон додавање WebAssembly поддршка на Kubernetes оркестраторот, овозможувајќи му истовремено да учествува во покренувањето на безсерверски функции кои се извршуваат во форма на контејнери и WASM модули. Искористувајќи го стекнатото искуство од компаративната анализа на перформансите на различните WASM околини, како основа се користи онаа со најдобри перформанси, Wasmtime. Со користење посебна софтверска спојка дизајнирана за интеграција на WebAssembly извршната околина со Containerd контејнерското решение употребувано од страна на Kubernetes, но и со развој на нов Kubernetes оператор за подигнување WASM безсерверски функции, се демонстрира изводливоста на ова решение во пракса. Најважно од сè е што не се прават никакви директни промени во API-то на самиот Kubernetes, задржувајќи директна компатибилност со целата постоечка свита на Kubernetes алатки. Резултатите од спроведеното тестирање на перформансите покажуваат дека WASM модулите имаат барем 2 пати побрзо време на стартување и во Kubernetes средина, споредено со традиционалните контејнери, дури и во случаи кога контејнерските слики се веќе преземени и локално достапни.

Опишаното проширување на Kubernetes за извршување и WebAssembly модули може да се спроведе врз кој било постоечки или нов Kubernetes кластер. Но, сепак, за да се изготви решение кое истовремено би можело да функционира на повеќе од една локација и да се протега низ облакот и работ, потребно е да се дизајнира робусна стратегија за меѓусебно мрежно поврзување.

Воведувањето на Wireguard како соодветен протокол за дефинирање виртуелни приватни мрежи повторно го оживеа ова истражувачко поле, особено во насока на дефинирање решенија за сестрано поврзување, каде што секој јазол во мрежата има директна виртуелна врска со секој останат учесник. На овој начин се овозможува најкратка комуникациска патека, помало доцнење, елиминација на централни податочни јазли како тесни грла, но и повисоко ниво на безбедност. Воведувањето централна контролна рамнина која не е дел од податочната патека, туку учествува исклучиво во координацијата и размената на клучеви меѓу јазлите исто така може да придонесе кон напуштање на потребата од рачна конфигурација при хоризонтално скалирање на инфраструктурата.

Со цел евалуација на моменталните VPN решенија кои нудат сестрано поврзување преку централна контролна рамнина, дефинирани се строги критериуми задоволени од три софтвери: Headscale, Netbird и ZeroTier. Опсежната анализа на нивните перформанси и дополнителни функционалности се фокусира на повеќе аспекти: работа во идеални мрежни услови, работа во мрежни услови со пречки од типот на загуба на пакети или прекумерно доцнење и севкупно корисничко искуство. Сите тестови се изведуваат во контекст на реалното сценарио за кое би се употребувале решенијата, т.е. како подмрежа врз која би била воспоставена Kubernetes надмрежа за меѓусебна комуникација на Kubernetes јазлите. Резултатите покажуваат дека Netbird има најдобри сурови перформанси кога станува збор за идеални мрежни услови, со брзини кои се споредливи дури и со оние кои се постигнуваат без никакво користење на VPN решение. Но, кога станува збор за порестриктивни околини, Netbird исклучиво со своето UDP тунелирање нуди ограничено справување со агресивни политики на огнените ѕидови или NAT. Headscale и ZeroTier не само што имаат поддршка за тунелирање на VPN сообраќајот низ TCP, маскирајќи го, туку се справуваат подобро и при постоењето поголеми доцнења во мрежата. Заедничко за сите решенија, а воедно и еден од предусловите за вклучување во

анализата, е тоа што се со отворен код и може да се поставaт на сопствена инфраструктура без никакви дополнителни зависности. Сепак, начинот на кој ова се постигнува е различен и со варијабилна тежина. Headscale го нуди најдоброто корисничко искуство и најробусното заобиколување и справување со мрежни препреки, но помала сурова брзина од Netbird поради Wireguard имплементацијата која работи во корисничкиот домен на оперативниот систем. Но, од друга страна, пак, благодарејќи на ваквата одлука Headscale овозможува дефинирање напредни листи за контрола на пристапот, со што се овозможува регулирано сестрано поврзување и користење една иста Headscale инстанца за повеќе изолирани, оддалечени инфраструктури.

Робусно решение за сестрано поврзување овозможува непречена и високо безбедна врска меѓу различни географски дистрибуирани јазли на еден ист Kubernetes кластер, поставени на работ од мрежата или во облакот. За координација меѓу повеќе Kubernetes кластери, дел од посебни екосистеми, потребно е воведување дополнителна логика во улога на унифицирачка контролна рамнина, управувајќи со поединечните контролни рамнини на придружените кластери.

Визијата за повеќенаменска архитектура на безсерверски платформи во облак-раб екосистемот вклучува обединување на различни пресметковни заедници претставени од инфраструктури под контрола на независни даватели на услуги. Овие заедници се протегаат низ работ на мрежата и во облакот и се обединети во една целина. Со нудење единствен интерфејс за комуникација со целосниот екосистем сочинет од различните пресметковни заедници, се цели кон овозможување подобро корисничко искуство, намалување на зависноста од конкретни трети даватели на услуги, а притоа овозможувајќи и поголема еконимичност и повисоки перформанси.

Преку изведување 7 функционални барања, влечејќи поука од направените истражувања и добиени резултати, се дефинираа неопходните аспекти за имплементирање унифициран екосистем над повеќе облак-раб пресметковни заедници. Архитектурата се темели на 4 основни столба: вмрежување, оркестрација, кластерирање и извршни околини. Користејќи ги резултатите и решенијата претставени како дел од претходните истражувања опишани во оваа дисертација, дизајнирана е повеќенаменска архитектура која ги задоволува изнесените потреби. Архитектурата е валидирана во пракса преку создавање на инфраструктура сочинета од 6 кластери, поставени во 3 различни земји на 2 континента, обединети во единствен екосистем. Модуларноста може да се гледа како можност за идно проширување на архитектурата со дополнителни извршни околини или за ажурирање на постојните, како што тие дополнително ќе се развиваат. Дополнително, се остава и отворена врата за други подобрувања, како поефикасни алгоритми за распределба на задачите, поддршка за нови компјутерски архитектури и хардвер, без притоа да бидат потребни интервенции во најгорниот слој претставен од безсерверските функции поставени од крајните корисници.

Кога станува збор за поставување функции од крајни корисници и употреба на решението во пракса, потребна е алатка која ги користи интерфејсите за апликативни програми на унифицираниот екосистем слој за овозможување лесно инстанцирање нови функции, во форма на каталог. Во овој поглед, со цел потврда на интероперабилноста на претставената архитектура во пракса, се разгледува nmaas, како готово решение. nmaas, првично развиен за работа во единечни Kubernetes кластери и подигнување апликации од готов каталог, може да работи во рамките на контролната рамнина на федерацијата. Преку воведување безсерверски функции во единствен каталог, се овозможува користење на целиот унифициран екосистем низ еден единствен прозорец, со едноставен кориснички интерфејс.

Како заклучок, со претставените истражувања и резултати се потврдуваат сите 5 хипотези поставени на самиот почеток. Идентификувањето, а потоа и валидацијата на WebAssembly како соодветна технологија за безсерверски функции служеше како инспирација за поврзување на оваа нова извршна околина со постоечки оркестратор за контејнери. На овој начин, за првпат во литературата, се овозможи постоечки контејнер оркестратор, Kubernetes, да врши оркестрација и на WebAssembly модули. Со дефинирањето и демонстрацијата како новите и иновативни решенија за виртуелни приватни мрежи можат да се користат за воспоставување на надмрежи, се обезбедува брзо, безбедно и скалабилно поврзување на различни инфраструктури помеѓу облакот и работ на мрежата. За крај, изведувањето на функционални барања и нивното соодветно задоволување преку референтна имплементација во пракса, резултираше со комплетирање на анализата и дизајнот на повеќенаменска архитектура на безсерверски платформи за транспарентни пресметки во облак-раб екосистемот, со што се надминуваат ограничувањата на досегашните алтернативни решенија.

9. КОРИСТЕНА ЛИТЕРАТУРА

[1] R. Buyya, C. S. Yeo, S. Venugopal, J. Broberg, and I. Brandic, 'Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility', *Future Generation Computer Systems*, vol. 25, no. 6, pp. 599–616, Jun. 2009, doi: 10.1016/j.future.2008.12.001.

[2] P. Mell and T. Grance, 'The NIST Definition of Cloud Computing', National Institute of Standards and Technology, NIST Special Publication (SP) 800-145, Sep. 2011. doi: 10.6028/NIST.SP.800-145.

[3] C. M. Mohammed and S. R. M. Zeebaree, 'Sufficient Comparison Among Cloud Computing Services: IaaS, PaaS, and SaaS: A Review', *International Journal of Science and Business*, vol. 5, no. 2, pp. 17–30, 2021.

[4] Y. Duan, G. Fu, N. Zhou, X. Sun, N. C. Narendra, and B. Hu, 'Everything as a Service (XaaS) on the Cloud: Origins, Current and Future Trends', in *2015 IEEE 8th International Conference on Cloud Computing*, Jun. 2015, pp. 621–628. doi: 10.1109/CLOUD.2015.88.

[5] N. Kratzke, 'A Brief History of Cloud Application Architectures', *Applied Sciences*, vol. 8, no. 8, p. 1368, Aug. 2018, doi: 10.3390/app8081368.

[6] 'AWS Lambda – Serverless Compute - Amazon Web Services', Amazon Web Services, Inc. Accessed: Apr. 26, 2021. [Online]. Available: https://aws.amazon.com/lambda/

[7] 'Azure Functions Serverless Compute | Microsoft Azure'. Accessed: Apr. 26, 2021. [Online]. Available: https://azure.microsoft.com/en-us/services/functions/

[8] 'IBM Cloud Functions - Overview'. Accessed: Apr. 26, 2021. [Online]. Available: https://www.ibm.com/cloud/functions

[9] 'Cloud Functions', Google Cloud. Accessed: Apr. 26, 2021. [Online]. Available: https://cloud.google.com/functions

[10] 'Apache OpenWhisk is a serverless, open source cloud platform'. Accessed: Apr. 26, 2021. [Online]. Available: https://openwhisk.apache.org/

[11] O. Ltd, 'OpenFaaS - Serverless Functions Made Simple', OpenFaaS - Serverless Functions Made Simple. Accessed: Dec. 15, 2022. [Online]. Available: https://www.openfaas.com/

[12] 'Kubeless'. Accessed: Dec. 15, 2022. [Online]. Available: https://kubeless.io/

[13] 'Getting started with IBM Cloud Functions'. Accessed: Apr. 26, 2021. [Online]. Available: https://cloud.ibm.com/docs/openwhisk?topic=openwhisk-getting-started

[14] *Azure/iotedge*. (Apr. 23, 2021). C#. Microsoft Azure. Accessed: Apr. 26, 2021. [Online]. Available: https://github.com/Azure/iotedge

[15] S. S. Gill *et al.*, 'Transformative effects of IoT, Blockchain and Artificial Intelligence on cloud computing: Evolution, vision, trends and open challenges', *Internet of Things*, vol. 8, p. 100118, Dec. 2019, doi: 10.1016/j.iot.2019.100118.

[16] M. S. Aslanpour *et al.*, 'Serverless Edge Computing: Vision and Challenges', in *2021 Australasian Computer Science Week Multiconference*, in ACSW '21. New York, NY, USA: Association for Computing Machinery, Feb. 2021, pp. 1–10. doi: 10.1145/3437378.3444367.

[17] P. K. Gadepalli, G. Peach, L. Cherkasova, R. Aitken, and G. Parmer, 'Challenges and Opportunities for Efficient Serverless Computing at the Edge', in *2019 38th Symposium on Reliable Distributed Systems (SRDS)*, Oct. 2019, pp. 261–2615. doi: 10.1109/SRDS47363.2019.00036.

[18] J. M. Hellerstein *et al.*, 'Serverless Computing: One Step Forward, Two Steps Back', in *CIDR 2019*, Monterey, CA, Dec. 2018. Accessed: Apr. 09, 2021. [Online]. Available: http://arxiv.org/abs/1812.03651

[19] T. Linden, R. Khandelwal, H. Harkous, and K. Fawaz, 'The Privacy Policy Landscape After the GDPR', *Proceedings on Privacy Enhancing Technologies*, vol. 2020, no. 1, pp. 47–64, Jan. 2020, doi: 10.2478/popets-2020-0004.

[20] 'AWS IoT Greengrass - Amazon Web Services', Amazon Web Services, Inc. Accessed: Apr. 26, 2021. [Online]. Available: https://aws.amazon.com/greengrass/

[21] 'IoT Hub | Microsoft Azure'. Accessed: Apr. 26, 2021. [Online]. Available: https://azure.microsoft.com/en-us/services/iot-hub/

[22] V. Kjorveziroski, S. Filiposka, and V. Trajkovik, 'IoT Serverless Computing at the Edge: A Systematic Mapping Review', *Computers*, vol. 10, no. 10, Art. no. 10, Oct. 2021, doi: 10.3390/computers10100130.

[23] V. Kjorveziroski *et al.*, 'IoT Serverless Computing at the Edge: Open Issues and Research Direction', *Transactions on Networks and Communications*, vol. 9, no. 4, Art. no. 4, Dec. 2021, doi: 10.14738/tnc.94.11231.

[24] V. Kjorveziroski, S. Filiposka, and V. Trajkovik, 'Serverless Platforms Performance Evaluation at the Network Edge', in *ICT Innovations 2021. Digital Transformation*, in Communications in Computer and Information Science. Cham: Springer International Publishing, 2022, pp. 160–172. doi: 10.1007/978-3-031-04206-5_12.

[25] V. Kjorveziroski and S. Filiposka, 'Kubernetes distributions for the edge: serverless performance evaluation', *J Supercomput*, vol. 78, no. 11, pp. 13728–13755, Jul. 2022, doi: 10.1007/s11227-022-04430-6.

[26] V. Kjorveziroski, C. Bernad, K. Gilly, and S. Filiposka, ‘Full-mesh VPN performance evaluation for a secure edge-cloud continuum’, *Software: Practice and Experience*, vol. 54, no. 8, pp. 1543–1564, 2024, doi: 10.1002/spe.3329.
[27] V. Kjorveziroski, A. Mishev, and S. Filiposka, ‘Evaluating IPv6 Support in Kubernetes’, in *2021 29th Telecommunications Forum (TELFOR)*, Belgrade, Serbia: IEEE, Nov. 2021, pp. 1–4. doi: 10.1109/TELFOR52709.2021.9653276.
[28] V. Kjorveziroski and S. Filiposka, ‘WebAssembly as an Enabler for Next Generation Serverless Computing’, *J Grid Computing*, vol. 21, no. 3, p. 34, Jun. 2023, doi: 10.1007/s10723-023-09669-8.
[29] V. Kjorveziroski and S. Filiposka, ‘WebAssembly Orchestration in the Context of Serverless Computing’, *J Netw Syst Manage*, vol. 31, no. 3, p. 62, Jul. 2023, doi: 10.1007/s10922-023-09753-0.
[30] V. Kjorveziroski, S. Filiposka, and A. Mishev, ‘Evaluating WebAssembly for Orchestrated Deployment of Serverless Functions’, in *2022 30th Telecommunications Forum (TELFOR)*, Nov. 2022, pp. 1–4. doi: 10.1109/TELFOR56187.2022.9983733.
[31] V. Kjorveziroski, C. Bernad Canto, K. Gilly, and S. Filiposka, ‘Implementing Multi-Access Edge Computing with Kubernetes’, 2022, Accessed: Feb. 27, 2023. [Online]. Available: https://repository.ukim.mk:443/handle/20.500.12188/25349
[32] V. Kjorveziroski, ‘Framework for a Multipurpose Remotely Accessible Laboratory for Education’, presented at the 19th International Conference on Informatics and Information Technologies, Skopje, North Macedonia, May 2022.
[33] V. Kjorveziroski and S. Filiposka, ‘Federated architecture for serverless platforms aimed at transparent execution in the edge-cloud continuum’, *IJCC*, vol. 14, no. 1, pp. 115–144, 2025, doi: 10.1504/IJCC.2025.145664.
[34] V. Kjorveziroski, P. V. Vuletic, Ł. Łopatowski, and F. Loui, ‘On-Demand Network Management with NMaaS: Network Management as a Service’, in *NOMS 2022-2022 IEEE/IFIP Network Operations and Management Symposium*, Apr. 2022, pp. 1–3. doi: 10.1109/NOMS54207.2022.9789815.
[35] B. Varghese and R. Buyya, ‘Next generation cloud computing: New trends and research directions’, *Future Generation Computer Systems*, vol. 79, pp. 849–861, Feb. 2018, doi: 10.1016/j.future.2017.09.020.
[36] N. El Ioini, D. Hästbacka, C. Pahl, and D. Taibi, ‘Platforms for Serverless at the Edge: A Review’, in *Advances in Service-Oriented and Cloud Computing*, vol. 1360, C. Zirpins, I. Paraskakis, V. Andrikopoulos, N. Kratzke, C. Pahl, N. El Ioini, A. S. Andreou, G. Feuerlicht, W. Lamersdorf, G. Ortiz, W.-J. Van den Heuvel, J. Soldani, M. Villari, G. Casale, and P. Plebani, Eds., Cham: Springer International Publishing, 2021, pp. 29–40. Accessed: Apr. 09, 2021. [Online]. Available: http://link.springer.com/10.1007/978-3-030-71906-7_3
[37] H. Shafiei, A. Khonsari, and P. Mousavi, ‘Serverless Computing: A Survey of Opportunities, Challenges and Applications’, *arXiv:1911.01296 [cs]*, Dec. 2019, Accessed: Feb. 09, 2021. [Online]. Available: http://arxiv.org/abs/1911.01296
[38] H. B. Hassan, S. A. Barakat, and Q. I. Sarhan, ‘Survey on serverless computing’, *J Cloud Comp*, vol. 10, no. 1, p. 39, Dec. 2021, doi: 10.1186/s13677-021-00253-7.
[39] R. Buyya *et al.*, ‘A Manifesto for Future Generation Cloud Computing: Research Directions for the Next Decade’, *ACM Comput. Surv.*, vol. 51, no. 5, pp. 1–38, Jan. 2019, doi: 10.1145/3241737.
[40] ‘Kubeflow’, Kubeflow. Accessed: Sep. 28, 2021. [Online]. Available: https://www.kubeflow.org/
[41] ‘Argo Workflows - The workflow engine for Kubernetes’. Accessed: Sep. 28, 2021. [Online]. Available: https://argoproj.github.io/argo-workflows/
[42] S. Risco, G. Moltó, D. M. Naranjo, and I. Blanquer, ‘Serverless Workflows for Containerised Applications in the Cloud Continuum’, *J Grid Computing*, vol. 19, no. 3, p. 30, Jul. 2021, doi: 10.1007/s10723-021-09570-2.
[43] M. Adhikari, T. Amgoth, and S. N. Srirama, ‘A Survey on Scheduling Strategies for Workflows in Cloud Environment and Emerging Trends’, *ACM Comput. Surv.*, vol. 52, no. 4, pp. 1–36, Sep. 2019, doi: 10.1145/3325097.
[44] L. Bittencourt *et al.*, ‘The Internet of Things, Fog and Cloud continuum: Integration and challenges’, *Internet of Things*, vol. 3–4, pp. 134–155, Oct. 2018, doi: 10.1016/j.iot.2018.09.005.
[45] N. El Ioini, A. El Majjodi, D. Hastbacka, T. Cerny, and D. Taibi, ‘Unikernels Motivations, Benefits and Issues: A Multivocal Literature Review’, in *Proceedings of the 3rd Eclipse Security, AI, Architecture and Modelling Conference on Cloud to Edge Continuum*, Ludwigsburg Germany: ACM, Oct. 2023, pp. 39–48. doi: 10.1145/3624486.3624492.
[46] A. Bocci, S. Forti, G.-L. Ferrari, and A. Brogi, ‘Secure FaaS orchestration in the fog: how far are we?’, *Computing*, vol. 103, no. 5, pp. 1025–1056, May 2021, doi: 10.1007/s00607-021-00924-y.
[47] J. Wen *et al.*, ‘An empirical study on challenges of application development in serverless computing’, in *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, in ESEC/FSE 2021. New York, NY, USA: Association for Computing Machinery, Aug. 2021, pp. 416–428. doi: 10.1145/3468264.3468558.

[48] K. Petersen, S. Vakkalanka, and L. Kuzniarz, ‘Guidelines for conducting systematic mapping studies in software engineering: An update’, *Information and Software Technology*, vol. 64, pp. 1–18, Aug. 2015, doi: 10.1016/j.infsof.2015.03.007.

[49] E. Al-Masri, I. Diabate, R. Jain, M. H. Lam, and S. Reddy Nathala, ‘Recycle.io: An IoT-Enabled Framework for Urban Waste Management’, in *2018 IEEE International Conference on Big Data (Big Data)*, Seattle, WA, USA: IEEE, Dec. 2018, pp. 5285–5287. doi: 10.1109/BigData.2018.8622117.

[50] T. Pfandzelter and D. Bermbach, ‘IoT Data Processing in the Fog: Functions, Streams, or Batch Processing?’, in *2019 IEEE International Conference on Fog Computing (ICFC)*, Prague, Czech Republic: IEEE, Jun. 2019, pp. 201–206. doi: 10.1109/ICFC.2019.00033.

[51] S. Zhang, X. Luo, and E. Litvinov, ‘Serverless computing for cloud-based power grid emergency generation dispatch’, *International Journal of Electrical Power & Energy Systems*, vol. 124, p. 106366, Jan. 2021, doi: 10.1016/j.ijepes.2020.106366.

[52] M. Gorlatova, H. Inaltekin, and M. Chiang, ‘Characterizing task completion latencies in multi-point multi-quality fog computing systems’, *Computer Networks*, vol. 181, p. 107526, Nov. 2020, doi: 10.1016/j.comnet.2020.107526.

[53] M. Salehe, Z. Hu, S. H. Mortazavi, I. Mohomed, and T. Capes, ‘VideoPipe: Building Video Stream Processing Pipelines at the Edge’, in *Proceedings of the 20th International Middleware Conference Industrial Track*, Davis CA USA: ACM, Dec. 2019, pp. 43–49. doi: 10.1145/3366626.3368131.

[54] A. Christidis, R. Davies, and S. Moschoyiannis, ‘Serving Machine Learning Workloads in Resource Constrained Environments: a Serverless Deployment Example’, in *2019 IEEE 12th Conference on Service-Oriented Computing and Applications (SOCA)*, Kaohsiung, Taiwan: IEEE, Nov. 2019, pp. 55–63. doi: 10.1109/SOCA.2019.00016.

[55] L. Baresi, D. Filgueira Mendonça, and M. Garriga, ‘Empowering Low-Latency Applications Through a Serverless Edge Computing Architecture’, in *Service-Oriented and Cloud Computing*, F. De Paoli, S. Schulte, and E. Broch Johnsen, Eds., in Lecture Notes in Computer Science. Cham: Springer International Publishing, 2017, pp. 196–210. doi: 10.1007/978-3-319-67262-5_15.

[56] M. Großmann, C. Ioannidis, and D. T. Le, ‘Applicability of Serverless Computing in Fog Computing Environments for IoT Scenarios’, in *Proceedings of the 12th IEEE/ACM International Conference on Utility and Cloud Computing Companion*, in UCC ’19 Companion. New York, NY, USA: Association for Computing Machinery, Dec. 2019, pp. 29–34. doi: 10.1145/3368235.3368834.

[57] A. Albayati, N. F. Abdullah, A. Abu-Samah, A. H. Mutlag, and R. Nordin, ‘A Serverless Advanced Metering Infrastructure Based on Fog-Edge Computing for a Smart Grid: A Comparison Study for Energy Sector in Iraq’, *Energies*, vol. 13, no. 20, p. 5460, Oct. 2020, doi: 10.3390/en13205460.

[58] F. Huber and M. Mock, ‘Toci: Computational Intelligence in an Energy Management System’, in *2020 IEEE Symposium Series on Computational Intelligence (SSCI)*, Canberra, ACT, Australia: IEEE, Dec. 2020, pp. 1287–1296. doi: 10.1109/SSCI47803.2020.9308324.

[59] L. F. Herrera-Quintero, J. C. Vega-Alfonso, K. B. A. Banse, and E. C. Zambrano, ‘Smart ITS Sensor for the Transportation Planning Based on IoT Approaches Using Serverless and Microservices Architecture’, *IEEE Intelligent Transportation Systems Magazine*, vol. 10, no. 2, pp. 17–27, Summer 2018, doi: 10.1109/MITS.2018.2806620.

[60] E. Jonas *et al.*, ‘Cloud Programming Simplified: A Berkeley View on Serverless Computing’, *arXiv:1902.03383 [cs]*, Feb. 2019, Accessed: Jul. 30, 2020. [Online]. Available: http://arxiv.org/abs/1902.03383

[61] P. K. Gadepalli, S. McBride, G. Peach, L. Cherkasova, and G. Parmer, ‘Sledge: a Serverless-first, Light-weight Wasm Runtime for the Edge’, in *Proceedings of the 21st International Middleware Conference*, in Middleware ’20. New York, NY, USA: Association for Computing Machinery, Dec. 2020, pp. 265–279. doi: 10.1145/3423211.3425680.

[62] A. Hall and U. Ramachandran, ‘An execution model for serverless functions at the edge’, in *Proceedings of the International Conference on Internet of Things Design and Implementation*, in IoTDI ’19. New York, NY, USA: Association for Computing Machinery, Apr. 2019, pp. 225–236. doi: 10.1145/3302505.3310084.

[63] C. Cicconetti, M. Conti, and A. Passarella, ‘Low-latency Distributed Computation Offloading for Pervasive Environments’, in *2019 IEEE International Conference on Pervasive Computing and Communications (PerCom*, Kyoto, Japan: IEEE, Mar. 2019, pp. 1–10. doi: 10.1109/PERCOM.2019.8767419.

[64] J. Patman, D. Chemodanov, P. Calyam, K. Palaniappan, C. Sterle, and M. Boccia, ‘Predictive Cyber Foraging for Visual Cloud Computing in Large-Scale IoT Systems’, *IEEE Transactions on Network and Service Management*, vol. 17, no. 4, pp. 2380–2395, Dec. 2020, doi: 10.1109/TNSM.2020.3010497.

[65] B. Wang, A. Ali-Eldin, and P. Shenoy, ‘LaSS: Running Latency Sensitive Serverless Computations at the Edge’, in *Proceedings of the 30th International Symposium on High-Performance Parallel and Distributed Computing*, New York, NY, USA: Association for Computing Machinery, 2020, pp. 239–251. Accessed: Sep. 02, 2021. [Online]. Available: https://doi.org/10.1145/3431379.3460646

[66] I. Pelle, F. Paolucci, B. Sonkoly, and F. Cugini, 'Latency-Sensitive Edge/Cloud Serverless Dynamic Deployment Over Telemetry-Based Packet-Optical Network', *IEEE Journal on Selected Areas in Communications*, vol. 39, no. 9, pp. 2849–2863, Sep. 2021, doi: 10.1109/JSAC.2021.3064655.

[67] T. Elgamal, 'Costless: Optimizing Cost of Serverless Computing through Function Fusion and Placement', in *2018 IEEE/ACM Symposium on Edge Computing (SEC)*, Seattle, WA, USA: IEEE, Oct. 2018, pp. 300–312. doi: 10.1109/SEC.2018.00029.

[68] C. Cicconetti, M. Conti, and A. Passarella, 'A Decentralized Framework for Serverless Edge Computing in the Internet of Things', *IEEE Transactions on Network and Service Management*, pp. 1–1, 2020, doi: 10.1109/TNSM.2020.3023305.

[69] P. Karhula, J. Janak, and H. Schulzrinne, 'Checkpointing and Migration of IoT Edge Functions', in *Proceedings of the 2nd International Workshop on Edge Systems, Analytics and Networking*, in EdgeSys '19. New York, NY, USA: Association for Computing Machinery, Mar. 2019, pp. 60–65. doi: 10.1145/3301418.3313947.

[70] C. Cho, S. Shin, H. Jeon, and S. Yoon, 'QoS-Aware Workload Distribution in Hierarchical Edge Clouds: A Reinforcement Learning Approach', *IEEE Access*, vol. 8, pp. 193297–193313, 2020, doi: 10.1109/ACCESS.2020.3033421.

[71] S. Agarwal, M. A. Rodriguez, and R. Buyya, 'A Reinforcement Learning Approach to Reduce Serverless Function Cold Start Frequency', in *2021 IEEE/ACM 21st International Symposium on Cluster, Cloud and Internet Computing (CCGrid)*, May 2021, pp. 797–803. doi: 10.1109/CCGrid51090.2021.00097.

[72] I. Wang, E. Liri, and K. K. Ramakrishnan, 'Supporting IoT Applications with Serverless Edge Clouds', in *2020 IEEE 9th International Conference on Cloud Networking (CloudNet)*, Nov. 2020, pp. 1–4. doi: 10.1109/CloudNet51028.2020.9335805.

[73] J. Kim and K. Lee, 'FunctionBench: A Suite of Workloads for Serverless Cloud Function Service', in *2019 IEEE 12th International Conference on Cloud Computing (CLOUD)*, Milan, Italy: IEEE, Jul. 2019, pp. 502–504. doi: 10.1109/CLOUD.2019.00091.

[74] A. Palade, A. Kazmi, and S. Clarke, 'An Evaluation of Open Source Serverless Computing Frameworks Support at the Edge', in *2019 IEEE World Congress on Services (SERVICES)*, Milan, Italy: IEEE, Jul. 2019, pp. 206–211. doi: 10.1109/SERVICES.2019.00057.

[75] A. Das, S. Patterson, and M. Wittie, 'EdgeBench: Benchmarking Edge Computing Platforms', in *2018 IEEE/ACM International Conference on Utility and Cloud Computing Companion (UCC Companion)*, Zurich: IEEE, Dec. 2018, pp. 175–180. doi: 10.1109/UCC-Companion.2018.00053.

[76] L. Baresi and D. Filgueira Mendonca, 'Towards a Serverless Platform for Edge Computing', in *2019 IEEE International Conference on Fog Computing (ICFC)*, Prague, Czech Republic: IEEE, Jun. 2019, pp. 1–10. doi: 10.1109/ICFC.2019.00008.

[77] L. Baresi, D. F. Mendonça, M. Garriga, S. Guinea, and G. Quattrocchi, 'A Unified Model for the Mobile-Edge-Cloud Continuum', *ACM Trans. Internet Technol.*, vol. 19, no. 2, pp. 1–21, Apr. 2019, doi: 10.1145/3226644.

[78] S. Yang, K. Xu, L. Cui, Z. Ming, Z. Chen, and Z. Ming, 'EBI-PAI: Towards An Efficient Edge-Based IoT Platform for Artificial Intelligence', *IEEE Internet Things J.*, pp. 1–1, 2020, doi: 10.1109/JIOT.2020.3019008.

[79] T. Rausch, W. Hummer, V. Muthusamy, A. Rashed, and S. Dustdar, 'Towards a Serverless Platform for Edge {AI}', presented at the 2nd {USENIX} Workshop on Hot Topics in Edge Computing (HotEdge 19), 2019. Accessed: Mar. 23, 2021. [Online]. Available: https://www.usenix.org/conference/hotedge19/presentation/rausch

[80] B. Cheng, J. Fuerst, G. Solmaz, and T. Sanada, 'Fog Function: Serverless Fog Computing for Data Intensive IoT Services', in *2019 IEEE International Conference on Services Computing (SCC)*, Jul. 2019, pp. 28–35. doi: 10.1109/SCC.2019.00018.

[81] M. Zhang, C. Krintz, and R. Wolski, 'STOIC: Serverless Teleoperable Hybrid Cloud for Machine Learning Applications on Edge Device', in *2020 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops)*, Austin, TX, USA: IEEE, Mar. 2020, pp. 1–6. doi: 10.1109/PerComWorkshops48775.2020.9156239.

[82] C. Cicconetti, M. Conti, A. Passarella, and D. Sabella, 'Toward Distributed Computing Environments with Serverless Solutions in Edge Systems', *IEEE Commun. Mag.*, vol. 58, no. 3, pp. 40–46, Mar. 2020, doi: 10.1109/MCOM.001.1900498.

[83] Z. Huang, Z. Mi, and Z. Hua, 'HCloud: A trusted JointCloud serverless platform for IoT systems with blockchain', *China Communications*, vol. 17, no. 9, pp. 1–10, Sep. 2020, doi: 10.23919/JCC.2020.09.001.

[84] D. Pinto, J. P. Dias, and H. S. Ferreira, 'Dynamic Allocation of Serverless Functions in IoT Environments', in *2018 IEEE 16th International Conference on Embedded and Ubiquitous Computing (EUC)*, Oct. 2018, pp. 1–8. doi: 10.1109/EUC.2018.00008.

[85] C. Avasalcai, C. Tsigkanos, and S. Dustdar, 'Resource Management for Latency-Sensitive IoT Applications with Satisfiability', *IEEE Transactions on Services Computing*, pp. 1–1, 2021, doi: 10.1109/TSC.2021.3074188.
[86] W. Ling, L. Ma, C. Tian, and Z. Hu, 'Pigeon: A Dynamic and Efficient Serverless and FaaS Framework for Private Cloud', in *2019 International Conference on Computational Science and Computational Intelligence (CSCI)*, Las Vegas, NV, USA: IEEE, Dec. 2019, pp. 1416–1421. doi: 10.1109/CSCI49370.2019.00265.
[87] R. Wolski, C. Krintz, F. Bakir, G. George, and W.-T. Lin, 'CSPOT: portable, multi-scale functions-as-a-service for IoT', in *Proceedings of the 4th ACM/IEEE Symposium on Edge Computing*, Arlington Virginia: ACM, Nov. 2019, pp. 236–249. doi: 10.1145/3318216.3363314.
[88] T. Quang and Y. Peng, 'Device-driven On-demand Deployment of Serverless Computing Functions', in *2020 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops)*, Mar. 2020, pp. 1–6. doi: 10.1109/PerComWorkshops48775.2020.9156140.
[89] G. Tricomi, Z. Benomar, F. Aragona, G. Merlino, F. Longo, and A. Puliafito, 'A NodeRED-based dashboard to deploy pipelines on top of IoT infrastructure', in *2020 IEEE International Conference on Smart Computing (SMARTCOMP)*, Bologna, Italy: IEEE, Sep. 2020, pp. 122–129. doi: 10.1109/SMARTCOMP50058.2020.00036.
[90] T. Pfandzelter and D. Bermbach, 'tinyFaaS: A Lightweight FaaS Platform for Edge Environments', in *2020 IEEE International Conference on Fog Computing (ICFC)*, Sydney, Australia: IEEE, Apr. 2020, pp. 17–24. doi: 10.1109/ICFC49376.2020.00011.
[91] S. Nastic *et al.*, 'A Serverless Real-Time Data Analytics Platform for Edge Computing', *IEEE Internet Computing*, vol. 21, no. 4, pp. 64–71, 2017, doi: 10.1109/MIC.2017.2911430.
[92] P. Persson and O. Angelsmark, 'Kappa: serverless IoT deployment', in *Proceedings of the 2nd International Workshop on Serverless Computing*, in WoSC '17. New York, NY, USA: Association for Computing Machinery, Dec. 2017, pp. 16–21. doi: 10.1145/3154847.3154853.
[93] M. Zhang, F. Wang, Y. Zhu, J. Liu, and Z. Wang, 'Towards cloud-edge collaborative online video analytics with fine-grained serverless pipelines', in *Proceedings of the 12th ACM Multimedia Systems Conference*, New York, NY, USA: Association for Computing Machinery, 2021, pp. 80–93. Accessed: Sep. 02, 2021. [Online]. Available: https://doi.org/10.1145/3458305.3463377
[94] A. Luckow, K. Rattan, and S. Jha, 'Pilot-Edge: Distributed Resource Management Along the Edge-to-Cloud Continuum', in *2021 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, Jun. 2021, pp. 874–878. doi: 10.1109/IPDPSW52791.2021.00130.
[95] W.-T. Lin, F. Bakir, C. Krintz, R. Wolski, and M. Mock, 'Data Repair for Distributed, Event-based IoT Applications', in *Proceedings of the 13th ACM International Conference on Distributed and Event-based Systems*, Darmstadt Germany: ACM, Jun. 2019, pp. 139–150. doi: 10.1145/3328905.3329511.
[96] P. Datta, P. Kumar, T. Morris, M. Grace, A. Rahmati, and A. Bates, 'Valve: Securing Function Workflows on Serverless Computing Platforms', in *Proceedings of The Web Conference 2020*, Taipei Taiwan: ACM, Apr. 2020, pp. 939–950. doi: 10.1145/3366423.3380173.
[97] I. Pelle, J. Czentye, J. Doka, A. Kern, B. P. Gero, and B. Sonkoly, 'Operating Latency Sensitive Applications on Public Serverless Edge Cloud Platforms', *IEEE Internet Things J.*, pp. 1–1, 2020, doi: 10.1109/JIOT.2020.3042428.
[98] N. Kratzke, 'A Brief History of Cloud Application Architectures', *Applied Sciences*, vol. 8, no. 8, p. 1368, Aug. 2018, doi: 10.3390/app8081368.
[99] B. Trach, O. Oleksenko, F. Gregor, P. Bhatotia, and C. Fetzer, 'Clemmys: towards secure remote execution in FaaS', in *Proceedings of the 12th ACM International Conference on Systems and Storage*, in SYSTOR '19. New York, NY, USA: Association for Computing Machinery, May 2019, pp. 44–54. doi: 10.1145/3319647.3325835.
[100] *gwsystems/sledge-serverless-framework*. (Apr. 26, 2021). C. The Embedded and Operating Systems group at GWU. Accessed: Apr. 27, 2021. [Online]. Available: https://github.com/gwsystems/sledge-serverless-framework
[101] *gwsystems/aWsm*. (Apr. 26, 2021). C. The Embedded and Operating Systems group at GWU. Accessed: Apr. 27, 2021. [Online]. Available: https://github.com/gwsystems/aWsm
[102] 'Nuclio', nuclio. Accessed: Apr. 27, 2021. [Online]. Available: https://nuclio.io/
[103] 'Firecracker – Lightweight Virtualization for Serverless Computing', Amazon Web Services. Accessed: Apr. 27, 2021. [Online]. Available: https://aws.amazon.com/blogs/aws/firecracker-lightweight-virtualization-for-serverless-computing/
[104] 'Firecracker – Secure and fast microVMs for serverless computing'. Accessed: Apr. 27, 2021. [Online]. Available: https://firecracker-microvm.github.io/
[105] F. Manco *et al.*, 'My VM is Lighter (and Safer) than your Container', in *Proceedings of the 26th Symposium on Operating Systems Principles*, in SOSP '17. New York, NY, USA: Association for Computing Machinery, Oct. 2017, pp. 218–233. doi: 10.1145/3132747.3132763.

[106] 'Projects | Unikernels'. Accessed: Apr. 27, 2021. [Online]. Available: http://unikernel.org/projects/
[107] C. Cicconetti, *ccicconetti/serverlessonedge*. (Apr. 23, 2021). C++. Accessed: Apr. 27, 2021. [Online]. Available: https://github.com/ccicconetti/serverlessonedge
[108] 'Horizontal Pod Autoscaling', Kubernetes. Accessed: Dec. 26, 2021. [Online]. Available: https://kubernetes.io/docs/tasks/run-application/horizontal-pod-autoscale/
[109] M. Zhang, *Heronalps/STOIC*. (Jul. 01, 2020). Jupyter Notebook. Accessed: Apr. 27, 2021. [Online]. Available: https://github.com/Heronalps/STOIC
[110] D. Pinto, *duartepinto/serverless-iot*. (Oct. 31, 2018). JavaScript. Accessed: Apr. 27, 2021. [Online]. Available: https://github.com/duartepinto/serverless-iot
[111] smartfog, *smartfog/fogflow*. (Apr. 27, 2021). Go. Accessed: Apr. 27, 2021. [Online]. Available: https://github.com/smartfog/fogflow
[112] *MAYHEM-Lab/cspot*. (Mar. 26, 2021). C. MAYHEM-Lab. Accessed: Apr. 27, 2021. [Online]. Available: https://github.com/MAYHEM-Lab/cspot
[113] 'deib-polimi/A3-E', GitHub. Accessed: Apr. 27, 2021. [Online]. Available: https://github.com/deib-polimi
[114] *SCAR - Serverless Container-aware ARchitectures*. (Sep. 04, 2021). Python. GRyCAP. Accessed: Sep. 04, 2021. [Online]. Available: https://github.com/grycap/scar
[115] *OSCAR - Open Source Serverless Computing for Data-Processing Applications*. (Jul. 30, 2021). Vue. GRyCAP. Accessed: Sep. 04, 2021. [Online]. Available: https://github.com/grycap/oscar
[116] 'scar/examples/mask-detector-workflow at master · grycap/scar', GitHub. Accessed: Sep. 04, 2021. [Online]. Available: https://github.com/grycap/scar
[117] cavasalcai, *Decentralized Resource Management for Latency-Sensitive IoT Applications with Satisfiability*. (Feb. 24, 2021). Python. Accessed: Sep. 04, 2021. [Online]. Available: https://github.com/cavasalcai/Decentralized-Resource-Management
[118] *OpenFogStack/tinyFaaS*. (Apr. 08, 2021). Python. OpenFogStack. Accessed: Apr. 27, 2021. [Online]. Available: https://github.com/OpenFogStack/tinyFaaS
[119] *MDSLab/stack4things*. (Dec. 02, 2019). MDSLab. Accessed: Apr. 27, 2021. [Online]. Available: https://github.com/MDSLab/stack4things
[120] *Sched-Sim*. Accessed: Apr. 27, 2021. [Online]. Available: https://git.dsg.tuwien.ac.at/serverless-edge-ai/sched-sim
[121] C. Cicconetti, *ccicconetti/etsimec*. (Mar. 12, 2021). C++. Accessed: Apr. 27, 2021. [Online]. Available: https://github.com/ccicconetti/etsimec
[122] 'Qinling - OpenStack'. Accessed: Apr. 27, 2021. [Online]. Available: https://wiki.openstack.org/wiki/Qinling
[123] 'Node-RED'. Accessed: Dec. 05, 2023. [Online]. Available: https://nodered.org/
[124] 'K3s: Lightweight Kubernetes'. Accessed: Sep. 05, 2021. [Online]. Available: https://k3s.io/
[125] 'MicroK8s - Zero-ops Kubernetes for developers, edge and IoT | MicroK8s', microk8s.io. Accessed: Sep. 05, 2021. [Online]. Available: http://microk8s.io
[126] *EricssonResearch/calvin-base*. (Feb. 02, 2021). Python. Ericsson Research. Accessed: Apr. 27, 2021. [Online]. Available: https://github.com/EricssonResearch/calvin-base
[127] S. Dahmen-Lhuissier, 'ETSI - Multi-access Edge Computing - Standards for MEC', ETSI. Accessed: Apr. 27, 2021. [Online]. Available: https://www.etsi.org/technologies/multi-access-edge-computing
[128] Y. Gan *et al.*, 'An Open-Source Benchmark Suite for Microservices and Their Hardware-Software Implications for Cloud & Edge Systems', in *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, in ASPLOS '19. New York, NY, USA: Association for Computing Machinery, Apr. 2019, pp. 3–18. doi: 10.1145/3297858.3304013.
[129] R. F. Hussain, M. A. Salehi, and O. Semiari, 'Serverless Edge Computing for Green Oil and Gas Industry', in *2019 IEEE Green Technologies Conference(GreenTech)*, Apr. 2019, pp. 1–4. doi: 10.1109/GreenTech.2019.8767119.
[130] 'Apache JMeter - Apache JMeter™'. Accessed: Dec. 07, 2023. [Online]. Available: https://jmeter.apache.org/
[131] A. Das, *akaanirban/edgebench*. (Jul. 21, 2020). Accessed: Apr. 27, 2021. [Online]. Available: https://github.com/akaanirban/edgebench
[132] *kmu-bigdata/serverless-faas-workbench*. (Apr. 19, 2021). Python. BigData Lab. in KMU. Accessed: Jan. 14, 2023. [Online]. Available: https://github.com/kmu-bigdata/serverless-faas-workbench
[133] S. Brenner and R. Kapitza, 'Trust more, serverless', in *Proceedings of the 12th ACM International Conference on Systems and Storage*, in SYSTOR '19. New York, NY, USA: Association for Computing Machinery, May 2019, pp. 33–43. doi: 10.1145/3319647.3325825.
[134] 'The Serverless Application Framework | Serverless.com', serverless. Accessed: Apr. 27, 2021. [Online]. Available: https://serverless.com/
[135] *serverless/serverless on GitHub*. (Apr. 27, 2021). JavaScript. Serverless. Accessed: Apr. 27, 2021. [Online]. Available: https://github.com/serverless/serverless

[136] ‘AWS Serverless Application Repository - Amazon Web Services’, Amazon Web Services, Inc. Accessed: May 22, 2021. [Online]. Available: https://aws.amazon.com/serverless/serverlessrepo/
[137] K. Djemame, M. Parker, and D. Datsev, ‘Open-source Serverless Architectures: an Evaluation of Apache OpenWhisk’, in *2020 IEEE/ACM 13th International Conference on Utility and Cloud Computing (UCC)*, Dec. 2020, pp. 329–335. doi: 10.1109/UCC48980.2020.00052.
[138] *openfaas/faasd*. (May 30, 2021). Go. OpenFaaS. Accessed: May 31, 2021. [Online]. Available: https://github.com/openfaas/faasd
[139] ‘faasd - OpenFaaS’. Accessed: May 31, 2021. [Online]. Available: https://docs.openfaas.com/deployment/faasd/
[140] *OpenFaaS Rust HTTP Template*. (Feb. 15, 2023). Rust. openfaas-incubator. Accessed: Feb. 27, 2023. [Online]. Available: https://github.com/openfaas-incubator/rust-http-template
[141] *OpenFaaS Golang HTTP templates*. (Feb. 07, 2023). Shell. OpenFaaS. Accessed: Feb. 27, 2023. [Online]. Available: https://github.com/openfaas/golang-http-template
[142] *openfaas/python-flask-template*. (May 04, 2021). Python. OpenFaaS. Accessed: May 09, 2021. [Online]. Available: https://github.com/openfaas/python-flask-template
[143] ‘2021 Kubernetes Adoption Survey’. Accessed: Dec. 26, 2021. [Online]. Available: https://www.purestorage.com/content/dam/pdf/en/analyst-reports/ar-portworx-pure-storage-2021-kubernetes-adoption-survey.pdf
[144] P. Kayal, ‘Kubernetes in Fog Computing: Feasibility Demonstration, Limitations and Improvement Scope : Invited Paper’, in *2020 IEEE 6th World Forum on Internet of Things (WF-IoT)*, New Orleans, LA, USA: IEEE, Jun. 2020, pp. 1–6. doi: 10.1109/WF-IoT48130.2020.9221340.
[145] ‘Software conformance(Certified Kubernetes)’, Cloud Native Computing Foundation. Accessed: Dec. 26, 2021. [Online]. Available: https://www.cncf.io/certification/software-conformance/
[146] J. Li, S. G. Kulkarni, K. K. Ramakrishnan, and D. Li, ‘Analyzing Open-Source Serverless Platforms: Characteristics and Performance’, *arXiv:2106.03601 [cs]*, pp. 15–20, Jul. 2021, doi: 10.18293/SEKE2021-129.
[147] A. Eiermann, M. Renner, M. Großmann, and U. R. Krieger, ‘On a Fog Computing Platform Built on ARM Architectures by Docker Container Technology’, in *Innovations for Community Services*, G. Eichler, C. Erfurth, and G. Fahrnberger, Eds., in Communications in Computer and Information Science. Cham: Springer International Publishing, 2017, pp. 71–86. doi: 10.1007/978-3-319-60447-3_6.
[148] ‘Knative’, Knative. Accessed: Jan. 15, 2023. [Online]. Available: https://knative.dev/
[149] *openfaas/faas*. (Dec. 26, 2021). Go. OpenFaaS. Accessed: Dec. 26, 2021. [Online]. Available: https://github.com/openfaas/faas
[150] ‘Autoscaling - OpenFaaS’. Accessed: Dec. 26, 2021. [Online]. Available: https://docs.openfaas.com/architecture/autoscaling/
[151] ‘Container Storage Interface (CSI) for Kubernetes GA’, Kubernetes. Accessed: May 20, 2022. [Online]. Available: https://kubernetes.io/blog/2019/01/15/container-storage-interface-ga/
[152] R. Kumar and M. C. Trivedi, ‘Networking Analysis and Performance Comparison of Kubernetes CNI Plugins’, in *Advances in Computer, Communication and Computational Sciences*, S. K. Bhatia, S. Tiwari, S. Ruidan, M. C. Trivedi, and K. K. Mishra, Eds., in Advances in Intelligent Systems and Computing. Singapore: Springer, 2021, pp. 99–109. doi: 10.1007/978-981-15-4409-5_9.
[153] ‘Kubespray – Deploy a Production Ready Kubernetes Cluster’. Accessed: Dec. 26, 2021. [Online]. Available: https://kubespray.io/#/
[154] A. Hat Red, ‘Ansible is Simple IT Automation’. Accessed: Apr. 03, 2022. [Online]. Available: https://www.ansible.com
[155] H. Galal, ‘Introduction to K3s’, SUSE Communities. Accessed: Feb. 18, 2022. [Online]. Available: https://www.suse.com/c/rancher_blog/introduction-to-k3s/
[156] ‘K3s System Requirements’, Rancher Labs. Accessed: Feb. 17, 2022. [Online]. Available: https://rancher.com/docs/k3s/latest/en/installation/installation-requirements/
[157] ‘Kubeadm System Requirements’, Kubernetes Docs. Accessed: Feb. 17, 2022. [Online]. Available: https://kubernetes.io/docs/setup/production-environment/tools/kubeadm/install-kubeadm/
[158] ‘High availability (HA) | MicroK8s’, microk8s.io. Accessed: Feb. 18, 2022. [Online]. Available: http://microk8s.io
[159] ‘MicroK8s System Requirements’, microk8s.io. Accessed: Feb. 17, 2022. [Online]. Available: http://microk8s.io
[160] ‘OpenFaaS Helm Chart for Kubernetes’, GitHub. Accessed: Dec. 26, 2021. [Online]. Available: https://github.com/openfaas/faas-netes
[161] ‘Longhorn’, Longhorn. Accessed: May 17, 2025. [Online]. Available: https://longhorn.io/
[162] J. Dogan, *rakyll/hey*. (Dec. 26, 2021). Go. Accessed: Dec. 26, 2021. [Online]. Available: https://github.com/rakyll/hey

[163] S. Eismann *et al.*, 'A Case Study on the Stability of Performance Tests for Serverless Applications', *arXiv:2107.13320 [cs]*, Jul. 2021, Accessed: Nov. 29, 2021. [Online]. Available: http://arxiv.org/abs/2107.13320
[164] 'Prometheus - Monitoring system & time series database'. Accessed: May 22, 2021. [Online]. Available: https://prometheus.io/
[165] Prometheus, 'Alertmanager | Prometheus'. Accessed: Dec. 26, 2021. [Online]. Available: https://prometheus.io/docs/alerting/latest/alertmanager/
[166] A. Haas *et al.*, 'Bringing the web up to speed with WebAssembly', in *Proceedings of the 38th ACM SIGPLAN Conference on Programming Language Design and Implementation*, Barcelona Spain: ACM, Jun. 2017, pp. 185–200. doi: 10.1145/3062341.3062363.
[167] 'WebAssembly Language Support Matrix', Fermyon Technologies (@FermyonTech). Accessed: Aug. 29, 2022. [Online]. Available: https://www.fermyon.com
[168] W. Wang, 'Empowering Web Applications with WebAssembly: Are We There Yet?', in *2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, Nov. 2021, pp. 1301–1305. doi: 10.1109/ASE51524.2021.9678831.
[169] Z. Wang, J. Wang, Z. Wang, and Y. Hu, 'Characterization and Implication of Edge WebAssembly Runtimes', in *2021 IEEE 23rd Int Conf on High Performance Computing & Communications; 7th Int Conf on Data Science & Systems; 19th Int Conf on Smart City; 7th Int Conf on Dependability in Sensor, Cloud & Big Data Systems & Application (HPCC/DSS/SmartCity/DependSys)*, Dec. 2021, pp. 71–80. doi: 10.1109/HPCC-DSS-SmartCity-DependSys53884.2021.00037.
[170] W. Wang, 'How Far We've Come – A Characterization Study of Standalone WebAssembly Runtimes', presented at the IISWC 2022, Austin, TX, USA, Nov. 2022.
[171] *WASI filesystem*. WebAssembly. Accessed: Aug. 29, 2022. [Online]. Available: https://github.com/WebAssembly/wasi-filesystem
[172] *WASI Sockets*. Rust. WebAssembly. Accessed: Aug. 29, 2022. [Online]. Available: https://github.com/WebAssembly/wasi-sockets
[173] *wasi-threads*. (Oct. 26, 2022). WebAssembly. Accessed: Nov. 09, 2022. [Online]. Available: https://github.com/WebAssembly/wasi-threads
[174] *WebAssembly System Interface – Proposals*. Rust. WebAssembly. Accessed: Aug. 29, 2022. [Online]. Available: https://github.com/WebAssembly/WASI/blob/bac366c8aeb69cacfea6c4c04a503191bf1cede1/Proposals.md
[175] P. Gackstatter, P. A. Frangoudis, and S. Dustdar, 'Pushing Serverless to the Edge with WebAssembly Runtimes', in *2022 22nd IEEE International Symposium on Cluster, Cloud and Internet Computing (CCGrid)*, May 2022, pp. 140–149. doi: 10.1109/CCGrid54584.2022.00023.
[176] *Component Model design and specification*. (Nov. 09, 2022). Python. WebAssembly. Accessed: Nov. 09, 2022. [Online]. Available: https://github.com/WebAssembly/component-model
[177] J. Long, H.-Y. Tai, S.-T. Hsieh, and M. J. Yuan, 'A lightweight design for serverless Function-as-a-Service', *IEEE Softw.*, vol. 38, no. 1, pp. 75–80, Jan. 2021, doi: 10.1109/MS.2020.3028991.
[178] D. Hockley and C. Williamson, 'Benchmarking Runtime Scripting Performance in Wasmer', in *Companion of the 2022 ACM/SPEC International Conference on Performance Engineering*, in ICPE '22. New York, NY, USA: Association for Computing Machinery, Jul. 2022, pp. 97–104. doi: 10.1145/3491204.3527477.
[179] A. Jangda, B. Powers, E. Berger, and A. Guha, 'Not So Fast: Analyzing the Performance of WebAssembly vs. Native Code', May 31, 2019. doi: 10.5555/3358807.3358817.
[180] J. Ménétrey, M. Pasin, P. Felber, and V. Schiavoni, 'WebAssembly as a Common Layer for the Cloud-edge Continuum', in *Proceedings of the 2nd Workshop on Flexible Resource and Application Management on the Edge*, Jul. 2022, pp. 3–8. doi: 10.1145/3526059.3533618.
[181] S. Murphy, L. Persaud, W. Martini, and B. Bosshard, 'On the Use of Web Assembly in a Serverless Context', in *Agile Processes in Software Engineering and Extreme Programming – Workshops*, M. Paasivaara and P. Kruchten, Eds., in Lecture Notes in Business Information Processing. Cham: Springer International Publishing, 2020, pp. 141–145. doi: 10.1007/978-3-030-58858-8_15.
[182] N. Mäkitalo *et al.*, 'WebAssembly Modules as Lightweight Containers for Liquid IoT Applications', in *Web Engineering*, M. Brambilla, R. Chbeir, F. Frasincar, and I. Manolescu, Eds., in Lecture Notes in Computer Science. Cham: Springer International Publishing, 2021, pp. 328–336. doi: 10.1007/978-3-030-74296-6_25.
[183] Stephen, *Awesome WebAssembly Runtimes*. (Nov. 09, 2022). Accessed: Nov. 09, 2022. [Online]. Available: https://github.com/appcypher/awesome-wasm-runtimes
[184] 'WasmEdge'. Accessed: Aug. 29, 2022. [Online]. Available: https://wasmedge.org/
[185] 'Wasmtime'. Accessed: Aug. 29, 2022. [Online]. Available: https://wasmtime.dev/
[186] 'Wasmer - The Universal WebAssembly Runtime'. Accessed: Aug. 29, 2022. [Online]. Available: https://wasmer.io/

[187] 'containerd – An industry-standard container runtime with an emphasis on simplicity, robustness and portability'. Accessed: Feb. 27, 2023. [Online]. Available: https://containerd.io/
[188] *containers/crun*. C. Accessed: Aug. 29, 2022. [Online]. Available: https://github.com/containers/crun
[189] *'Distroless' Container Images*. (Nov. 09, 2022). Starlark. GoogleContainerTools. Accessed: Nov. 09, 2022. [Online]. Available: https://github.com/GoogleContainerTools/distroless
[190] 'hound - crates.io: Rust Package Registry'. Accessed: Jan. 15, 2023. [Online]. Available: https://crates.io/crates/hound
[191] A. N. Simon, *bild*. (Jan. 12, 2023). Go. Accessed: Jan. 15, 2023. [Online]. Available: https://github.com/anthonynsimon/bild
[192] 'n-body (Benchmarks Game)'. Accessed: Jan. 15, 2023. [Online]. Available: https://benchmarksgame-team.pages.debian.net/benchmarksgame/performance/nbody.html
[193] 'Prime Numbers - The Algorithms'. Accessed: Jan. 15, 2023. [Online]. Available: https://the-algorithms.com
[194] A. Lok, *simsearch*. (Jan. 12, 2023). Rust. Accessed: Jan. 15, 2023. [Online]. Available: https://github.com/andylokandy/simsearch-rs
[195] S. Potapov, *Whatlang*. (Jan. 13, 2023). Rust. Accessed: Jan. 15, 2023. [Online]. Available: https://github.com/greyblake/whatlang-rs
[196] *zip-rs*. (Jan. 12, 2023). Rust. zip-rs. Accessed: Jan. 15, 2023. [Online]. Available: https://github.com/zip-rs/zip
[197] 'Supported WASM And WASI Proposals - WasmEdge Runtime'. Accessed: Nov. 09, 2022. [Online]. Available: https://wasmedge.org/book/en/features/proposals.html
[198] 'FreeBSD Manual Pages – clang - the Clang, C, C++ and Objective-C compiler'. Accessed: Jan. 15, 2023. [Online]. Available: https://www.freebsd.org/cgi/man.cgi?query=clang++&sektion=1&manpath=FreeBSD+9.0-RELEASE
[199] M. Lukša, *Kubernetes in action*. Shelter Island, NY: Manning Publications Co, 2018.
[200] A. Agache *et al.*, 'Firecracker: Lightweight Virtualization for Serverless Applications', presented at the 17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20), 2020, pp. 419–434. Accessed: Sep. 05, 2022. [Online]. Available: https://www.usenix.org/conference/nsdi20/presentation/agache
[201] M. Sebrechts, T. Ramlot, S. Borny, T. Goethals, B. Volckaert, and F. De Turck, 'Adapting Kubernetes controllers to the edge: on-demand control planes using Wasm and WASI', Sep. 02, 2022, *arXiv*: arXiv:2209.01077. doi: 10.48550/arXiv.2209.01077.
[202] 'wasmCloud Documentation'. Accessed: Aug. 29, 2022. [Online]. Available: https://wasmcloud.dev/
[203] 'Introducing Spin', Spin Documentation. Accessed: Aug. 29, 2022. [Online]. Available: https://spin.fermyon.dev
[204] 'Nomad | HashiCorp Developer', Nomad | HashiCorp Developer. Accessed: May 24, 2025. [Online]. Available: https://developer.hashicorp.com/nomad
[205] 'The frictionless WebAssembly platform for writing microservices and web apps', The Fermyon Platform. Accessed: May 24, 2025. [Online]. Available: https://www.fermyon.dev
[206] page.head.extra.author, 'What is Bindle'. Accessed: May 24, 2025. [Online]. Available: https://www.fermyon.com/blog/bindle-what-is-it
[207] 'Helm Documentation'. Accessed: Jan. 18, 2022. [Online]. Available: https://helm.sh/docs/
[208] 'Runtime Class', Kubernetes. Accessed: Feb. 27, 2023. [Online]. Available: https://kubernetes.io/docs/concepts/containers/runtime-class/
[209] 'pREST | Instant RESTFul APIs for your data | Join data across databases, REST services to build powerful modern applications'. Accessed: Feb. 27, 2023. [Online]. Available: https://prestd.com/
[210] 'Stack - OpenFaaS'. Accessed: Mar. 08, 2023. [Online]. Available: https://docs.openfaas.com/architecture/stack/
[211] 'Client in reqwest::blocking - Rust'. Accessed: Mar. 04, 2023. [Online]. Available: https://docs.rs/reqwest/latest/reqwest/blocking/struct.Client.html
[212] 'Multi-access Edge Computing (MEC): Study on MEC support for alternative virtualization technologies'. ETSI MEC ISG. Accessed: May 20, 2022. [Online]. Available: https://www.etsi.org/deliver/etsi_gr/MEC/001_099/027/02.01.01_60/gr_MEC027v020101p.pdf
[213] G. Nencioni, R. G. Garroppo, and R. F. Olimid, '5G Multi-access Edge Computing: Security, Dependability, and Performance', *arXiv:2107.13374 [cs]*, Jul. 2021, Accessed: May 20, 2022. [Online]. Available: http://arxiv.org/abs/2107.13374
[214] V. Aggarwal and B. Thangaraju, 'Performance Analysis of Virtualisation Technologies in NFV and Edge Deployments', in *2020 IEEE International Conference on Electronics, Computing and Communication Technologies (CONECCT)*, Jul. 2020, pp. 1–5. doi: 10.1109/CONECCT50063.2020.9198367.
[215] R. Perez *et al.*, 'A Performance Comparison of Virtualization Techniques to Deploy a 5G Monitoring Platform', in *2021 Joint European Conference on Networks and Communications 6G Summit (EuCNC/6G Summit)*, Jun. 2021, pp. 472–477. doi: 10.1109/EuCNC/6GSummit51104.2021.9482570.

[216] A. Randazzo and I. Tinnirello, 'Kata Containers: An Emerging Architecture for Enabling MEC Services in Fast and Secure Way', in *2019 Sixth International Conference on Internet of Things: Systems, Management and Security (IOTSMS)*, Oct. 2019, pp. 209–214. doi: 10.1109/IOTSMS48152.2019.8939164.

[217] 'KubevIRT - Building a Virtualization API for Kubernetes', KubeVirt.io. Accessed: May 20, 2022. [Online]. Available: https://kubevirt.io//

[218] R. Harkanson, Y. Kim, J.-Y. Jo, and K. Pham, 'Effects of TCP Transfer Buffers and Congestion Avoidance Algorithms on the End-to-End Throughput of TCP-over-TCP Tunnels', vol. 800, pp. 401–408, 2019, doi: 10.1007/978-3-030-14070-0_55.

[219] S. Mackey, I. Mihov, A. Nosenko, F. Vega, and Y. Cheng, 'A Performance Comparison of WireGuard and OpenVPN', in *Proceedings of the Tenth ACM Conference on Data and Application Security and Privacy*, in CODASPY '20. New York, NY, USA: Association for Computing Machinery, Mar. 2020, pp. 162–164. doi: 10.1145/3374664.3379532.

[220] X. Wei *et al.*, 'Research on Using Dynamic Thread Pool to Improve the Performance of VPN Gateway', in *2022 7th International Conference on Computer and Communication Systems (ICCCS)*, Apr. 2022, pp. 566–570. doi: 10.1109/ICCCS55155.2022.9846591.

[221] J. A. Donenfeld, 'WireGuard: Next Generation Kernel Network Tunnel', in *Proceedings 2017 Network and Distributed System Security Symposium*, San Diego, CA: Internet Society, 2017. doi: 10.14722/ndss.2017.23160.

[222] J. Whited and J. Tucker, 'Userspace isn't slow, some kernel interfaces are!', Tailscale. Accessed: Jun. 25, 2023. [Online]. Available: https://tailscale.com/blog/throughput-improvements/

[223] 'Source Code For OpenVPN Community Edition', OpenVPN. Accessed: Jun. 25, 2023. [Online]. Available: https://openvpn.net/source-code/

[224] 'Setting Up Your Own Certificate Authority (CA)', OpenVPN. Accessed: Jun. 25, 2023. [Online]. Available: https://openvpn.net/community-resources/setting-up-your-own-certificate-authority-ca/

[225] J. A. Donenfeld, 'Quick Start - WireGuard'. Accessed: Jun. 25, 2023. [Online]. Available: https://www.wireguard.com/quickstart/

[226] F. Pohl and H. D. Schotten, 'Secure and Scalable Remote Access Tunnels for the IIoT: An Assessment of openVPN and IPsec Performance', in *Service-Oriented and Cloud Computing*, F. De Paoli, S. Schulte, and E. Broch Johnsen, Eds., in Lecture Notes in Computer Science. Cham: Springer International Publishing, 2017, pp. 83–90. doi: 10.1007/978-3-319-67262-5_7.

[227] E. Dekker and P. Spaans, 'Performance comparison of VPN implementations WireGuard, strongSwan, and OpenVPN in a 1 Gbit/s environment'.

[228] T. Goethals, D. Kerkhove, B. Volckaert, and F. D. Turck, 'Scalability evaluation of VPN technologies for secure container networking', in *2019 15th International Conference on Network and Service Management (CNSM)*, Oct. 2019, pp. 1–7. doi: 10.23919/CNSM46954.2019.9012673.

[229] J. Paillisse, A. Barcia, A. Lopez, A. Rodriguez-Natal, F. Maino, and A. Cabellos, 'A Control Plane for WireGuard', in *2021 International Conference on Computer Communications and Networks (ICCCN)*, Jul. 2021, pp. 1–8. doi: 10.1109/ICCCN52240.2021.9522315.

[230] P. Gunda and S. D. Voleti, *Performance evaluation of wireguard in kubernetes cluster*. 2021. Accessed: Feb. 18, 2023. [Online]. Available: http://urn.kb.se/resolve?urn=urn:nbn:se:bth-21167

[231] Z. Wang, M. Goudarzi, J. Aryal, and R. Buyya, 'Container Orchestration in Edge and Fog Computing Environments for Real-Time IoT Applications', in *Computational Intelligence and Data Analytics*, R. Buyya, S. M. Hernandez, R. M. R. Kovvur, and T. H. Sarma, Eds., in Lecture Notes on Data Engineering and Communications Technologies. Singapore: Springer Nature, 2023, pp. 1–21. doi: 10.1007/978-981-19-3391-2_1.

[232] D. Mlynka, 'IoT device management using Kubernetes', 2022, [Online]. Available: https://is.muni.cz/th/x3jnk/fi-pdflatex.pdf

[233] D. Falcone, 'Designing a scalable network overlay for Kubernetes multi-cluster topologies', laurea, Politecnico di Torino, 2021. Accessed: Apr. 22, 2023. [Online]. Available: https://webthesis.biblio.polito.it/20504/

[234] 'Virtual Kubelet'. Accessed: Jun. 25, 2023. [Online]. Available: https://virtual-kubelet.io/

[235] K. Subratie, S. Aditya, and R. J. Figueiredo, 'EdgeVPN: Self-organizing layer-2 virtual edge networks', *Future Generation Computer Systems*, vol. 140, pp. 104–116, Mar. 2023, doi: 10.1016/j.future.2022.10.007.

[236] A. Keranen, C. Holmberg, and J. Rosenberg, 'Interactive Connectivity Establishment (ICE): A Protocol for Network Address Translator (NAT) Traversal', p. RFC8445, Jul. 2018, doi: 10.17487/RFC8445.

[237] P. S. Junior, D. Miorandi, and G. Pierre, 'Good Shepherds Care For Their Cattle: Seamless Pod Migration in Geo-Distributed Kubernetes', *2022 IEEE 6th International Conference on Fog and Edge Computing (ICFEC)*, pp. 26–33, May 2022, doi: 10.1109/ICFEC54809.2022.00011.

[238] C. Chee, *Awesome WireGuard*. (Jun. 25, 2023). Accessed: Jun. 25, 2023. [Online]. Available: https://github.com/cedrickchee/awesome-wireguard

[239] HarvsG, *Compare WireGuard Mesh Tools*. (Jun. 24, 2023). Accessed: Jun. 25, 2023. [Online]. Available: https://github.com/HarvsG/WireGuardMeshes
[240] Tailscale, 'Network access controls (ACLs)', Tailscale. Accessed: Jun. 25, 2023. [Online]. Available: https://tailscale.com/kb/1018/acls/
[241] J. Font, *Headscale ACL Support*. (Jun. 25, 2023). Go. Accessed: Jun. 25, 2023. [Online]. Available: https://github.com/juanfont/headscale
[242] 'juanfont/headscale: An open source, self-hosted implementation of the Tailscale control server'. Accessed: Jun. 25, 2023. [Online]. Available: https://github.com/juanfont/headscale
[243] 'netbirdio/netbird: Connect your devices into a single secure private WireGuard®-based mesh network with SSO/MFA and simple access controls.' Accessed: Jun. 25, 2023. [Online]. Available: https://github.com/netbirdio/netbird
[244] 'zerotier/ZeroTierOne: A Smart Ethernet Switch for Earth'. Accessed: Jun. 25, 2023. [Online]. Available: https://github.com/zerotier/ZeroTierOne
[245] 'gravitl/netmaker: Netmaker makes networks with WireGuard. Netmaker automates fast, secure, and distributed virtual networks.' Accessed: Jun. 25, 2023. [Online]. Available: https://github.com/gravitl/netmaker
[246] *Tailscale on GitHub*. (Jun. 25, 2023). Go. Tailscale. Accessed: Jun. 25, 2023. [Online]. Available: https://github.com/tailscale/tailscale
[247] 'firezone/firezone: WireGuard®-based VPN server and firewall'. Accessed: Jun. 25, 2023. [Online]. Available: https://github.com/firezone/firezone
[248] *WireGuard Easy*. (Jun. 25, 2023). HTML. wg-easy. Accessed: Jun. 25, 2023. [Online]. Available: https://github.com/wg-easy/wg-easy
[249] 'Stormblest / mistborn · GitLab', GitLab. Accessed: Jun. 25, 2023. [Online]. Available: https://gitlab.com/cyber5k/mistborn
[250] L. Antunes, *wesher - Wireguard Overlay Mesh Network Manager*. (Jun. 23, 2023). Go. Accessed: Jun. 25, 2023. [Online]. Available: https://github.com/costela/wesher
[251] *innernet - A private network system that uses WireGuard under the hood*. (Jun. 24, 2023). Rust. Tonari, Inc. Accessed: Jun. 25, 2023. [Online]. Available: https://github.com/tonarino/innernet
[252] *slackhq/nebula*. (Jun. 25, 2023). Go. Slack. Accessed: Jun. 25, 2023. [Online]. Available: https://github.com/slackhq/nebula
[253] 'Configure MTU to maximize network performance | Calico Documentation'. Accessed: Jun. 25, 2023. [Online]. Available: https://docs.tigera.io/calico/latest/networking/configuring/mtu
[254] Tailscale, 'DERP Servers', Tailscale. Accessed: Jun. 25, 2023. [Online]. Available: https://tailscale.com/kb/1232/derp-servers/
[255] Tailscale, 'Custom DERP Servers', Tailscale. Accessed: May 07, 2023. [Online]. Available: https://tailscale.com/kb/1118/custom-derp-servers/
[256] R. Mahy, P. Matthews, and J. Rosenberg, 'Traversal Using Relays around NAT (TURN): Relay Extensions to Session Traversal Utilities for NAT (STUN)', p. RFC5766, Apr. 2010, doi: 10.17487/rfc5766.
[257] *Coturn TURN server*. (Jun. 25, 2023). C. coturn. Accessed: Jun. 25, 2023. [Online]. Available: https://github.com/coturn/coturn
[258] 'netbird/infrastructure_files/management.json.tmpl at b524a9d49d001564b5818abe426be9689aa56ff3 · netbirdio/netbird'. Accessed: Jun. 25, 2023. [Online]. Available: https://github.com/netbirdio/netbird/blob/b524a9d49d001564b5818abe426be9689aa56ff3/infrastructure_files/management.json.tmpl#L4
[259] 'Add force relay conn env var for debug purpose by pappz · Pull Request #904 · netbirdio/netbird', GitHub. Accessed: Jun. 25, 2023. [Online]. Available: https://github.com/netbirdio/netbird/pull/904
[260] 'Protocol Design Whitepaper | ZeroTier Documentation'. Accessed: Jun. 25, 2023. [Online]. Available: https://docs.zerotier.com/zerotier/manual/
[261] 'Network Controllers | ZeroTier Documentation'. Accessed: Jun. 25, 2023. [Online]. Available: https://docs.zerotier.com/self-hosting/network-controllers
[262] 'Private Root Servers | ZeroTier Documentation'. Accessed: Jun. 25, 2023. [Online]. Available: https://docs.zerotier.com/zerotier/moons
[263] 'ZeroTierOne/tcp-proxy at dev · zerotier/ZeroTierOne', GitHub. Accessed: Jun. 25, 2023. [Online]. Available: https://github.com/zerotier/ZeroTierOne
[264] 'tc(8) - Linux manual page'. Accessed: Jun. 25, 2023. [Online]. Available: https://man7.org/linux/man-pages/man8/tc.8.html
[265] F. Lombardo, S. Salsano, A. Abdelsalam, D. Bernier, and C. Filsfils, 'Extending Kubernetes Networking to make use of Segment Routing over IPv6 (SRv6)', Jan. 03, 2023, *arXiv*: arXiv:2301.01178. Accessed: Apr. 20, 2023. [Online]. Available: http://arxiv.org/abs/2301.01178

[266] *k8s-bench-suite*. (Apr. 17, 2023). Shell. infraBuilder. Accessed: May 07, 2023. [Online]. Available: https://github.com/InfraBuilder/k8s-bench-suite

[267] M. Trevisan, D. Giordano, I. Drago, and A. S. Khatouni, 'Measuring HTTP/3: Adoption and Performance', in *2021 19th Mediterranean Communication and Computer Networking Conference (MedComNet)*, Jun. 2021, pp. 1–8. doi: 10.1109/MedComNet52149.2021.9501274.

[268] 'Introduction | ZeroTier Documentation'. Accessed: Jun. 28, 2023. [Online]. Available: https://docs.zerotier.com/self-hosting/introduction/

[269] D. Xie, Y. Hu, and L. Qin, 'An Evaluation of Serverless Computing on X86 and ARM platforms: Performance and Design Implications', in *2021 IEEE 14th International Conference on Cloud Computing (CLOUD)*, Sep. 2021, pp. 313–321. doi: 10.1109/CLOUD53861.2021.00045.

[270] F. Lumpp, F. Barchi, A. Acquaviva, and N. Bombieri, 'On the Containerization and Orchestration of RISC-V architectures for Edge-Cloud computing', *Proceedings of the 3rd Eclipse Security, AI, Architecture and Modelling Conference on Cloud to Edge Continuum*, pp. 21–28, Oct. 2023, doi: 10.1145/3624486.3624490.

[271] 'Introduction - The Cluster API Book'. Accessed: Aug. 15, 2023. [Online]. Available: https://cluster-api.sigs.k8s.io/

[272] 'IMT Vision – Framework and overall objectives of the future development of IMT for 2020 and beyond', [Online]. Available: https://www.itu.int/dms_pubrec/itu-r/rec/m/R-REC-M.2083-0-201509-I!!PDF-E.pdf

[273] 'Open, Multi-Cloud, Multi-Cluster Kubernetes Orchestration | karmada'. Accessed: Aug. 15, 2023. [Online]. Available: https://karmada.io/

[274] 'Resource Propagating | karmada'. Accessed: Aug. 15, 2023. [Online]. Available: https://karmada.io/docs/userguide/scheduling/resource-propagating/

[275] 'Overlay networking | Calico Documentation'. Accessed: Jun. 25, 2023. [Online]. Available: https://docs.tigera.io/calico/latest/networking/configuring/vxlan-ipip

[276] 'Configure BGP peering | Calico Documentation'. Accessed: Aug. 15, 2023. [Online]. Available: https://docs.tigera.io/calico/latest/networking/configuring/bgp

[277] M. Suzuki, T. Miyasaka, D. Purkayastha, Y. Fang, Q. Huang, and J. Zhu, 'Enhanced DNS Support towards Distributed MEC Environment'. [Online]. Available: https://www.etsi.org/images/files/ETSIWhitePapers/etsi-wp39-Enhanced-DNS-Support-towards-Distributed-MEC-Environment.pdf

[278] A. S. M. Rizvi, L. Bertholdo, J. Ceron, and J. Heidemann, 'Anycast Agility: Network Playbooks to Fight {DDoS}', presented at the 31st USENIX Security Symposium (USENIX Security 22), 2022, pp. 4201–4218. Accessed: Aug. 15, 2023. [Online]. Available: https://www.usenix.org/conference/usenixsecurity22/presentation/rizvi

[279] 'nmaas Documentation'. Accessed: May 24, 2025. [Online]. Available: https://docs.nmaas.eu/

[280] 'GÉANT', GÉANT | Networks - Services - People. Accessed: May 24, 2025. [Online]. Available: https://geant.org/

[281] T. Kormaník and J. Porubän, 'Exploring GitOps: An Approach to Cloud Cluster System Deployment', in *2023 21st International Conference on Emerging eLearning Technologies and Applications (ICETA)*, Oct. 2023, pp. 318–323. doi: 10.1109/ICETA61311.2023.10344182.