Universidad del País Vasco
Euskal Herriko Unibertsitatea

ZIENTZIA ETA TEKNOLOGIA FAKULTATEA
FACULTAD DE CIENCIA Y TECNOLOGÍA

Gradu Amaierako Lana / Trabajo Fin de Grado
Ingeniaritza Elektronikoko Gradua / Grado en Ingeniería Electrónica

Kubernetes eta bere ekosistema

Azterketa teoriko eta praktikoa

Egilea / Autor/a:
Jon Gabirondo López
Zuzendaria / Director/a:
Iñigo Arredondo López de Guereñu

Leioa, 2020ko ekainaren 17a / Leioa, 17 de junio de 2020

Gaien aurkibidea

1. kapitulua

Sarrera eta helburuak

Internet bidez eskainitako zerbitzuen kopuruak gorakada handia jasan du azken urteetan [1, 2]. Horietako batzuen helburua ludikoa den bitartean, besteek burutzen dituzten ekintzak eguneroko bizitzarako ezinbestekoak dira: banku-eragiketak sarean egitea ahalbidetzen duten zerbitzuak, etxetik lan egiteko plataformak edota gailuak urrunetik kontrolatzeko programak, azken horien adibide aproposak dira. Haien publikoaren hazkundeak eta publikoak horiekiko duten menpekotasunaren ondorioz, ezinbestekoa izan da zerbitzuak zein haiek exekutatzen dituzten zerbitzariak berdiseinatzea. Zerbitzuen atzean dauden aplikazioen garapenaren aldetik, elementu bakar batez osatutako aplikazio "monolitikoak" sortu eta argitaratu beharrean, mikro-zerbitzuetan oinarritutako aplikazioak garatu dira azkenaldian: elementu txikiagotan banatuz, aplikazioen garapena eta mantenua errazten da. Zerbitzariei dagokionez, makina bakarrean zerbitzu bakar bat eskaintzetik, makina ezberdinez osatutako sistemetan aplikazio ugari exekutatzera pasa da. Hainbat abantaila eskaintzen ditu planteamendu horrek: eskaerakopuruaren aldaketen (batez ere hazkundearen) arabera azpiegitura moldatzeko aukera eman eta makina anitzetan oinarritutako zerbitzu sendoak sor daitezke, zerbitzari bakarraren funtzionamenduaren menpe ez daudenak. Potentzia eta konplexutasun handiagoko egitura horiek kudeatzeko Kubernetes sortu zen: Google-ek garatutako kode irekiko plataforma horrek, mikro-zerbitzuetan oinarritutako aplikazioak klusterretan exekutatu eta mantentzeko beharrezko softwarea eskaintzen du [3]. Gaur egun, geroz eta enpresa gehiagok erabiltzen dute Kubernetes haien zerbitzuetan eta, ondorioz, erabilera erraztu edota funtzionalitateak gehitzeko helburua duten tresna ugari sortu dira Kubernetes-en inguruan.

Lan honen helburua Kubernetes-en inguruko azterketa teoriko eta praktiko bat burutzea da. Teknologia horren oinarrizko funtzionamendua azaltzeaz gain, beste hainbat tresnen jarduera berrikustea ere zati teorikoaren xedeak dira. Alderdi praktikoari dagokionez, ingurune profesional batean erabil zitekeen kluster bat kudeatu eta bertan garatutako aplikazio bat mantentzea bilatu da, teorian landutako tresnak implementatuz. Webgune dinamikoek azaldu ohi duten egitura[1] izango du aipaturiko aplikazioak, landutako kontzeptuak eta egindako inplementazioak beste adibide batzuetan aplikagarriak direla bermatuz.

Horretarako, arkitektura ezberdinetako makinez osatutako kluster bat Kubernetes-en bidez kudeatuko da, haren atzean dauden zenbait kontzeptu azaldu eta bere funtzio-

[1] Webgune dinamikoek unean eguneratutako edukiak erakusten dituzte [4]. Eguraldia erakusten duten web orriak, sare sozialak edota egunkarien webguneak adibide aproposak izan daitezke.

nalitateak probatzeko. Aplikazio baten garapena eta mantenua landuko dira ideia teorikoak praktikan ipintzeko: Kubernetes-ek eskainitako objektuez abiatuta, haren funtzionalitateak zabaltzen dituzten plataformak aztertuko dira, elkarrren arteko integrazioan arreta berezia ipiniz.

Txosten honen lehen atalean, gainontzeko testua ulertu ahal izateko lagungarriak izan daitezkeen hainbat azalpen ematen dira, aplikazioen garapenaren ingurukoak, hain zuzen ere. Segidan, Kubernetes-en atzean dagoen filosofia aurkezten da eta Kubernetes osatzen duten oinarrizko objektuak azaltzen dira, tresna osagarrien deskribapenarekin amaituz. Bigarren atalean aldiz, kluster baten inplementazioa jasotzen da. Behin Kubernetes martxan izanda, Kubernetes zein beste plataformen funtzionamendua argituko duen aplikazio baten garapena eta mantenua aztertzen da. Amaitzeko, lortutako emaitzak baloratu eta Ingenieritza Elektronikoarekin lotutako aplikazio posibleak eztabaidatzen dira.

2. kapitulua

Garapena: oinarri teorikoa

2.1 Oinarrizko kontzeptuak

Dokumentazio ofizialaren arabera, Kubernetes software-edukiontzitan sartutako aplikazioen argitarapena, eskalaketa eta kudeaketa automatizatzeko kode irekiko sistema bat da [5].

Definizio hori zein jarraian emango diren azalpenak ondo uler daitezen, komenigarria da Kubernetes-en funtzionamenduan sakondu aurretik oinarrizko kontzeptu batzuk finkatzea.

Kubernetes aplikazioekin lan egitera bideratuta dago. Aplikazio bat, sarrera batzuk hartu, horiek prozesatu eta irteera bat ematen duen software zatia da.

Aplikazio konplexuak sinpleagoak diren beste batzuk konbinatuz garatzen dira. Ohikoa da aplikazioa bi atal nagusitan banatzea: *front end* edo interfazea —datuak aurkeztu edota sartzeko aukera ematen duena— eta *back end* edo motorea —datuen kudeaketaz arduratzen dena—. Planteamendu horren ideia orokorra honakoa da: interfazeak erabiltzailearen sarrera-datuak jaso eta motorearen ezaugarriei egokitzen den formatu batera bihurtzen ditu. Haiek prozesatu ostean, motoreak erantzuna interfazeari bidaltzen dio, erabiltzaileak uler dezakeen formatu batera itzul dezan [6]. Bi zatiak independenteki garatu eta mantentzea ahalbidetzen du abstrakzio-maila horrek.

Aplikazioen arteko komunikazioa aplikazioak programatzeko interfazeen (APIen, ingelesez *Application programming interface*) bidez egiten dira. API batek software zati bat beste aplikazio batetik liburutegi bat bezala erabiltzea ahalbidetzen duen interfazea da, funtzio, azpirrutina eta metodoez osatutakoa [7]. APIek software ezberdinen integrazioa errazten dute, aplikazio konplexuak sortzeko aukera emanez.

APIak, normalean, banandutako konputazio-sistemetan argitaratzen dira, hau da, elkarren artean sarearen bidez konektatzen eta komunikatzen diren makina ezberdinek osatutako sistemetan exekutatzen dira. Sistema horiek oso fidagarriak izan behar dira, APIak eguneroko bizitzarako geroz eta beharrezkoagoak baitira: besteak beste, sarearen norberaren profila balioztatzeko, deiak egiteko edota dokumentu ezberdinekin lan egiteko erabiltzen dira. Ezin dute huts egin eta eskuragarri egon behar dira beti, sistemaren zati batek huts egitean eta eguneraketa- eta mantentze-lanek diharduten bitartean ere. Bestalde, sarean dauden horrelako zerbitzuek geroz eta erabiltzaile gehiago

dituztenez, funtsezkoa da eskalagarriak izatea, APIa argitaratzen duen sistema berdiseinatu gabe bere ahalmena handitzeko aukera eskainiz.

Banandutako konputazio-sistemak modu arrakastatsuan sortzeko eta argitaratzeko softwarea eskaintzen du Kubernetes-ek, azaldutako zehaztapen guztiak betez. Zehazki, batera lan egiteko konektatutako ordenagailuen sistemak kudeatzen ditu: klusterrak.

Hasierako definizioan, software-edukiontziak ("edukiontziak" hemendik aurrera) ere aipatu dira. Edukiontziak aplikazioen iturburu-kodea eta liburutegiak jasotzen dituen software-unitateak dira. Horrela, aplikazioak konputazio-ingurune ezberdinetan modu azkar eta fidagarrian exekuta daitezke [8].

Gainera, liburutegiek sortutako arazo asko konpon ditzake inplementazio horrek. Esaterako, aplikazioa garatzen den ingurunean (garapen-ingurunean) eta martxan ipini nahi den amaierako ingurunean (produkzio-ingurunean) liburutegien bertsio ezberdinak edukitzeak funtzionamendua esperotakoa ez izatea sor dezake. Bestalde, makina berdinetan liburutegi berdinak erabiltzen dituzten programak egonez gero, aplikazio guzti horiek liburutegien bertsio berdinak erabiltzera behartzen dira. Aplikazioak haien inguruneekin enkapsulatu eta isolatzean, arazo horiek guztiak ekiditen dira.

Kubernetes edukiontzien kudeaketa automatikoaz arduratzen da, aplikazioen fidagarritasuna hobetu eta garapenerako behar diren baliabideak murriztuz. Edukiontzien hasieratzea automatizatzeaz gain, zerbitzu eta edukiontzien egoera etengabe behatzen du, huts egindako baliabideak behar izanez gero berrabiaraziz. Akatsik antzemanez gero, jasotako eskaerak ondo dauden edukiontzietara berbidaltzeko gai ere bada.

2.2 Kubernetes-en egitura

2.2.1 Filosofia

Laburki bada ere, garrantzitsua da Kubernetes-en atzean dauden zenbait ideia eta kontzeptu azaltzea, plataformaren egituran eta lan egiteko moduan eragin handia baitaukate [9].

Aldaezintasuna

Ordenagailu eta software-sistemak elementu editagarritzat hartu ohi izan dira, aldaketak une bateko sistemaren egoeraren eguneraketa inkrementalak bezala aplikatuz. Ondorioz, oso zaila da sistema aurreko egoera batera itzultzea, haren egoera ordura arte egindako aldaketa guztien ondorioa baita. Sistema aldaezinetan, aldiz, aldaketa bat egitean sistemaren irudi guztiz berri bat sortzen da. Prozedura horren gakoa aldaketak burutzerakoan egindako moldaketen historia jasotzean datza, atzera itzultzeko aukera bermatzen baitu.

Aldaezinak diren edukiontziak kudeatzen ditu Kubernetes-ek: aplikazioren batean aldaketa bat eginez gero, eguneraketa jasotzen duen aplikazioaren kopia berri bat sortzen da, lehendik zegoena ezabatuz.

Konfigurazio deklaratiboa

Kubernetes-en definitutako objektuek desiratutako elementuak deskribatzen dituzte eta konfigurazio deklaratiboan oinarritzen dira. Nahi den egoerara iristeko makinak exekutatu beharreko aginduak eman ordez (konfigurazio inperatiboan egiten denez), lortu nahi den egoera deskribatzen da konfigurazio deklaratiboan. Kubernetes-en kasuan, sortu nahi den objektuaren egoera YAML fitxategitan deskribatzen da [10]. Fitxategi horiek bertsioen kontrol-sistema[1] baten bidez kudeatzeak aldaketak desegitea errazten du.

Sistema auto-konpontzaileak

Sistema auto-konpontzaileen (ingelesezko *self-healing system*) adibide bat da Kubernetes: etengabe ekintzak burutzen ditu une horretako egoerak desiratutako egoerarekin bat egiten duela ziurtatzeko. Ondorioz, sistema hasieratzeaz gain, perturbazio eta hutsegiteez babestuko du, haren kudeaketa eta mantenua asko erraztuz.

Hardwarearekiko abstrakzioa

Ingurune oso ezberdinetan erabil daiteke Kubernetes: arkitektura anitzetako makina fisikoetan oinarritutako *bare metal*[2] klusterretan erabiltzeaz gain, makina birtualez osatutakoak edota hodeieko zerbitzuen hornitzaile (ingelesezko *cloud service provider*)[3] ezberdinek eskainitako klusterrak ere kudea ditzake.

Ingurunea dena-delakoa izanda ere, Kubernetes-en funtzionamendua berdina (edo oso antzekoa) da kasu guztietan. Hardwarearekiko azaltzen duen abstrakzio horrek, aplikazioen garapena eta mantenua asko errazten du: konfigurazio-fitxategiez baliatuz ia berehalakoa baita garapen-ingurunetik produkzio-ingurunera migratzea (hardwarea ezberdina izan arren).

2.2.2 Oinarrizko funtzionamendua

Desiratutako elementuak (aplikazio eta zerbitzuak, adibidez) deskribatzeko Kubernetes-en APIaren objektuak erabiltzen dira. Horiek sortzeko Kubernetes-en bezero ofiziala erabiltzen da normalean: *kubectl*. Kubectl Kubernetes-en APIarekin elkarreragitea ahalbidetzen duen komando-lerroko interfazea da eta aipatutako objektuak kudeatzeaz gain klusterraren egoeraren inguruko informazioa jasotzeko ere erabil daiteke [14].

Desiratutako elementuak finkatu ostean, Kubernetes-en Kontrol Planoak klusterraren egitura desiratutakoarekin bat etortzeko hartu beharreko ekintzak burutuko ditu. Klusterrean exekutatzen diren prozesu batzuen bilduma da Kontrol Planoa. Prozesu horien artean hiruk —*kube-apiserver*, *kube-controller-manager* eta *kube-scheduler*—

[1] Bertsioen kontrol-sistema bat kodean egindako modifikazioen erregistroa jasoz fitxategietan egindako aldaketak jarraitzen laguntzen duen software bilduma da [11]. Gaur egun erabiliena Git da [12].

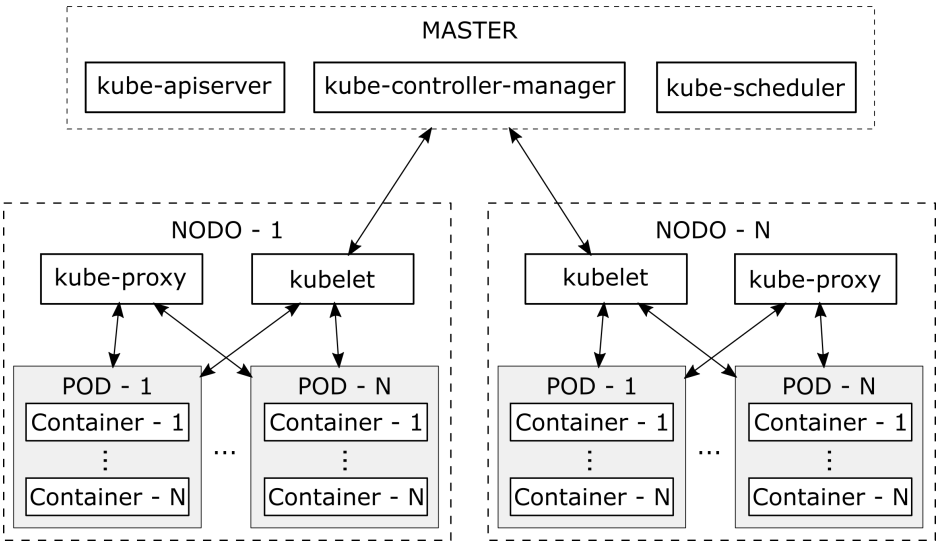
[2] *Bare metal server:* Bare metal zerbitzari edota kluster bat bere software eta hardware propioa dituen zerbitzari bat da. Gaur egun termino hau erabiltzen da birtualizazioan eta hodei-ostalaritzan (*cloud hosting*) oinarritutako zerbitzarietaz ezberdintzeko.

[3] Hodeieko zerbitzuen hornitzaile bat hodeian oinarritutako plataforma, azpiegitura, aplikazio edota biltegiratze-zerbitzuak eskaintzen dituen enpresa bat da [13].

Kubernetes-en Masterra osatzen dute eta normalean "master" izena jasotzen duen nodo bakar batean exekutatzen dira[4]. Kubernetes-ekin elkarrekintza bat izatean, master nodoaren aurka lan egiten da: eskatutako egoera interpretatu eta aztertzen du, egoera horretako objektuak zein makinatan exekutatuko diren erabaki eta haien egoera kontrolatuko du. Funtsean, hori da aipatutako prozesuen eginkizuna [15, 16, 17].

Masterrak ez diren nodoetan Kontrol Planoko beste bi prozesuak exekutatzen dira: *kubelet* —Kubernetes-en Masterrarekin komunikatzen dena— eta *kube-proxy* —nodo bakoitzean Kubernetes-en zerbitzuak inplementatzeko proxy zerbitzaria[5]—. Kubernetes-en konfigurazioaren inguruko datu guztiak jasotzen dituen giltza-balio datu-basea da Kontrol Planoaren azken elementua: *etcd* [18].

Laburbilduz, 2.1. irudian ikus daitekeen nagusi-morroi egituran oinarritzen da Kubernetes (nagusiari, sarritan, *master* izenaz deitzen zaio). Klusterraren barruan exekutatu nahi diren elementuak masterrean finkatu ostean, bertako Kontrol Planoko zerbitzuek elementu horiek hasieratu eta martxan daudelaz arduratuko dira etengabe. Horrela, langile nodoren batean arazorik egonez gero, bertan exekutatzen ziren baliabideak beste nodoetara automatikoki mugituko dira.



2.1. irudia: Kubernetes-en oinarrizko elementuek osatzen duten egituraren diagrama sinplifikatua.

2.2.3 Objektuak

Lehen aipatu denez, sistemaren desiratutako egoera deskribatzeko Kubernetes-en objektuak erabiltzen dira: edukiontzitan sartutako zein aplikazio dauden martxan, aplikazio horiek eskura dituzten baliabideak eta haien funtzionamenduaren inguruko politikak (berrarabiarazteko edo eguneratzeko politikak, adibidez) finkatzen dituzte. Objektu horiek sortu ostean, Kubernetes etengabe egongo da lanean objektuak existitzen direla ziurtatzeko. Honakoak dira oinarrizkoak:

[4]Eskuragarritasun eta erredundantzia arrazoiengatik masterra bikoiztuta egon daiteke.

[5]Proxy zerbitzaria: zerbitzu baten erabiltzailearen eta zerbitzariaren artean bitartekari lanak egiten dituen software edo gailua.

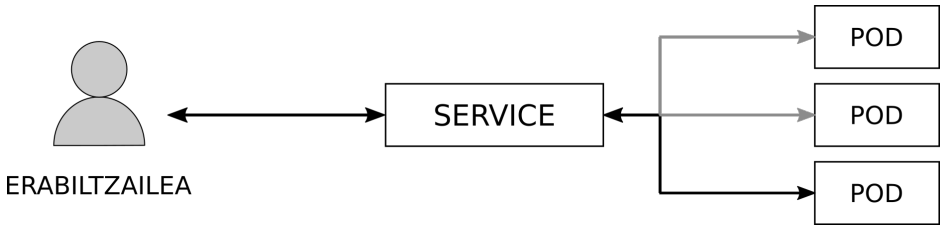
Pod

Pod bat Kubernetes-eko aplikazio baten exekuzio-unitaterik txikiena da [19]. Pod batek aplikazioaren edukiontziak, biltegiratze-baliabideak, IP helbide bakar bat eta edukiontzien funtzionamendua finkatzeko arauak biltzen ditu. Pod bakoitzak aplikazioaren kopia bakarra exekutatzen du. Aplikazioaren kopia asko exekutatu nahi izanez gero, nahi adina Pod sortu behar dira: horri *erreplikazioa* deitzen zaio eta normalean erreplikatutako Pod-ak talde bezala sortu eta kudeatzen dira *Controller* (hemendik aurrera "Kontroladore") izeneko abstrakzioen bidez.

Kontuan izan behar da Pod-ak aldaezinak eta, ondorioz, iragankorrak direla: Pod batek arazoren bat duenean edota edukiontziak eguneratzean, Pod-a ezabatu eta berri bat abiarazten da.

Service

Service objektuek Pod-etako zerbitzuetara sartzea ahalbidetzen dute. Pod-ak iragankorrak direnez, normalean Service-ek Pod-en *selector* eremua erabiltzen dute nahi diren Pod-ak aukeratzeko eta horiei IP eta portu finkoak esleitzeko [20]. Service eta Pod-en arteko elkarrekintza 2.2. irudian beha daiteke: erabiltzailearen eta Pod-en arteko bitartekari lanak egiten ditu Service-ak, erabiltzailearen eskaerari erantzungo dion Pod-a hautatu eta harek sortutako emaitza itzuliz.



2.2. irudia: Service baten oinarrizko funtzionamenduaren diagrama sinplifikatua. Eskuragarri dauden Pod-ekin dituen konexioak grisez agertzen diren bitartean, erabiltzailearen eskaerak eta haren erantzunak jarraitutako bidea beltzez azaltzen da.

Service mota ezberdinak existitzen dira, aipagarrienak hauek izanik:

- ClusterIP: Service-a kluster barruan bakarrik eskuragarri ipintzen du, automatikoki hautatutako IP eta portuetan (hau defektuzko portaera da).
- NodePort: Service-a klusterretik kanpo erakusten du, nodo bakoitzeko IP-an eta portu estatiko batean.
- LoadBalancer: Service-a klusterretik kanpo erakusten du IP eta portu finkoen bidez.

Volume

Pod-en izaerak bi arazo planteatzen ditu biltegiratzeari dagokionez. Alde batetik, edukiontzien barruko fitxategiak Pod-a bizirik dagoenean bakarrik existitzen dira, Pod-a berrabiaraziz gero galduz. Bestalde, askotan beharrezkoa da Pod baten barruan edukiontzi ezberdinek elkarren artean fitxategiak partekatzea [21]. *Volume* objetuek arazo horiei aurre egiten diete: Pod-etatik bereizita dauden biltegiratze-objektuak dira, independenteki sortu eta ezabatzen direnak eta Pod ezberdinetatik kontsulta daitezkeenak. Volume mota asko daude, erabilera eta kasu ezberdinei egokitzen direnak [21].

Namespace

Namespace-ak kluster baten barruan sortzen diren kluster birtualak dira [22]. Proiektu ezberdinak banatzeko erabiltzen dira, klusterraren kudeaketa erraztuz. Objektuak biltzen dituen karpeten moduan uler daitezke: Kontrol Planoko Pod guztiak *kube-system* Namespace-aren barruan exekutatzen dira, adibidez.

Secret

Secret-ak informazio sentikorra (pasahitzak, esaterako) gordetzeko erabiltzen diren objektuak dira [23]. Pod-etatik eskuragarri egon daitezen, edukiontzitan muntatutako Volume-etan fitxategi bezala edota edukiontziaren ingurune-aldagai bezala (*environment variable*[6], ingelesez) atxiki daitezke.

2.2.4 Kontroladoreak

Kontroladoreak aurreko atalean azaldutako objektuetan oinarritzen dira eta funtzionalitate gehigarriak eskaintzen dituzte. Jarrian elementu horien oinarrizko ezaugarriak zehazten dira:

ReplicaSet

Pod-en erreplika kopuru bat mantentzeaz arduratzen den elementurik sinpleena [24].

Deployment

Pod eta ReplicaSet-en eguneraketa deklaratiboak kudeatzen dituzte [25]. Desiratutako elementuen egoera definitu ostean, Deployment-aren Kontroladorea elementu horiek modu kontrolatu batean sortzeaz edota eguneratzeaz arduratzen da.

StatefulSet

Deployment-en antzera lan egiten du baina sortutako Pod-ak modu ordenatuan sortu eta bakarrak izateaz arduratzen da [26]. Pod horiek ez dira trukakorrak elkarren artean, berrabiarazpenetan ere galtzen ez den identifikadore bat baitaukate.

DaemonSet

Nodo guztietan Pod baten kopia bat egoteaz arduratzen den elementua [27].

Job

Job batek Pod bat edo batzuk sortzen ditu eta haien lana ondo amaitzen dutelaz arduratzen da [28]. Job-ak behin bakarrik exekutatu behar diren lanak burutzeko erabiltzen dira, beste Pod batzuek behar duten ingurunea prestatzeko, adibidez.

[6] *Environment variable*: Ordenagailu batean exekutatzen diren prozesuetan eragina eduki dezaketen aldagai dinamikoak dira, prozesua exekutatzen den ingurunearen parte direnak.

2.3 Erabilitako tresnak

Azken urteetan, Kubernetes-en inguruan plataforma ezberdinez osatutako ekosistema aberats eta konplexua sortu da. Tresna horiek sistemaren kudeaketa errazteaz gain, funtzionalitate gehigarriak ere eskaintzen dituzte. Atal honetan plataformen oinarrizko helburuak bakarrik azalduko dira, 3. atalean haien funtzionamenduan sakonduz.

Aipatzekoa da tresna gehienak Kubernetes-en agerpenaren aurretik ere existitzen zirela eta ez daudela Kubernetes-ekin bakarrik lan egiteko pentsatuta, independenteki edota beste ingurune batzuetan erabil baitaitezke.

2.3.1 Kubeadm

Kubeadm Kubernetes-en kluster bat sortzeko erabil daitekeen tresna da. Erabiltzeko erraza izateaz gain, ingurune oso ezberdinetan lan egiteko aproposa da (arkitektura ezberdinetako makinak onartzen ditu eta hodeiko baliabideak kudeatzeko ere erabil daiteke). Klusterreko nodo guztietan instalatu ostean, masterrean Kontrol Planoa hasieratu eta gero, gainontzeko nodoak hari lotu behar zaizkio Kubernetes-en klusterra sortzeko [29].

2.3.2 Docker Engine

Docker Engine edukiontzitan sartutako aplikazioak exekutatzeko softwarea da. Kasu honetan, nodoetan Pod-en barruan aurkitzen diren edukiontziak exekutatzeko erabiltzen da. Docker-en edukiontziak 2013an argitaratu ziren Docker Engine kode-irekiko proiektuaren barruan eta edukiontzitan exekutatutako aplikazioentzako plataforma estandarra izateaz gain, irudiak arinak eta seguruak direnez, antzeko plataformen artean popularrena da [30, 31]. Irudi horiek Dockerfile izeneko konfigurazio-fitxategien bidez adierazi eta sortzen dira.

Aipagarria da Docker Engine eta bere irudien filosofia lehen azaldutako Kubernetes-en kontzeptuekin bat datorrela, deklaratiboki sortutako irudi aldaezinak baitira.

2.3.3 Flannel

Flannel Kubernetes-en sarearen kudeatzaile bat da, arkitektura anitzetako klusterretan lan egin dezakeena [32]. Sarearen kudeaketa Kubernetes-en funtsezko alderdia da. Makina berdinetan aplikazio ugari exekuta daitezkeenez, arazo handienetako bat aplikazioek portu ezberdinak erabiltzen dituztela ziurtatzean datza, bestela gatazkak sor baitaitezke. Horri aurre egiteko inplementazio ezberdinak erabil daitezke[7], horien artean Flannel egonik.

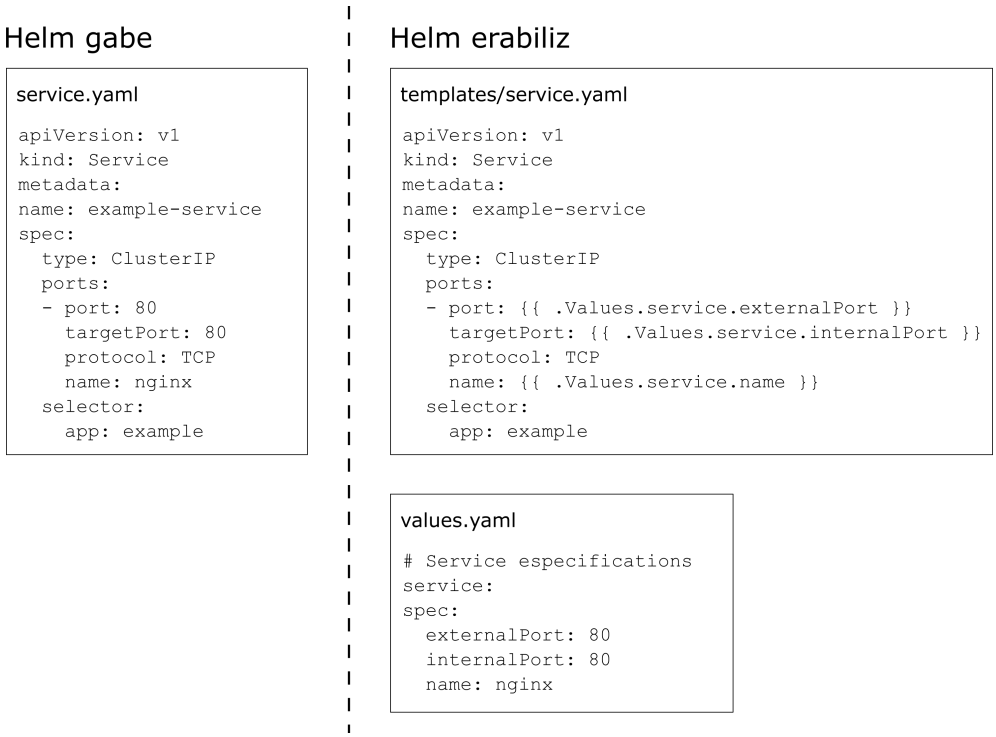
[7] Beste aukeren artean ezagunenak Calico edo Weave Net izan daitezke, baina haiek zenbait konpatibilitate arazo dituzte MetalLB-rekin [33].

2.3.4 MetalLB

MetalLB bare metal klusterretan *load balancing*-a[8] (hemendik aurrera "zama-banaketa") inplementatzen duen plataforma da. Zama-banatzaile batek sarrera-puntu estatiko bakarra eskaintzen du aplikazioa osatzen duten Pod guztietara iristeko. Kubernetes-ek, defektuz, hodeiko plataformetan oinarritutako klusterretan bakarrik jartzen ditu martxan LoadBalancer erako Service-ak. Bare metal klusterretan horrelako elementuak erabiltzeko, beharrezkoa da MetalLB erabiltzea [35]. LoadBalancer-ek erabil ditzaketen IPak MetalLB-en finkatzen dira, Service-a sortzean horietako bat erabiliz. Horrela, aukeratutako aplikazioa beti IP eta portu batean egongo da eskuragarri.

2.3.5 Helm

Helm Kubernetes-en argitaratutako aplikazioen kudeaketa errazten duen plataforma da [36]. Lehen azaldu denez, Kubernetes-eko objektuak konfigurazio-fitxagien bidez sortzen dira. Aplikazioak garatu ahala, ohikoa da Pod mota ezberdinak eta Service ugari izatea. Eguneraketak egitean fitxategi asko editatu behar izatea suposatzen du horrek. Helm Chart-en bidez fitxategi guzti horiek txantiloi bihur daitezke, 2.3. irudiko "templates/service.yaml" fitxategian ikus daitekeen bezala. Chart-etan definitutako eremuen balioak aipaturiko irudiko "values.yaml" fitxategiaren antzeko batean finkatzen dira, aplikazio osoaren konfigurazioa fitxategi bakar baten bidez burutuz. Horregatik, Helm oso erabilgarria da aplikazioak partekatzerako orduan eta, ondorioz, hemen azaldutako plataforma askoek haien softwarea banatzeko erabiltzen dute.



2.3. irudia: Service baten konfigurazio-fitxategiak Helm gabe eta Helm erabiliz. Helm bidez aplikazio baten parametro guztiak "values.yaml" bezalako konfigurazio-fitxategi bakarretik finka daitezke.

[8] *Load balancing*: lan multzo bat makina anitzez osatutako sistema batean banatuz sistemaren eraginkortasuna handitzea bilatzen duen prozesua [34]. Kubernetes-en kasuan LoadBalancer Service-ek jasotako deiak aplikazioaren Pod ezberdinen artean banatzen dituzte.

2.3.6 Redis

Redis RAM memorian gordetzen den datu-egitura biltegi bat da, datu-base, cache edota mezu-broker[9] bezala erabil daitekeena [38]. Kasu honetan datu-base lanak burutuko ditu, Redis Cluster-ak datuen banaketa eta Redis Sentinel-ak eskuragarritasun handia eskaintzen baitute.

Redis Cluster-ak nagusi-morroi egitura inplementatzen da: Redis-en nodo batzuk master-ak izango dira eta gainontzekoak haien morroi bezala definitzen dira. Horrela, datuak Redis-en nodo ezberdinen artean bana daitezke, azpitalde batek huts eginez gero ere lanean jarraituz. Idazketa-prozesuak master-en aurka egiten dira eta idatzitako informazioa master horren morroietara banatzen da. Morroi nodoetan, aldiz, irakurketa-prozesuak bakarrik daude baimenduta [39].

Inplemetazio horren ezaugarririk interesgarriena Redis-en master batek huts egitean bere morroietako batek masterraren postua hartzen duela da, irakurketa- eta idazketa-eragiketen eskuragarritasuna bermatuz. Aipaturiko ordezkapena Redis Sentinel aplikazioaren bitartez egiten da. Sentinel-ak Redis-en master eta morroien funtzionamendua etengabe behatzen du eta masterrean akatsik topatzean, hura ordezkatzeko beharrezko prozedura jartzen du martxan [40].

Nahiz eta proiektu honen helburua ez izan, Redis oso aproposa da Gauzen Internet-eko (ingelesezko *Internet of Things* (Iot)) gailuetan erabiltzeko, memoria eta CPU gutxi kontsumitzeaz gain (Raspberry Pi plaketan exekuta daiteke, adibidez) Redis-en bosgarren bertsioa datu-transmisioaren egitura denbora-serie aplikazioetarako diseinatu baita [41].

2.3.7 Istio

Istio zerbitzuak konektatu, babestu, kontrolatu eta behatzeko plataforma da. Aplikazioen osagai ezberdinak zerbitzu-sare baten bidez kudeatzen ditu, trafikoaren kontrola, zama-banaketa eta prozesuen inguruko datuen bilketa eta erakusketa eskainiz, besteak beste [42]. Ezaugarri guzti horiek aplikazioen garapenerako eta mantenurako eskaintzen dituzten onuren ondorioz, oso erabilia da enpresa eta erakunde handietan [43].

2.3.8 Kiali

Kiali Istioren zerbitzu-sareak behatu eta kontrolatzeko erabiltzen den plataforma da. Sarearen inguruko informazioa biltzeaz gain, sarearekin lan egiteko interfaze-grafiko ezberdinak eskaintzen ditu eta konfigurazio-akatsak identifikatzen ditu [44].

2.3.9 Prometheus

Prometheus klusterraren eta bertan exekutatzen diren aplikazioen inguruko informazioa biltzea ahalbidetzen duen kode-irekiko plataforma da. Denboran zehar datuak jasotzeko aukeraz gain, haien arteko operazioak burutzeko, bisualizatzeko, biltegiratzeko eta beste plataformetara exportatzeko aukerak eskaintzen ditu [45].

[9] Mezu-broker bat (ingelesezko *message broker*) bi sistema ezberdinen arteko mezuak lenguaia batetik bestera itzultzen dituen bitartekari bat da [37].

2.3.10 Grafana

Grafana milaka enpresek haien azpiegiturak eta aplikazioak behatzeko erabiltzen duten plataforma da [46]. Datu-iturri ezberdinekin lan egin dezake (horien artean, Prometheus) eta jasotako datuak malgutasun handia eskaintzen duten interfaze-grafikoen bidez erakuts ditzake. Grafana-k abisu-sistema bat ere inplementatzen du, aurretik finkatutako baldintza batzuk betetzean aktibatzen dena.

3. kapitulua

Garapena: azterketa praktikoa

Atal honetan bare metal kluster batean Kubernetes instalatu eta konfiguratuko da. Jarraian, teorian landutako kontzeptuak praktikan ipintzeko eta Kubernetes-en oinarrizko objektuen funtzionamendua probatzeko, aplikazio baten garapena eta mantenua simulatuko dira. Garapena etapa ezberdinetan banatuko da: aplikazioaren eta Kubernetes beraren konfigurazioak pixkanaka zailduko dira, azken urratsetan teorian azaldutako plataforma ezberdinak integratuz. Prozesu horretan erabilitako tresna guztien bertsioak A eranskinean jasotzen dira eta konfigurazio-fitxategi guztiak proiektuaren GitHub-eko biltegian aurki daitezke: jongablop/OlimpoServer.

3.1 Klusterraren inplementazioa

3.1.1 Erabilitako gailuak

Inplementatuko den klusterra lau nodoz osatutako bare metal zerbitzari heterogeneo[1] bat izango da. Kubernetes-en Kontrol Planoa master-nodo bakarrean exekutatuko da eta gainontzeko hirurak langile-nodoak izango dira. Master bezala AMD64 arkitekturako prozesadorea eta 4GB RAM dituen ordenagailua erabili den bitartean, langile-nodoak heterogeneoagoak dira: langiletako biek AMD64 prozesadoreak eta 4GB RAM dituzte eta hirugarrena, aldiz, 4GB-eko RaspberryPi 4 Model B bat da, ARM arkitekturako prozesadorea duena.

Gailu horien bidez erabilera oso anitzak izan ditzakeen azpiegitura bat simulatu da. Raspberry-ak merkeak direnez, klusterraren kostu gutxiko eskalagarritasun eta mantenua bermatzen dute. Gainera, ARM prozesadoreak IoT-en oso erabiliak direnez, nodo hauek sentsore ezberdinen datuak jaso eta prozesatzeko erabil daitezke. Beste nodoetako ordenagailuak, aldiz, ohikoagoak eta fidagarriagoak dira eta aplikazio ezberdinak aldi berean zerbitzatzeko potentzia nahikoa dute.

3.1.2 Sarearen konfigurazioa

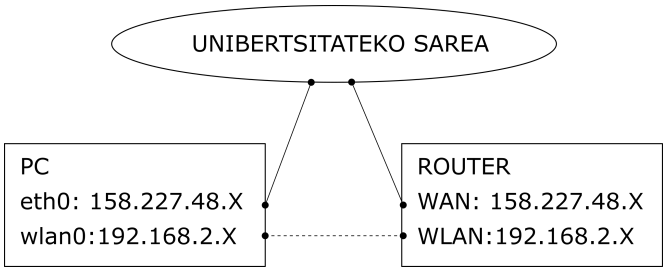
Sarearen konfigurazioa da bare metal kluster bat inplementatzeak dakarren zailtasunik handiena, ez bakarrik nodo guztiak fisikoki interfaze ezberdinen bidez lotu behar direlako, baizik eta, klusterrean argitaratutako aplikazioak kanpotik eskuragarri egon daitezen, Kubernetes bera zein instalatutako plataformak (segurtasunarekin eta trafikoaren

[1] Arkitektura ezberdinetako prozesadoreak dituzten makinez osatutako zerbitzaria.

kudeaketarekin erlazionatutakoak, batez ere) bereziki konfiguratu behar direlako.

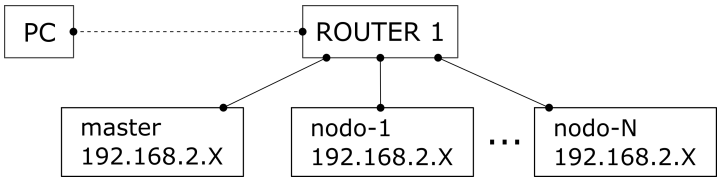
Klusterra fisikoki laborategi batean ipini denez, laborategiko sarearen egoerak guztiz baldintzatu du klusterraren barne-sarearen inplementazioa. Bi izan dira konfiguratzerako orduan finkatutako helburu nagusiak: alde batetik, lortutako sistema produkzioan, ingurune erreal batean, erabiliko litzatekeen baten antzekoena izatea eta, bestetik, laborategiko sarearekiko ahalik eta independenteena izatea.

Azken helburu hori betetzeak lanaren errepikakortasuna eta sendotasuna bermatuko lituzke: behin klusterra laborategian lan egiteko moduan konfiguratu ostean, aldaketa minimo batzuk egitea nahikoa izango litzateke beste ingurune batean martxan ipintzeko.



3.1. irudia: Laborategiko sarearen diagrama. Lerro etenak haririk gabeko konexioak adierazteko erabili dira eta lerro jarraituak kable bidezko konexioak adierazteko.

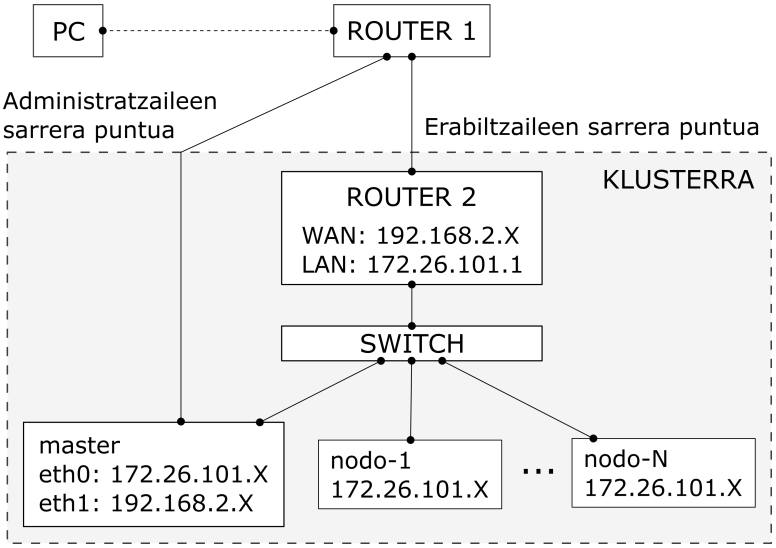
Laborategiko sarean bi elementu nagusi daude, 3.1. irudian erakusten denez: ordenagailu bat eta router bat. Biak, noski, unibertsitateko sarera (eta, ondorioz, Internetera) konektatuta daude. Kable bidez unibertsitateko sarera lotuta dago PCa, bere IPa finkoa izanik. Abantaila handi bat eskaintzen du horrek: sarearen egitura egoki batekin eta PCa bera ondo konfiguratuz gero, posible izango litzateke ekipo hori klusterrerako sarbide bezala erabiltzea. Gainera, ordenagailu hori routerrera Wi-Fi bidez konektatzen denez, posible da PCaren bitartez routerrera lotutako gailu guztiak ikustea.



3.2. irudia: Sarearen inplementazio sinple baten diagrama.

Egoera horren aurrean, klusterra osatzen duten nodo guztiak laborategiko routerrera konektatzea aukera bat izango litzateke (ikus 3.2. irudia). Horrela, klusterreko nodo guztiak sarean eskuragarri egongo lirateke eta plateatutako sistemak laborategiko routerrarengan menpekotasun osoa izango luke. Inplementazio horrek ez lituzke lehen finkatutako helburuak beteko, beraz.

Hori hobetzeko asmoz, IP eremu ezberdin batean lan egingo duen bigarren router bat ipintzea erabaki da, 3.3. irudian ageri denez. Gainera, nodoen arteko komunikazioak routerrean sortutako lan-karga txikitzeko switch bat gehitu da router horren eta klusterreko nodoen artean.



3.3. irudia: Klusterrera sartzeko bi sarrera dituen inplementazioaren diagrama.

Inplementazio berri horretan klusterreko nodo guztiek azken router horrek emandako IPak izango dituzte, 3.3. irudian azaldu bezala. Horrela, nahiz eta klusterraren kokapena aldatu eta bigarren router hori beste sare batera konektatu, kluster osoko sarea martxan ipintzeko nahikoa izango litzateke routerra konfiguratzea (bere IPa aldatzea).

Gainera, Kubernetes-ek klusterreko routerraren eta nodoen artean *firewall*[2] bat inplementatzen duenez, beharrezkoa izan da masterrari beste sare-txartel bat ipintzea, laborategiko routerrari zuzenean lotzeko. Firewall-ak kluster kanpotik barrura doazen konexio guztiak eteten ditu, bereziki baimendutako portu batzuetatik doazenak izan ezik. Klusterraren sarearen barrutik egindako dei guztiak, nahiz eta Kubernetes-en klusterraren kanpo dauden makina batetik egindakoak izan, baimenduta daude. Orokorrean SSH zerbitzuetan[3] erabiltzen diren 22 portua blokeatuta dago, hori irekita egotea segurtasun-arazo larri bat izan baitaiteke [47]. Masterreko bigarren txartela laborategiko ordenagailua dagoen sarera konektatzean ordenagailu horretatik masterrera sartzea ahalbidetzen da.

Konfigurazio horrekin klusterra bi sareetara konektatzen da: bat zerbitzuak emateko (klusterrean argitaratutako aplikazioak erakusteko) eta bestea mantenurako. Egitura hori ingurune profesionaletan erabiltzen da, nahiz eta zenbaitetan mantenurako sarea *Virtual Private Network* (VPN)[4] baten bidez konfiguratu [48, 49].

Plantetutako egiturak hasieran finkatutako helburu guztiak betetzen dituenez, hori da hemendik aurrera erabiliko dena.

[2] *Firewall*: Firewall bat, trafiko jakin bat sare batera sartu edota saretik irteten eragozten duen gailua da. Sarearen kanpoko erasoak ekiditeko erabiltzen dira normalean.

[3] SSH: Babestutako konexio baten bidez urruneko zerbitzari batera sartzea ahalbidetzen duen protokoloari eta hura inplementatzeko programari SSH deritzo.

[4] *Virtual Private Network (VPN)*: Sare publiko baten barruan sare pribatu bat hedatzea ahalbidetzen duen teknologia da: horren bidez, sare ezberdinetan konektatutako gailuek, sare berdinean baleude bezala burutu ditzakete datu-transmizioak.

3.2 Kubernetes-en oinarrizko konfigurazioa

Zenbait konfigurazio egin behar dira Kubernetes instalatu aurretik. Sareari dagokionez, nodoei IP finkoak esleitzea ezinbestekoa da klusterrak funtzionatu dezan. Kasu honetan 172.26.101.121 - 172.26.101.124 IPak erabili dira, lehena masterraren helbidea izanik. Bestalde, Kubernetes inplementatzeko Docker hautatu da edukiontziak exekutatzeko plataforma bezala.

Kubernetes-en instalazioa burutzeko kubeadm erabili da eta sarearen kudeaketaz arduratzeko Flannel instalatu da. Kubernetes-en LoadBalancer Service-ak probatu ahal izateko MetalLB instalatu da eta Service horiek izango dituzten IP-ak 172.26.101.240 - 172.26.101.250 eremuan egoteko konfiguratu dira, klusterrean nodo gutxi egotea aurreikusten denez eremu hori libre egongo baita. Nodoen kopurua oso handia balitz eta IP horiek eskuragarri egon beharko balira, routerraren eremuaren barruan dagoen beste azpieremu bat aukeratzea posible izango litzateke.

3.3 Adibide baten argitarapena

Ohiko egitura duen aplikazio bat argitaratzea da atal honen helburua, teorian azaldutako plataformen konfigurazioa eta elkarren arteko integrazioa landuz.

Garrantzia Kubernetes-ek eta beste plataformen funtzionamenduaren azalpenek izateko, erabiltzaile-interfaze sinple bat duen web-zerbitzu bat sortuko da. Erabiltzailea web helbide batera sartzean, webgunea zenbatetan bisitatu den jasotzen duen kontagailua eguneratuko da eta bere balioa erakutsiko du.

Nahiz eta erabiltzailearen ikuspuntutik zerbitzu ia tribial bat izan, horrelako aplikazio bat Kubernetes-en barruan garatzeak mikro-zerbitzuen (web zerbitzuak eta datubaseak, kasu honetan) funtzionamendua ulertzeko aukera ematen du eta, gainera, inplementatuko diren tresna eta egiturak kasu askoz konplexuagoetan aplikagarriak izango dira.

Azalpenak sinpleak eta jarraiterrazak izateko, aplikazioaren argitarapena pausoka egingo da, une bakoitzean topatutako arazoei aurre eginez eta aplikazioaren egitura pixkanaka zailduz. Amaieran lortuko den aplikazioak produkzio-ingurune batean erabiliko litzatekeen baten oso antzekoa izatea bilatuko da.

Bi atal nagusi izango ditu aplikazioak: *frontend* edo interfazea eta *backend* edo motorea. Lehen azaldu denez, interfazea erabiltzailearekin kontaktuan dagoen zatia da eta, motorea, aldiz, datuen prozesaketaz arduratzen dena.

Bi atal horiek, nahiz eta elkarrekin aplikazio bakar bat osatu, normalean independenteki garatzen dira eta eskaera kopuru ezberdinak jasotzen dituzte. Biak Pod berdinetan egongo balira, interfazea aldatzean motorearen argitarapen berri bat egin beharko litzateke eta ezin izango litzateke zerbitzu bakoitza bere aldetik eskalatu. Horregatik, horrelako aplikazio bat sortzean ezinbestekoa da bi atal horiek Pod ezberdinetan sortzea.

Kasu honetan, interfaze bezala Flask-en oinarritutako web-aplikazio bat erabiliko da.

Flask Python-en idatzitako lan-ingurune minimalista bat da, web-aplikazioak modu erraz eta sinple batean sortzea ahalbidetzen duena [50].

Klusterreko nodoetan banandutako eta eskuragarritasun-handiko datu-base bat inplementatuko da, Redis-en oinarritutakoa [51]. Datu-base horren helburua datuak modu iraunkor batean mantentzea izango da: nahiz eta Pod batek huts egin, gordetako datuak eskuragarri egon beharko dira ahalik eta denbora gehienean.

3.3.1 Lehenengo iterazioa: aplikazioaren oinarrizko argitarapena

Ezer baino lehen, "app" izeneko Namespace berri bat sortuko da aplikazioa klusterrean bizi diren gainontzeko aplikazioetatik bereizita mantentzeko. Nahiz eta derrigorrezkoa ez izan, pare bat abantaila eskaintzen ditu horrek: alde batetik, Namespace barruko Pod guztiei aldi berean etiketak ipintzea ahalbidetuko du, gainontzeko Pod-ei eragin gabe. Bestetik, probak amaitzean nahikoa izango da Namespace-a ezabatzea aplikazioaren objektu guztiak aldi berean ezabatzeko, klusterraren mantenua asko erraztuz.

Redis argitaratzeko Helm erabiliko da, GitHub-eko Helm-en biltegitik plataformaren garatzaileek prestatutako taulak eta konfigurazio-fitxategia deskargatuz[5] [54]. Redis produkzioan erabili ahal izateko behar diren parametroak daude definituta YAML fitxategi horretan: sortuko diren Pod-en kopurua, esleitutako diskoen edukiera edota segurtasun-baldintzak konfigurazio-fitxategi hori modifikatuz kontrola daitezke.

Konfigurazio horri pare bat modifikazio egingo zaizkio: datuak klusterreko disko gogorretan gordetzeko konfiguratu, Redis Cluster eta Sentinel aktibatu (ikus 2.3.6. atala) eta sortzen diren objektu guztiak "app" Namespace-an kokatuko dira. Gainera, hasierako proba hau egiteko bi morroi baino ez dira erabiliko. Motoreak datu-basea RAM memorian eta disko gogorrean gordeko duenez, *NodeSelector* hautatzailearen bidez Redis-eko Pod guztiak AMD64 prozesadorea duten nodoetan (PCetan) bakarrik exekutatzea finkatu da, aplikazioaren atal hori ahalik eta sendoena izan dadin. Nodoari eta nahi diren Pod-ei aipaturiko hautatzailearen giltza-balio bikote berdinak esleituz Pod-ak zein nodotan exekutatuko diren aukera daiteke.

Datu-basea disko gogorretan gordetzeko, beharrezkoa da Volume-ak sortu eta Pod-ei lotzea. Redis instalatzean *PersistentVolume* (PV) motako Volume-ak Pod-ekin lotzen dituzten *PersistentVolumeClaim* (PVC) izeneko objektuak sortzen dira. Egitura horrekin, Volume horiek ez dira zuzenean Pod-aren barnean gordetzen, kanpoko baliabide bat bezala kudeatzen dira. Horrela, Pod-ak huts eginez gero, gordetako datuak ez dira galtzen.

Aipagarria da, Redis-en konfigurazioan disko zurrunen erabilera aktibatuz gero, Pod-ak ez direla hasieratzen beharrezko PV-ak sortu gabe (Redis-eko nodo bakoitzeko bat). Desiratutako egoera eta une horretako egoera etengabe konparatzeaz gain, desiratutako egoera egingarria den ala ez ere behatzen du Kubernetes-ek, aplikazioen hutsegitea sor dezaketen ekintzak ekidinez. Kasu honetan, Volume-ak sortzean Redis-eko Pod guztiak martxan ipintzen dira.

[5] Oharra: Helm proiektuaren egoeraren ondorioz, zeinak 2020ko martxoaren 9an aipaturiko biltegia zaharkitutzat hartu zuen [52], une horretatik aurrera Bitnami-k eskainitako taulak erabili dira [53].

Une hau aplikazioaren interfazea martxan ipintzeko egokia da, datu-baseak ondo funtzionatzen duela ziurtatu eta lehenengo probak egiteko. Interfazearen edukiontzia Dockerfile baten bidez sortu da (ikusi 2.3.2. atala) eta Docker-en erregistro batera igo da, nodoek irudia bertatik har dezaten. Urrats honen helburua datu-basearen argitarapena denez, interfazearen Pod bakarra sortu da. Pod hori ARM arkitekturako nodoetan exekutatzeko finkatu da, lehen datu-basearekin egindakoaren antzera. Interfazeek (eta, bereziki, erabilitako adibide tribial honek) pisu txikia izaten dute normalean eta Raspberry-ak oso egokiak izan daitezke lan horiek burutu eta beste nodoen lan-karga gutxitzeko. Flask-eko aplikazioa Python-eko fitxategi bakar bat exekutatzean abiaraz daitekeenez, fitxategi hau osorik *ConfigMap* izeneko Volume batean sartzea erabaki da, bere argitarapena erraztuz.

ConfigMap-ak Pod-en barruan muntatzen dira. Horrela, bertan definitutako Pythoneko programa edukiontziko memorian eskuragarri egongo da. Lehen aipatutako Dockerfilean, funtsean, edukiontzian sistema eragile arin bat [55] eta Python instalatzeko eskatzen da. Ostean, Flask-en instalazioa egiteko eta bere memoriako toki batean dagoen aplikazioa (ConfigMap-aren bidez bertan kokatu dena) exekutatzeko agintzen zaio.

ConfigMap-a sortzean Flask-eko aplikazioa Redis-eko datu-basearekin lotu behar da. Horretarako, 3 parametro nagusi behar dira: datu-basearen helbidea eta bertara konektatzeko erabili behar dituen portua eta pasahitza (ikus 3.4. irudia). Lehenengo biak Redis instalatzean agertutako mezutik lor daitezke: argitarapenak sortutako Service baten izena agertuko da bertan (Namespace-ko Service-en barruan eskuragarri egongo dena), hari konektatzeko portuak ere adieraziz. Sentinel-a aktibatu denez, bi portu egongo dira eskuragarri: 6379 eta 26379. Lehena irakurtzeko bakarrik erabil daiteke eta 26379 portua, aldiz, irakurri/idazteko erabil daiteke. Lehenengo proba honetan 26379 portua erabiliko da, datu-baseko datuak behar bezala eguneratzen direlaz ziurtatzeko.

Pasahitzari dagokionez, argitarapena egitean Secret objektu bat ere sortzen da. Secret horren barruan Redis-ekin konektatzeko pasahitza aurki daiteke, hain zuzen ere. Edukiontzi barruan pasahitza ingurune-aldagai bezala eskuragarri izateko nahikoa da Secret hori interfazearen Pod-ari gehitzea. Horrela, Flask-eko aplikaziotik erabili ahalko da.

Une honetan oso interesgarria da sortutako aplikazioa masterretik at eskuragarri ipintzea, Kubernetes-ek eskaintzen duen portuak berbidaltzeko (ingelesezko *port-forwarding* [6]) baliabidea erabiliz [56]. Ezer gehiago konfiguratu gabe probak egitea ahalbidetzen du horrek, aplikazioaren egoera egokia lortu arte oso erabilgarria dena. Service baten bidez aplikazioa eskuragarri ipintzea hurrengo urratsetan landuko da.

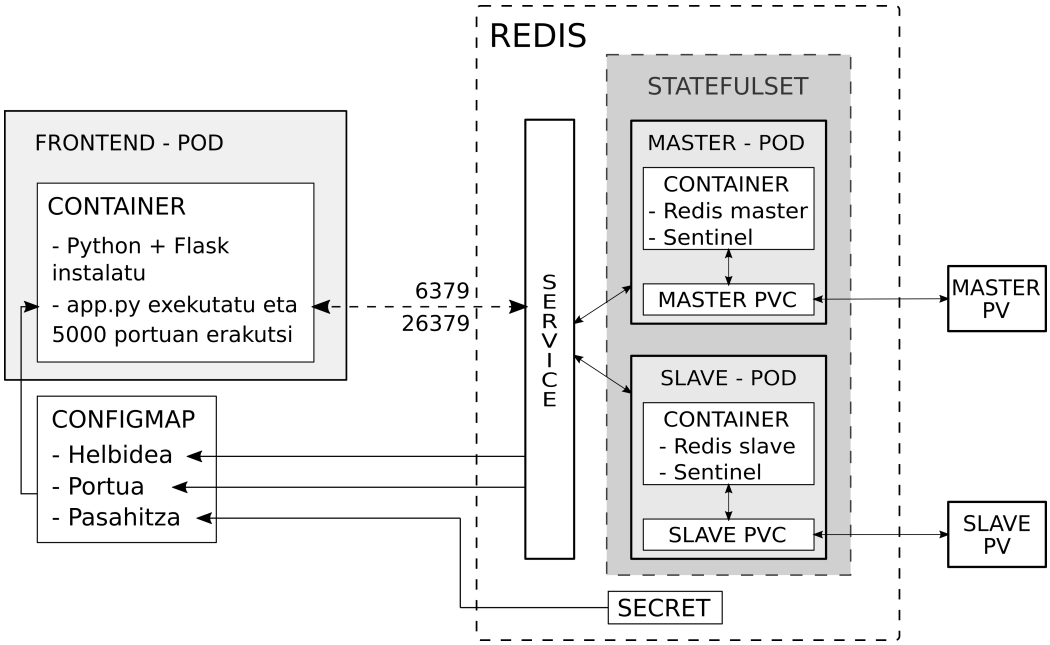
Flask-ek defektuz 5000 portura iritsitako eskaerei erantzuten dienez, klusterreko edozein nodotako 5000 portura iritsitako trafikoa aplikaziora berbidaliko da. Masterreko nabigatzailetik edota cURL[7] erabiliz http://localhost:5000 helbidera joz gero, konta-

[6] *Port-forwarding*: Sare bateko nodo baten portu bat nodo batetik beste batera berbidaltzearen ekintza. Horren bidez, kanpoko erabiltzaile bat sareko IP pribatu baten portura irits daiteke .

[7] cURL: fitxategien transferentziari bideratutako software proiektua bat da, libreria batez (libcurl) eta komandoen interprete batez (curl) osatutakoa. cURL-en erabilera nagusia fitxategien transferentzia edota gainbegiralerik gabeko operazioen segidak automatizatzea da eta erabiltzaile batek web nabigatzaile batean egindako ekintzak simulatzeko tresna aproposa izan daiteke.

gailu bat duen mezu labur bat agertuko da web nabigatzailean (*curl* komandoa erabiliz gero exekutatutako kontsolan egongo da ikusgai). Helbidera egindako eskaera bakoitzak kontagailuaren balioa eguneratuko du, guztira jasotako eskaera kopurua erakutsiz.

Aplikazioaren funtzionamendu eta konfigurazioa 3.4. irudian agertzen da laburtuta. Redis-i dagokionez, aipagarria da datuak gordetzen dituzten Volume-ak Redis-en instalazioaren independenteak direla eta, ondorioz, Redis-en konfigurazioan egidako aldaketek ez dietela zuzenean eragiten. Gainera, Redis-en Service-en inguruko ideia bat ere ematen da: Service bakar batek jasotzen ditu datu-basera doazen dei guztiak eta bertan Redis-en zein nodora berbidali erabakitzen da. Bestalde, interfazearen konfigurazioaren zehaztasunak ere agertzen dira: Flask datu-basearekin konektatu ahal izateko, beharrezkoa da ConfigMap-ean datu-basearen Service-aren izena eta bertara sartzeko portua eta pasahitza gordetzen duen ingurune-aldagaiaren izena finkatzea.



3.4. irudia: Aplikazioaren konfigurazio eta funtzionamenduaren diagrama sinplifikatua.

Lehen atal honi amaiera emateko Redis-en morroi berri bat argitaratu da, Redis-en konfigurazio-fitxategia aldatu eta Helm bidez eguneratuz. Aurreko kasuetan bezala, beharrezkoa izan da Volume berri bat sortzea morroiaren Pod-a martxan jartzeko. Behin morroi guztiak martxan daudenean Redis-en zergatia justifikatuko duen proba egin da: kontsola batetik zerbitzura eskaerak egiten ari diren bitartean[8] Redis-en masterra itzali da, eskuragarri egon behar diren kopien kopurua zeron finkatuz. Horrela, masterraren Pod-a itzali ostean ez da berriro berrabiaraziko. Redis-eko Sentinelek masterra desagertu delaz ohartu eta beste morroi bat jarri dute bere partez. Trantsizio hau bat eta hiru minutu bitartean gertatzen da eta denbora-tarte horretan erabiltzaileak errore bat jasotzen du (zerbitzuak 500 HTTP egoera-kodea itzultzen du: *Internal Server Error* [57]). Jarraian masterrak erreplika bat izateko finkatuz gero, masterra berriro ere martxan ipintzen da, baina morroi baten postua hartzen du, une horretako datuen egoera master berritik jasoz.

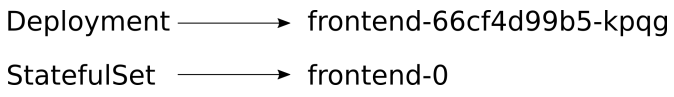
[8]Hori egiteko *watch* eta *curl* komandoak konbinatu dira, bi segundutik behin zerbitzuari eskaerak egiten joateko

3.3.2 Bigarren iterazioa: aplikazioa zerbitzu baten bidez erakustea

Bigarren pausu honetan egindako aplikazioa Service baten bidez argitaratzea bilatuko da, orain arte probak egiteko erabilitako portu-berbidalketa alde batera utzi eta sendoagoa den zerbitzu bat erabiliz.

Kasu honetan zama-banatzaile bat erabiliko da, aplikazioaren interfazera deiak egiteko sarrera-puntu bakarra eskainiko duenak. Zama-banatzaileak MetalLB-ren eremuan dagoen IP bat eta aukeratutako portu bat ipiniko ditu eskuragarri, 3.6. irudiko goikaldean ikus daitekeenez. Nabigatzaile edota kontsolatik helbide horretara sartuz gero, aplikazioak erantzungo du, lehen portu-berbidalketarekin egiten zuenaren antzera.

Aurreko urratsetako interfazearen Pod-a StatefulSet batekin ordezkatuko da hori egin ahal izateko. Aipatutako zama-banatzailea erabiltzeko eta interfazearen nahi adina kopia egiteko finka daiteke bertan. Interfazearen Pod-ek sortzeko jarraitzen duten ordenak garratzirik ez duenez eta Pod horiek ez dutenez memoria iraunkorrekin lan egiten, kasu honetan egokiena Deployment bat erabiltzea izango litzateke. Hala ere, StatefulSet bat aukeratu da Pod-ak ordenean sortu eta izen ulergarriak edukitzeak Kubernetes-en funtzionamendua ulertzeko lagungarria baita (ikus 3.5. irudia).



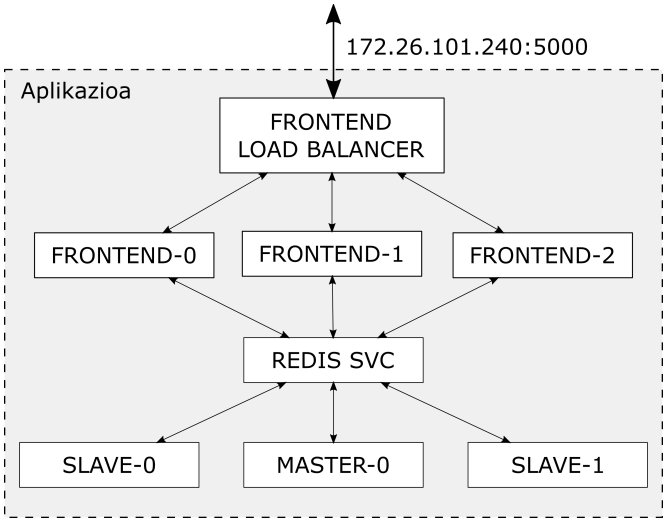
3.5. irudia: Deployment eta StatefulSet banen bidez sortutako Pod-en izenen adibidea.

Interfazearen funtzionamendua datu-basearenaren antzekoa izango da orain: interfazearen hiru kopia argitaratuko dira eta jasotako deiak hiru kopietako batera bidaliko ditu Service-ak, 3.6. irudian ageri denez. Horrelako argitarapen bat egitean, une oro argitarapenaren egoera eta administratzaileak finkatutakoa konparatzen ditu Kubernetes-ek. Horrela, Pod bat itzaliz gero, bi egoerak ezberdinak direlaz ohartu eta beste bat sortuko du.

Gainera, Flask-eko aplikazioan aldaketa txiki bat egin da zerbitzua zenbatetan ikusi den erakusteaz gain zein Pod jo den erakusteko. ConfigMap-a aldatzean Pod-ak ez dira bakarrik eguneratzen, beharrezkoa da denak berrabiaraztea. Hori egiteko modurik errazena StatefulSet-aren kopia kopurua zerora jartzea da, ostean berriro ere nahi bezainbeste kopia sortzeko. Horrela, Pod guztiak itzali eta berriro beste batzuk abiaraziko dira (kasu honetan 3), sortu berri den konfigurazioa kargatuz. Orain, honelakoa da nabigatzailean jasotzen den mezua:

```
Hello World! I'm running in Pod frontend-2 and I have been seen 972
    times.
```

Helbideari deiak egitean kontagailuaren zenbakia handituz joango da eta "frontend-X" aldatuz joango da zama-banatzaileak deia Pod batera edo bestera berbildaltzen duen heinean.



3.6. irudia: Zama-banatzailea konfiguratu ostean lortutako aplikazioaren arkitekturaren diagrama sinplifikatua.

3.3.3 Hirugarren iterazioa: aplikazioaren sendotzea

Kubernetes-ek eskaintzen dituen zenbait funtzionalitate aztertu eta inplementatzea izango dira atal honen helburu nagusiak. Jarraian ikusiko den bezala, Pod-en funtzionamenduarekin erlazionatutako parametroen balioak kasuan-kasuan hautatu beharko dira, hemen azaldutakoak aplikazio honi hobekien datozkionak baino ez izanik.

Aplikazioaren sendotasuna hobetzeak ez du esan nahi erroreak ekiditea bilatzen denik. Aplikazioen perfekzioa bilatzea ez dago Kubernetes-en filosofiaren atzean, baizik eta arazoak eta erroreak beti gertatuko direla onartuta, akats horiek aurreikusi, automatikoki detektatu eta aurre egiteko sistemak eraikitzea bilatzen du.

Atal honetan, beraz, egindako aplikazioaren interfazeak edota datu-baseak uneren batean partzialki zein guztiz kale egingo duela onartu da. Akats horien antzematea eta konpontzea ahalik eta azkarren egitea bilatu da, eskainitako zerbitzua eskuragarri ez dagoen denbora minimizatuz.

Redis-en konfigurazioari dagokionez, dokumentazio ofizialaren arabera, hiru master-morroi bikote argitaratu beharko lirateke gutxienez Redis-en kluster baten abantaila guztiak izateko [58]. Hala ere, Redis-ek ez du argitarapen hori egiteko soluzio errazik eskaintzen: adibide honetako lehen atalean ikusi den bezala, master baten eta bere morroien argitarapena berehalakoa den bitartean, beste aplikazio bat konfiguratu beharko litzateke master ezberdinen arteko interakzio egokia bermatzeko.

Lan honen izaera ez denez Redis-en inguruko lan monografiko batena, konfigurazio sinpleago bat inplementatuko da, kluster egoki batetik espero daitekeen portaera[9] imita dezakeena eta eskuragarri dagoen Kubernetes-en klusterraren mugapenak kontuan hartzen dituena.

[9]Master ugariko Redis-en kluster batean, idatzi zein irakurtzean latentzia arazorik ez egotea espero da: nahiz eta idazketa-prozesu asko aldi berean egin nodo guztietan aurki daitekeen informazioa berdina izanez. Datu-basearen zerbitzua nodoetako batek kale egitean ere eskuragarri egon beharko da.

Kasu honetan Redis-eko Pod-ak bi nodoetan baino ezin direnez exekutatu, master bakarra eta bere hiru morroi baino ez dira argitaratuko, nodo bietan binaka banatuz. Horrela, nodo bateko datu-basearen Pod bat hilez gero, berriro berrabiaraztean nodo horretan bertan egongo litzatekeen Pod-etik hartu ahalko luke datu-basearen une horretako egoera. Transmisio hori beti izango da sare bidezkoa baino azkarragoa eta, ondorioz, arazo gutxienak eman ditzakeena.

Orain arte erabilitako Redis-en instalazioaz gain, beste bat ere eskuragarri topa daiteke Github-en: Redis-ha [59]. Erabilgarritasun Handiko Redis-en konfigurazioak (ingelesezko *Redis High Availability*) une honetan argitaratutakoaren antzeko datu-base bat sortzen du, baina zenbait parametroen balioemgatik ezberdintzen dira. Parametro horiek finkatzen dute, hain zuzen ere, Redis-en instalazioak errore baten aurrean izango duen portaera.

Bi dira zehazki aldatu beharreko parametroak: lehenak, *downAfterMilliseconds* parametroak, masterrarekin konexioa galtzen denetik master berri bat aukeratzen hasi bitartean pasa beharreko denbora finkatzen du. Minutu bateko itxarote-denbora (defektuz finkatutakoa) hamar segundura murriztuz, arazoak gainditzeko erreakzio-denbora gutxitzen da. Aldi berean paraleloan egin daitezkeen sinkronizazio kopurua finkatzen du bigarrenak. Jatorrizko konfigurazioan bezala sinkronizazioa banaka egin beharrean hirunaka egitean, nodo guztiak aldi berean eguneratzen dira, guztietan jasotako datuak beti berdinak direla ziurtatuz.

Bi parametro horiek sintonizatuta datu-basearen ohiko funtzionamendua eta arazo baten aurrean izango duen portaera hobetu dira. Egindako aldaketen ondorioak kuantifikatzeko asmoz, masterraren akatsak simulatu dira: sistema osoa martxan dagoen egoera batetik hasita (Pod guztiak bizirik daudela), datu-baseko masterraren Pod-a itzali da. Bitartean, segundu bateko periodoarekin aplikazioaren zerbitzura deiak egin dira, zerbitzuak ondo erantzundako azken deiaren denbora eta berriro martxan hasi osteko ondo erantzundako lehenengo deiaren denborak jasoz. Horrela, zerbitzua eskuragarri egon ez den denbora-tartea kalkulatu da. Atal honetan egindako aldaketen eragina neurtu nahi denez, 3.3.2. ataleko konfigurazioarekin eta oraingoarekin egin dira neurketak, inplementazio bakoitzean bost aldiz errepikatuz eta lortutako emaitzak alderatuz.

Kalkulatutako denbora-tarteen bataz bestekoak kalkulatzeaz gain, desbideraketa estandarren balioak 3.1. adierazpenaren bidez kalkulatu eta emaitza guztiak 3.1. taulan jaso dira.

$$\mathrm{s} = \sqrt{\frac{1}{N-1}\sum_{i=1}^{N}(x_i - \bar{x})^2} \tag{3.1}$$

non N egindako neurketa kopurua, x_i neurketa bakoitza eta $\bar{x}$ neurketen batez bestekoa diren.

	t_1(s)	t_2(s)	t_3(s)	t_4(s)	t_5(s)	t_{bb}(s)	s (s)
3.3.2. ataleko konfigurazioa	47	54	56	64	56	55.4	6.07
3.3.3. ataleko konfigurazioa	46	47	14	46	9	32.4	19.16

3.1. taula: Inplementatutako bi konfigurazioek datu-basearen masterraren hutsegitetik berreskuratzeko behar izandako denborak. Datuak segundu bateko tartearekin hartu dira.

Aplikazioak datu-basearen masterraren hutsegite batetik berreskuratzeko behar izandako denborak hobetu dituzte egindako aldaketek, 3.1. taulan jasotako datuek argi uzten dutenez. Horren adierazpidea da inplementazio berriaren kasurik okerrenak (zeinetan zerbitzua 47 segunduz dagoen erorita) eta inplementazio zaharraren kasurik onenak hutsegite-denbora berdinak dituztela.

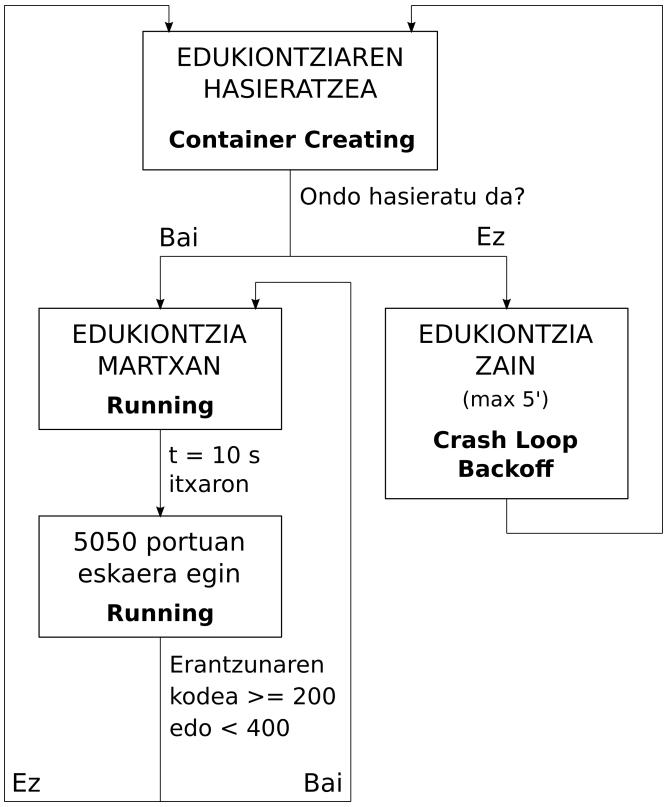
Aipatzekoa da ere, azken inplementazioarekin egindako neurketek azaldutako desbideraketa estandarra. Nahiz eta lan honen helburua ez den Redis-en funtzionamendua aztertzea, morroietako Sentinel-en erregistroak aztertzean desbideraketaren jatorria Sentinel-en arteko komunikazioa dela jakin da: Sentinel-ek master berria aukeratzeko egiten duten lehenengo komunikazioak huts eginez gero, berriro saiatu aurretik segundu batzuk zain gelditzen dira. Horregatik, denbora-tarte laburrak komunikazio hori hasiratik ondo egin den kasuari dagozkio (agian nodo berdinean exekutatzen ari diren bi Pod-en artekoa) eta beste tarteak, aldiz, komunikazioaren hutsegitearen osteko itxarote-denboraren ondorio dira.

Bestalde, interfazearen sendotzeari dagokionez, aplikazioen funtzionamenduaren inguruko datuak biltzeko edota trafikoa kudeatzeko erabilitako plataformetan ohikoa den "sidekar" egitura planteatu da [60, 61]. Horrela, Pod bakoitzean beste edukiontzi bat sortuko da interfazearen edukiontziaz gain. Edukiontzi horren barruan lehen erabilitako Flask-en oinarritutako aplikazioaren kopia bat abiarazi da, pare bat berezitasun dituenak: alde batetik, aplikazioak dei bat jasotzen duenean datu-basean gordetako datua irakurriko du, ezer idatzi gabe. Bestetik, aplikazioak 5050 portura egindako deiei baino ez die erantzungo. Datu-basearen egoeraren inguruko informazioa jasotzea ahalbidetzen du konfigurazio horrek, aplikazio osoaren funtzionamenduan eragin minimoa izanik.

Jarraian bi edukiontzi horientzat "bizitza-probak" konfiguratu dira (ingelesez *Liveness Probe*). Orokorrean, proba horiek ekintzaren bat egiten dute nahi den edukiontzia martxan dagoela egiaztatzeko: komando jakin bat exekutatu edota portu batean protokolo zehatz bateko eskaera bat egin, esaterako. Eskaera horren erantzuna hainbat aldiz okerra bada ("erantzun okerra" exekutatutako aginduaren izaeraren araberakoa izango da), edukiontzia berrabiarazi egingo da. Edukiontzia zenbaitetan segidan berrabiarazten bada edota ez bada berriro martxan ipintzeko gai, errorea emango du. Errore hori sortaraztea oso interesgarria izan daiteke: askotan, nahiz eta edukiontziak martxan egon, baliteke barne-errore bat gertatu izana eta ohiko funtzionamendua etenda egotea. Erabiltzailea arazo horretaz ohartu ezean, administratzaileak aplikazioa lanean ari dela ikusten du, Pod-ek ez baitute errorerik ematen. Bizitza-probak konfiguratzea funtsezkoa izan daiteke, ez bakarrik arazoak detektatzeko, baizik eta, zenbaitetan, arazo horiei automatikoki erantzuteko ere.

Zehazki, kasu honetan, 10 segundutik behin HTTP eskaera bat egingo diote 5050 portuari bi edukiontziek. Horrela, aplikazioa datu-basetik datuak irakurtzeko gai ez bada,

interfazeko bi edukiontziak berrabiarazi egingo dira edota Pod-a erroreko egoera batera pasako da[10]. Adibidez, interfazea berrabiarazi gabe Redis-en instalazio osoa aldatzean aipaturiko kasua gertatzen da, interfazearen Pod-ek ez baitute Secret-ean gordetako pasahitza berria automatikoki hartzen.



3.7. irudia: Edukiontzi bakar bat duen Pod baten bizitzaren zikloa azaltzen duen fluxu-diagrama sinplifikatua. Une bakoitzeko Pod-aren egoera letra lodiz adierazi da.

Edukiontzi bakarreko Pod baten etapa ezberdinak azaltzen dira 3.7. irudian ikus daitekeen fluxu-diagraman. Pod berdinean edukiontzi ugari egonez gero (atal honetan garatutako adibidean bezala), Pod-aren egoera edukiontzi guztien egoeren araberakoa izango litzateke (Pod-a "Running" egongo litzateke baldin eta soilik baldin bere barneko edukiontzi guztiak ondo exekutatzen arituko balira, esaterako). Diagraman ikusten denez, edukiontzia ondo hasieratuz gero 10 segundutik behin bere bizitza-proba exekutatuko du eta berrabiarazi egingo da proba horren emaitza egokia ez bada. Edukiontzia ondo hasieratzeko gai ez bada, berriro berrabiarazten saiatu aurretik zain egoteko egoerara joango da. Atzerapen horren iraupenak esponentzialki haziko dira (hasieran 10 s, gero 20 s, 40 s etab.) 5 minutuko mugara iritsi arte. Uneren batean edukiontzia hasieratzeko gai bada, 10 minutuz martxan egon beharko da atzerapen-kontagailu hori berrabiarazteko.

Hutsegiteak automatikoki konpontzeaz gain, egoera oker bati aurre egiteko erreakziodenbora hobetzea ere bilatu da. Horretarako, interfazearen Pod-ak itzali aurretik itxa-

[10] Aplikazioak eskaerari 200 edo handiagoa edota 400 baino txikiagoa den HTTP egoera kode batez erantzunez gero proba arrakastatsua izango da eta porrota izango da bestela.

ron beharreko denbora kendu da (edukiontziek itzalteko agindua jasotzen dutenetik Pod-a ezabatu arte 30 segundu igarotzen dira defektuz). Parametro hori kritikoa izan daiteke aplikazio batzuen kasuan, edukiontzi batzuek amaieratze-denbora luzea behar baitute haien lana behar bezala uzteko (datuak idazten dituzten edukiontziek, esaterako). Interfazearen edukiontziek arazo hori ez dutenez, berehala itzali daitezke. Interfazearen Pod-ek hutsegite baten ostean berrabiarazteko behar duten denbora murrizten du sintonizazio horrek, jarraian ikusiko denez.

Oraingoan ere zenbait neurketa egin dira, interfazearen hutsegiteak zerbitzuaren funtzionamenduan izango lukeen eragina behatzeko. Hutsegite hori simulatzeko interfazearen Pod guztiak ezabatu eta berriro martxan ipini dira, 3.1. taulako datuak lortzeko jarraitutako prozeduraren antzera. Kasu honetan hauek dira jasotako datuak:

	t_1(s)	t_2(s)	t_3(s)	t_4(s)	t_5(s)	t_{bb}(s)	s (s)
3.3.2. ataleko konfigurazioa	47	47	50	46	47	47	1.52
3.3.3. ataleko konfigurazioa	14	9	14	9	12	12	2.51

3.2. taula: Inplementatutako bi konfigurazioek interfazearen hutsegitetik berreskuratzeko behar izandako denborak.

Denbora-tarteak nabarmenki laburtzen dira inplementazio berrirarekin, 3.2. taulan ikusten denez. Kubernetes-en potentziaren beste ikuspegi bat eskaintzen du horrek: konfigurazio erraz eta itzulgarri baten bidez Pod-en funtzionamendua bertan exekutatzen den aplikazioaren beharrei egoki daiteke, zeinak aplikazioen funtzionamenduan eragin handia izan dezakeen.

Azpimarratu behar da Pod-ak eguneratzean Kubernetes-ek duen jarraibidea. Pod-en konfigurazioan aldaketaren bat egon bada edota errore baten aurrean hainbat Pod berrabiarazi behar baditu, *Rolling Update* bat egiten du Kubernetes-ek: aplikazioari dagozkion martxan egon beharreko Pod-en kopurua ahal den neurrian betez, Pod-en eguneraketak sistema osoaren funtzionamenduan izango duen eragina minimizatzen saiatzen da. Adibidez, 3 Pod-ek osatutako Deployment batean aldaketa bat eginez gero, konfigurazio berria duen laugarren Pod bat sortzen da eta hori martxan egon arte ez dira gainontzekoak berrabiarazten. Jarraian Pod bakoitza banaka berrabiarazten hasiko da une oro eskuragarri dauden Pod-en kopurua 3 izanik.

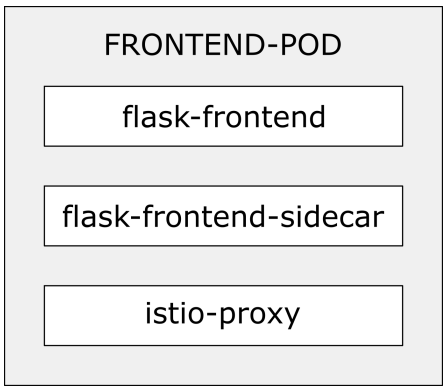
Lan egiteko era hori oso interesgarria da klusterraren eta bertan exekutatzen diren aplikazioen mantenuaren ikuspuntutik, aplikazioak modu ez-intrusibo batean eguneratzea ahalbidetzen baitu (zerbitzuak ahalik eta denbora laburrenean etenez). Gainera, eguneraketak inplementatutako konfigurazioak akatsik aurkeztuko balu, automatikoki bertan behera utziko litzateke.

3.3.4 Laugarren iterazioa: Istio-ren argitarapena

Teorian ikusi denez (ikusi 2.3.7 atala), Istio klusterren segurtasuna, kontrola eta monitorizazioa burutzeko tresnak eskaintzen dituen plataforma oso erabilia da. Istio-ren populartasuna eta aipaturiko tresnen potentzia izango dira atal honetan plataforma hori lantzearen arrazoi nagusiak.

Kluster batean Istio martxan jartzea ez da berehalako prozesu bat: argitarapena burutzeko eman beharreko pausuak klusterraren izaeraren zein bertan exekutatzen diren aplikazioen menpe egongo dira [62].

Topatutako lehenengo arazoa klusterreko nodoekin erlazionatuta dago: Istio-ren Pod-ak ezin dira ARM arkitekturako prozesadoreetan exekutatu. Nahiz eta Pod gehienak AMD64 prozesadoredun bi PCtan exekuta daitezkeen, Istio-ren "istio-proxy" edukiontzia (besteak beste Pod-en monitorizazioaz eta trafikoaren kontrolaz arduratzen dena) ezin da edonon exekutatu, argitaratutako Pod bakoitzaren barruan exekutatu behar baita.



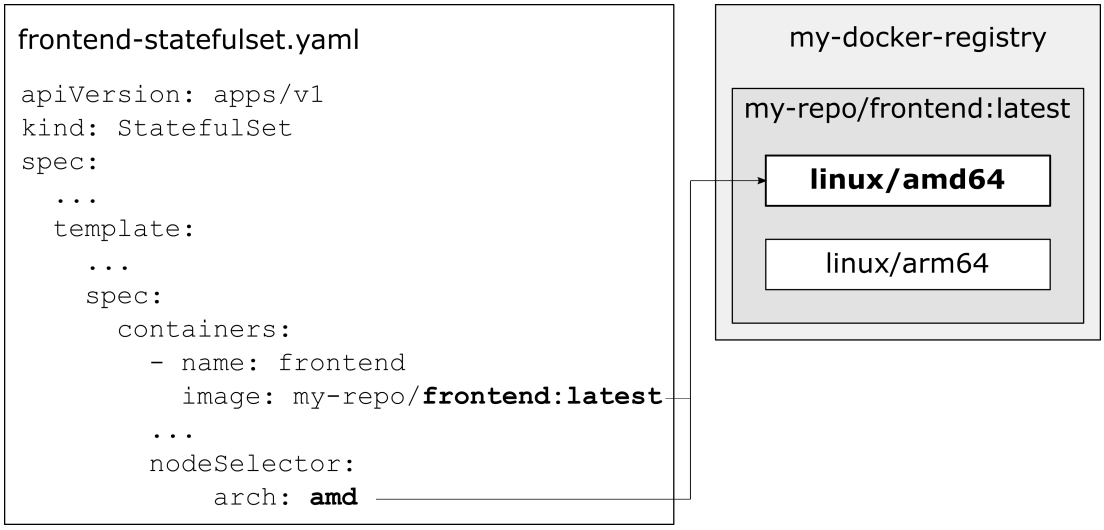
3.8. irudia: Klusterrean Istio instalatu ostean interfazearen Pod-ek duten egitura: aurretik sortutako edukiontziez gain "istio-proxy" edukiontzia ere exekutatzen da.

Orain arte garatutako aplikazioan Istio integratu ostean interfazearen Pod-ek izan beharko luketen egitura beha daiteke 3.8. irudian. Pod horiek, lehen aipatu bezala, ARM prozesadoredun nodoetan (Raspberry Pi-etan) exekutatzen dira. Ez dagoenez "istio-proxy" edukiontziaren ARM bertsiorik, edukiontzi horiek ezin dira hasieratu eta Pod-ak ez dira guztiz martxan ipintzen.

Aurreko ataletan argi ikusi da posible dela, orokorrean, arkitektura anitzetan oinarritutako kluster batentzako aplikazioak garatu eta argitaratzea. Une honetatik aurrera adibidetzat garatutako aplikazioa arkitektura bakarreko nodoetan bakarrik exekutatuko da, bai motorea baita interfazea ere. Horrela, Istio eta beste zenbait plataformen funtzionamenduak aztertu ahalko dira.

Interfazearen Pod-ak arkitektura bateko nodoetatik beste arkitekturetako nodoetara bidaltzeko 3.3.1. atalean aipatutako nodo-hautatzaileak erabili dira, bertan ARM arkitekturari dagokion etiketa AMD64 arkitekturari dagokionagatik ordezkatuz.

Aldaketa hori ahalik eta modu eraginkorrenean egiteko, oso interesgarria da interfazearen arkitektura anitzetako irudiak erabiltzea. Nahiz eta prozedura hori hemen azaldutakoa baino konplexuagoa izan, funtsean, makina bakarrean nahi diren arkitekturetarako irudiak sortzean datza, denak Docker-eko erregistro batera igo eta etiketa berdinarekin markatuz [63]. Horrela, nahikoa izango litzateke nodoaren hautatzailea aldatzea Pod-a nahi den nodoetan martxan ipintzeko, erabili beharreko irudiaren bertsio zehatza adierazi gabe.



3.9. irudia: Arkitektura anitzetako irudiak jasotzen dituen Docker-eko erregistroaren eta Kubernetes-en arteko integrazioaren diagrama sinplifikatua.

Aipagarria da 3.9. irudian agertzen den *image* eremuan ez dela finkatzen zein arkitekturako edukiontzia deskargatu behar den. Horrela, Kontrol Planoak konfigurazio-fitxategia interpretatzean, edukiontzia AMD64 arkitekturako prozesadoredun nodoei esleituko die, nodeSelector eremuko irizpideari jarraituz. Nodoek, automatikoki, haien arkitekturari dagokion edukiontzia aukeratuko dute erregistrotik deskargatzerako orduan.

Inplemetazio horrek aplikazioa asko sendotu eta garapena errazten du, sortzen diren irudiak klusterreko nodo guztietan exekutagarriak izatea ahalbidetzen baitu.

Lehen azpimarratu bezala, klusterrean argitaratutako aplikazioek ere Istio-ren instalazioa eta martxan jartzea zail dezakete [62]. Istio segurtasunaz arduratzen denez, Kubernetes-en defektuz egin daitezkeen zenbait ekintza blokeatzen ditu eta horiek burutzeko baimenak bereziki eman behar dira. Kasu honetan Redis-en instalazioa moldatu behar izan da Pod-ek diskoetan idatzi eta irakurtzeko baimenak izateko eta datu-baseko Pod-ek elkarren artean komunikatu ahal izateko.

Bi konfigurazio horiek moldatu ostean, aplikazioa martxan dago berriro eta Istio-k eskainitako sarearen kudeaketarako elementu berriak erabil daitezke aplikazioa kanpotik eskuragarri ipintzeko.

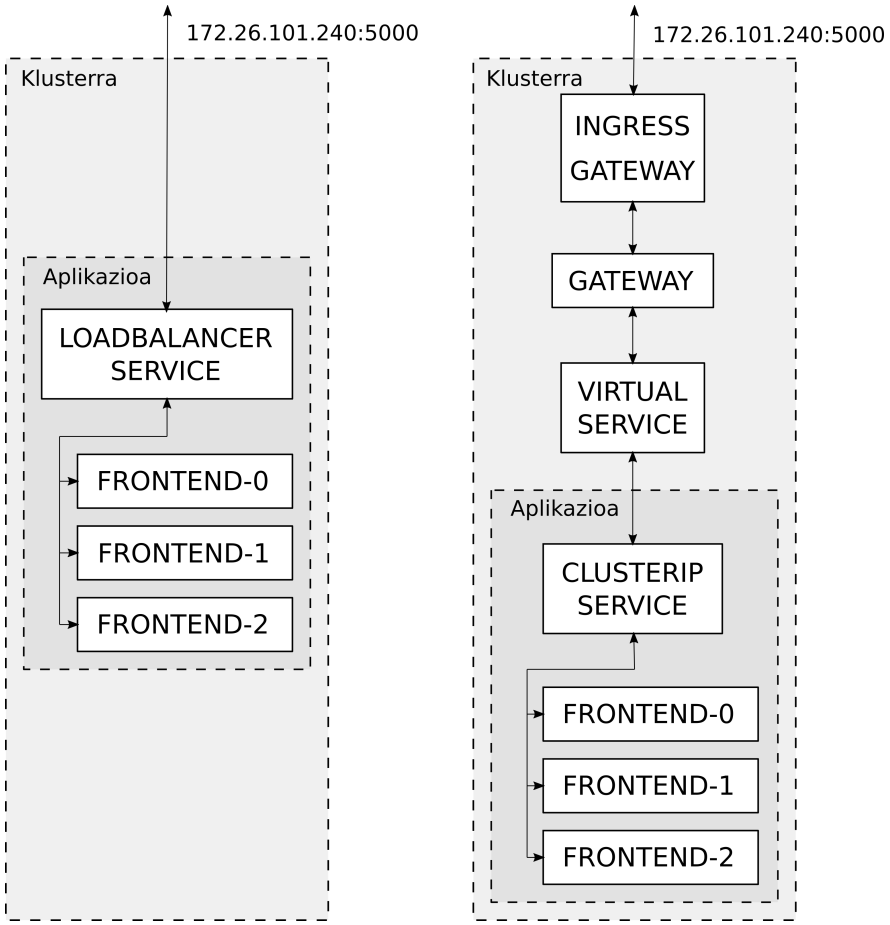
Service ezberdinak argitaratzen ditu Istio-k: monitorizazioko aplikazioetara sartzeko, Istio-ren barne-kudeaketarako eta kanpoko deiak onartzeko Service-ak, besteak beste.

Istio-k proposatutako sareak aurretik argitaratutako zama-banatzailearen funtzio antzekoak egiten ditu (kanpoko erabiltzailearen begietara, batez ere) baina kontrol eta abstrakzio aukera gehiago eskainiz (egitura konplexuak egitea ahalbidetzen duena). Istio-ren *Ingress Gateway*-ek klusterreko aplikazioetara sartzeko portuak irekitzen dituzte, zama-banatzaileak egiten duenaren antzera (ikus 3.10. irudia), baina portu horiek ez daude zuzenean aplikazio batekin lotuta: LoadBalancer batek deiak aplikazio bakarrera berbidal ditzakeen bitartean, Ingress Gateway bat ez da arduratzen deiak

zehazki amaierako puntura bidaltzeaz [64]. Elementu horrek aplikazio ezberdinetara zuzendutako konexioen sarrera-puntu bezala jokatzen du, trafikoaren kontrola eta monitorizazioa erraztuz.

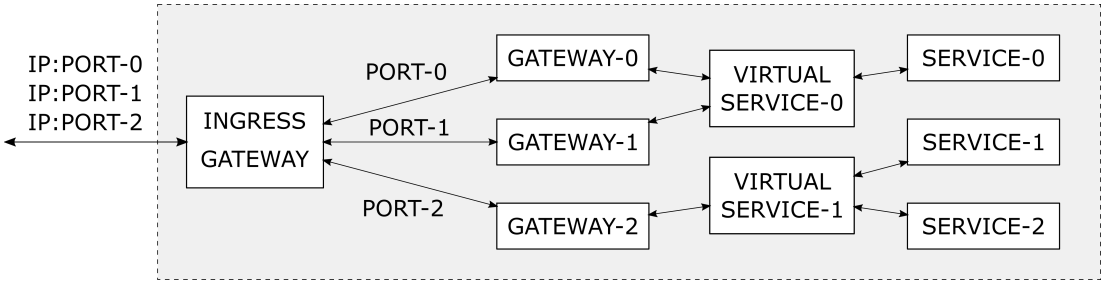
Sarrera-ate horren ostean beste *Gateway* bat ipintzen da, zeinak sarrerako ateko aukeratutako portu batzuetako trafikoa bakarrik jasoko duen [65]. Jarraian *Virtual Service* bat argitaratzen da, hautatutako Gateway batzuetako portu batzuetara iritsitako deiak dagozkien aplikazioetara berbidaliko dituenak [66].

Elementu berri horien funtzionamendua irudikatzeko 3.10. irudia prestatu da, orain arte erabilitako Service-aren eta Istio-ren sarearen bidez lortutako egituren funtzionamenduak konparatzeko balio dezakeena:



3.10. irudia: Aplikazio berdinaren argitarapena Kubernetes-en Service baten bidez (ezkerrean) eta Istio-ren zerbitzu-sarearen bidez (eskuina).

Begi-bistakoa denez, Istio-ren bidez sortutako sarea aurretik inplementatutakoa baino konplexuagoa da, baina konfigurazio hori Istio-k eskainitako trafikoaren kontrol- eta behaketa-tresnak erabiltzeko beharrezkoa da eta aplikazio konplexuagoen beharrei egokitzeko aukera ematen dute, hurrengo irudian erakusten den bezala:



3.11. irudia: Istio-ren zerbitzu-sarearen inplementazio posible bat, sarbide bakarretik aplikazio ezberdinetara doazen konexioak kudeatzen dituena.

Nahiz eta 3.11. irudian azaldutako adibidea benetan ez inplementatu, Istio-ren bidez posible izango litzateke horrelako egitura bat argitaratu eta kontrolatzea. Funtsean, 3.11. irudian portu ezberdinak irekita dituen helbide bat agertzen da. Bertako portuetara iritsitako deiak Gateway ezberdinetara iristen dira eta bertatik Virtual Service-k dagozkien aplikazioetara berbidaltzen dituzte. Azken berbidalketa hori bereziki zehatza izan daiteke, eskaera egindako URL-aren araberakoa, adibidez [66].

Orain arte sortutako aplikazioa Istio-rekin guztiz integratzeko aldaketa bakar bat gehiago egin behar da. Istio Kubernetes-eko Deployment-ekin lan egiteko pentsatuta dago eta StatefulSet-ekin azaltzen duen integrazioa konplexuagoa da [67]. Horregatik, interfazearentzat erabilitako StatefulSet-a Deployment bategatik ordezkatu da eta hori erakusteko erabiltzen zen Service-a LoadBalancer motakoa izatetik ClusterIP izatera pasa da (3.10. irudian azaltzen denez), jada ez baita kanpotik eskuragarri egon behar. Aldaketa horiek berehalakoak izan dira StatefulSet-en eta Deployment-en konfigurazioa (eta funtzionamendua) antzekoa baita. Horrela, aplikazioa 3.10. irudiko ezkerreko diagramaren egitura izatetik eskuinekoa izatera pasa da.

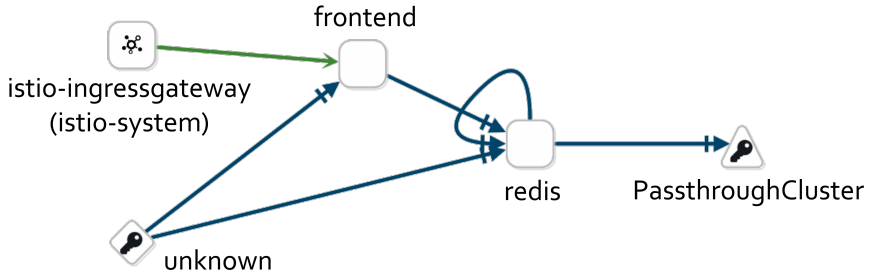
Laburbilduz, atal honetan Istio instalatu eta lehendik argitaratutako aplikazioarekin integratu da. Horrek, lehen esan bezala, kontrol eta monitorizazio tresna gehiago jarriko ditu klusterraren administratzailearen eskura, hurrengo ataletan landuko direnak.

3.3.5 Bosgarren iterazioa: zerbitzu-sarearen kudeaketa Kiali erabilita

Kiali Istio-ren zerbitzu-sareen monitorizazioa eta kudeaketa errazten duen interfaze-grafikoa da. Istio-ren bidez instala daiteke eta web-nabigatzailetik Istio-ren sarearen barruan argitaratutako aplikazioen inguruko informazioa jaso edota editatzeko aukera ematen du.

Aplikazioen egoeraren inguruko datuak eskaintzeaz gain (eskuragarri dauden Pod-en kopurua, konfigurazioan topatutako akatsak etab.), grafika ezberdinen bidez zerbitzuek jasotako trafikoaren inguruko informazioa ere azaltzen du. Nahiz eta ezaugarri horiek oso interesgarriak izan, Istio-ren zerbitzu-sareak grafikoki erakusteko gaitasuna da Kiali-ren berezitasuna.

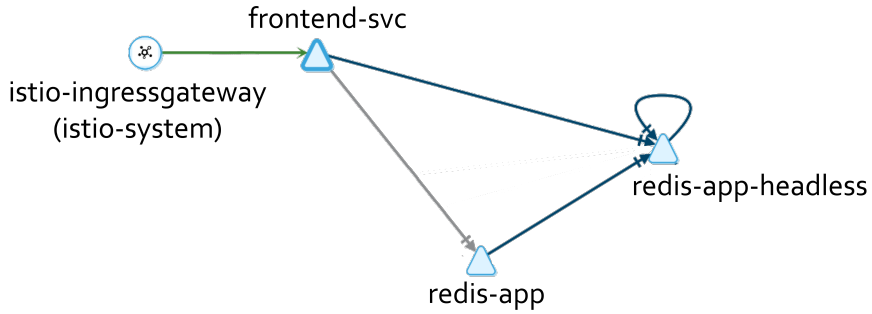
Kiali-k sortutako aplikazioaren eskema 3.12. irudian beha daiteke. Grafoko hiru nodo dira bereziki interesgarriak: kanpoko deiak jasotzen dituen Ingress Gateway-a, interfazearen aplikazioa eta Redis-en aplikazioa. Beste biek zerbitzu-saretik kanpo dauden Kubernetes-ek sortutako aplikazioekin konexioak adierazten dituzte.



3.12. irudia: Garatutako aplikazioaren diagrama, Kiali-rekin sortutakoa. Interfazea ("frontend" nodoa), datu-basea ("redis" nodoa) eta sarrera-puntua agertzeaz gain, horiek klusterreko beste elementuekin lotzen dituzten sarrera eta irteera-puntuak ere ikus daitezke.

Nodoen arteko loturen koloreei dagokionez, berdea HTTP konexioak adierazteko erabiltzen da, urdina TCP konexioetarako, laranjak nodoen arteko konexioak trafiko-arazoak daudela adierazten du eta grisa nodoen arteko konexioa existitzen dela baina trafikorik ez dagoela adierazten du. Nodoen formek ere esanahia dute: triangeluek zerbitzuak adierazten dituzte, karratuek aplikazioak edo Pod-ak eta borobilak sarrera-puntuentzat erabiltzen dira.

Kiali-k aplikazio baten inguruko adierazpen grafiko ezberdinak sor ditzake, nahi diren datuak zehazki behatzeko aukera emanez. Hori oso erabilgarria izan daiteke aplikazio konplexuen funtzionamendua ulertzerako orduan.

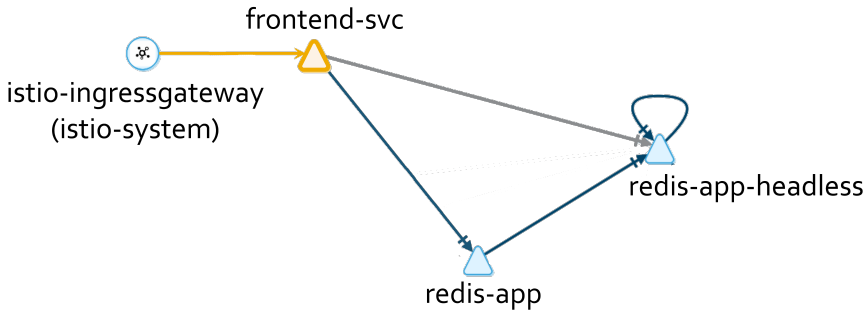


3.13. irudia: Garatutako aplikazioaren zerbitzu ezberdinen arteko konexioak erakusten dituen diagrama, Kiali-rekin sortutakoa.

Garatutako aplikazioa osatzen duten zerbitzuen diagrama ageri da 3.13. irudian. Adierazpen horrek Redis-en funtzionamenduaren inguruko informazio asko eskaintzen du, elementu ezberdinen arteko konexioak erakusten baititu. Esaterako, argi gelditzen da aplikazioaren interfazea Redis-ekin "redis-headless" zerbitzuaren bidez bakarrik komunikatzen dela. Zerbitzu horrek kanpoko eskaerei erantzutzeaz gain Redis-eko nodoen sinkronizazioa ere burutzen du, zerbitzuak bere buruarekin duen konexioak adierazi bezala. Aipatzekoa da ere "redis-app" zerbitzua interfazearekin konektatuta egon arren Redis-eko master nodoak huts egitean baino ez duela bertatik trafikorik jasotzen. Ondorioz, konexio horretan trafikoa izateak datu-basean arazoren bat egon delaren seinaletzat erabil daiteke, jarraian landuko denez.

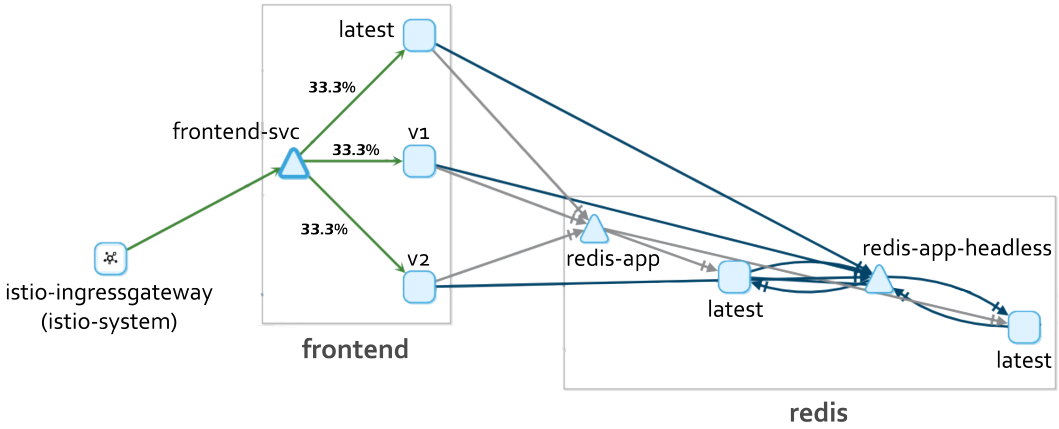
Redis-eko masterrak huts egitean lortutako egoera 3.14. irudiko diagraman ikus daite-

ke: sarrera-puntura iritsitako deiek errore bat jasoko lukete master berria aukeratu eta datu-base guztia berriro sinkronizatu bitartean (horregatik agertzen da horiz konexio hori) eta "redis-app" zerbitzua "redis-app-headless" zerbitzuarekin konektatuko litzateke aukeratutako masterraren berri emateko. Hori guztia gertatu ostean sistema 3.13. irudiko egoerara itzuliko litzateke.



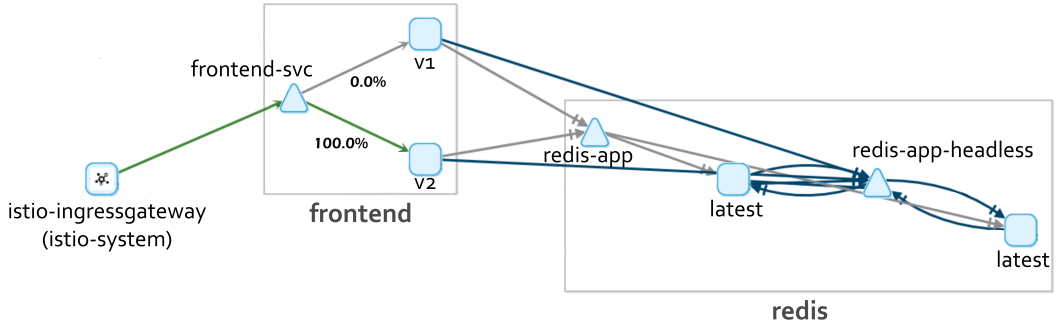
3.14. irudia: Redis-en masterrak huts egitean sistemaren egoera azaltzen duen diagrama, Kiali-rekin lortutakoa.

Aplikazio bati zuzendutako trafikoa aplikazioaren bertsio ezberdinen artean banatzeko aukera ematen du Istio-k. Ezaugarri hori oso interesgarria izan daiteke aplikazioak eguneratzerako orduan: aplikazioaren bertsio bakarretik hasita bere bi kopia sor daitezke, batek une horretako bertsioa izango duena eta besteak egin nahi diren aldaketak jasotzen dituena. Behin Pod guztiak (zaharrak eta berriak) martxan daudela, trafikoa hiruetara proportzio berdinean bidaliko litzateke (ikus 3.15. irudia).



3.15. irudia: Interfazearen eguneraketa baten hasieran sistemak izango lukeen egoeraren diagrama, Kiali-rekin lortutakoa. Interfazearen hiru bertsio agertzen dira diagraman: "latest" interfazearen hasierako egoera da, "v1" horren kopia bat eta "v2" interfazearen eguneraketa. Trafikoa hiru bertsioetara proportzio berdinean bideratzen da.

Jasotako deietatik bertsio bakoitzari berbidali beharreko deien portzentaia Virtual Service-a aldatuz edota beste bat sortuz finka daiteke. Hasiera-puntu egoki bat trafiko guztia lehenengo bertsiodun aplikaziora bidaltzea izango litzateke eta, pixkanaka, egin beharreko proba guztiak egokiak izanez gero, trafiko guztia amaierako bertsiora bidaltzen amaitu.



3.16. irudia: Interfazearen eguneraketa baten amaieran sistemak izango lukeen egoeraren diagrama, Kiali-rekin lortutakoa. Trafiko guztia "v2" bertsiora bideratzen da.

Etorkizunean beste eguneraketa bat eginez gero prozesua askoz sinpleagoa izango litzateke, hasieratik bi bertsioetara trafikoa berbidal dezakeen Virtual Service-a konfiguratuta baitago.

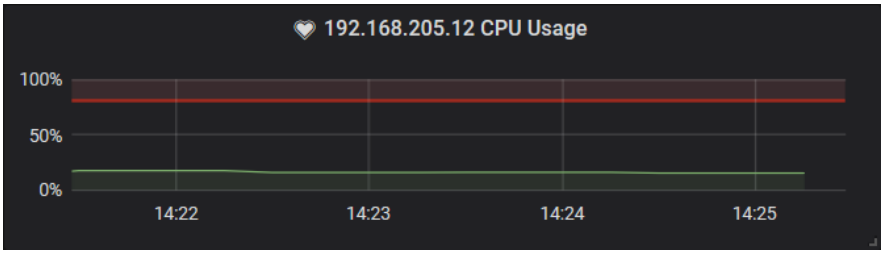
Garrantzitsua da, adibide horren ostean, zenbait ideia argi gelditzea: alde batetik, kontuan eduki behar da egindako eguneraketa interfazearen konfigurazioan egindako aldaketa bat izateaz gain, aplikazioaren eskalaketa (eskuragarri egon behar diren Pod-en kopuruaren aldaketa) bat ere izan zitekeela. Horregatik, oso garrantzitsua da Kubernetes-en barruan sortu nahi diren aplikazioen zati ezberdinak ondo identifikatu eta haien portaerak kontuan edukitzea Pod-ak diseinatzerako orduan. Kasu honetan, ez luke zentzurik izango Redis-en baliabideak eta interfazeak batera eskalatzeak, jasotzen duten lan-karga, orokorrean, ezberdina baita. Bestalde, azpimarratu behar da azaldutako prozesu horretan aplikazioaren zati baten eguneraketa integral bat burutu dela eta trantsizio osoan ez dela eskainitako zerbitzua eten. Aplikazio osoa antzerako prozedura bat jarraituz egunera zitekeen, zerbitzuaren eskuragarritasuna inoiz kaltetu gabe.

3.3.6 Seigarren iterazioa: klusterraren monitorizazioa Prometheus eta Grafana erabilita

Nahiz eta Kiali-k zerbitzuen inguruko datuak jaso, ohikoa da klusterraren egoera ondo kontrolatzeko informazio gehiago behar izatea. Horregatik, datu horiek jasotzeko Prometheus erabili da. Istio-ren saretik kanpo dauden aplikazioen edota hardwarearen datuak biltzeko beharrezkoa da Prometheus-en beste argitarapen bat egitea, Istio-rekin integratuta datorrenak zerbitzu-sarearen inguruko datuak baino jasotzen ez baititu. Klusterraren egoera osoaren informazioa bi iturri horiek erabiliz bil daiteke.

Datu horiek guztiak modu erosoan bistaratzeko Grafana erabiliko da. Istio-k bere barneko aplikazioak behatzeko eskainitako aginte-mahaiez (ingelesezko *dashboard*-etik) gain, klusterraren hardwarea eta aplikazioaren egoera behatzea ahalbidetzen duten panelez osatutako beste bat sortu da.

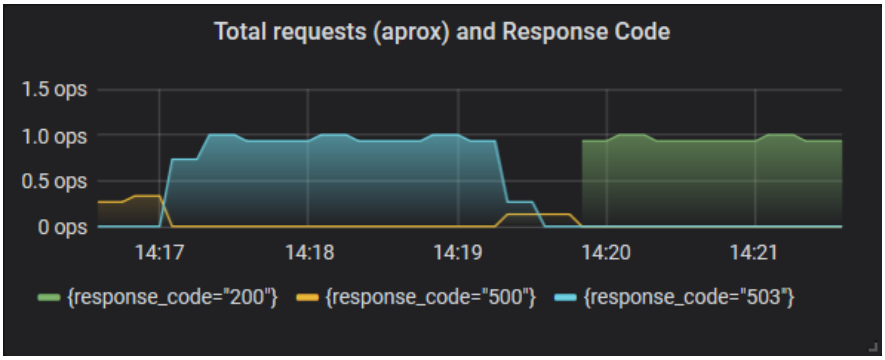
3.17. irudia: Klusterreko langile nodoen CPUaren erabilera, erabilitako RAM memoria eta disko zurrunetan betetako edukiera azaltzen dituen panela.



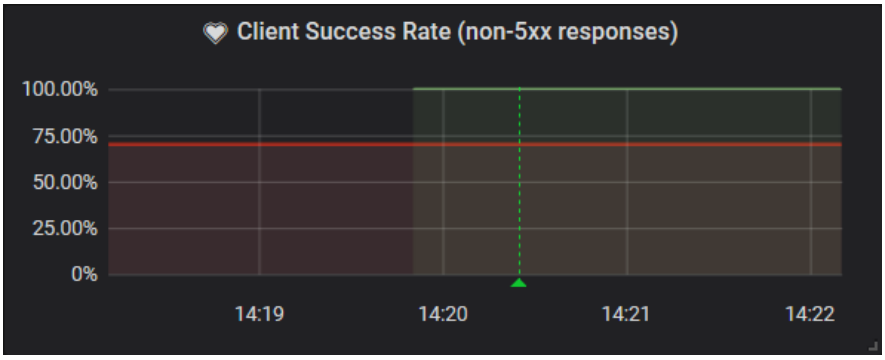
3.18. irudia: Nodo bateko CPUak denboran zehar izandako lan-karga behatzeko panela. Lerro gorriak jakinarazpena aktibatzeko CPU-ak izan beharreko lan-karga minimoa adierazten du.

Hardwarea behatzeko erabilitako panel batzuk 3.17 eta 3.18. irudietan beha daitezke. Aipagarriena "192.168.205.11 CPU Usage" panelean gorriz agertzen den lerroa da. Berdez ageri den lerroa (nodoaren prozesadorearen lan-karga adierazten duena) bertara iristen denean notifikazio bat sortzen da, nodo horren prozesadorea lan gehiegi egiten ari delaz ohartaraziz. Alerta horiek posta elektronikora edota Slack-eko kanal batera berbidali daitezke, besteak beste. Nodoen RAM memoriaren erabilera behatzeko panelak ere sortu dira, antzerako alertekin doitutakoak.

Gainera, Kiali-rekin lan egitean zerbitzuen funtzionamenduaz lortutako ezagutzak erabili daitezke garatutako aplikazioaren aginte-mahaia sortzeko. Adibidez, 3.19 eta 3.20. irudietako bi panelak erabil daitezke interfazearen funtzionamendua behatzeko: lehenengoan denboran zehar jasotako dei guztiak ikus daitezke, dei horiek jasotako erantzunkodea kolore ezberdinen bidez adierazten dituenak. Bigarrenean aldiz, dei guztien artean ondo erantzundakoen proportzioa agertzen da. Panel horretan beste alarma bat ezarri da, ondo erantzundako deiak guztien % 70-era iristen ez bada aktibatzen dena.



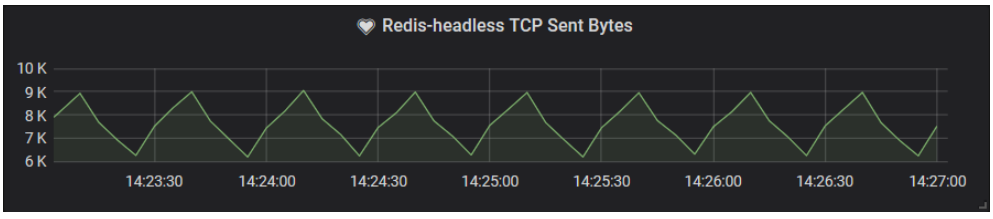
3.19. irudia: Denboran zehar jasotako deien kopurua haiek jasotako erantzun-kodearen arabera. Irudian adierazten denez, berdez markatu dira 200 kodea jasotako deiak, horiz 500 kodea jasotakoa eta urdinez 503 kodea jaso dituztenak. Ardatz bertikaleko "ops" unitetateak "eragiketa-segunduko" adierazi nahi du.



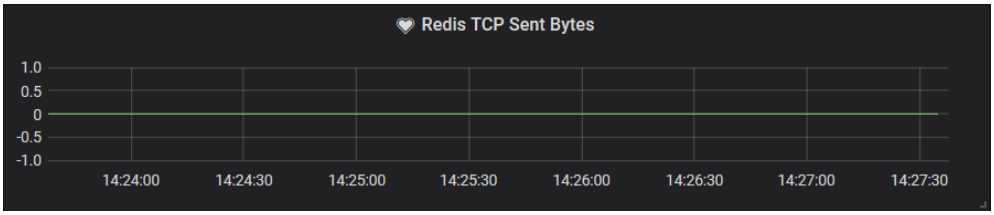
3.20. irudia: Denboran zehar jasotako dei guztietatik ondo erantzundako deien proportzioa (berdez). Lerro gorriak jakinarazpena ez aktibatzeko ondo erantzundako dei minimoen portzentaia adierazten du.

Redis-en datu-basea behatzeko beste lau panel sortu dira, "redis-app-headless" eta "redis-app" zerbitzuek jaso eta bidalitako byten kopurua ikustarazteko. Redis-en nodoak sinkronizatzeaz arduratzen den "redis-app-headless" Service-ak etengabe datuak bidaltzen dituela ikus daiteke 3.21. irudian, 3.3.5. atalean ikusitakoarekin bat etorriz. Horregatik, zerbitzu horrek trafikorik ez duenean piztuko den alerta bat konfiguratu da, hori Redis-eko nodoen arteko sinkronizaziorik ez dagoelaren seinale izan baitaiteke.

Bestalde, "redis-app" zebitzua bere ohiko egoeran dagoela 3.22. irudiko panelean beha daiteke. Zerbitzu horrek Redis-eko masterrak huts egitean bakarrik jasotzen duenez trafikoa, sortu den alarmak trafikorik egonez gero abisatzen du.



3.21. irudia: Denboran zehar "redis-app-headless" zerbitzuak bidalitako byten kopurua.



3.22. irudia: Denboran zehar "redis-app" zerbitzuak bidalitako deien kopurua.

3.3.7 Oharra

Azken bi iterazioak burutzeko ez da ordura arte erabilitako klusterra erabili, makina birtualetan oinarritutako bat baizik. Aldaketa horren helburua aplikazioen garapenean erabili ohi diren inguruneak zein lan egiteko moduak simulatu eta probatzea izan da.

Sarritan garapenerako eta produkziorako klusterrak ez dira makina berdinetan oinarritzen eta, askotan, garapenerako erabilitako klusterrak makina birtualen bidez inplementatzen dira. Bi klusterrak berdin konfiguratuz gero, aplikazio bat batetik bestera migratzea berehalako lana izan beharko litzateke.

Makina birtualetan oinarritutako inplementazioak abantaila asko eskaintzen ditu: kluster osoa makina fisiko bakarrean exekuta daiteke eta klusterraren inguruko aldaketa sakonak (nodoetako sistema eragilea aldatzea, nodoetan memoria gehiago jartzea etab.) probatzeko aukera eskaintzen du, atzera itzultzea askoz errazagoa baita. Makinen mantenua eta sarearen kudeaketa ere kluster fisikoan baino errazagoa da. Aurkan, makina bakarrean kluster osoa (edo nodo minimo batzuk behintzat) martxan ipintzeko beharrezko hardwarea potentea izan behar da, batez ere prozesadore eta memoriak, eta sarearen kudeaketa ez da bare metal kluster batean egingo litzatekeenaren berdina, kluster birtualetik fisikorako trantsizioa zailduz.

Kasu honetan, makina birtualak sortzeko eta konfiguratzeko Vagrant erabili da [68], nahiz eta beste edozein aukera baliagarria izan zitekeen (Vagrant bidez sortutako makinak Ansible bidez konfigura zitezkeen edo inplementazio osoa Terraform-en bidez egin zitekeen, esaterako). Migrazioa burutzean konfigurazio-deklaratiboaren potentzia agerian gelditu da: aurreko egoerara iristeko nahikoa izan da kluster zaharretik orain arte erabilitako fitxategi guztiak jaso eta kluster berrian exekutatzea.

4. kapitulua

Ondorioak

Lan honetan Kubernetes-en oinarrizko funtzionamendua deskribatzeaz gain, kluster heterogeneo bat Kubernetes bidez kudeatu da, baita aplikazio baten garapena landu ere. Prozesu guztian erabilitako konfigurazio-fitxategiak proiektuaren GitHub-eko biltegian jaso dira: jongablop/OlimpoServer.

Oso garrantzitsua da Kubernetes-ek azpiegiturarekiko eskaintzen duen independentzia eta abstrakzioa. Makina anitzez osatutako klusterrak kudeatzeaz gain, Kubernetesen atzean dagoen filosofiaren eta lan egiteko moduen ondorioz, erraza da proiektuak azpiegitura batetik bestera migratzea, lan honetan egiaztatu denez. Kubernetes bidez ebatz daitezkeen arazoen eta aplikazio-eremuen sorta asko zabaltzen du horrek: klusterra sortzeko erabilitako azpiegitura eta proiektuaren helburu eta zehaztapenak edonolakoak izanik, kasu gehienetan posible izango da Kubernetes modu arrakastatsu batean inplementatzea.

Teorian landutako kontzeptuak praktikan ipintzeko, web-aplikazio baten garapena eta argitarapena aztertu dira: interfazea Flask ingurunean sortuz eta datuak gordetzeko Redis-en oinarritutako datu-basea erabiliz, interfaze-motore egituradun aplikazioaren sorrera eta mantenua landu dira. Argitarapenari dagokionez, webgunea klusterretik kanpo IP eta portu finkoen bidez erakutsi da, produkzio-ingurunetan egiten den moduan. Argi gelditu da edukiontzitan oinarritutako aplikazio konplexu eta eskalagarriak sortzeko aproposa dela Kubernetes, bere helburuak bikainki betetzen baititu: aplikazioaren atalak independenteki garatzeko ingurune eta tresna egokiak eskaintzeaz gain, horiek argitaratzeko baliabide aberatsak ere baditu.

Garapenean zehar Kubernetes-en konfigurazioa aplikazioaren beharreri egokitu da, moldaketa horien emaitzak 3.3.3. atalean jasoz: Kubernetes-ek burutzen dituen prozesuen parametroak sintonizatzean, hutsegitetik berreskuratzeko denborak % 50 - 70 bitartean murriztu dira.

Hurrengo urratsetan Kubernetes-ekin integra daitezkeen zenbait plataforma inplementatu dira. Istio instalatzeak aplikazioaren barne-egitura aldatzea behartu du, baina lortutako onurak hainbat dira: produkzio-ingurune konplexuagoak probatzeko aukera eskaintzeaz gain, zerbitzu-sarea Kiali bidez kudeatu eta Prometheus eta Grafana bidez behatu ahal izan da. Kiali-k zerbitzu-sarea behatu eta kontrolatzeko interfaze aberatsak ditu eta Prometheus eta Grafana konbinatzeak klusterraren egoeraren inguruko informazio asko aldi berean bistaratzeko aukera ematen du. Tresna guzti horiek erabi-

lita posible da klusterraren hutsegite eta arazoak ekidin eta aurreikustea.

Lanari amaiera emateko Vagrant ere probatu da, makina birtualetan oinarritutako kluster bat sortzeko erabili dena. Kluster batetik besterako migrazioa berehalakoa izateak lehen aipatutakoa azpimarratzen du: Kubernetes-ek azpiegitura oso ezberdinetan sortutako klusterrak kudea ditzake eta aplikazio baten garapen eta argitarapenaren prozesu osoan da baliagarria.

Etorkizunari begira, sakontzeke gelditu diren hainbat bide jarrai daitezke proiektu hau oinarritzat hartuz. Ingeniaritza Elektronikoko informatikaren adarrarekin lotutako alderdiari dagokionez, interesgarria izango litzateke azpiegituren diseinu eta mantenuan erabiltzen diren plataformak aztertzea. Kluster bakarrarekin lan egin beharrean, elkarren artean konektatutako makina askorekin lan egiten da horrelako inplementazioetan, behar diren klusterrak sare horren barruan sortuz. Azpiegitura horien baliabideen kostua ere azter zitekeen. Bestalde, Ingeniaritza Elektronikoari zuzenago erlazionatutako lan ugari ere egin daitezke, bai irakaskuntzarekin lotutakoak baita ikerkuntzarekin lotutakoak ere. Moodle bezalako aplikazioen zerbitzari-lanak egiteaz gain, Jupyter proiektuan oinarritutako irakaskuntza-proiektuak garatzeko aproposa da Kubernetes, plataforma horren baliabideak horrelako klusterretan instalatzeko prestatuta baitaude. Hurrengo proiektu batean Kubernetes erabiliko da Jupyter-en oinarritutako zenbait irakaskuntza-proiektuen zerbitzaria kudeatzeko [69, 70]. Ikerkuntzan aldiz, fitxategien kontrol-sistemak argitaratzeko erabiltzeaz gain, IoT sistemetako datuen bilketa eta bisualizazioa burutzeko erabiltzen dira Kubernetes, Grafana, Redis eta Prometheus, besteak beste. Industria 4.0-ren inguruko lanak burutzeko oinarrizko ezagutzak biltzen ditu lan honek, beraz.

Aipatzekoa da ere, plataforma horiek eta teorian landutako kontzeptuak, Kubernetesekiko independenteki, proiektu oso anitzetan erabil daitezkeela: Docker-ek aplikazioak garatzeko ingurune isolatuak sortzea ahalbidetzen du, Grafana-k datu-base oso ezberdinetan jasotako datuak ikusteko aukera ematen du eta Redis sentsore batzuetatik datozen datuen cache bezala erabil daiteke, adibidez.

Laburbilduz, aplikazioak garatu eta argitaratzeko plataforma potente eta malgua da Kubernetes. Nahiz eta inplementazioa konplexua izan, oso erabilgarria den eta izango den tresna dela ulertu behar da: orain dagoen joera hurrengo urteetan jarraituz gero, sarean eskuragarri dauden zerbitzuetan oinarritutako lan-sistemak ohikoagoak izango dira. Sistema horiek kudeatzeko tresna aproposak edukiontziak direnez, haiek kudeatzeko plataformak geroz eta beharrezkoagoak izango dira.

A eranskina

Erabilitako plataformen bertsioak

Erabilitako bertsioa jasotzea ezinbestekoa da egindako lana errepikagarria izan dadin. Gainera, proiektuaren garapenean zehar bertsioekin erlazionatutako hainbat arazo izan dira: erabilitako plataformak etengabe garapenean daudenez, bertsio berriak aurretik zegoenarekin edota beste plataformekin bateragarriak ez izatea gerta liteke. Arazo horiek egun batzuen buruan konpontzen dira gehienetan, garatzaileek akatsaren zuzenketa argitaratzean. Beste batzuetan ordura arteko dokumentazio osoa ere zaharkitua geratzen da eta bertsio berriari egokitzeak aste edota hilabete batzuetako lana suposatzen du.

Proiektu honek iraun bitartean, 2019-2020 ikasturtean, erabilitako plataforma nagusien bertsioak hurrengo taulan biltzen dira:

Plataforma	**Bertsioa**
Kubernetes	1.18
Kubeadm	1.18
Kubectl (client eta server)	1.18
Docker (client eta server)	19.03.8
Flannel	0.11.0
MetalLB	0.7.3
Flask	1.1.1
Helm	3.1.2
Redis	5.0.7
Istio	1.5.1
Kiali	1.15.2
Prometheus	2.15.1
Grafana	6.5.2

A.1. taula: Erabilitako plataformen bertsioak.

Azpimarragarria da erabilitako zenbait tresna haien garapenaren hasierako egoera batean daudela, A.1. taulan ikus daitekeenez. Haien egonkortasuna edo beste softwareekiko integrazioa zail zezakeen horrek, baina lan honetan ez da horrelako arazorik egon.

Bibliografia

[1] "Total number of Websites," internetlivestats.com. [Online]. Hemendik eskuratua: https://www.internetlivestats.com/total-number-of-websites (2020/05/05ean kontsultatua).

[2] "Internet Growth Statistics," internetworldstats.com, 2020. [Online]. Hemendik eskuratua: https://www.internetworldstats.com/emarketing.htm (2020/05/05ean kontsultatua).

[3] "The History of Kubernetes on a Timeline," risingstack.com, 2018. [Online]. Hemendik eskuratua: https://blog.risingstack.com/the-history-of-kubernetes (2020/05/05ean kontsultatua).

[4] P. Christensson, "Dynamic Website Definition," techterms.com, 2009. [Online]. Hemendik eskuratua: https://techterms.com/definition/dynamicwebsite (2020/05/05ean kontsultatua).

[5] The Kubernetes Authors, "Production-Grade Container Orchestration," kubernetes.io, 2020. [Online]. Hemendik eskuratua: https://kubernetes.io (2020/04/26ean kontsultatua).

[6] "Frontend vs Backend," geeksforgeeks.org. [Online]. Hemendik eskuratua: https://www.geeksforgeeks.org/frontend-vs-backend (2020/04/11ean kontsultatua).

[7] Red Hat, "Integration - What is an API?" redhat.com, 2020. [Online]. Hemendik eskuratua: https://www.redhat.com/en/topics/api/what-are-application-programming-interfaces (2020/04/26an kontsultatua).

[8] "Contenedores en Google," cloud.google.com. [Online]. Hemendik eskuratua: https://cloud.google.com/containers (2020/04/11ean kontsultatua).

[9] B. Burns, K. Hightower, eta J. Beda, "Introduction," in *Kubernetes: Up and Running*. O'Reilly Media, 2017, 17–37. orr.

[10] "YAML: YAML Ain't Markup Language," yaml.org. [Online]. Hemendik eskuratua: https://yaml.org (2020/04/11ean kontsultatua).

[11] GeeksforGeeks Contributors, "Version Control Systems," geeksforgeeks.org. [Online]. Hemendik eskuratua: https://www.geeksforgeeks.org/version-control-systems (2020/04/11ean kontsultatua).

[12] "Git," git-scm.com. [Online]. Hemendik eskuratua: https://git-scm.com/ (2020/04/11ean kontsultatua).

[13] "What is a cloud service provider?" azure.microsoft.com, 2020. [Online]. Hemendik eskuratua: https://azure.microsoft.com/en-us/overview/what-is-a-cloud-provider (2020/04/11ean kontsultatua).

[14] The Kubernetes Authors, "Conceptos," kubernetes.io, 2019. [Online]. Hemendik eskuratua: https://kubernetes.io/es/docs/concepts (2020/04/12an kontsultatua).

[15] The Kubernetes Authors, "kube-proxy," kubernetes.io, 2020. [Online]. Hemendik eskuratua: https://kubernetes.io/docs/reference/command-line-tools-reference/kube-proxy (2020/04/12an kontsultatua).

[16] The Kubernetes Authors, "kube-scheduler," kubernetes.io, 2020. [Online]. Hemendik eskuratua: https://kubernetes.io/docs/reference/command-line-tools-reference/kube-scheduler (2020/04/12an kontsultatua).

[17] The Kubernetes Authors, "kube-controller-manager," kubernetes.io, 2020. [Online]. Hemendik eskuratua: https://kubernetes.io/docs/reference/command-line-tools-reference/kube-controller-manager (2020/04/12an kontsultatua).

[18] The Kubernetes Authors, "Kubernetes Components," kubernetes.io, 2020. [Online]. Hemendik eskuratua: https://kubernetes.io/docs/concepts/overview/components (2020/04/12an kontsultatua).

[19] The Kubernetes Authors, "Pod Overview," kubernetes.io, 2019. [Online]. Hemendik eskuratua: https://kubernetes.io/docs/concepts/workloads/pods/pod-overview (2020/04/12an kontsultatua).

[20] The Kubernetes Authors, "Service," kubernetes.io, 2020. [Online]. Hemendik eskuratua: https://kubernetes.io/docs/concepts/services-networking/service (2020/04/12an kontsultatua).

[21] The Kubernetes Authors, "Namespaces," kubernetes.io, 2020. [Online]. Hemendik eskuratua: https://kubernetes.io/docs/concepts/overview/working-with-objects/namespaces (2020/04/12an kontsultatua).

[22] The Kubernetes Authors, "Volumes," kubernetes.io, 2020. [Online]. Hemendik eskuratua: https://kubernetes.io/docs/concepts/storage/volumes (2020/04/12an kontsultatua).

[23] The Kubernetes Authors, "Secrets," kubernetes.io, 2020. [Online]. Hemendik eskuratua: https://kubernetes.io/docs/concepts/configuration/secret (2020/06/16ean kontsultatua).

[24] The Kubernetes Authors, "ReplicaSet," kubernetes.io, 2020. [Online]. Hemendik eskuratua: https://kubernetes.io/docs/concepts/workloads/controllers/replicaset (2020/04/13an kontsultatua).

[25] The Kubernetes Authors, "Deployments," kubernetes.io, 2020. [Online]. Hemendik eskuratua: https://kubernetes.io/docs/concepts/workloads/controllers/deployment (2020/04/13an kontsultatua).

[26] The Kubernetes Authors, "StatefulSets," kubernetes.io, 2020. [Online]. Hemendik eskuratua: https://kubernetes.io/docs/concepts/workloads/controllers/statefulset (2020/04/13an kontsultatua).

[27] The Kubernetes Authors, "DaemonSets," kubernetes.io, 2020. [Online]. Hemendik eskuratua: https://kubernetes.io/docs/concepts/workloads/controllers/daemonset (2020/04/13an kontsultatua).

[28] The Kubernetes Authors, "Jobs - Run to Completion," kubernetes.io, 2020. [Online]. Hemendik eskuratua: https://kubernetes.io/docs/concepts/workloads/controllers/jobs-run-to-completion (2020/04/13an kontsultatua).

[29] The Kubernetes Authors, "Creating a single control-plane cluster with kubeadm," kubernetes.io, 2020. [Online]. Hemendik eskuratua: https://kubernetes.io/docs/setup/production-environment/tools/kubeadm/create-cluster-kubeadm (2020/04/13an kontsultatua).

[30] Docker Inc., "What is a Container?" docker.com, 2020. [Online]. Hemendik eskuratua: https://www.docker.com/resources/what-container (2020/04/13an kontsultatua).

[31] "Developer Survey Results 2019," insights.stackoverflow.com, 2020. [Online]. Hemendik eskuratua: https://insights.stackoverflow.com/survey/2019 (2020/04/13an kontsultatua).

[32] CoreOS, "Flannel," github.com/coreos, 2020. [Online]. Hemendik eskuratua: https://github.com/coreos/flannel (2020/04/13an kontsultatua).

[33] "Network Addon Compatibility," metallb.universe.tf, 2020. [Online]. Hemendik eskuratua: https://metallb.universe.tf/installation/network-addons (2020/04/13an kontsultatua).

[34] F5, "What Is Load Balancing?" nginx.com. [Online]. Hemendik eskuratua: https://www.nginx.com/resources/glossary/load-balancing (2020/04/26ean kontsultatua).

[35] "MetalLB," metallb.universe.tf, 2020. [Online]. Hemendik eskuratua: https://metallb.universe.tf (2020/04/13an kontsultatua).

[36] Cloud Native Computing Foundation, "Helm - The package manager for Kubernetes," https://helm.sh. [Online]. Hemendik eskuratua: https://helm.sh (2020/04/13an kontsultatua).

[37] IBM Cloud Education, "Message Brokers," ibm.com, 2020. [Online]. Hemendik eskuratua: https://www.ibm.com/cloud/learn/message-brokers (2020/04/13an kontsultatua).

[38] Redis Labs, "Introduction to Redis," redis.io. [Online]. Hemendik eskuratua: https://redis.io/topics/introduction (2020/04/14ean kontsultatua).

[39] Redis Labs, "Redis cluster tutorial," redis.io. [Online]. Hemendik eskuratua: https://redis.io/topics/cluster-tutorial (2020/04/14ean kontsultatua).

[40] Redis Labs, "Redis Sentinel Documentation," redis.io. [Online]. Hemendik eskuratua: https://redis.io/topics/sentinel (2020/04/14ean kontsultatua).

[41] Redis Labs, "Redis on ARM," redis.io. [Online]. Hemendik eskuratua: https://redis.io/topics/ARM (2020/04/14ean kontsultatua).

[42] Istio Authors, "What is Istio?" istio.io, 2020. [Online]. Hemendik eskuratua: https://istio.io/docs/concepts/what-is-istio (2020/04/14ean kontsultatua).

[43] T. Krazit, "Is Google cooling on open-source foundations?" protocol.com, 2020. [Online]. Hemendik eskuratua: https://www.protocol.com/google-open-source-istio (2020/04/14ean kontsultatua).

[44] "Service mesh observability and configuration," kiali.io. [Online]. Hemendik eskuratua: https://kiali.io (2020/04/14ean kontsultatua).

[45] Prometheus Authors, "From metrics to insight," prometheus.io, 2020. [Online]. Hemendik eskuratua: https://prometheus.io (2020/04/14ean kontsultatua).

[46] Grafana Labs, "The open observability platform," grafana.com, 2020. [Online]. Hemendik eskuratua: https://grafana.com (2020/04/14ean kontsultatua).

[47] "SSH : Security Vulnerabilities," cvedetails.com. [Online]. Hemendik eskuratua: https://www.cvedetails.com/vulnerability-list/vendor_id-120/SSH.html (2020/05/05ean kontsultatua).

[48] V. Chemitiganti, "Kubernetes Concepts and Architecture," in *The Gorilla Guide to Kubernetes in the Enterprise*. ActualTech Media, 2019. [Online]. Hemendik eskuratua: https://platform9.com/blog/kubernetes-enterprise-chapter-2-kubernetes-architecture-concepts

[49] L. Nikoltsios, "How to secure the Kubernetes API behind a VPN," intruder.io, 2018. [Online]. Hemendik eskuratua: https://www.intruder.io/blog/how-to-secure-the-kubernetes-api-behind-a-vpn (2020/02/23an kontsultatua).

[50] A. Ronacher and contributors, "Flask," palletsprojects.com, 2015. [Online]. Hemendik eskuratua: https://palletsprojects.com/p/flask (2020/02/29an kontsultatua).

[51] "Redis," redis.io. [Online]. Hemendik eskuratua: https://redis.io (2020/02/29an kontsultatua).

[52] "Helm Charts: Status of the Project," github.com/helm. [Online]. Hemendik eskuratua: https://github.com/helm/charts#deprecation-timeline (2020/03/14ean kontsultatua).

[53] "Redis," github.com/bitnami. [Online]. Hemendik eskuratua: https://github.com/bitnami/charts/tree/master/bitnami/redis (2020/03/14ean kontsultatua).

[54] "Redis," github.com/helm. [Online]. Hemendik eskuratua: https://github.com/helm/charts/tree/master/stable/redis (2020/02/29an kontsultatua).

[55] Alpine Linux Development Team, "Alpine Linux," alpinelinux.org, 2018. [Online]. Hemendik eskuratua: https://alpinelinux.org/about (2020/03/14ean kontsultatua).

[56] The Kubernetes Authors, "Use Port Forwarding to Access Applications in a Cluster," kubernetes.io, 2019. [Online]. Hemendik eskuratua: https://kubernetes.io/docs/tasks/access-application-cluster/port-forward-access-application-cluster (2020/02/29an kontsultatua).

[57] Colaboradores de MDN, "Códigos de estado de respuesta HTTP," developer.mozilla.org, 2020. [Online]. Hemendik eskuratua: https://developer.mozilla.org/es/docs/Web/HTTP/Status (2020/03/04ean kontsultatua).

[58] Redis Labs, "Redis cluster tutorial," redis.io. [Online]. Hemendik eskuratua: https://redis.io/topics/cluster-tutorial (2020/04/01ean kontsultatua).

[59] The Helm Proyect, "redis-ha," github.com/helm. [Online]. Hemendik eskuratua: https://github.com/helm/charts/tree/master/stable/redis-ha (2020/04/04ean kontsultatua).

[60] Google Cloud Platform, "Using Prometheus," cloud.google.com. [Online]. Hemendik eskuratua: https://cloud.google.com/monitoring/kubernetes-engine/prometheus (2020/04/08an kontsultatua).

[61] Istio Authors, "Sidecar," istio.io, 2020. [Online]. Hemendik eskuratua: https://istio.io/docs/reference/config/networking/sidecar (2020/04/08an kontsultatua).

[62] Istio Authors, "Applications FAQ," istio.io, 2019. [Online]. Hemendik eskuratua: https://istio.io/faq/applications (2020/03/26ean kontsultatua).

[63] "Building multi-arch docker images," lobradov.github.io, 2018. [Online]. Hemendik eskuratua: https://lobradov.github.io/Building-docker-multiarch-images (2020/03/23an kontsultatua).

[64] Istio Authors, "Ingress Gateways," istio.io, 2020. [Online]. Hemendik eskuratua: https://istio.io/docs/tasks/traffic-management/ingress/ingress-control (2020/03/25ean kontsultatua).

[65] Istio Authors, "Gateway," istio.io, 2020. [Online]. Hemendik eskuratua: https://istio.io/docs/reference/config/networking/gateway (2020/03/25ean kontsultatua).

[66] Istio Authors, "Virtual Service," istio.io, 2020. [Online]. Hemendik eskuratua: https://istio.io/docs/reference/config/networking/virtual-service (2020/03/25ean kontsultatua).

[67] The Container Security Blog on StackRox, "How to Make Istio Work with Your Apps," securityboulevard.com, 2019. [Online]. Hemendik eskuratua: https://securityboulevard.com/2019/11/how-to-make-istio-work-with-your-apps (2020/03/25ean kontsultatua).

[68] "Introduction to Vagrant," vagrantup.com. [Online]. Hemendik eskuratua: https://www.vagrantup.com/intro (2020/06/16ean kontsultatua).

[69] J. Gabirondo-López, "jongablop/minervalab: First version of minervalab," 2020. [Online]. Hemendik eskuratua: https://doi.org/10.5281/zenodo.3893455 (2020/06/14ean kontsultatua).

[70] J. Gabirondo-López eta J. M. Igartua, "jongablop/callistolab: 1.1," 2019. [Online]. Hemendik eskuratua: https://doi.org/10.5281/zenodo.3523220 (2020/06/14ean kontsultatua).