POLITECNICO DI TORINO

Corso di Laurea Magistrale in Ingegneria Informatica - Computer Networks and Cloud Computing

Politecnico di Torino

Tesi di Laurea Magistrale

Sistemi di gestione multi-cluster: Controllo e gestione di orchestratori distribuiti

Relatore

Prof. Fulvio Giovanni RISSO

Tutor aziendale

Dr. Guido VICINO

Candidato

Francesco GALLO

Ottobre 2021

Sommario

Nel 2021, secondo il *Flexera 2021 State of Cloud report* [1], il 78% delle aziende IT basa i propri servizi sul cloud ibrido, utilizzando insieme piattaforme private e uno o più piattaforme pubbliche grazie ai principali public cloud providers. L'approccio maggiormente adottato dalle aziende è di tipo "*one private and multiple public*", rendendo di fatto necessaria la capacità di poter gestire i propri workload sulla propria rete di cluster in maniera centralizzata.
L'obiettivo è introdurre nel mercato uno strumento che permetta di collegare questi cluster in un unico "logico" cluster ed effettuare tutte quelle operazioni che venivano fatte sul singolo, adesso in un contesto multi-cloud. Questo evita di replicare le stesse procedure più volte, doverle riadattare al singolo vendor e interfacciarsi con ogni singolo cluster in maniera diretta.
Purtroppo quello degli strumenti multi-cloud e multi-cluster è un mercato ancora acerbo e sono poche le soluzioni che offrono un prodotto valido e completo. L'obiettivo di questa tesi quindi, è quello di analizzare i più validi strumenti di gestione multi-cluster, studiarne le features e, dopo averli provati in un contesto multi-cloud, definirne pro e contro.
Nello specifico, gli strumenti analizzati saranno principalmente due: **Red Hat Advanced Cluster Management** (soluzione con target enterprise di Red Hat) e **Rancher** (soluzione open-source e license-free).
Sarà poi proposto un caso d'uso dimostrativo, sulla base dell'esperienza lavorativa maturata all'interno di **Blue Reply**, al fine di mostrare com'è possibile risolvere un problema di propagazione di uno stack di logging, sia utilizzando gli strumenti multi-cloud analizzati che non. Per ogni soluzione presentata saranno evidenziati requisiti, costi, vantaggi e svantaggi, definendo quindi quale approccio presenta un rapporto qualità/costi maggiore.

Ringraziamenti

Ringrazio infinitamente i miei genitori, amici e tutti gli affetti più vicini, per avermi accompagnato e sostenuto in tutti i miei studi.

Indice

Elenco delle tabelle

Elenco delle figure

Acronimi

RHACM
Red Hat Advanced Cluster Management

VM
Virtual Machine

IAAS
Infrastructure As A Service

PAAS
Platform As A Service

SAAS
Software As A Service

IDS
Intruder Detection System

NAT
Network Address Translator

K8s
Kubernetes

QoL
Quality of Life

IoT
Internet of Things

AI

Artificial Intelligence

CI/CD

Continous Integration Continous Delivery

OCP

OpenShift Container Platform

GCP

Google Cloud Platform

Capitolo 1

Introduzione

1.1 Il Cloud nel 2021

L'adozione di un business model che preveda l'utilizzo di uno o più cloud pubblici da parte delle aziende, IT e non, è ormai una strategia diffusa. Stando al *Flexera 2021 State of Cloud report* [1], il 36% delle aziende spende più di $1M al mese e la metà di queste mantiene i propri workloads interamente sul cloud pubblico.
Per Cloud pubblico si intende l'insieme d'infrastrutture che erogano servizi verso aziende o consumer privati, offerti da un provider esterno secondo un piano di fatturazione "*pay-as-you-go*" o mensile. Come anche documenta il *Magic Quadrant for Cloud Infrastructure and Platform Services 2021* di Gartner [2], il mercato del cloud pubblico è in mano a tre principali provider: Amazon con **AWS**, Google con **Google Cloud Platform** e Microsoft con **Azure**. Ognuno di questi provider offre un proprio set di servizi che è possibile suddividere in tre categorie principali:

- **IaaS - Infrastructure as a Service**: L'insieme di servizi che permettono di creare, distruggere, migrare virtual machines. Include anche la gestione dell'intera infrastruttura a supporto: rete, storage e sicurezza.

- **PaaS - Platform as a Service**: Soluzioni ready-to-go che offrono al customer la possibilità di rendere attivi applicativi web, server di posta o database senza preoccuparsi degli aspetti tecnici relativi al loro provisioning. Sarà compito del provider occuparsi del layer sottostante permettendo al customer di risparmiare risorse in termini di costi e tempo.

- **SaaS - Software as a Service**: Un insieme di servizi che il provider offre come funzionalità di applicativi proprietari che stanno in esecuzione sulla propria infrastruttura. Solitamente l'interazione col customer avviene tramite applicazione web single-page.

Figura 1.1: Magic Quadrant for Cloud Infrastructure and Platform Services - Fonte: Gartner [2]

Tra i servizi che maggiormente spiccano, ci sono sicuramente quelli relativi al provisioning di virtual machines, cluster Kubernetes, CDN e storage cloud-hosted. Bisogna però notare che l'offerta non è diretta alle sole aziende IT, bensì esistono servizi mirati a chi opera in altri settori come quello del video streaming. L'esempio più comune è quello di **Netflix**, che grazie ai servizi di video encoding e CDN di Amazon AWS, riesce a trasmettere i propri contenuti in tutto il mondo.
Uno dei principali motivi del successo del cloud pubblico è sicuramente il risparmio in termini di risorse e costi. Scegliere di spostare o depositare i propri workload sul cloud significa non doversi preoccupare di costi di manutenzione, facilities e, sopratutto, di risorse umane.
Infatti molte aziende che non dispongono di una forte componente IT, possono

ignorare gran parte dei processi tecnici di più basso livello, dovendo lavorare a un livello di astrazione più alto.
Altro aspetto che necessita una menzione riguarda le aziende che operano nel campo dell'**edge computing**, ora in forte crescita grazie alla diffusione del 5G. In questo campo, il target principale è la *low-latency* e la vicinanza geografica all'utente finale, due proprietà che sono caratteristiche dei big cloud vendor. Grazie alla loro enorme rete globale, riescono a coprire, sebbene con diversi livelli di efficienza, quasi tutte le regioni del globo.

1.2 Il Multi-Cloud

Col passare degli anni, dalla nascita dei principali cloud provider, le aziende IT di tutto il mondo hanno cominciato a spostare i loro workload e dati sul cloud pubblico. Contemporaneamente però, erano molte le aziende che possedevano già una propria infrastruttura privata e ciò significava definire una strategia che permettesse interazione tra il cloud privato e il pubblico.
Nel 2021, sono pochissime le aziende che mantengono tutti i propri workload interamente su un solo provider o un singolo tipo di cloud. Secondo il "*Flexera 2021 State of the Cloud Report*" [1], il 92% delle aziende ha adottato una strategia multi-cloud e di queste, solo il 10% interamente sul cloud pubblico mentre l'82% preferisce il cloud ibrido.
Questi grandi numeri riguardo l'adozione del multi-cloud non dicono nulla sull'interazione tra questi cloud, infatti, specialmente all'inizio molti hanno preferito utilizzare il cloud pubblico come alternativa alla costosa estensione delle risorse del proprio datacenter. Di fatto, continuano a rimanere "*slegati*" dati e risorse, e si sente la necessità di uno strumento che permetta d'interfacciarsi su più cloud come se fosse uno solo.
Ecco che nascono le prime soluzioni di gestione multi-cluster, con supporto al multi-cloud, anche se l'offerta è molto ridotta. Tra le poche disponibili, si menzionino quelle che verranno analizzate in seguito, ovvero **Red Hat Advanced Cluster Management**, d'ora in poi **RHACM o ACM**, e **Rancher**.

Ciò che però accomuna le varie soluzioni è:

- La possibilità di poter creare una rete di cluster in vita su cloud diversi così da doversi interfacciare con un singolo cluster "logico".
- Effettuare il deployment di applicativi in maniera distribuita da una singolo fonte (come Git repository o Helm chart). Comunemente con pieno supporto all'approccio GitOps.
- Singolo punto d'interazione con lo strumento tramite web UI.
- Monitoraggio centralizzato di risorse e salute, con una granularità da interi cluster ai singoli pod.

Bisogna però menzionare che una strategia multi-cloud introduce un carico in termini di complessità, pertanto è necessario che questo approccio venga prima valutato attentamente. Le condizioni che portino all'adozione una soluzione multi-cloud possono essere molteplici, le più comuni sono:

- **Acquisizioni o fusioni**: In questi casi l'adozione al multi-cloud è meno costosa della migrazione verso il singolo.
- **Latenze ridotte**: Ogni cloud provider ha una propria rete di copertura e offre vicinanze geografiche diverse verso gli utenti.
- **Obblighi legali sui dati**: Bisogna preoccuparsi della giurisdizione che opera nelle regioni in cui i dati vengono immagazzinati.
- **Disaster Recovery**: In caso di guasto o indisponibilità di servizio di un provider, i propri dati e le applicazioni resteranno operative grazie alle repliche su un secondo.
- **Evitare il "cloud vendor lock-in"**: Con il termine lock-in si intende quella condizione per cui un'azienda si ritrova costretta a utilizzare lo stesso cloud provider. Questo può succedere quando le proprie applicazioni dipendono da capabilities proprietarie di un vendor.

Si noti però che ogni cloud provider offre un insieme di servizi e strumenti che sono indipendenti da quelli offerti dagli altri provider. Doversi interfacciare su cloud diversi significa anche dover possedere skill su più provider, e ciò non è detto che sia in linea con le risorse in possesso da un'azienda.

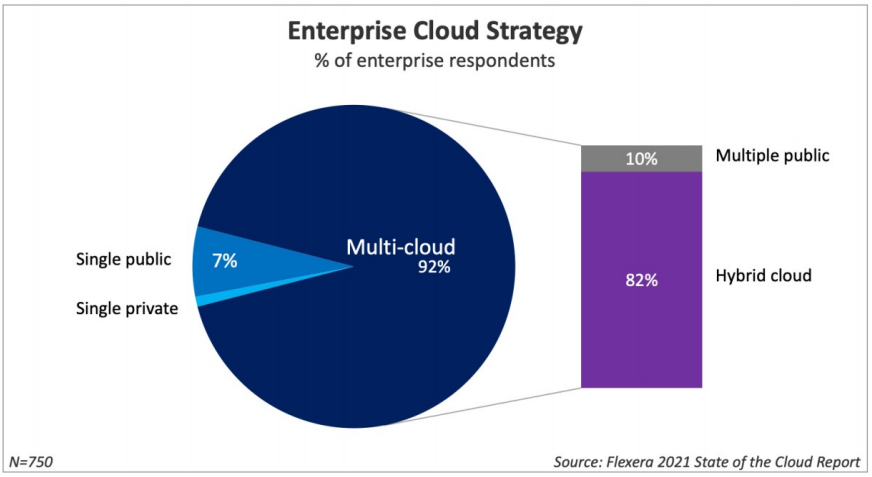

Figura 1.2: Strategie utilizzate dalle varie aziende in termini di multi-cloud - Fonte: Flexera[1]

Capitolo 2

Strategie di orchestrazione su più piattaforme cloud

2.1 Il multi-cloud come soluzione

Come menzionato precedentemente, il doversi interfacciare con più cloud contemporaneamente è una realtà oramai consolidata. Ciò implica dover distribuire anche i propri workload su più piattaforme, che queste siano pubbliche o private.
Con la nascita di **Docker** nel 2013, le aziende IT hanno cominciato a migrare le proprie applicazione da una struttura monolitica a quella a microservizi, la quale rende le applicazioni completamente portatili, *ready-to-deploy* e indipendenti.
Con **Kubernetes** nel 2014 invece, grazie anche al supporto offerto dai principali vendor, le aziende hanno iniziato a effettuare il provisioning di cluster Kubernetes sia su piattaforme pubbliche che private. Da questo momento in poi, il doversi interfacciare con ognuno di questi cluster significava replicare più volte i propri workload.
È divenuto possibile automatizzare questi processi tramite strumenti di automation come **Ansible** o **Jenkins**, che però risultano avere un target molto generico, il che rende le performance di queste soluzioni non ottimali in contesti di soli cluster Kubernetes.
Quindi diventa necessario per le aziende disporre di uno strumento che permetta d'interfacciarsi, tramite una singola interfaccia o via linea di comando, con una moltitudine di cluster su diverse piattaforme.
Questo è facilitato dall'architettura di Kubernetes che è basata su REST Api, ciò si traduce in pura comunicazione HTTP, adatta perfettamente alla rete internet.

2.1.1 Le esigenze di chi si approccia al multi-cloud

Il primo ostacolo in un contesto multi-cloud è quello della creazione della propria rete di cluster e gestione della suddetta come unica entità. Tale procedura deve naturalmente risultare vantaggiosa rispetto alla controparte single-cloud, quindi di semplice realizzazione e compatibile con una moltitudine di vendor. Le soluzioni single-cluster prevedono il mantenimento dei kubeconfig in locale e connessioni one-to-one tramite o linea di comando (*kubectl*) o tools di terze parti (*Visual Studio Code* o *Lens*).
Il continuo "**context-switching**" [1] introduce ritardi in termini di tempo e un critico abbassamento della **Quality of Life** dell'intera procedura.
In contesti di produzione, gestire una moltitudine di cluster come singola rete potrebbe significare un aumento della complessità fuori scala, dovendo porre attenzione a un quantitativo di risorse dell'ordine delle migliaia. Ecco perchè, tra le esigenze di un possibile approccio multi-cloud, bisogna tener conto della "**observability**" ovvero la capacità di poter osservare tutti i cluster in modo efficace. Più nel dettaglio:

- Poter visionare tutte le risorse, come pods, deployment o services, e la loro salute[2]. Eventualmente filtrarli e/o raggrupparli per namespace o label.
- Poter accedere a una panoramica delle risorse hardware utilizzate dai cluster in termini di CPU, memoria e disco. Mostrando anche quelli che sono i limiti di tali risorse e le capacità massime dei cluster.
- Poter essere in grado di effettuare ciò che si è espresso nei punti precedenti, ma a un livello "*multicluster-wide*", quindi accedere a informazioni relative a salute e risorse utilizzate per ogni cluster nella sua interezza.

[1] Qui il termine context-switching deve essere inteso con il suo significato "Kubernetes", e non con quello canonico appartenente ai calcolatori elettronici. Infatti si menzioni il comando "kubectl config use-context clusterA" e derivati, atti alla gestione del cluster di destinazione dei propri comandi

[2] Risorse consumate, stato del pod, etc.

Una volta realizzata una rete di cluster multi-cloud, funzionante e operativa, l'esigenza fondamentale di una figura DevOps o IT Operations, è quella di poter configurare un singolo workload da propagare e distribuire su uno, più o tutti i cluster della rete. Ciò significa che deve essere necessario un back-end solido capace d'inviare comandi ai vari API server dei cluster, saper interpretare le loro risposte e mostrare un riepilogo in maniera chiara all'utente. Oltretutto, deve poter accettare come source per i manifest più alternative, come repository online o helm charts, e saper reagire a eventi esterni come ad esempio delle modifiche alla repository, in modo da poter garantire **CI/CD**.
Infine, si evidenzi anche la capacità di estendere i cluster in maniera dinamica, spesso trascurata ma di vitale importanza quando si è già in possesso di uno o più cluster. L'alternativa di migrare tutti i propri contenuti da un cluster esistente a uno nuovo, risulta spesso onerosa o impossibile da effettuare.

2.1.2 Gli use case più comuni

I contesti in cui viene valutata una soluzione multi-cloud possono essere molteplici, ma non tutti si adattano bene. Si noti infatti che un approccio simile, nonostante i vantaggi, presenta dei costi e overhead non indifferenti: viene introdotta della complessità relativa all'aumento dei cluster e alla loro lontananza, sia logica che fisica.
In questa sezione saranno analizzati gli use case più comuni di un approccio multi-cloud, evidenziando i contesti in cui tale approccio risulta vantaggioso.

- **Acquisizioni o fusioni**: Spesso, a seguito di acquisizioni o fusioni, si ottene la gestione di cluster su dei vendor o piattaforme diverse da quelle già adottate. Inoltre, si noti che, oltre ai cluster, vengono introdotte anche nuove risorse umane con skill tecniche su quelle stesse piattaforme: migrare tutti i workload sulla "propria" piattaforma significa costi e ritardi dovuti alla formazione.

- **Evitare il "lock-in"**: Come già introdotto nel primo capitolo di questa tesi, in un contesto multi-cloud, un fattore importante quando si scelgono le piattaforme che ospiteranno i cluster e le applicazioni, è il **lock-in**. Ovvero evitare che si diventi "vittima" di un singolo vendor, e le cause possono essere molteplici: capabilities proprietarie del suddetto o impossibilità di migrare i propri workload a causa del loro numero.

- **Obblighi legali**: Uno dei vantaggi offerti dai principali vendor è la capacità di coprire una moltitudine di regioni grazie alla loro enorme rete e all'alto numero di datacenter. Ciò significa che, a seguito di dinamiche legali di origine economica o politica, dei dati potrebbero non essere mantenuti in determinate regioni, proprio per la giurisdizione a cui sono sottoposti.

- **Sfruttare i punti di forza di ogni vendor**: Uno dei contesti di maggior utilizzo dell'approccio multi-cloud è quello in cui si cerca di distribuire le proprie applicazioni sui vari vendor a seconda dei loro punti di forza.

Quelli sopra elencati, sono i contesti più comuni in cui una soluzione multi-cloud risulta vantaggiosa o una scelta obbligata. Molti altri invece, nonostante questo approccio si adatti e funzioni bene, prevedono che gli svantaggi e i costi risultino essere troppo alti. Si pensi a contesti dove l'approccio multi-cloud viene valutato solo per disaster ricovery, ovvero mantenere i workload su più provider così che, in caso di disservizio di uno, gli applicativi rimangano disponibili. Purtroppo è una scelta anacronistica in quanto, nel 2021, i principali cloud provider offrono servizi di *high availability* sia a livello intra che inter-regionale, che un loro completo disservizio risulta essere un evento molto raro.

2.1.3 Capabilities fondamentali

Dopo aver chiarito quali sono le esigenze tecniche in un contesto multi-cloud e gli use case in cui l'approccio multi-cloud risulta essere una soluzione ottimale, verranno quindi riepilogati i requisiti funzionali di un gestore di orchestratori distribuiti.

- **Capabilities indispensabili:**
  - **Creazione della rete multicluster**: Lo strumento deve permettere di creare dei cluster su piattaforme remote, effettuando il provisioning in autonomia di tutte le risorse necessaria. Deve essere anche possibile importare quelli già esistenti o eliminarli. È importante che la comunicazione tra cluster avvenga anche in presenza di entità di rete come firewall, reverse proxy o load balancer.
  - **Observability cluster/multicluster-wide**: Lo strumento deve presentare una pagina web in cui siano accessibili i dati relativi sia ai cluster nella loro interezza che alle risorse in essi contenute. Più nello specifico, i dati accessibili riguardano: salute dei pod, possibilità di filtrare le risorse, risorse hardware consumate, allocate, limiti e capacità massima.
  - **Workload multi-cluster con supporto a cluster selectors e CICD**: Lo strumento deve presentare una sezione per la configurazione di workloads da propagare su uno o più cluster. Deve essere possibile scegliere tra più source di manifests (git repository, helm charts, etc), cluster di destinazione e deve essere disponibile il supporto al CICD. In relazione a quest'ultimo, lo strumento, in autonomia, deve essere in grado di aggiornare le release in vita sui cluster a seconda di eventi esterni come un nuovo push sulla repository di riferimento.

  - **Compatibilità alle principali piattaforme cloud pubbliche e private**: Lo strumento deve supportare, nelle sue funzioni principali, l'interazione con i principali cloud provider, sia questi pubblici che privati. I più importanti: AWS, Azure, GCP, IBM Cloud, vSphere.
  - **Architettura centralizzata**: Lo strumento, al fine di migliorare la QoL dei processi d'interazione con la rete multi-cluster, deve presentare un'architettura centralizzata. Nello specifico, deve essere possibile accedere a tutte le sue funzionalità tramite singolo **point-of-contact**, implementato o tramite CLI o web UI.

- **Capabilities opzionali**:
  - **Sistemi integrati di backup e recovery**: Lo strumento può offrire nativamente, quindi senza installazioni o configurazioni ulteriori, dei processi sia di backup che di recovery, così da favorire la disaster recovery. Questi possono riguardare i database etcd o i singoli cluster nella loro interezza.
  - **Scheduling dei workload**: Lo strumento può offrire la possibilità di automatizzare i workload configurati in maniera periodica.
  - **Open license**: Lo strumento non deve presentare costi di licenza, direttamente o indirettamente a causa di componenti di cui necessita.
  - **Spegnimento e accensione remota dei cluster**: Deve essere possibile spegnere e accendere i cluster in maniera remota e senza perdite di dati. Questo permette di risparmiare sulla fatturazione per quei cluster di development o test, i quali spesso non richiedono di rimanere online nelle ore notturne.

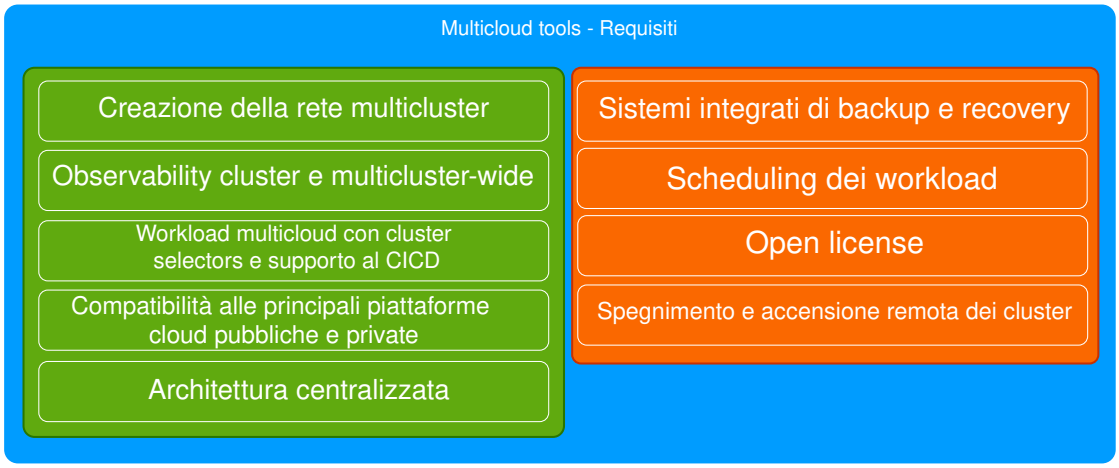

Figura 2.1: Panoramica dei requisiti: in verde quelli indispensabile, in rosso quelli opzionali

2.2 Lo stato dell'arte

2.2.1 Strumenti di gestione di orchestratori distribuiti - Tipologie ed esempi

A oggi, il mercato di questi strumenti è ancora molto acerbo e l'offerta generale non è ampia. Le aziende che provano a sviluppare e distribuire una piattaforma di multi-cluster management spesso utilizzano soluzioni di tipo SaaS. Altre invece, seguono un approccio "self-managed" dove l'applicativo o l'architettura generale dello strumento è in mano a chi lo possiede.
Molte però sono soluzioni che non trovano assoluta adozione da parte dei customer, sopratutto per la scarsità di funzionalità che riescono a offrire. La maggior parte di queste, infatti, si rivelano essere semplici *single-pane-of-glass* della propria rete multi-cluster, senza forti funzionalità d'interazione. Un esempio simile che verrà riportato nel documento è **Platform9 Kubernetes Management**.
Per ciò che concerne le funzionalità, si noti che quasi tutte le soluzioni propongono una forte componente di observability, sopratutto per ciò che riguarda le risorse Kubernetes o hardware. Viene offerta la possibilità di collegare i cluster al cosiddetto "hub centrale" o di crearne di nuovi su uno o più cloud provider, a seconda delle compatibilità.
Non tutti però prevedono workload distribuiti sulla rete multicloud, colonna portante di un buon prodotto. In aggiunta a ciò, questo modello SaaS prevede anche dei costi per il servizio e il mantenimento del suddetto, che risulta essere un overhead economico importante quando si hanno già a carico una moltitudine di cluster.

Le soluzioni invece maggiormente adottate, sopratutto per la loro varietà di capabilities offerte, sono le cosidette "Self-managed", ovvero soluzioni che richiedono installazione e configurazione sulle risorse del tenant. Quest'ultimo si deve preoccupare di tutti gli aspetti sia tecnici che logistici relativi al provisioning dell'applicativo e del suo mantenimento. Generalmente si trattano di applicativi "*plug-and-play*" (helm chart, operators) con piena compatibilità per la piattaforma sottostante, che questa sia Kubernetes o derivati, come OpenShift. Lato funzionalità, i prodotti in questione presentano un'ampia offerta sotto tutti gli aspetti più importanti, sopratutto: workload distribuiti, observability e monitoring. Alcuni nomi di grande rilievo sono: "**Red Hat Advanced Cluster Management**", "**Rancher**" e "**IBM Cloud Pak for Multicloud Management**" (*si noti che le prime due soluzioni elencate precedentemente verranno analizzate successivamente nel documento*).

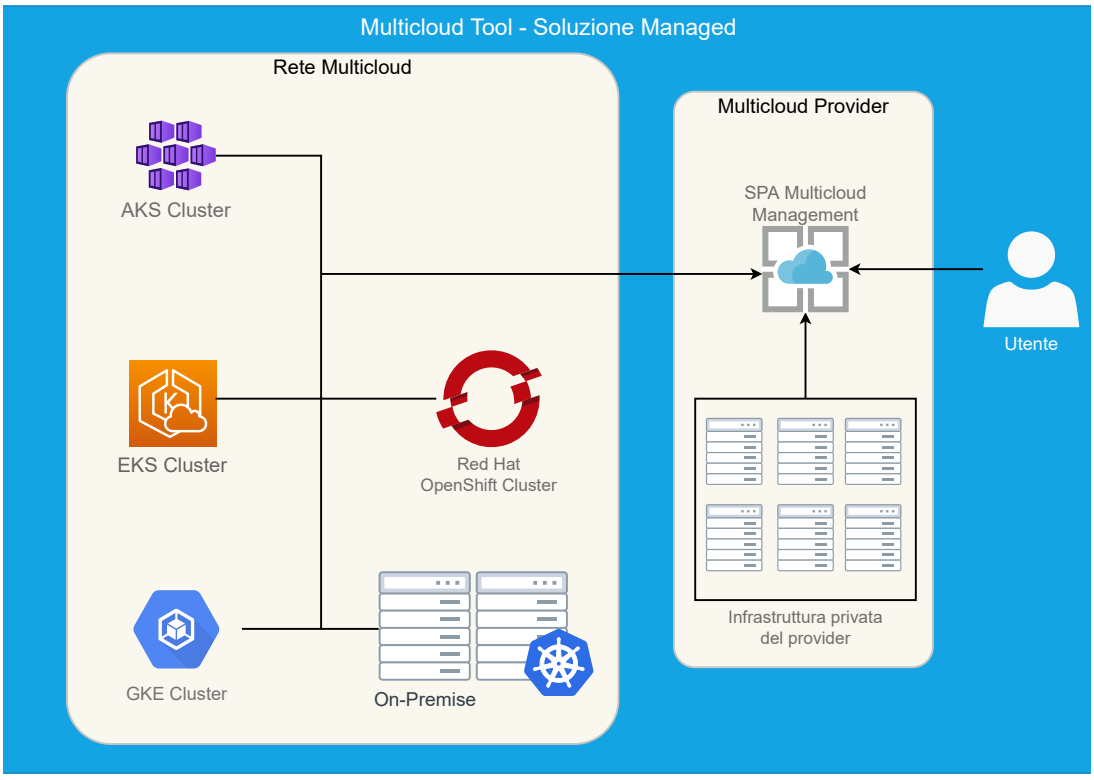

Figura 2.2: Architettura generale di una piattaforma di multi-cluster management managed. Si noti che il management tool viene offerto da un'infrastruttura privata del provider, che quindi offre il servizio all'utente secondo un modello SaaS.

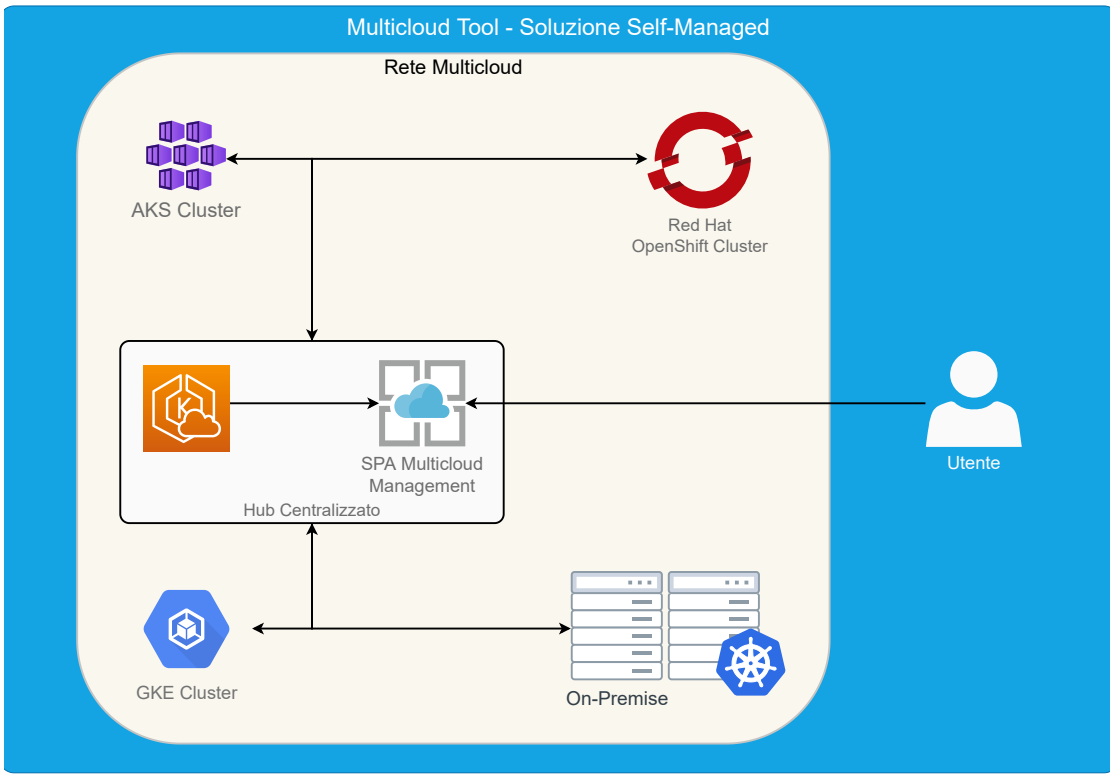

Figura 2.3: Architettura generale di una piattaforma di multi-cluster management self-managed. Si noti che il management tool viene offerto da un cluster interno alla rete multi-cluster, quindi in gestione all'utente stesso.

2.2.2 Alternative "single-cloud"

In quei contesti dove, per motivi di natura economica o logistica, non è possibile adottare un approccio multi-cloud o, date le condizioni, tale approccio risulta essere svantaggioso, vengono adottati business process di automazione.
Gli strumenti più usati e solidi sono sicuramente: **Ansible**, **Jenkins** e **Terraform**, i quali oramai presentano una forte compatibilità con i cluster Kubernetes, riuscendo a trattare i suddetti come veri e propri host di destinazione, al pari di VM. L'utilizzo di questi strumenti diventa necessario a causa dell'esigenza di poter distribuire e tenere aggiornati gli applicativi in esecuzione sui propri cluster, nonostante questi siano mantenuti da provider diversi e divisi dal punto di vista del networking.
Si noti però che questi strumenti sono sviluppati per poter interagire con diversi tipi di host, non solo cluster Kubernetes, e questa loro estrema compatibilità e adattabilità si traduce in un'interazione non ottimale in contesti puramente K8s. Continua a rimanere insoddisfatta la necessità di un entrypoint centralizzato da cui monitorare tutti i vari cluster, spesso risolto lasciando che siano i cluster stessi

ad "auto-monitorarsi" tramite sistemi di monitoring e alerting, notificando criticità o warning tramite mail o simili.
Volendo analizzare invece i costi, questi spesso coincidono con il provisioning di una *Ansible Tower* o server *Jenkins*, e quindi del provisioning di una singola VM. Il contributo più importante proviene dalla preparazione e testing dei singoli workload, *playbooks* per Ansible e *pipeline* per Jenkins, la cui realizzazione è ben più impegnativa della procedura offerta da uno strumento multi-cluster. Infine la loro adozione è dovuta, nella maggior parte dei casi, dal fatto che questi esistono già nelle infrastrutture di un'azienda e ciò permette di risparmiare sul provisioning e installazione.
Viene citato **Jenkins X**, che insieme ad altri nuovi strumenti cloud native come **Tekton**, sta creando ciò che sarà il nuovo standard de-facto dei tools DevOps-ITO per il cloud. Più nel dettaglio, Jenkins X presenta un set di strumenti, tra cui Tekton, installabile su cluster Kubernetes tramite Helm chart e che offre, secondo tutti i regimi di un'applicazione cloud-native, possibilità di: workload multi-cloud e multi-cluster tramite pipeline Tekton, CI/CD e, opzionalmente, log aggregation e monitoring.

2.2.3 Considerazioni finali sul fog ed edge computing

Tutte le considerazioni fatte precedentemente, sono state fatte in un contesto di "**Central Cloud**" o **Core della rete**, ovvero quella sezione della rete occupata dai "mega datacenter". Mentre non bisogna escludere che è ai confini della rete, o "edge", dove si affacciano gli utenti finali, i quali con il passare degli anni e il progredire degli standard tecnologici, richiedono latenze sempre più basse.
Una prima innovazione in ambito tecnologico è stato il 5G, che nel 2021 ha già una sua rete stabile, sia customer che enterprise, e permette ai provider di offrire alte performance all'edge della rete. Questo, unito all'ormai standard raggiunto dai cluster Kubernetes, prevede che gli applicativi delle principali applicazioni come **IoT**, **Cloud-gaming** o **AI Driving**, siano gestiti da cluster posti all'edge. Si noti però che questi applicativi presentano maggiore "delicatezza" rispetto alle applicazioni in vita nel core, in quanto il loro contesto d'utilizzo prevede forti responsabilità. Si porti l'esempio di applicativi che gestiscono le auto a guida autonoma, in tal caso un aumento nella latenza o disservizio provoca un disastroso abbassamento delle performance e/o danni a terzi. Questo comporta la necessità lato business di dover gestire e monitorare questi cluster in maniera efficiente, e spesso l'unica soluzione è un approccio distribuito su una rete multi-cluster distribuita.

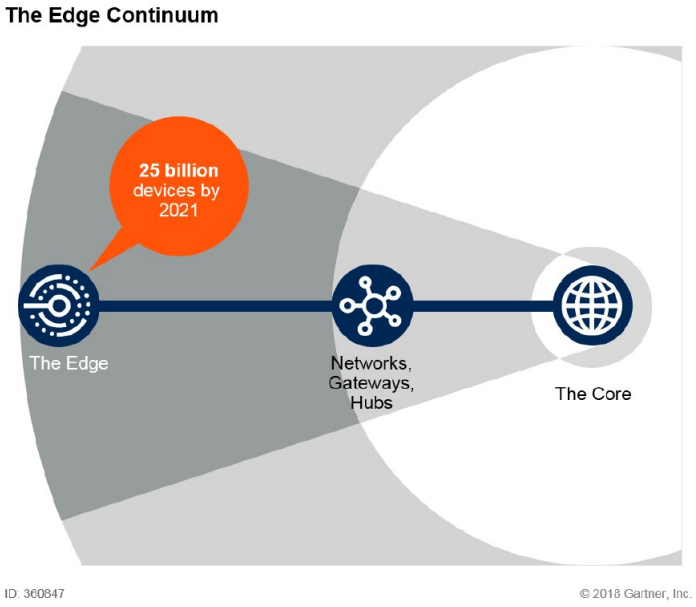

Figura 2.4: Distribuzione degli utenti nella rete - Fonte: Gartner 2018[3]

Capitolo 3

Analisi dei principali Multi-Cluster Management Systems

Dopo aver introdotto i concetti fondamentali e l'importanza del cloud nel 2021, e aver approfondito nel dettaglio il multi-cloud come soluzione, in questo capitolo verranno presentati due strumenti multi-cloud che sono stati scelti per la loro forte offerta in termini di funzionalità. L'analisi prevede di analizzare i seguenti aspetti:

- **Requisiti e installazione**: Vengono prese in considerazione tutte le risorse fisiche, come CPU, memoria e spazio su disco, e logiche, come versioni K8s richieste o piattaforme da cui dipendono. Quindi la possibilità di utilizzare il prodotto su architetture vecchie o meno. All'analisi si aggiunge anche la procedure d'installazione, inclusi aspetti come facilità, tempistiche e, se necessaria, configurazione.

- **Supporto e compatibilità**: Nello specifico, quanti cloud provider sono supportati dal prodotto, sia questi pubblici o privati.

- **Architettura e raggiungibilità dei cluster**: Avendo lavorato in un ambiente multi-cloud, si sono affrontate anche problematiche relative alla raggiungibilità tra i cluster, in contesti articolati come nel caso di reti private e protette da entità come firewall o proxy.

- **Creazione della propria rete multi-cluster**: Sono state utilizzate, su una topologia di prova, tutte le funzionalità relative alla creazione di nuovi cluster da zero su infrastrutture pubbliche. In seguito, anche quelle relativa all'import di cluster pre-esistenti, all'aggiornabilità e alla distruzione dei cluster della

propria rete. Nello specifico, si è posta l'attenzione su aspetti come facilità nell'effettuare la procedura, chiarezza ed efficienza.

- **Observability e monitoring**: Quanto bene e chiaramente lo strumento mostri tutte le informazioni relative a risorse e salute dei cluster in un'interfaccia grafica. Vengono prese in considerazione anche eventuali feature aggiuntive come filtraggio di risorse per progetto, namespace, label o simili.
- **Workload distribuiti**: Per ogni strumento, sono state analizzate le capabilities relative alla distribuzione di un workload su più cluster. Nel dettaglio, sono state tenute in considerazione aspetti come: supporto al CICD, compatibilità con source diverse, presenza di cluster selectors ed efficacia e chiarezza della procedura.
- **Backup, Recovery e Migration**: Se previste, sono state utilizzate le procedure di backup, recovery e di migrazione dello strumento.

Al fine di garantire una buona analisi, si è deciso di adottare una strategia di tipo ibrido, collocando quindi i propri cluster sia su provider pubblici che su un datacenter privato. Nello specifico:

- **On-Premise**: Un host VMware vSphere su cui si è effettuato il provisioning delle VM. Il datacenter è protetto all'interno della rete aziendale e il traffico da e verso internet è gestito da un proxy + firewall.
- **Pubblico**: Il provider utilizzato è stato Azure, il quale ha ospitato tutti i cluster relativi alla parte "public" della rete multi-cloud.

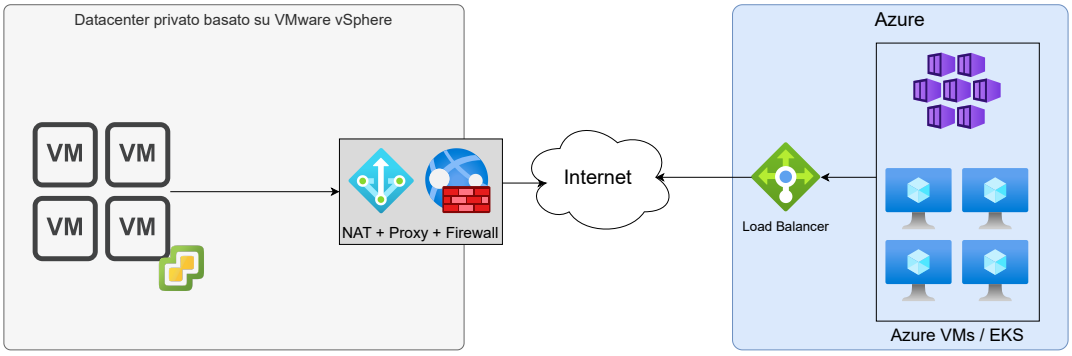

Figura 3.1: Topologia generale su cui i prodotti multi-cloud sono stati analizzati

Prima di procedere con l'analisi, viene definito il workflow generale che è stato seguito durante tutto il processo di analisi:

1. Vengono pianificati: topologia, numero di cluster e i relativi provider, considerando costi, tagli di macchine e numero di VMs da utilizzare. Infine sono state studiate le comunicazioni e le connessioni necessarie alla rete multi-cloud per funzionare, così da inserire se necessario NAT o proxies.

2. Vengono creati i primi cluster, a seconda della piattaforma di riferimento (OpenShift o Kubernetes), e verificato la loro salute e funzionamento.

3. In accordo con la topologia decisa al passo uno, viene installato il multi-cloud tool nel cluster delegato ad hub centralizzato per la rete. Quindi il prodotto viene installato, configurato e ne viene verificato il corretto funzionamento.

4. Una volto creati i cluster e installato il prodotto, viene creata la rete multi-cloud importando il secondo cluster, o se non fosse disponibile l'import, viene creato attraverso il prodotto.

5. Dopo aver approfondito le source supportate dallo strumento relativamente a propagazione di un workload, vengono preparate le suddette con un deployment di prova.

6. Vengono configurati uno o più workload a seconda di quante source sono supportate dallo strumento. Per ogni workload, questo viene fatto attraverso il wizard guidato dello strumento e, se supportato, vengono utilizzati cluster selector e/o features addizionali.

7. Viene monitorata la propagazione del deployment attraverso l'interfaccia e, una volta propagato, vengono esplorate le capabilities di osservazione, quindi si verifica la salute del cluster e risorse. Qui inoltre vengono utilizzate, se disponibili, altre feature come shell interattiva per i pod o shell interattiva per il cluster.

8. Se è supportata la CI/CD, questa viene verificata andando a effettuare modifiche sulla source di riferimento e controllando che quest'ultime vengano propagate sul cluster.

9. Infine, viene lasciato spazio a feature proprietarie o esclusive del prodotto.

Figura 3.2: Schema generale del workflow adottato per analizzare gli strumenti

3.0.1 L'approccio GitOps

Prima di procedere con l'analisi dei principali strumenti, è necessario definire prima cosa sia e come funzioni l'approccio GitOps, il quale sarà largamente usato nella propagazione di applicativi negli strumenti presentati.
L'approccio GitOps è un'implementazione del **Continous Integration - Continous Delivery** (CICD) che si integra fortemente con l'utilizzo di git e repository. Viene largamente usata in contesti cloud native e pone le sue basi su un principio fondamentale, quello di avere una repository il cui stato deve riflettere quello desiderato in un ambiente di produzione. A supporto di ciò deve essere predisposto un **automated process** che si occupa di riflettere eventuali cambiamenti dal repository all'ambiente di produzione.
Negli strumenti analizzati in seguito, questo automated process sarà già integrato tra le capabilities di propagazione di un workload distribuito.

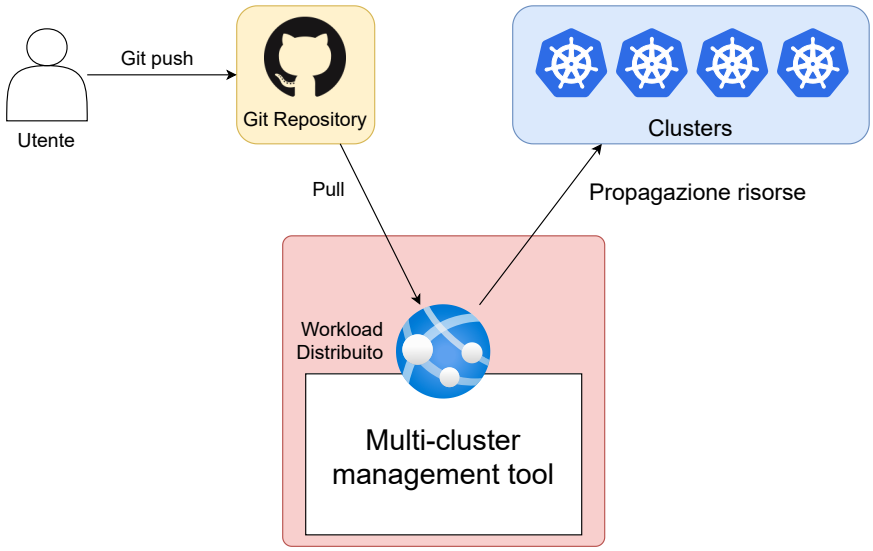

Figura 3.3: Funzionamento generale di un approccio GitOps inegrato a un sistema di gestione multi-cluster

3.1 RHACM – La soluzione multi-cloud proposta da Red Hat per OpenShift

3.1.1 Introduzione

Secondo il "The Forrester Wave™: Multicloud Container Development Platforms, Q3 2020" [4], **RedHat-IBM** è leader nel settore in quanto a piattaforme per la gestione e lo sviluppo di container in contesti di cloud ibrido e multi-cloud. Il prodotto di riferimento è **RedHat OpenShift Container Platform**, una piattaforma Kubernetes-based con un target puramente enterprise.
Il prodotto offre, in soluzione "*bundled*", un set di funzionalità che risultano aggiuntive rispetto a quelle che si trovano in un cluster Kubernetes vanilla. Tra le funzionalità più di rilievo si elencano:

- Sistema di monitoring integrato basato su Prometheus.
- Supporto al logging system **EFK** (Elasticsearch - Fluentd - Kibana) tramite CRD.
- Sistema di security context integrato che permette di personalizzare i permessi per account.
- Un sistema di autenticazione integrato con supporto a **LDAP** esterni e Active Directory.
- Sistema di funzionalità aggiuntive, o *Operators*, basato su marketplace,
- Integra una SDN proprietaria che offre un sistema di route e ingress controller.

Il prodotto però richiede, per motivi d'integrazione, che sia installato su macchine **Red Hat Enterprise Linux CoreOS**, una distribuzione open-source Linux che integra strumenti dediti alla gestione di container e virtualizzazione. Essendo OpenShift creato per lavorare su un singolo OS, ciò rende l'integrazione tra i due molto rigida e garantendo performance e stabilità sotto tutti i punti di vista.

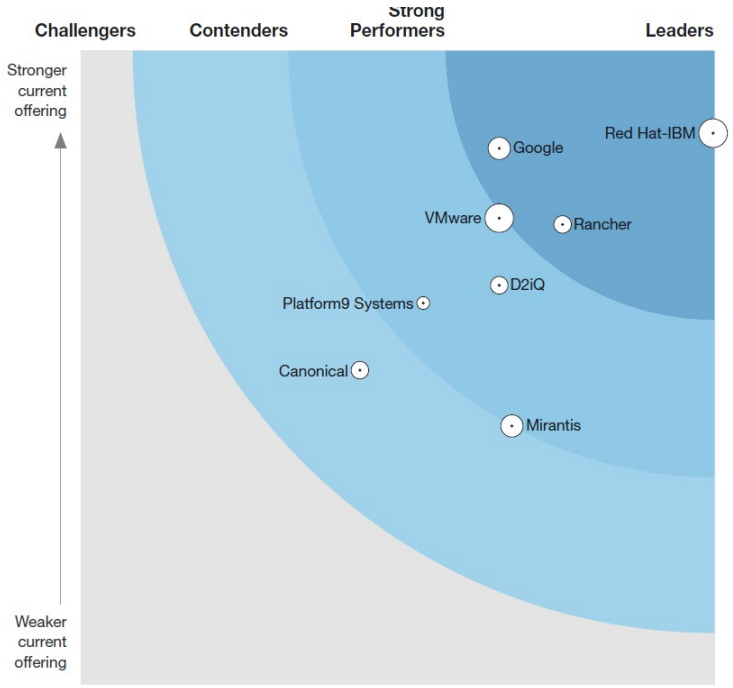

Figura 3.4: Principali piattaforme per lo sviluppo di container in contesti di cloud ibrido e multi-cloud. Si noti come RedHat e Rancher rientrino tra i leader del settore, a differenza di Platform9 Systems - Fonte: The Forrester Wave™: Multicloud Container Development Platforms, Q3 2020[4]

Con **Red Hat Advanced Cluster Management for Kubernetes**, Red Hat ha voluto introdursi nel mercato dei sistemi di gestione di cluster distribuiti puntando sulla forte diffusione delle piattaforme su cui pone le basi il prodotto, ovvero OpenShift.
Infatti RHACM si propone come un Operator di facile installazione per qualsiasi cluster OpenShift che soddisfi i requisiti minimi. Lo strumento propone un'interfaccia grafica capace di offrire visibilità e controllo in maniera completa su tutte le fasi di lifecycle di cluster e applicazioni, distribuiti su una rete multi-cloud.

I principali punti di forza dichiarati sono:

- Viene introdotta visibilità tra i cluster, permettendo all'approccio DevOps di poter operare in maniera efficiente.
- Possibilità di distribuire applicazioni su larga scala
- Gestione centralizzata del ciclo dei vita dei cluster della propria rete, e ciò include: creazione, aggiornamento e distruzione, sia su bare-metal, piattaforme di virtualizzazione o infrastrutture pubbliche.
- Raccoglie automaticamente dati e metriche su salute, risorse e log dai cluster così da poter effettuare auditing.

3.1.2 Architettura, costi e requisiti

Come introdotto precedentemente, il prodotto prevede uno stack tecnologico interamente firmato e sviluppato da Red Hat, nel dettaglio:

- **L1 - Sistema operativo RHCOS**: Ciò garantisce tutte le primitive che Red Hat ha ottimizzato ad-hoc per poter interagire al meglio con OpenShift.
- **L2 - OpenShift Container Platform**: Core engine di ACM, che integra Kubernetes e ne estende le funzionalità grazie a CRD predisposte da Red Hat in fase d'installazione.
- **L3 - Operatore RHACM**: Il prodotto in questione, che viene mantenuto da OpenShift come un canonico operator e che espone tutte le funzionalità che verranno analizzate meglio in seguito.

Sul fronte invece della topologia della rete, il prodotto prevede una struttura a stella, dove il centro coincide con il cluster su cui è in esecuzione l'operator. Tale cluster viene definito come "**hub centrale**" mentre tutti i cluster rimanenti che sono connessi al suddetto, vengono definiti come "**managed cluster**". Tutte le comunicazioni tra i cluster avvengono bidirezionalmente su due endpoint principali:

1. **API Server**: Banalmente, per poter effettuare tutte le richieste primitive offerte sia da API native di K8s, OpenShift o custom dell'utente. L'endpoint esposto risulta by default: *https://api.<**FQDN**>:6443*
2. **Endpoint di autenticazione**: Questo è un endpoint nativo di OpenShift che espone la rotta verso il sistema di autenticazione del cluster. L'endpoint esposto risulta by default: *https://oauth-openshift.apps.<**FQDN**>*

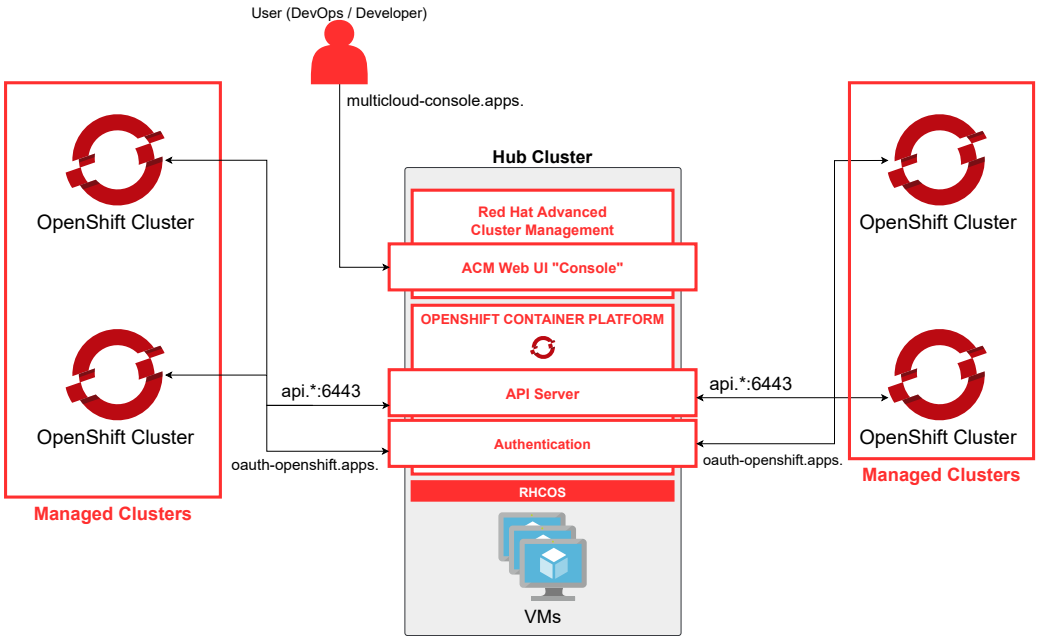

Figura 3.5: Architettura di Red Hat Advanced Cluster Management

Attenzione: *Si noti come le comunicazioni avvengano in entrambi i sensi e non sia possibile modificare porta o FQDN dell'API e authentication server. Quindi è di estrema importanza considerare eventuali NAT, firewall o reverse proxies che possano cambiare gli header dei pacchetti di rete. In caso di reverse proxies, si tengano in considerazione eventuali problemi dovuti allo scambio di certificati SSL:*

- *Eventuali certificati su endpoint esposti dal reverse proxy devono risultare attendibili. Certificati self-signed impediranno il peering tra i due cluster.*
- *Eventuali hostname contenuti nel certificato diversi da quelli esposti, potrebbero creare conflitti con l'hostname di destinazione contenuto nell'header HTTP.*

Per ciò che concerne invece i costi del prodotto, questi non sono per nulla banali, tutt'altro. Red Hat Advanced Cluster Management eredita gli stessi requisiti hardware della piattaforma OpenShift che prevedono[1]:

- Numero di nodi master: 3 (*minimo*) o più
- Numero di nodi worker: 2 o più
- CPU: 6 vCPU
- RAM: 16 GB
- Disco: 120 GB
- Versione Openshift: 4.3 o superiore

Come si evince dai requisiti, **OpenShift Container Platform** è altamente costoso in termini di risorse e non prevede single-node cluster, ciò lo rende inadatto a contesti di testing, sperimentazione o sviluppo. Nonostante Red Hat offra una piattaforma di e-learning con laboratori interattivi su un loro cluster, diviene veramente costoso il training e la sperimentazione, sopratutto in ambienti con poche risorse, come quello accademico.
I costi relativi alle risorse fisiche non sono gli unici, infatti se si vuole ricevere supporto ufficiale su un cluster OpenShift è necessario l'acquisto di una licenza.[2].

[1]I requisiti qui riportati sono approssimativi e stimati sulla base dei requisiti di Red Hat OpenShift sulle diverse piattaforme (AWS, GCP, Azure, vSphere, Bare-metal)

[2]Si noti che Red Hat fornisce, per cluster, una licenza di self-evaluation di 60 giorni.

3.1.3 Ambiente e topologia in fase di analisi

Qui di seguito verranno evidenziati, nel dettaglio, i componenti dell'intera topologia della rete multi-cluster ed eventuali componenti esterne.

- **Infrastruttura privata basata su VMware vSphere**: La componente on-premise della topologia si basa su un host all'interno di un'infrastruttura privata di **Blue Reply**. L'host in questione sfrutta **VMware vSphere** per effettuare il provisioning di virtual machines. Per quanto concerne la connettività, a causa di componenti come NAT, reverse proxy e Firewall, che gestiscono i flussi in ingresso e in uscita, è stato necessario predisporre una sesta macchina, definita come "helper node", per mantenere servizi come DNS, NAT e proxy.
  - **Virtual Machines**: Le virtual machines sono macchine AMD64 su cui è stata installata la versione 4.3 di RHCOS e sono "segregate" in una sotto-rete virtuale, esposte all'esterno grazie all'helper node che ricopre anche il ruolo di gateway. Grazie all'abbondanza di risorse hardware, sono stati superati di gran lunga i requisiti minimi richiesti da OpenShift. *Si noti che è consigliabile fornire risorse extra al cluster hub, poiché RHACM introduce un overhead in termini di hardware consumato.* La versione di OpenShift installata è la 4.3[3].
  - **Helper Node**: Il nodo cosiddetto di "helper" è stato necessario sia a fini d'installazione in ambiente privato, poiché erano necessari: server DNS, HAProxy, NAT, default Gateway, web server e secure tunneling endpoint. Proprio in merito a quest'ultimo, si evidenzi che è stato necessario a causa di NAT e proxies posti in uscita dalla rete del datacenter. I dettagli di queste problematiche verranno approfondite in seguito.

[3]Non si è potuto installare una versione più recente a causa d'incompatibilità tra gli script d'installazione e la versione di vSphere.

- **Infrastruttura pubblica basata su Microsoft Azure**: Per la componente pubblica si è scelto di utilizzare Azure per agevolazioni logistiche. Tutte le risorse all'interno del gruppo di risorse sono state create a posteriori, poiché il cluster è stato creato attraverso RHACM che si è occupato di effettuare il provisioning di tutte le componenti e della loro configurazione. Di seguito, le più importanti:
  - **Virtual Machines**: Le virtual machines scelte sono basate su architettura AMD64, e i tagli scelti sono in linea con i requisiti minimi. La versione di OpenShift installata è la 4.7.
  - **Gruppo di sicurezza di rete**: Componente che copre sia il ruolo di firewall che di NAT, permettendo o impedendo determinate connessioni da e verso l'esterno.
  - **Sistema di bilanciamento del carico**: Il servizio di load balancing offerto da Azure, qui permetteva la distribuzione delle richieste sui soli noi master.
  - **Zona DNS privata**: Zona DNS interna per la risoluzione di hostname privati.
- **Componenti extra**:
  - **Dominio pubblico**: È stato necessario registrare un dominio pubblico per poter raggiungere i vari cluster. Questo poiché i cluster OpenShift non ammettono raggiungibilità via IP, bensì necessitano del FQDN del cluster con cui collegarsi in fase di creazione della rete multi-cluster.
  - **Nginx Reverse proxy**: È stato necessario predisporre a posteriori un reverse proxy Nginx per tradurre le richieste da Azure all'hub da 6443 a 443. Ciò è dovuto all'utilizzo del tunnel sicuro tramite Ngrok e ciò verrà approfondito in seguito.

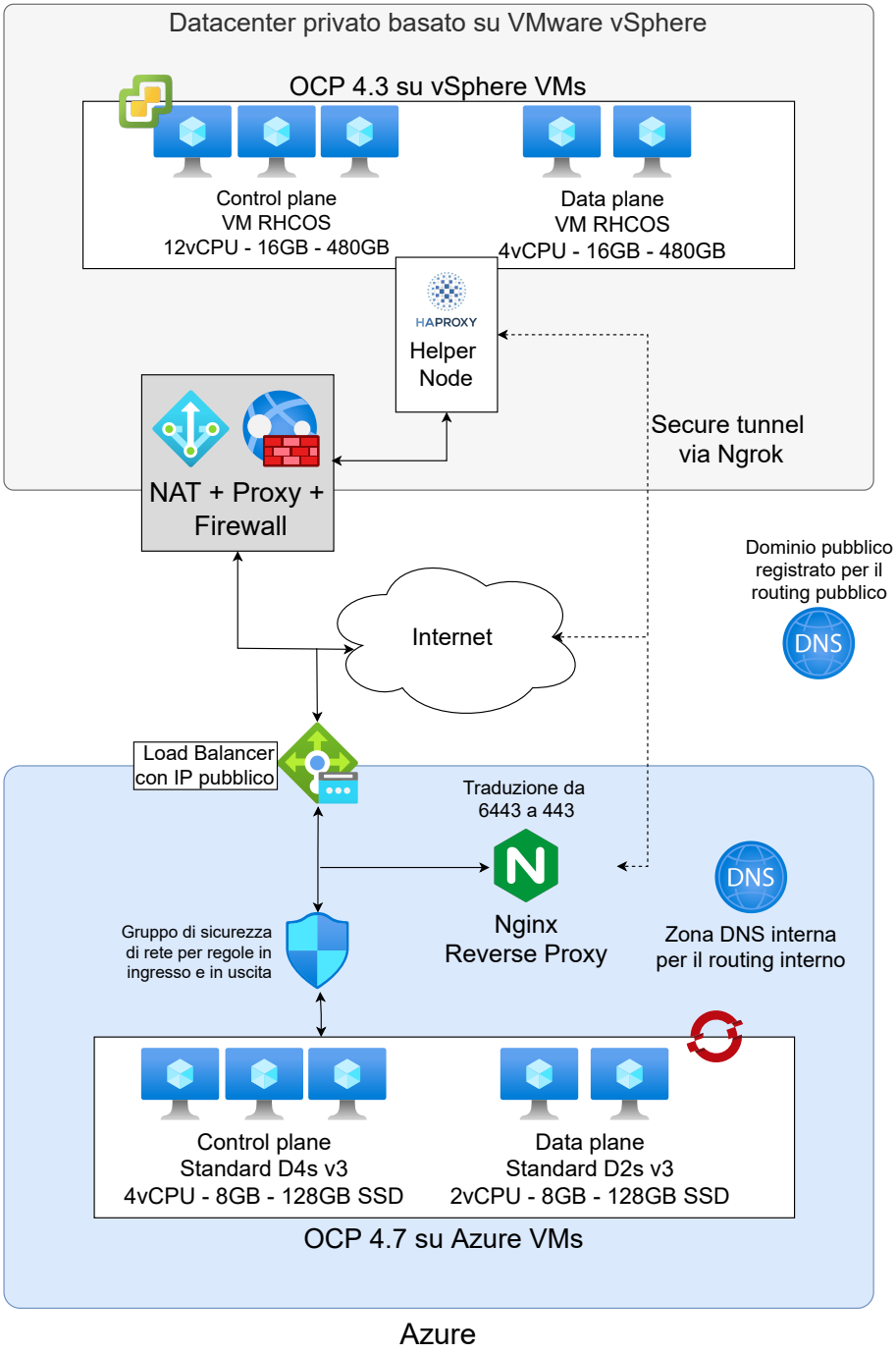

Figura 3.6: Architettura e topologia adottate durante l'analisi di Red Hat Advanced Cluster Management

3.1.4 Installazione dello strumento

Red Hat Advanced Cluster Management si presenta come un Operator installabile facilmente dall'OperatorHub nativo di OpenShift. La procedura d'installazione non prevede grandi configurazioni o complicanze, infatti se sono soddisfatti i requisiti di versione della piattaforma OpenShift, basterà avviare l'installazione e sarà OpenShift che si occuperà del provisioning.
Una volta conclusa l'installazione però è richiesto che si crei una CRD "**MulticlusterHub**", che contiene tutta la logica relativa al hub centralizzato, tra cui l'interfaccia web e la rotta esposta per l'utilizzo del prodotto.
Durante l'analisi, sono emersi i seguenti punti riguardo il processo d'installazione:

- Il prodotto presenta un processo d'installazione guidato e molto semplice.
- Offre la possibilità di scegliere il namespace di destinazione, e le strategie di aggiornamento (automatico o manuale) dell'operator stesso.
- È possibile installarlo anche tramite CLI od offline, ottenendo il pacchetto dai repository ufficiali.
- Viene creato un managed cluster definito come "*local-cluster*" che corrisponde al hub centralizzato, infatti questo viene gestito da ACM come un normale cluster della rete.

Per concludere, si evidenzi che non sono stati riscontrate limitazioni o problematiche di alcun tipo.

3.1.5 Gestione della rete multicluster - Creazione, Aggiornamento e Distruzione

Creazione o import di un cluster

Il passo fondamentale e successivo all'installazione è espandere la propria topologia, e ciò può avvenire in due modi: creando un cluster da zero o importandone uno già pre-esistente.
Per ciò che concerne la creazione di un cluster, lo strumento prevede di configurare preventivamente una **Provider Connection**, un oggetto contenente tutte le informazioni indispensabili all'accesso su un'utenza cloud, oppure un **Bare Metal Asset**, che invece è necessario nel caso in cui si volesse creare il cluster in contesto bare-metal.
Una volta configurata una PC, è possibile accedere al wizard guidato per la creazione di un cluster sulle seguenti piattaforme:

- AWS
- Azure
- GCP
- vSphere
- Bare-metal

Il processo è altamente configurabile e prevede tutti gli aspetti relativi a:

- Versione di OpenShift - *Si noti che sono presenti solo le ultime versioni di OCP, e dipendono dal provider che si è scelto.*
- Dimensione del cluster - È possibile definire numero di nodi e tagli delle relative macchine, così da poter dimensionare il cluster a seconda dei propri utilizzi e anche la fatturazione relativa.
- Networking - Sono configurabili tutti gli aspetti relativi a nome dominio, CIDR e plugin di rete da utilizzare

Una volta configurata e avviata la creazione, autonomamente il prodotto provvederà alla creazione di tutte le risorse necessarie al funzionamento del cluster sul provider di destinazione. Una volta completata, questo verrà automaticamente importato all'interno della rete.
Lato import la procedura è naturalmente più breve e molto meno configurabile, infatti una volta scelto il nome ed eventuali labels, verrà generato un comando da lanciare sul cluster di destinazione. Questo genererà le risorse necessarie all'import sul cluster, che autonomamente si collegheranno all'endpoint dell'api server dell'hub sulla porta 6443.

Aggiornamento di un cluster

Se un cluster dovesse avere una versione non **latest** di OpenShift, questo verrà notificato sulla pagina sommario dei cluster connessi. È necessario indicare la versione a cui aggiornare il cluster, e questo comincerà il processo in maniera del tutto trasparente.
Ciò è dovuto al principale vantaggio di basare tutti i propri cluster su una piattaforma dedicata, che integra in sé un sistema di aggiornamento solido. *Si noti però che, nonostante la bontà del sistema di aggiornamento, è fortemente consigliare effettuare i backup dell'etcd dei vari nodi.*

Distruzione e detachment di un cluster

Per dimensionare la propria rete, ACM offre pure funzionalità di distruzione e rimozione di un cluster.
La **distruzione** è possibile solo per i cluster che sono stati creati attraverso ACM e il processo rimuoverà il cluster permanentemente, non permettendo più di recuperarlo in futuro.
La **rimozione** invece prevede di rimuovere il cluster dalla propria rete, non interferendo in alcun modo sul suo ciclo di vita. *Si noti però, come riportato anche nella documentazione, che alcune risorse create durante l'import sopravvivano al processo di rimozione, costringendo quindi a effettuare un intervento manuale. Red Hat al riguardo offre uno script che permette la rimozione automatica dei residui, che però dopo un'esperienza personale, non si è dimostrato molto efficace.*

Connettività tra i cluster e problematiche riscontrate

È necessario evidenziare, in quest'analisi, anche le criticità relative alla connettività tra i cluster e le problematiche riscontrate.
In fase di pianificazione, a causa di una documentazione molto povera lato architetturale, non erano state previste alcune componenti come il tunnel server **Ngrok** e il reverse proxy a monte del cluster su Azure (*si faccia riferimento alla fig. 3.5*). In fase d'import del cluster su Azure, non risultava possibile completare la procedura e sono state riscontrate molte difficoltà a effettuare operazione di debug a causa di log poco chiari.
Dopo un'approfondita analisi è stato chiaro che:

- La comunicazione tra cluster prevede l'interazione di due endpoint da entrambe le parti:
  - Api server - *https://api.<FQDN>:6443*
  - Authentication server *https://oauth-openshift.apps.<FQDN>*

In aggiunta a ciò, la comunicazione risulta essere bidirezionale (**il che non è scontato, come si vedrà durante l'analisi di Rancher in seguito**), e questo trovava criticità con le componenti di rete poste in ingress sul datacenter privato, tra cui NAT e Reverse Proxy.
Nel dettaglio, le connessioni in ingresso verso vSphere presentavano le seguenti problematiche:

- Il certificato aziendale installato sul proxy imponeva che anche l'FQDN del cluster debba essere coerente ai nomi DNS. In caso contrario, il cluster su Azure riscontrava errori di sicurezza e terminava la connessione. Inoltre viene richiesto un certificato con wildcard fino a L3 o L4, così da coprire endpoint come ***.apps** o **api**.

- Il reverse proxy, in alcune configurazioni, prevede di redirigere il traffico sull'IP andando a eliminare ciò che di fatto è l'hostname e il path, che risultano indispensabili al routing di OpenShift per riconoscere il servizio da chiamare.

- Impossibile personalizzare porta ed endpoint dell'api server, questo costringe l'infrastruttura esterna a doversi adattare a OpenShift e ciò non è sempre possibile.

Non potendo quindi esporre direttamente su internet le rotte di autenticazione e api, si è scelto di adottare una soluzione che prevede:

- **Lato vSphere**: Installare un tool di secured tunnel (**Ngrok**) sul nodo helper così da esporre delle rotte su internet, grazie all'infrastruttura di Ngrok, con certificati attendibili forniti dal suddetto. Unico accorgimento è che tali tunnel gestiscono traffico o HTTP o HTTPS sulle relative porte 80 e 443, e ciò è un problema per l'api server che espone la 6443. Per ovviare a ciò, è stato necessario inserire nella configurazione di HAProxy, una regola di reindirizzamento del traffico verso l'api server con porta 6443.

- **Lato Azure**: Il comando d'import generato da ACM continuerà a indicare come indirizzo dell'api server la porta 6443 che, per i motivi spiegati prima, non viene esposta su internet. Quindi per coprire anche il traffico da Azure verso vSphere, è bastato installare un reverse proxy Nginx che automaticamente, in caso di traffico verso quell'endpoint, traduca la porta da 6443 a 443, risolvendo di fatto il problema.

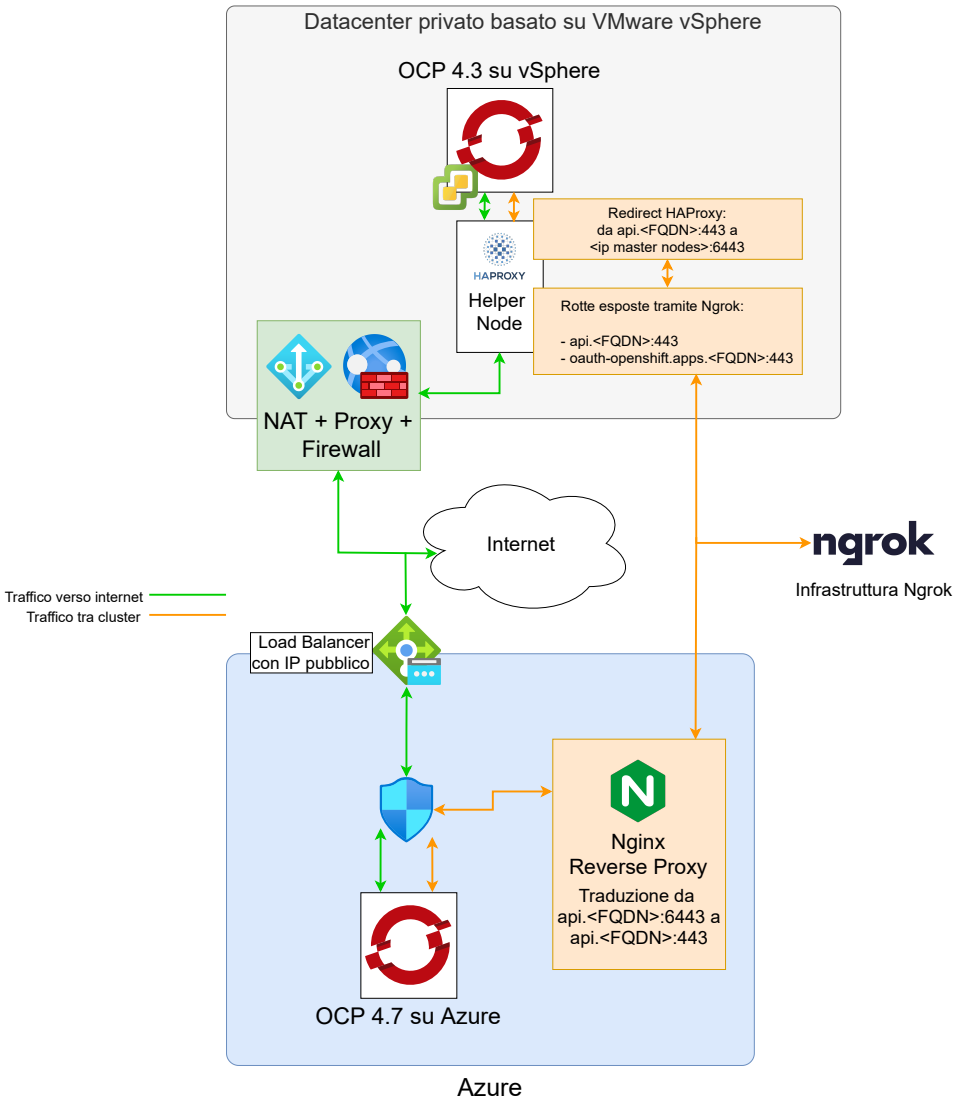

Figura 3.7: Workaround adottato per il traffico tra i cluster, così da ovviare alle complicanze introdotte da NAT e reverse proxy del datacenter.

3.1.6 Gestione dei workload multicloud

Componenti principali e la loro interazione

L'architettura che Red Hat ha definito per ciò che concerne la gestione di applicazioni distribuite, si compone di quattro componenti principali:

- **Application**: Oggetto proprio di RHACM che definisce il concetto di applicazione. Contiene tutte le risorse che sono necessarie all'applicativo e permette all'operator di riconoscere quali applicativi sono attualmente distribuiti tramite ACM, e con quali label.
- **Channel**: Risorsa che corrisponde all'astrazione di un canale verso la source dell'applicativo, sia questo sotto forma di manifest Kubernetes in una repository **Git**, **Helm chart**, **Template** o **Object storage**.
- **Subscription**: Risorsa che permette il collegamento tra l'Application, Channel e Placement Rule. Grazie a essa, i cluster sapranno dove reperire i manifests. *Si noti che è possibile definire più Subscription per singola Application.*
- **Placement Rule**: Oggetto che contiene tutte le espressioni da soddisfare affinché un cluster sia incluso nella propagazione dell'applicativo di riferimento.

Il prodotto prevede che l'utente non debba occuparsi del provisioning dei singoli componenti[4] e della loro interazione, bensì è necessario solo definire i parametri principali come la source dei manifest e le regole di cluster selection.
Una volta che è stato effettuato il provisioning di tutte le componenti, ACM, grazie alla Placement Rule, riconosce quali cluster scegliere come "destinazioni" dell'applicativo, mentre la Subscription effettua il watching della sorgente, e applicando una strategia **CI/CD** aggiorna le risorse installate in locale. Questa logica viene applicata in tutti i cluster di destinazione che dispongono di una porzione di risorse utili allo scambio di messaggi con l'hub centrale, installate in fase di creazione o import.

[4]Si noti però che ciò non preclude all'utente la possibilità di applicare gli YAML di ogni componente in maniera del tutto manuale.

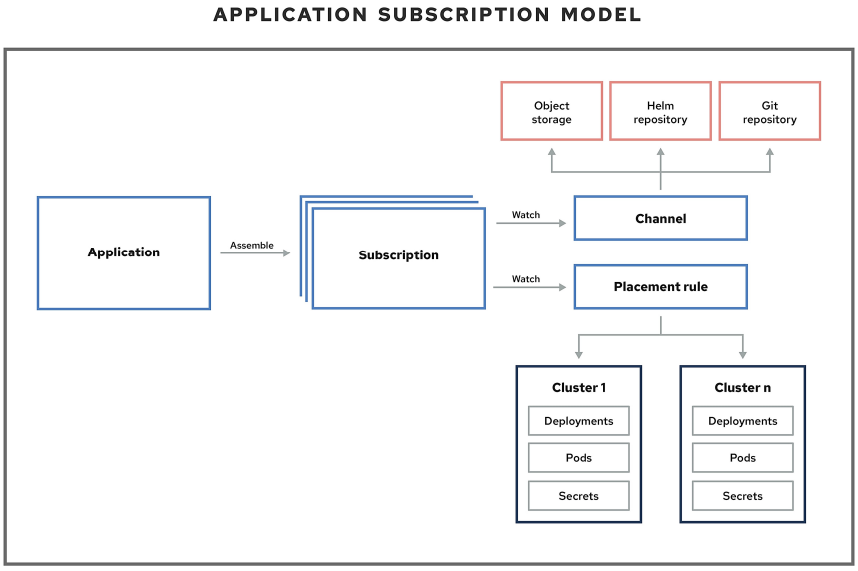

Figura 3.8: Componenti della gestione di applicativi in ACM - Fonte: Documentazione RHACM

Creazione di un workload distribuito

Il processo di creazione di un workload distribuito si presenta come procedura guidata all'interna della web console, oramai punto principale d'utilizzo dell'operator. Vengono di seguito riportati parametri e sezioni principali:

- **Informazioni Generali**: Vengono richiesti nome dell'applicativo e namespace di destinazione.
- **Repository location per le risorse**: Questa sezione prevede di aggiungere una o più source da cui ottenere le risorse. Le opzioni selezionabili sono: Git, Helm, Object storage.
  - **Git**: In caso di manifests contenuti in una repository git, è indispensabile indicare l'indirizzo della repository ed eventuali credenziali, se questa fosse privata. È possibile inoltre indicare il path e il branch, così da poter creare una gerarchia di cartelle e poter riunire più applicativi in una singola repository.
  - **Helm**: In caso Helm chart, basterà indicare gli stessi parametri necessari alla CLI di helm per l'installazione di un chart, quindi indirizzo della repository e nome del chart. Purtroppo questa opzione risulta essere ben più povera rispetto alla precedente, in quanto non è possibile personalizzare

il chart con valori custom rendendo, di fatto, inutile l'utilizzo di questa opzione. Più avanti verrà analizzata la problematica e presentati alcuni workaround.

  - **Object Storage**: Opzione molto simile alla Git repository, solo che si fa riferimento a un Object storage tramite indirizzo e credenziali.

- **Cluster Selection**: Di grande importanza la sezione di selezione dei cluster. ACM offre opzioni molto generiche in caso si voglia propagare l'applicazione su tutti o un cluster, ma anche più complesse tramite "*label-matching*". Quest'ultima prevede di effettuare dei match sulle label assegnate ai cluster della propria rete.

- **Scheduling e Time Window**: Si noti anche che ACM offre come funzionalità puramente aggiuntiva, la possibilità di configurare una finestra temporale nel quale rendere operativo o meno l'applicativo. Offre granularità in termini di fuso orario, giorni della settimana e ora del giorno.

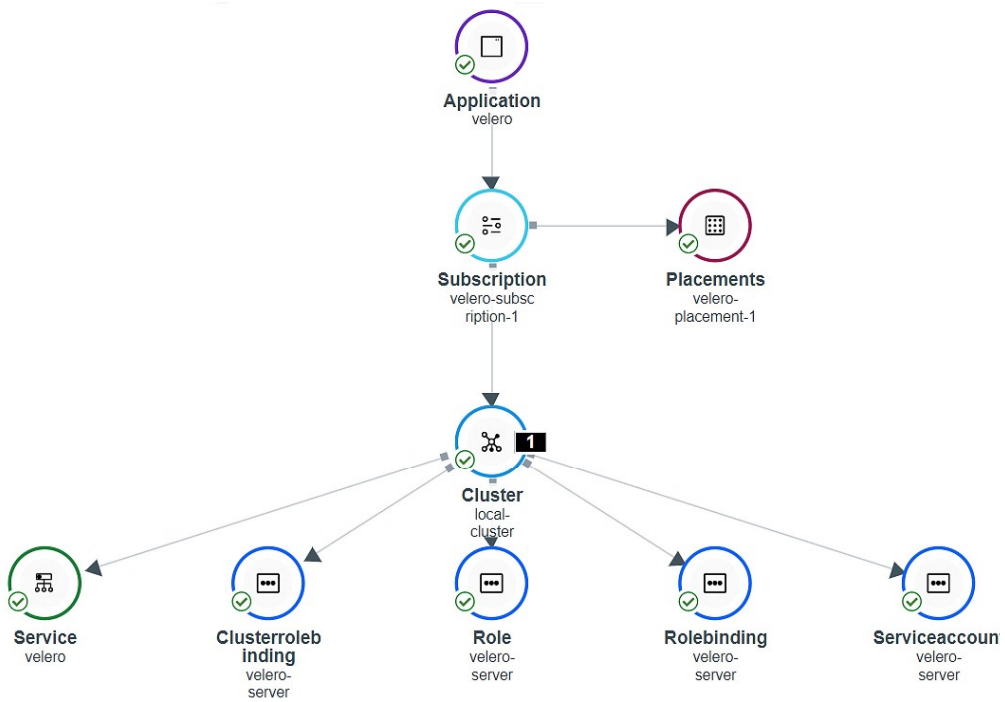

Figura 3.9: Esempio di topologia di un'applicazione propagata tramite ACM

Dopo aver iniziato il processo di propagazione, verranno create tutte le componenti descritte nella sezione precedente, e sarà presentata una topologia di tutte le risorse Kubernetes relative all'applicativo. Dall'Application di ACM fino ai manifest recuperati dalla source configurata, e per ognuna di queste risorse viene mostrato lo stato (*Creata, In creazione, Warning, Errore*) per ogni cluster in cui è in esecuzione.

Considerazioni finali - Punti di forza e limiti riscontrati

Durante tutto il lavoro di analisi, sono stati configurati più workload, sia tramite helm che git, tutti attraverso la procedura guidata. Quest'ultima si presenta sicuramente molto chiara e semplice, valorizzando sicuramente la QoL e la user experience. Non è stato mai necessario dover riallineare o modificare lo yaml prodotto dalla procedura.
È stata inoltre verificata la bontà della CI/CD, che rivela, tramite delle *git pull* se l'ultimo commit coincide o meno con quello mantenuto in memoria. In caso contrario vengono automaticamente propagate tutte le variazioni, allineando tutte le istanze dell'applicazione.
Per quanto concerne le diverse source selezionabili:

- Recuperare i manifest tramite **Git** è l'opzione più flessibile e configurabile in quanto si ha il pieno controllo della repository, tanto che è possibile inserire in una repository sia manifest che helm charts e questi verranno automaticamente riconosciuti da ACM e installati di conseguenza. In accordo con la documentazione, è consigliato strutturare le proprio repository in modo che sia possibile associare a ogni sotto cartella una singola applicazione, in quanto è permesso configurare il path di riferimento di una git repository. Così facendo è possibile, all'interno della stessa repository, differenziare uno o più applicativi e personalizzarli di conseguenza.

- Riguardo invece all'utilizzo come source di un **Helm chart**, l'analisi ha evidenziato numerosi limiti che rendono la scelta ancora acerba. Si evidenzi infatti che un Helm chart è un bundle di Kubernetes manifests parametrizzato che in fase d'installazione viene compilato con dei valori scelti dall'utente. Purtroppo in ACM non è presente la possibilità d'inserire dei values personalizzati, rendendo quindi **inefficace** la scelta di utilizzare tale opzione. In aggiunta a ciò si noti che, spesso risulta fallace l'installazione di un helm chart generico, a meno che questo non sia stato creato appositamente per una piattaforma OpenShift, in quanto potrebbero mancare alcune risorse aggiuntive proprie di OpenShift, come per esempio i **Security Context**[5], che molto spesso hanno reso impossibile completare l'installazione.

In merito al secondo punto, durante l'analisi è stato scoperto che il watching della repository Git presenta un'intelligenza ben maggiore rispetto a quanto presente nella documentazione ufficiale.
È infatti possibile inserire nella repository un Helm chart[6] che, se provvisto di un file *values.yaml* personalizzato, verrà installato utilizzando i valori custom.
Ma il supporto agli Helm chart non è l'unico riscontrato, è presente anche il supporto a **Kustomize**, un sistema di parametrizzazione di template nativo per Kubernetes. Utilizzando un file *kustomize.yaml*, insieme ai manifest di riferimento, questo verrà correttamente "consumato" da ACM.
Come esposto precedentemente, l'utilizzo degli helm chart risulta impossibile se non integrandoli in una repository Git, anche se è presente il rischio di conflitti con i sistemi di sicurezza aggiuntivi introdotti da OpenShift. Una soluzione attuabile, possibile grazie alla flessibilità di utilizzare una repository Git, è rappresentata dall'utilizzo sia di Helm chart che di manifest predisposti *ad-hoc* per quel deployment, rendendo quindi l'applicativo *ready-to-go* per un ambiente OpenShift.
Per ciò che concerne invece il monitoring della propagazione dell'applicativo, sono emersi diversi problemi durante l'analisi, sopratutto quando si è utilizzato ACM per effettuare il deployment di applicativi più complessi, comprendenti CRDs. Il monitoring dei progressi tramite ACM è risultato inefficace, in quanto erano presenti forti ritardi di sincronizzazione rispetto all'effettivo stato sul cluster di destinazione. Oltretutto, spesso lo stato di provisioning delle CRDs risultava fermo quando queste erano state installate correttamente sul cluster di destinazione.

[5]I Security Context sono oggetti introdotti in OpenShift che permettono di gestire i permessi di pod, grazie a un sistema di tipo RBAC.

[6]Si noti che è necessario inserire tutti i file necessari all'installazione di un chart, tra cui templates, values e Chart.yaml

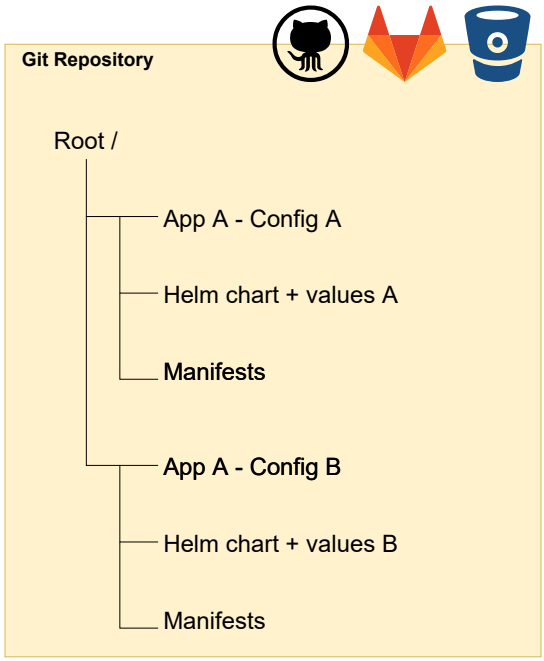

Figura 3.10: Possibile esempio di una gerarchia di cartelle in repository Git-based. Si noti come sia possibile sfruttare la gerarchizzazione per mantenere due copie della stessa applicazione con configurazioni diverse.

Durante il lavoro di analisi, è stato propagato un deployment **"Restic - Velero"** preparato ad hoc in un repository GitHub così da adattarsi alle misure di sicurezza di OpenShift. La propagazione è avvenuta con successo in circa 60-90 secondi, nonostante il monitoring della suddetta, risultando fuori sync, presenta dei ritardi nel mostrare i progressi.
Per concludere quindi, si elencano i **principali punti** emersi durante l'analisi:

- Wizard guidato di facile utilizzo, efficace e dall'alta configurabilità.
- Supporto a sorgenti su git repositories molto flessibile e potente. È possibile utilizzare manifest statici, template tramite Kustomize, Helm chart personalizzati o un soluzione ibrida tra queste.
- L'utilizzo di Helm chart tramite helm repository inefficace, manca la possibilità di usare valori personalizzati.
- Monitoring della propagazione poco performante. Presenta problema di sincronizzazione rispetto allo stato del cluster di destinazione e alcune risorse rimangono in pending anche se correttamente installate.
- Documentazione relativa alla gestione delle applicazioni, a tratti incompleta e poco accurata.

3.1.7 Backup, Recovery e Migration

In merito a strategie di **disaster recovery**, ACM non presenta alcuno strumento, né relativamente a Backup, Recovery, né Migration.

- In merito a Backup e Recovery, questo è comprensibile dato che già OpenShift presenta degli strumenti molto solidi, sotto forma di script e già predisposti all'interno del filesystem dei nodi RHCOS. Gli script prevedono di effettuare il backup dell'etcd e, in caso di failure, effettuare la recovery. *Si noti che, come in accordo con la documentazione ufficiale, il processo di recovery è possibile solo con backup effettuati con la stessa versione di OCP.*
- Red Hat non prevede alcuna metodologia definita per la migrazione dell'hub centrale. In caso si volesse cambiare hub, sarà necessario replicare le risorse sul nuovo cluster in maniera del tutto autonoma.

3.1.8 Considerazioni finali sul prodotto

Red Hat Advanced Cluster Management for Kubernetes si presenta come un prodotto molto stabile, grazie alle già forti fondamenta di OpenShift. Dopo averlo analizzato nei suoi aspetti più importanti, si evince immediatamente che lo strumento riesce nel suo intento. Le capabilities dichiarate hanno quasi tutte un riscontro pratico e inoltre, grazie al forte supporto di un'azienda leader come Red Hat, gode di piena compatibilità con tutti i principali vendor sul mercato.
I difetti più importanti sono stati riscontrati in fase di creazione della rete multi-cluster e propagazione dei workload distributi. In entrambi i casi, non è stato possibile trovare riscontro nella documentazione ufficiale in merito alle problematiche incontrate. Sono state necessarie interazioni dirette col supporto Red Hat per scoprire alcuni dettagli tecnici mancanti nella documentazione.
La sua forte stabilità e il suo ampio set di capabilities però hanno un costo non indifferente. Si ricorda che la piattaforma su cui pone le basi ACM, richiede che siano soddisfatti requisiti minimi esigenti che, nelle reti multi-cluster più complesse, portano a costi elevatissimi. Per questo motivo, è necessario che sia chiaro che tale prodotto **è consigliato solo per contesti enterprise**, e non è adatto ad ambienti di test, sviluppo o dove sia necessario poter spegnere i cluster in determinati momenti del giorno o modularli con piccoli tagli, eventualmente anche single-node.

Si evidenziano dunque i punti principali del prodotto, a seguito dell'analisi svolta:

- Insufficiente: La funzionalità è incompleta e inefficace, ciò la rende inutilizzabile per l'uso dichiarato dalla documentazione del prodotto.
- Sufficiente: La funzionalità non è completa ma presenta tutti i punti fondamentali e indispensabili per l'utilizzo base. Potrebbe presentare problematiche di criticità più o meno elevata, ma che comunque non ne compromettono il funzionamento di base.
- Buono: La funzionalità presenta tutti i punti fondamentali per il corretto funzionamento, è efficace e stabile. Si presenta però ancora migliorabile sotto alcuni punti.
- Ottimo: La funzionalità è completa sia di tutti gli aspetti fondamentali che di eventuali feature extra, il tutto in linea con quanto dichiarato, stabile ed efficiente.

Tabella 3.1: Riepilogo dei principali punti dello strumento RHACM a seguito dell'analisi

Funzionalità	Note	Valutazione
Installazione, compatibilità e costi		
Requisiti d'installazione	Relativi al solo Operator	Ottimo
Compatibilità con cloud vendor e piattaforme on-premise		Ottimo
Facilità d'installazione		Ottimo
Costi (licenza, risorse hardware, etc.)	Molto alti in termini di hardware. Disponibili licenze di self-evaluation.	Buono
Possibilità di spegnere e accendere i nodi	Richiede intervento manuale sul sistema di rinnovamento dei certificati.	Insufficiente
Gestione della rete multicluster		
Creazione, aggiornamento e distruzione di cluster remoti		Buono
Import e detachment di cluster remoti		Buono
Connettività tra cluster	Documentazione poco chiara. Poca adattabilità in datacenter privati, protetti da NAT e proxies.	Sufficiente
Gestione dei workload distribuiti		
Compatibilità in termini di source		Buono
Efficacia della procedura guidata	Semplice e chiara. Nessuna necessità di configurare manualmente file YAML	Buono
Efficacia della source Git Repository	Molto flessibile. Supporta Kustomize, Helm, Manifests.	Buono
Efficacia della source Helm Chart	Inefficace. Impossibile usare values custom.	Insufficiente
Efficacia della CI/CD		Ottimo
Efficacia della propagazione	Manca un sistema di report di eventuali errori nel caso la propagazione non riesca ad andare a buon fine.	Sufficiente

Monitoraggio dei progressi	Spesso fuori sincronia, in ritardo e non esatto.	Sufficiente
Cluster selectors	Semplice, basato su labels.	Buono
Observability		
Efficacia dell'interfaccia	Chiara, è possibile accedere a tutte le informazioni con efficacia.	Buono
Capabilities offerte	E' possibile filtrare le risorse, sia cluster che multicluster-wide, per vari campi.	Buono
Disaster Recovery		
Backup e Recovery	Ereditati da OCP.	Sufficiente
Migration	Non presente.	Insufficiente
Feature aggiuntive		
Time window per applicativi distribuiti		Buono
Shell interattiva via UI	Permette d'interagire con i cluster. Include *helm, oc, kubectl*	Ottimo
Documentazione		
Completezza e precisione	Mancano dettagli tecnici sulla comunicazione tra cluster. Non viene menzionata l'impossibilità di usare custom values per helm charts. Non viene menzionato che è possibile utilizzare Kustomize e Helm charts in una repository git.	Sufficiente

3.2 Rancher

3.2.1 Introduzione

Nella sezione precedente è stato proposto il prodotto di punta di Red Hat per la gestione di cluster distribuiti, e il motivo di questa scelta è stata la posizione leader della suddetta nel contesto del multicloud. Ma come già evidenziato dal report "*The Forrester Wave™: Multicloud Container Development Platforms,Q3 2020*" (*vedi fig. 3.3*), oltre Google e la già citata Red Hat, il terzo leader nel settore è **Rancher**. Rancher è sviluppato e mantenuto da **Rancher Labs**, un'azienda software con un enorme interesse verso l'open source. Infatti, oltre alla stabilità del prodotto, Rancher è stato scelto per la sua forte componente open source, che diventa quindi l'antitesi di RHACM.
Il prodotto si presenta come una soluzione altamente compatibile con le piattaforme cloud più diffuse, prevede piena integrazione con cluster Kubernetes managed (AKS, EKS, GKP), self-managed tramite IaaS, bare-metal e piattaforme private cloud come VMware.
Vengono elencate le funzionalità di maggior rilevanza:

- Pieno supporto a repository Helm, integrando un marketplace espandibile da cui installare tramite GUI chart e configurare values personalizzati.
- Offre un sistema di monitoring e logging integrato basato sui tools più diffusi come Prometheus, Grafana e Fluentd.
- Interfaccia centralizzata di controllo e monitoraggio della propria rete multi-cluster.
- Supporto a sistemi di autenticazione esterni come LDAP, Active Directory e SAML.

Rancher non richiede piattaforme di Kubernetes management, bensì Kubernetes nativo, rendendo quindi integrabile nella propria rete anche cluster *lightweight* a scopi di Iot e 5G. Si evidenzi però che il Kubernetes nativo non è più l'unica opzione quando si cerca un orchestratore di container, infatti Rancher Labs mantiene anche altri progetti come **K3S** e **RKE**.

- **Rancher Kubernetes Engine** è una distribuzione di Kubernetes interamente CNCF[7] compliant che permette l'installazione di K8s direttamente all'interno di container Docker. Ciò permette di facilitarne l'installazione e il mantenimento, in quanto aggiornamento o rimozione ereditano gli stessi vantaggi dei Docker container.
- **K3S** è una distribuzione di Kubernetes, anch'essa interamente CNCF compliant, che si presenta molto più light rispetto al Kubernetes classico. È stata sviluppata con il pieno interesse verso IoT, edge computing e tutti quei contesti a bassissima disponibilità di risorse computazionali, inoltre presenta pieno supporto per piattaforme ARM.

Grazie quindi all'enorme backend tecnologico mantenuto da Rancher Labs e ai bassi requisiti hardware, Rancher riesce ad avere piena integrazione su tutte le piattaforme Kubernetes, permettendo di costruire una rete multicloud altamente scalabile, a basso consumo di risorse e completamente license-free.

[7]Cloud Native Computing Foundation (CNCF) rappresenta uno standard realizzato per tutti i progetti open-source che mirano al cloud al fine di renderli competalmente vendor-neutral.

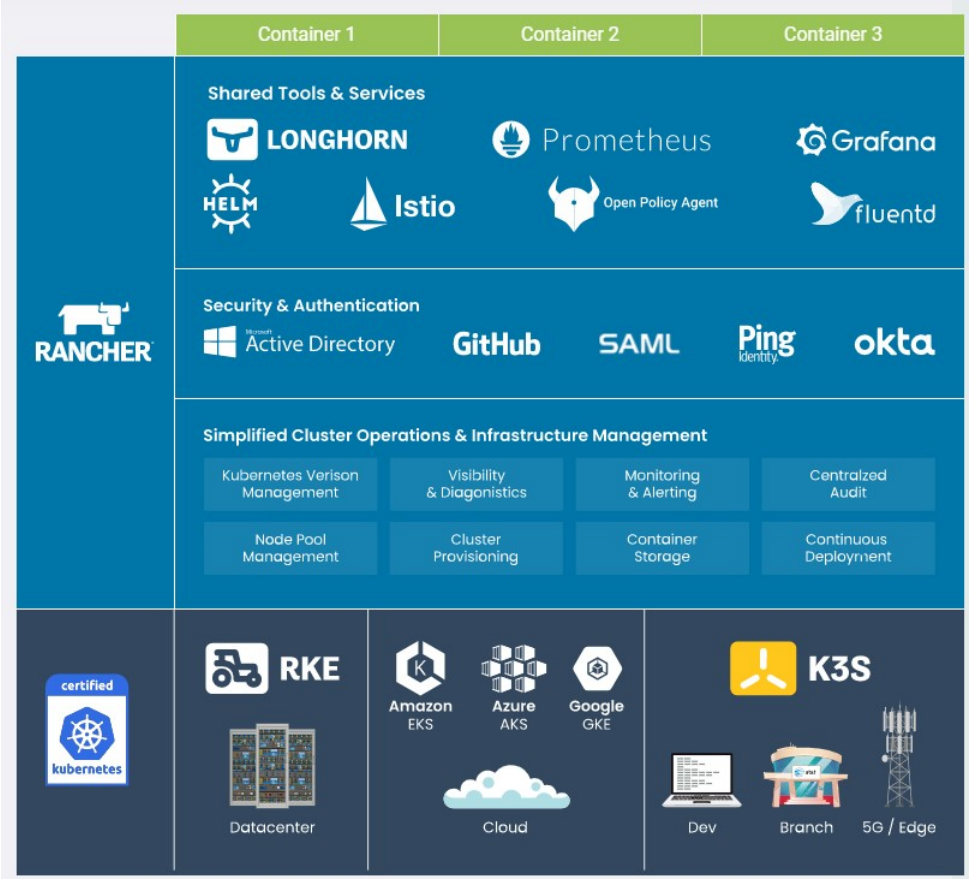

Figura 3.11: Riepilogo di Rancher: piattaforme supportate, features e tools integrati - Fonte: Rancher

3.2.2 Architettura, costi e requisiti

L'architettura del sistema prevede componenti sia sul hub centralizzato che nei cluster della rete, nel dettaglio le più importanti:

- Hub centrale o "**Rancher server**": Applicativo che offre tutte le capabilities di gestione di cluster e workload. È offerto come Docker container così da poter essere eseguito sia su singole VM (per ambienti di sviluppo o test) o in cluster dedicati (altamente raccomandato in contesti di produzione).
  - **Authentication proxy**: Componente che si occupa dell'autenticazione all'interno di Rancher, integrando sistemi esterni come AD o LDAP. Attua

anche funzionalità di proxying, adattando e reindirizzando le richieste verso le API dei cluster remoti.

  - **Cluster Controller**: Oggetto che si occupa di effettuare il watching dello stato di un cluster remoto così da allinearlo al "desired state". La comunicazione con il cluster di destinazione avviene tramite un tunnel.

- **Managed cluster**: Cluster all'interno della propria rete multicluster. Può avere fondamenta diverse, a seconda che si sia scelto K3S, RKE o Kubernetes nativo.

  - **Cluster agent**: Agent che coincide con l'endpoint del tunnel creato dal Cluster controller. Riceve le richieste verso le API del cluster dal Rancher server e le applica, propagando quindi workload e gestendo pod. Si occupa infine di condividere informazioni e metriche relative al cluster e alle sue risorse.

Data il grande numero di piattaforme Kubernetes che supporta, la comunicazione tra i cluster e il Rancher server non è definita da un unico modello, bensì da diversi a seconda del tipo di piattaforma utilizzata in downstream.
Questo permette, in fase di pianificazione architetturale, di poter valutare molte più opzioni e scegliere quella che si adatta meglio, sopratutto in contesti di datacenter privati protetti da NAT o proxies.

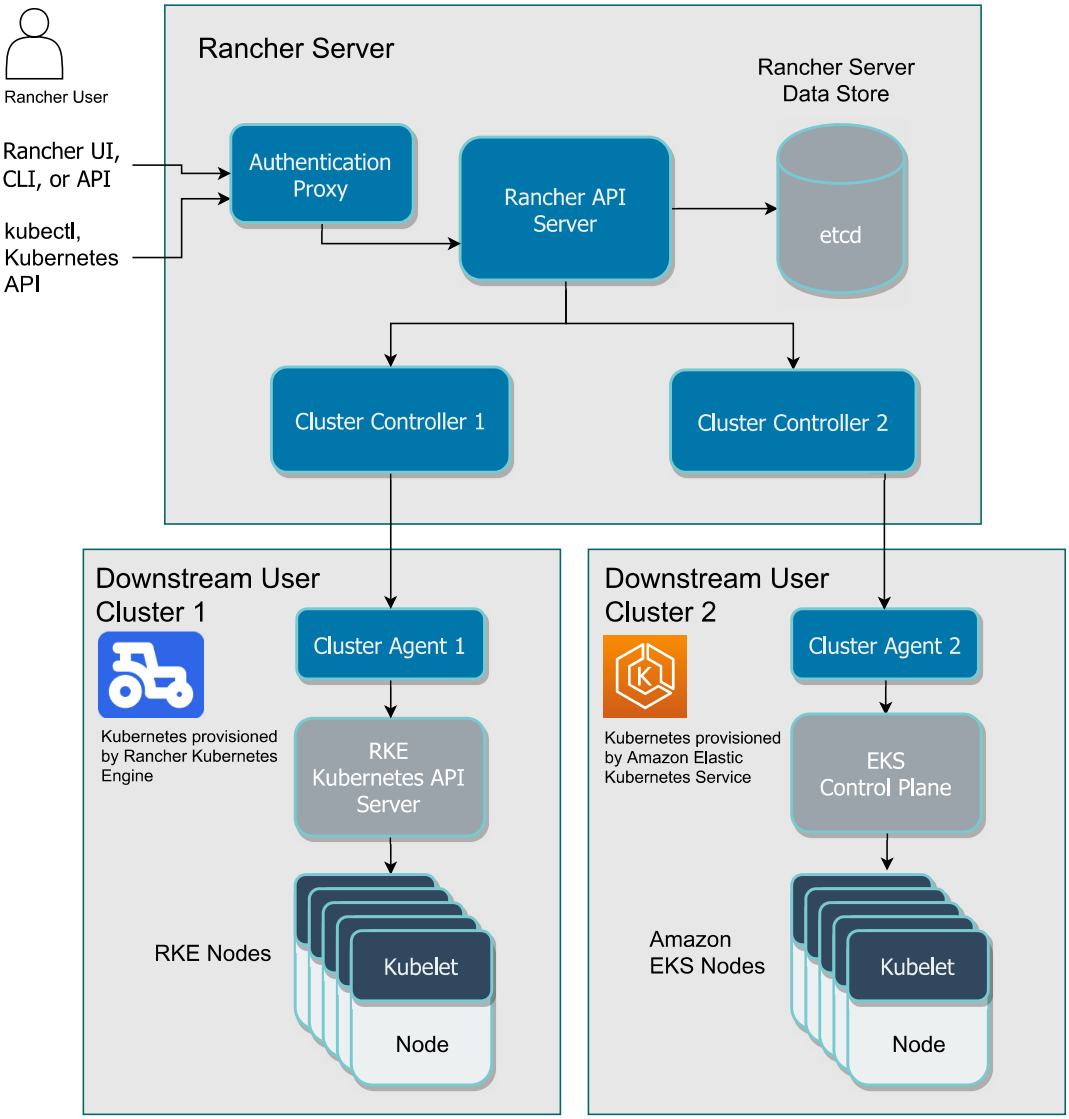

Figura 3.12: Architettura generale di Rancher - Fonte: Rancher

Come evidenziato dalla documentazione ufficiale, di cui si riporta un estratto in **figura 3.16**, la connettività tra cluster non è la stessa in tutti i casi. Infatti sono previsti requisiti diversi a seconda della natura del cluster in questione, si noti come per esempio un cluster RKE importato necessiti di una singola connettività verso la porta 443 del management plane. A differenza invece di un cluster AKS creato da Rancher, che richiede una connettività bidirezionale, coinvolgendo le porte 6443 e 443.

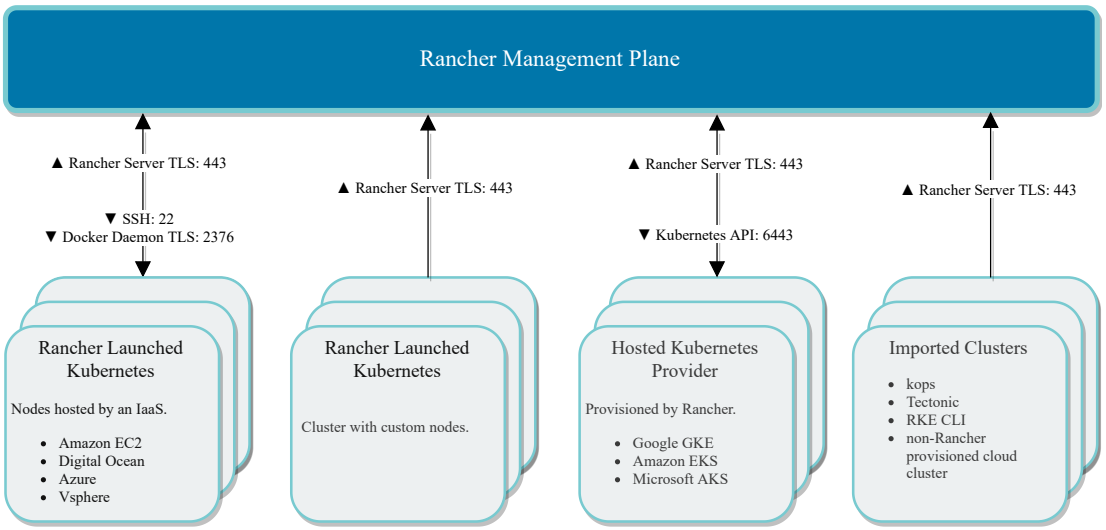

Figura 3.13: Panoramica della connettività tra cluster e Rancher server con dettagli sulle porte coinvolte - Fonte: Rancher

Essendo Rancher un prodotto con pieno supporto a varie piattaforme Kubernetes based, anche i suoi requisiti variano di conseguenza, mantenendo in ogni caso un regime molto basso.
Di seguito ne viene presentato un riepilogo[8]:

- Numero di nodi minimo: 1
- OS supportati: Ultime versioni delle principali release x86_64 Linux, come Ubuntu, CentOS, RHEL o SLES
- Versione Kubernetes: v1.19.14 o successive
- In caso si utilizzi K3S: v1.19.14+k3s1 o successive
- In caso si utilizzi RKE: Docker 19.03.x o successive

Mentre per ciò che concerne i requisiti hardware, questi devono essere modulati in base alla dimensione della propria rete multi-cluster. Segue una tabella riepilogativa che si applica al solo cluster centrale, in quanto le risorse richieste sono necessarie per poter gestire correttamente tutta la rete. Si noti come l'ordine di grandezza

[8]Si evidenzi che i requisiti e costi proposti fanno riferimento all'ultime versione di Rancher attuale, ovvero la 2.6.0

Dimensione	Clusters	Nodi	vCPUs	RAM
Piccolo	fino a 150	fino a 1500	2	8 GB
Medio	fino a 300	fino a 3000	4	16 GB
Grande	fino a 500	fino a 5000	8	32 GB
XL	fino a 1000	fino a 10000	16	64 GB
XXL	fino a 2000	fino a 20000	32	128 GB

Tabella 3.2: Panoramica dei requisiti hardware per il solo Rancher server modulati in base alla dimensione della propria rete

del numero di cluster e nodi, sia rispettivamente delle centinaia e migliaia già per le reti considerate "piccole". Da ciò si evince la forte componente di scalabilità del prodotto, che riesce a raggiungere queste caratteristiche con risorse estremamente basse (2 vCPUs e 8GB di Ram per nodo).
Infine, essendo un prodotto totalmente open-source non è previsto il pagamento di alcuna licenza, sia questa per l'utilizzo che per il supporto.

3.2.3 Ambiente e topologia in fase di analisi

La topologia scelta durante l'analisi è stata il frutto di scelte mirate al fine di ottenere una rete il più eterogenea possibile, nel dettaglio:

- **Infrastruttura privata basata su VMware vSphere**: Proprio come durante l'analisi di RHACM, anche qui per la componente privata è stato utilizzata un datacenter basato su vSphere. Tutta la componente di rete è rimasta invariata rispetto all'analisi precedente, ciò che è cambiato è il cluster on-premise. Infatti, a causa dell'esaurimento di risorse hardware sulla macchina host non è stato possibile predisporre un cluster multi-nodo. Questa limitazione, insieme alle problematiche derivate dalle varie componenti di NAT e proxy posti in egress, sono state superate predisponendo un cluster RKE single-nodo. Questo ha permesso:
  - Di ridurre il consumo di risorse hardware in quanto RKE prevede requisiti bassi grazie alla sua natura containerizzata.
  - Di evitare i problemi di connettività verso il Rancher server poiché, come da documentazione, un cluster RKE importato necessita un'unica connessione in outbound verso l'hub sulla porta 443.
- **Infrastruttura pubblica basata su Microsoft Azure**: Per la parte pubblica, anche qui è stato scelto Azure come provider. Nella parte pubblica della topologia sono stati predisposti il Rancher server e un cluster Kubernetes.
  - Rancher Server: L'hub centralizzato è stato predisposto su singola VM e installato via container come soluzione RKE. Questo ha permesso di ridurre al minimo i costi, dovendo gestire una rete multi-cluster molto piccola. La versione di Rancher utilizzata è stata la 2.5.7.
  - Cluster Azure: Un cluster single-node K8s esposto con IP pubblico.

Si noti come, a differenza dell'architettura prevista per l'analisi di RHACM, qui la struttura risulti molto più pulita, richiedendo nessuna risorsa extra. Ciò è stato possibile, come esposto prima, grazie all'utilizzo di RKE all'interno di vSphere e ai requisiti di connettività verso il Rancher server, che non richiedono una connessione bidirezionale. Nello specifico sono stati risparmiati: i costi di mantenimento dei tunnel Ngrok, il provisioning e la configurazione di un nodo helper con relativi servizi (HAProxy, DNS, etc.) e mantenimento e configurazione di un Nginx reverse proxy.
Come si evince facilmente, il risparmio in termini di tempo, costi e fatica non sono per niente banali, e questo è sicuramente un **punto importante dello strumento**.

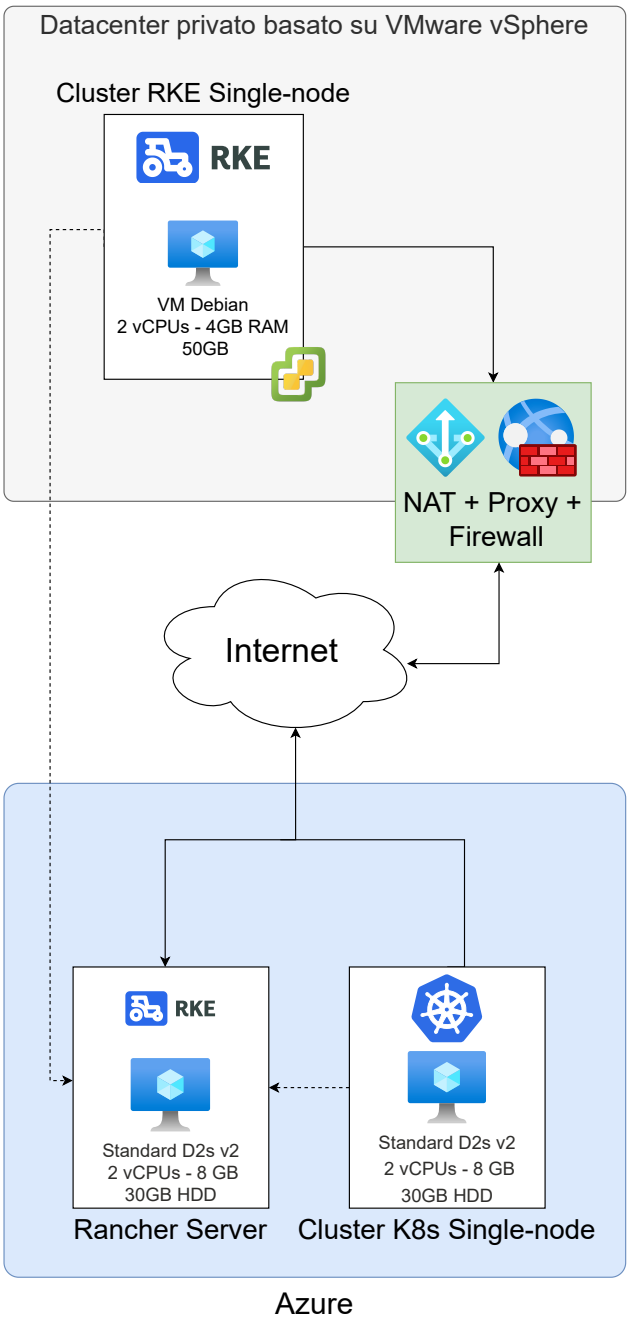

Figura 3.14: Architettura e topologia adottata durante la fase di analisi.

3.2.4 Installazione dello strumento

Rancher si presenta come un helm chart di facile installazione su un qualunque cluster Kubernetes o sotto forma di container Docker *ready-to-go*. Dopo averne analizzato i requisiti e predisposto un cluster Kubernetes o VM Linux con Docker con le risorse necessarie per la propria rete multi-cluster, è necessario o installare il chart o lanciare il container Docker affinchè lo strumento sia pronto all'utilizzo.
La scelta di aver predisposto il prodotto installabile tramite bundle come chart o immagini Docker, rende il processo d'installazione a minimo rischio di errori, sia umani sia dovuti a eventuali conflitti legati all'infrastruttura. Ed effettivamente si ha avuto riscontro di ciò in fase d'installazione, che non ha presentato problemi, grazie anche a una documentazione chiara e completa al riguardo.
Una volta installato, non è richiesta alcuna configurazione, infatti viene già predisposta una rotta verso la web UI di Rancher che, previa autenticazione, si dimostra già funzionante e con il cluster "local" già importato. In sintesi non sono state riscontrate limitazioni o problematiche durante la fase d'installazione.
Di seguto i **punti di maggior rilievo** riscontrati durante l'analisi:

- Installazione indipendente da ambiente di utilizzo che la rende difficilmente vittima di failure.
- L'installazione tramite immagine Docker, rende Rancher compatibile su qualsiasi sistema.
- Non è richiesta nessuna configurazione post installazione.
- Cluster locale già pre-importato nella rete.
- Documentazione ufficiale chiara e completa al riguardo.

3.2.5 Gestione della rete multi-cluster - Creazione, aggiornamento e distruzione

Creazione e registrazione di un cluster

La procedura di creazione di un nuovo cluster direttamente dal Rancher server si presenta **altamente** configurabile e offre un'altissima compatibilità in termini di vendor e piattaforme.
A seguito della configurazione delle cosiddette **Cloud Credentials**, ovvero le credenziali indispensabili per delegare a Rancher la creazione delle risorse sul cloud vendor, è possibile scegliere tra molte possibilità:

- **Cloud pubblico**
  - **Hosted Kubernetes**
    - AKS
    - GKE
    - EKS
  - **Self-managed Kubernetes**[9]
    - Amazon EC2
    - Azure
    - DigitalOcean
    - Linode
- **Cloud privato e bare-metal**: È possibile indicare i puntamenti alle macchine fisiche per l'installazione, se si dispone già di nodi pronti.

Il processo di configurazione di un nuovo cluster si presenta con una UI altamente pulita e un'altissima configurabilità di tutti gli aspetti del cluster. Che questo sia managed o self-managed, il wizard di Rancher lascia modificabili tutti i principali aspetti come:

- **Caratteristiche dei nodi**: È possibile configurare numero di nodi, formato delle macchine da utilizzare e, una volta creati uno o più pool di macchine, gestire quali di questi si dovranno occupare del ruolo di master, worker e chi dovrà mantenere un'istanza dell'etcd.
- **Risorse aggiuntive**: Se previsti, è possibile configurare gruppi di sicurezza di rete, regole sul traffico in ingresso/uscita, credenziali SSH e reti virtuali.
- **Versione di Kubernetes**

[9]Si noti come non sia disponibile il supporto a GCP per una soluzione IaaS

Una volta confermata la configurazione, Rancher in totale autonomia si occuperà di comunicare col provider scelto e di effettuare il provisioning di tutte le risorse necessarie. Saranno inoltre già predisposte tutte le risorse da installare sui cluster di destinazione al fine di permettere il tunneling con l'hub centrale. La procedure di import, o "**registrazione**" per come viene definita dalla documentazione, si presenta molto simile a quella già vista in RHACM.
A seguito della configurazione dei parametri necessari, Rancher fornirà un comando *kubectl* da eseguire sul cluster di destinazione che in maniera automatica creerà tutte le risorse necessarie alla comunicazione con l'hub centrale. A seguito di questo fase, il cluster risulterà correttamente registrato nella web UI del server. Si menzioni inoltre che, sia la procedura di creazione che di registrazione, prevedono la possibilità di associare dei label a ogni nuovo cluster, i quali saranno sfruttati per la cluster selection in fase di propagazione di workload distribuiti.

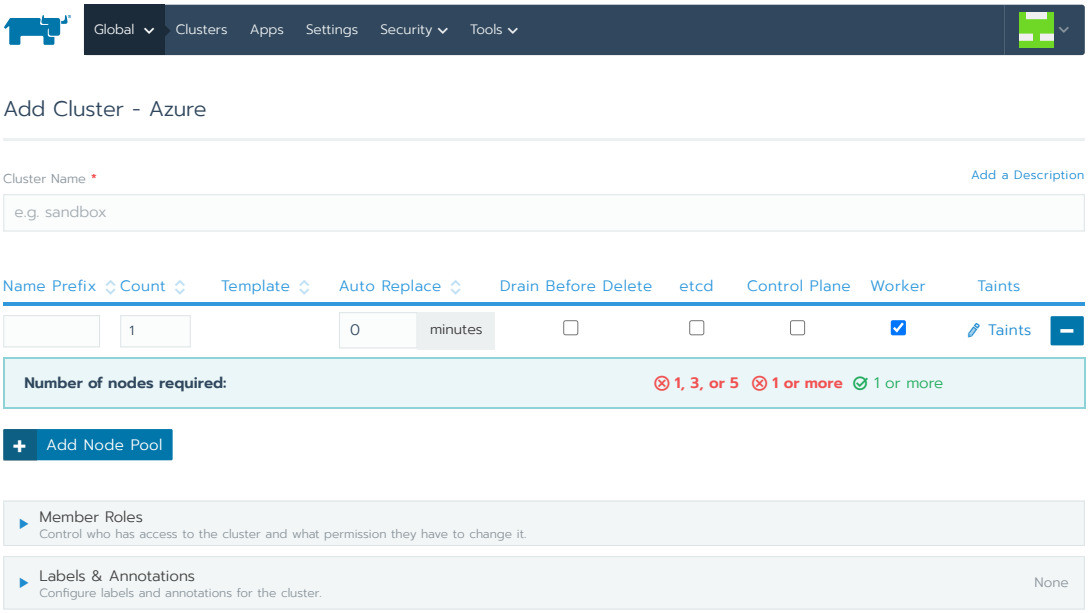

Figura 3.15: Web UI del Rancher Server relativa alla configurazione di un nuovo cluster su Azure tramite VM. Si noti come sia possibile associare a ogni pool quali il ruolo di control plane e/o data plane.

Durante il lavoro di analisi, la procedura di creazione della rete multi-cluster, a differenza di RHACM, non ha presentato alcun problema. Sopratutto per ciò che concerne il cluster single-node RKE mantenuto all'interno dell'infrastruttura privata. Come previsto dalla documentazione ufficiale, non ci sono stati problemi di connettività nonostante la rete privata.

Aggiornamento di un cluster

Il processo di aggiornamento dei cluster si presenta più limitato rispetto alla controparte Red Hat. Non avendo una piattaforma sottostante comune a tutti i cluster, se non l'orchestratore Kubernetes che ne fa da core, l'aggiornamento non risulta essere realizzabile.
L'unica eccezione la fanno i cluster RKE e K3s, che sotto questo punto di vista, presentano enormi funzionalità di aggiornamento, rollback, backup e recovery (*queste ultime analizzate in seguito*). La procedura viene offerta nativamente da RKE e K3s, che oltre a un sistema di aggiornamento solido, offrono anche l'eventuale possibilità di effettuare uno snapshot dell'etcd in caso di problematiche. Punto interessante è che, a differenza di RHACM, questa procedura **non provoca disservizio**.

Distruzione e rimozione di un cluster

Il processo di rimozione di un cluster dalla propria rete prevede che vengano eliminate tutte le risorse Kubernetes create sul cluster di destinazione affinché sia possibile effettuare il collegamento. Quindi non viene prevista in alcun modo la distruzione **totale** delle risorse computazionali (macchine virtuali, servizi Kubernetes hosted come AKS, EKS, GKE), lasciando all'utente la responsabilità di rimuovere manualmente tali risorse se non più necessarie.
Fanno eccezione invece i cluster esterni "registrati" dove il processo automatico di cleaning delle risorse non è completo e ne viene richiesta la rimozione in maniera totalmente manuale. In merito a quest'ultima funzionalità, si noti che è presente la stessa problematica in caso di "detachment" (e non rimozione) di un cluster in RHACM. Buona parte delle risorse nel cluster di destinazione rimangono ancora in vita, si evidenzi però che la documentazione a supporto di Rancher si presente altamente completa in merito, mostrando i dettagli relativi alla pulizia di risorse K8s, file, cartelle e interfacce di rete.

3.2.6 Gestione dei workload multi-cloud

Fleet - Logica e componenti

Per la propagazione dei workload su più cluster, Rancher sfrutta uno strumento "esterno", **Fleet**, che integra al suo interno già dalla fase d'installazione. Fleet, pienamente open-source e mantenuto da Rancher Labs, si presenta come **propagatore di workload distribuiti** che segue fortemente il paradigma **GitOps**[10].
Alla base del suo funzionamento, ci sono tre componenti principali:

- **GitRepo**: Oggetto che descrive su quale git repository effettuare il watching ed eventuali dettagli come: path della repository, credenziali e/o regole di cluster selection.
- **Bundle**: Unità interna usata per la gestione e la propagazione delle risorse individuate dalla scansione della git repository di riferimento. A seguito infatti del watching, viene generato un set di Bundles che rappresentano una o più risorse di diverso tipo (raw manifests, helm chart, kustomize template).
- **BundleDeployement**: Risorsa che viene creata ogni qual volta un Bundle viene propagato su un cluster remoto. Grazie a questo, Fleet è capace d'individuare eventuali discrepanze tra la versione in esecuzione sui downstream cluster e quella mantenuta nel repository.

Per effettuare l'impacchettamento delle risorse in Bundles e la loro gestione, Fleet converte le varie risorse da propagare in un Helm chart. Successivamente viene sfruttato Helm come engine sottostante, che si occuperà di tutti i processi d'installazione, aggiornamento e rimozione nei cluster di destinazione.

[10]Paradigma che fonde i principi del CI/CD all'utilizzo di repository git. Secondo il paradigma GitOps, l'immagine che viene mantenuta sul repository (generalmente nel **master** branch) deve essere la stessa in esecuzione all'interno degli host di destinazione.

Di seguito un dettaglio sull'architettura completa:

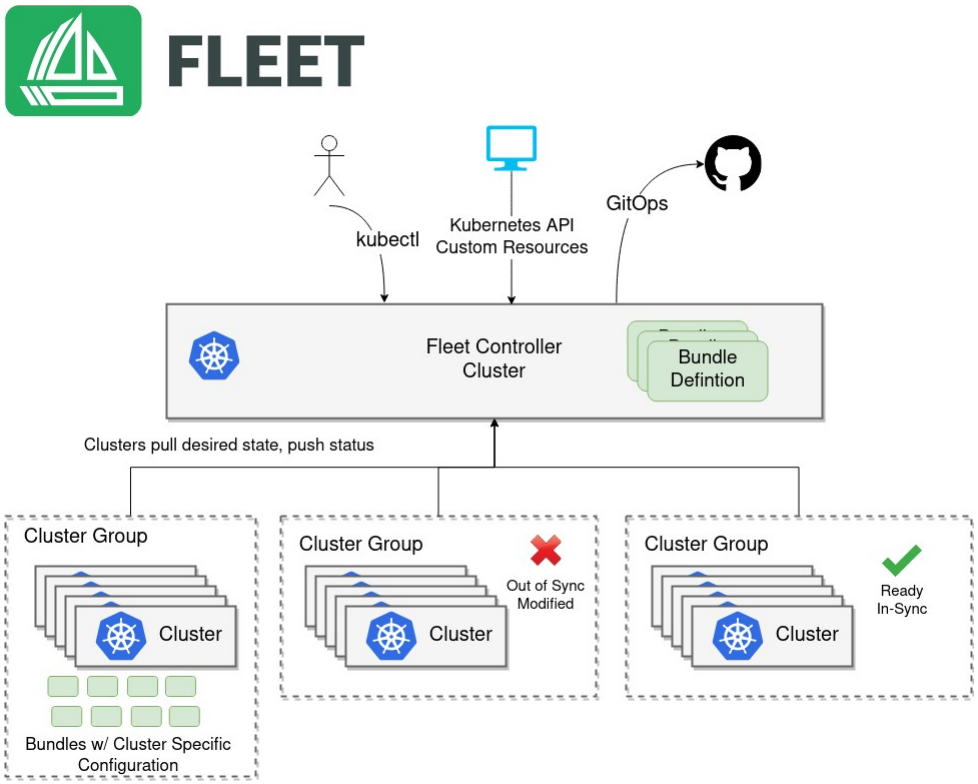

Figura 3.16: Architettura completa di Fleet - Fonte: Documentazione Fleet

Creazione di un workload distribuito

Il processo di creazione di un workload si presenta di facile esecuzione sia tramite web UI che tramite CLI, in quanto si tratta di applicare un singolo file YAML relativo all'oggetto **GitRepo**, automaticamente Fleet si preoccuperà della propagazione. Sia che si scelga l'approccio grafico che da terminale, i parametri principali da definire risultano essere:

- **Git repository**: Tutte le informazioni relative al repository di riferimento, tra cui URL, branch, path ed eventuali credenziali.
- **Cluster di destinazione**: Devono essere definite le regole per indicare quali cluster usare come destinazione del workload. L'espressioni prevedono matching basato sui label associati ai cluster in fase di creazione o registrazione.

Nonostante la struttura per la creazione del workload possa far trasparire un'eccessiva semplicità, la logica di watching e parsing delle risorse nella repository risulta molto più complessa e curata.
Si noti innanzitutto, che Fleet supporta tre diversi possibili tipi di risorse:

- **Raw manifests**
- **Template Kustomize**
- **Helm chart**: Sia questo locale nel repository che esterno, raggiungibile tramite riferimento nel file *fleet.yaml*.

La repository di riferimento **non ha vincoli in termini di struttura** e può definire una o più applicazioni tramite una di queste soluzioni o un ibrido tra le tre, favorendo ancora una volta la flessibilità. La logica di scanning di Fleet prevede di discendere su ogni path specificato in maniera indipendente, e una volta che tutti i file sono stati "consumati", Fleet si comporterà di conseguenza a seconda che questi siano *Chart.yaml*, *kustomization.yaml* o raw manifest.
Per rendere ancora più preciso e potente lo scanning del repository, è possibile inserire uno o più file *fleet.yaml*. Ognuno di questi corrisponderà a un nuovo oggetto Bundle e contiene dettagli relativi a:

- **Namespace e DefaultNamespace** - Namespace di destinazione e, per evitare problematiche relative a namespace già esistenti, indicazioni su una scelta di default da attuare in autonomia.
- **Dettagli a seconda del source utilizzato**:
  - **Kustomize** - Indicando il path contenente il file **kustomization.yaml**
  - **Helm chart** - In caso di Helm chart, possono essere definiti: url del chart, version e valori personalizzati.
- **RolloutStrategy** - Proprio come un classico deployment Kubernetes, sono definibili strategie di rollout.
- **Cluster selectors** - Sezione molto potente, che prevede d'indicare sia le espressioni per la cluster selection, che parametri per-cluster. Questi ultimi permettono quindi di personalizzare nativamente, senza l'uso di agenti esterni, l'applicazione da destinare su ogni cluster.

La fase di monitoraggio della propagazione si presenta molto più performante rispetto a RHACM, permettendo di seguire gli stati di ogni risorsa su ogni cluster in tempo reale. Inoltre, in caso di problemi durante il provisioning, eventuali errori e warning vengono reindirizzati verso il Rancher server così da evitare l'interazione coi cluster di destinazione.

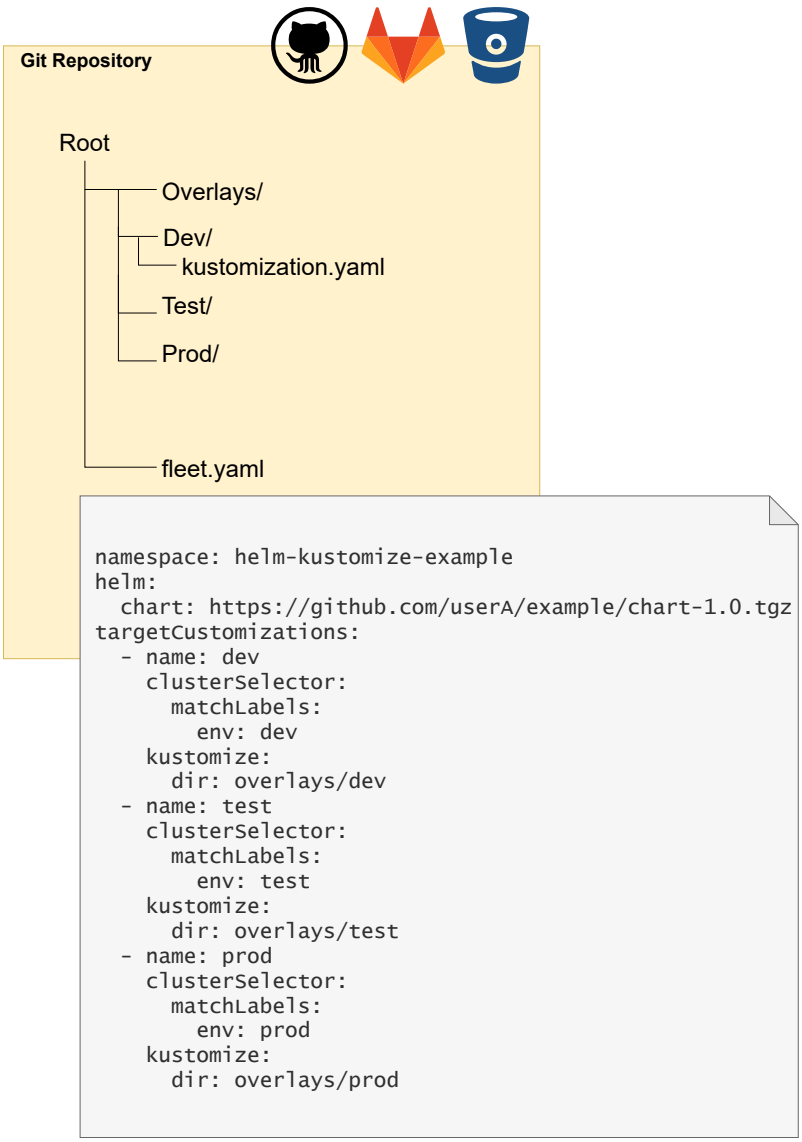

Figura 3.17: Esempio di un possibile repository che sfrutta un chart esterno da personalizzare tramite kustomize a seconda dell'ambiente del cluster di destinazione - Fonte: https://github.com/rancher/fleet-examples/

Considerazioni finali - Punti di forza e limiti riscontrati

L'analisi svolta ha portato alla luce molti punti di forza dello strumento che erano già previsti durante la fase di documentazione delle capabilities offerto. A differenza invece delle limitazioni che sono state poche e di minimo impatto per gli obbiettivi che Rancher si propone di raggiungere.
Durante l'analisi svolta sul prodotto, è stato preparato preventivamente un repository che prevedesse sia un approccio tramite helm locale, remoto che raw manifests su più cluster di destinazione, così da esplorare alcune delle possibilità offerte da Rancher.
La pianificazione della struttura del repository si è dimostrata di facile apprendimento, sopratutto grazie alla documentazione di supporto e agli esempi forniti da Rancher stesso. La possibilità di personalizzare tramite un singolo file (**fleet.yaml**) l'intera propagazione distribuita si rivela una soluzione altamente vincente, riducendo il quantitativo di file contenuti all'interno del repository e di conseguenza aumentandone la chiarezza.
Il processo di creazione e configurazione del workload è molto semplice ed essenziale, manca però di eventuali feature aggiuntive che invece sono state riscontrate nel prodotto Red Hat (*si consideri la possibilità di scheduling del workload*).
In merito al monitoraggio della propagazione, la soluzione di Rancher Labs presenta un feedback molto fedele e sincronizzato con lo stato dei cluster di downstream, mostrando anche errori di provisioning relativi alle risorse senza necessità alcuna d'interazione con il cluster remoto.
Anche con Rancher, come con ACM, è stato propagato come test un deployment **"Restic - Velero"** con performance ben superiori rispetto allo strumento Red Hat. Infatti in meno di 30 secondi tutte le risorse sono state propagate nel cluster di destinazione, in linea con quanto mostrato dal monitoring della propagazione, risultando quindi ben più efficace della controparte Red Hat.
Per concludere, di seguito i punti più importanti emersi a seguito dell'analisi:

- Propagazione del workload distribuito efficace, manca però di una più profonda configurabilità. Ad adesso (**Rancher 2.6**) sono presenti solo i parametri indispensabili.
- Altissima flessibilità di Fleet che permette di adattarsi a qualsisi tipologia di source. Le combinazioni delle soluzioni offerte permettono anche la parametrizzazione per cluster direttamente in un unico file.
- Monitoring della propagazione molto affidabile, fornisce anche gli errori per risorsa in caso di problematiche.
- Documentazione di supporto completa e dettagliata.

3.2.7 Backup, Recovery e Migration

La componente di disaster recovery offerta da Rancher si presenta ben più ampia e completa della controparte Red Hat, e le operazioni che permette sono eseguibili relativamene al solo Rancher Server, in quanto applicativo, che ai cluster basati su RKE.

- **Rancher Server** - Le operazioni di disaster recovery che operano sul Rancher server hanno il solo scopo di garantire continuità di servizio per il solo applicativo. Vengono omessi i dettagli relativi a Rancher in forma di container Docker, in quanto le procedure di disaster recovery sono relativo al Docker engine e non a Rancher stesso. **Si noti che non operano in termini dell'intero cluster "hub" in cui Rancher è in esecuzione.**
  - **Backup e Recovery** - Per abilitare il supporto ai backup è necessaria l'installazione di un operator offerto nel marketplace dello strumento. Previa installazione, direttamente dall'interfaccia grafica sarà possibile creare una risorsa **Backup** e memorizzarle in un volume persistente locale o, per maggior sicurezza, in un object storage remoto come S3 o Minio. Il processo di restore è esattamente analogo, attraverso un oggetto Recovery che, una volta indicati nome del backup e riferimenti della sua location, si preoccuperà di ripristinare lo stato del Rancher Server. **Si noti che il backup è ripristinabile solo sullo stesso cluster su cui è stato eseguito.**
  - **Migration** - Il processo di migrazione è una feature di rilievo, che permette di spostare il Rancher server da un cluster a un altro, a patto però che si mantenga l'FQDN. Il processo di migrazione prevede un backup già memorizzato su un object storage e l'installazione di un helm chart ad-hoc che permette di effettuare una "restore" sul nuovo cluster. A seguito di queste operazioni, è possibile procedere la canonica installazione.
- **Cluster basato su RKE** - Le operazioni di disaster recovery su un qualsiasi cluster di downstream possono essere effettuate solo se questo si basa su RKE, in quanto l'engine stesso prevede delle feature di backup e recovery. Parallelamente alle operazioni di upgrade del cluster, la natura container dell'engine, permette di effettuare facilmente snapshot del database etcd da memorizzare in locale o su storage remoti. Non sono previste per ovvie ragioni, operazioni di DR per qualsiasi piattaforma in quanto mancano by default oppure a mancare è un'integrazione forte con Rancher.

Per concludere, l'offerta di Rancher si presenta molto più completa e vasta sotto questo ambito, sia in merito al solo Rancher server che ai cluster RKE. È un limite previsto data l'assenza di una piattaforma sottostante universale (*a differenza di Openshift per ACM*) però la piena integrazione con RKE permette, laddove viene usato, di sfruttare appieno le capacità dello strumento.

3.2.8 Considerazioni finali sul prodotto

Rancher riesce, come orchestratore di cluster distribuiti, a soddisfare tutte le esigenze necessarie alla costruzione di una rete multicluster-multicloud, e alla propagazione di workload custom sui suddetti. È stato possibile, prima dell'analisi tecnica, valutare tutti i punti di forza, limiti e capabilities dichiarate da Rancher attraverso la documentazione ufficiale. Riguardo quest'ultima, si evidenzi che **non sono state riscontrate mancanze di dettagli, bensì si valuta la documentazione altamente completa e ricca di dettagli per ogni sezione**.
Principale punto di forza dello strumento risulta essere la flessiblità, capacità che trova modo di esprimersi sotto vari aspetti dello strumento. La gestione della rete multi-cluster presenta alta compatibilità con cloud vendor sia pubblici che privati, bare-metal e i requisiti si presentano già bassissimi. Una nota di merito è infatti diretta a **costi e requisiti**: Rancher permette di modulare sia l'hub che i cluster di downstream su diversi tagli di risorse hardware a seconda della dimensione della propria rete o dei workload da dover eseguire. In questo modo non esistono dei vincoli imposti, se non per i minimi requisiti necessari, e si ottiene un rapporto dimensione/costi davvero importante. **Si noti quindi che il prodotto in questione si presta bene tanto in contesti di sviluppo o ricerca che in contesti enterprise**, garantendo in entrambi i casi performance e stabilità. Grazie anche alla sua natura open-source e license-free, non sono richiesti costi di licenze e sono presenti più canali di supporto atti a garantire continua crescita del prodotto.
La sua posizione come leader del settore, insieme a Red Hat e Google, è dovuta anche all'intero set di prodotti che circondano e/o integrano Rancher, tra cui **RKE, K3s e Fleet**, ampliando quindi la potenza del prodotto con strumenti paralleli.

Si evidenziano dunque i punti principali del prodotto, a seguito dell'analisi svolta:

- Insufficiente: La funzionalità è incompleta e inefficace, ciò la rende inutilizzabile per l'uso dichiarato dalla documentazione del prodotto.
- Sufficiente: La funzionalità non è completa ma presenta tutti i punti fondamentali e indispensabili per l'utilizzo base. Potrebbe presentare problematiche di criticità più o meno elevata, ma che comunque non ne compromettono il funzionamento di base.
- Buono: La funzionalità presenta tutti i punti fondamentali per il corretto funzionamento, è efficace e stabile. Si presenta però ancora migliorabile sotto alcuni punti.
- Ottimo: La funzionalità è completa sia di tutti gli aspetti fondamentali che di eventuali feature extra, il tutto in linea con quanto dichiarato, stabile ed efficiente.

Tabella 3.3: Riepilogo dei principali punti dello strumento Rancher a seguito dell'analisi

Funzionalità	Note	Valutazione
Installazione, compatibilità e costi		
Requisiti d'installazione	Modulabili a seconda della dimensione della rete. Possibilità d'installazione si su Docker che Kubernetes	Ottimo
Compatibilità con cloud vendor e piattaforme on-premise	Presenta anche ottima integrazione con RKE e K3s	Ottimo
Facilità d'installazione		Ottimo
Costi (licenza, risorse hardware, etc.)	Hardware da valutre a secondi degli use-case. License-free	Ottimo
Possibilità di spegnere e accendere i nodi	Basandosi su K8s nativo, non presenta problemi a riguardo.	Buono
Gestione della rete multicluster		
Creazione, aggiornamento e distruzione di cluster remoti	Non è possibile distruggere le risorse su un cloud provider alla distruzione del cluster.	Buono
Import e detachment di cluster remoti		Ottimo

Connettività tra cluster	Altissima flessibilità per adattarsi in tutti i contesti	Ottimo
Gestione dei workload distribuiti		
Compatibilità in termini di source		Buono
Efficacia della procedura guidata	Semplice e chiara. Nessuna necessità di configurare manualmente file YAML	Buono
Efficacia della Git Repository	Molto potente. Fleet permette di combinare più approcci ottenendo risultati di rilievo.	Ottimo
Efficacia della CI/CD		Ottimo
Efficacia della propagazione		Buono
Monitoraggio dei progressi	SSempre in sync e vengono riportati eventuali errori direttamente nell'hub.	Ottimo
Cluster selectors	Semplice, basato su labels.	Buono
Observability		
Efficacia dell'interfaccia	Chiara e minimale.	Sufficiente
Disaster Recovery		
Disaster recovery per il Rancher Server	Molto utili e ben documentati	Ottimo
Disaster recovery per i cluster di downstream	Disponibile solo per cluster RKE	Sufficiente
Feature aggiuntive		
Shell interattiva via UI	Permette d'interagire con i cluster	Buono
Documentazione		
Completezza e precisione	Molto completa e precisa. Non è stata richiesta interazione con il supporto.	Ottimo

3.3 Platform9 Managed Kubernetes - Un esempio di alternative non-leader

Dopo aver presentato e analizzato due strumenti leader nel settore, rappresentanti rispettivamente delle soluzioni enterprise e open-source, si è ritenuto necessario documentare anche l'esperienza maturata con un terzo prodotto non-leader, al fine di dare una più completa visione del mercato nel 2021.
Lo strumento in questione è "**Platform9 Managed Kubernetes**", offerto da Platform9, il quale si presenta come un servizio di orchestrazione di cluster distribuiti ma seguendo un modello SaaS. Ciò che viene offerto è di fatto un hub centrale completamente gestito e mantenuto da Platform9 nella loro infrastruttura che espone verso l'utente un'interfaccia grafica da cui è possibile effettuare tutte le operazioni relative a gestione dei cluster e dei workload.

3.3.1 Capabilities

P9MK si presenta come un servizio **Multi-cloud Kubernetes** che punta a ridurre i costi e i tempi di creazione dei cluster Kubernetes e il loro import all'interno della rete multi-cluster. È proprio la gestione e il monitoring dei cluster distribuiti il punto maggiormente di rilievo dello strumento, che offre una vasta compatibilità con cloud provider, bare-metal server e piattaforme private. La web UI è l'unico punto d'interazione con lo strumento, e da essa è possibile anche monitorare la salute e le risorse dei propri cluster.

3.3.2 Limiti

Le limitazioni dello strumento invece sono tanto e importanti, a differenza di quelli analizzati nelle sezioni precedenti. Le limitazioni più importanti emerse durante l'analisi sono principalmente due:

- **Il modello SaaS**: L'adozione di un modello SaaS per l'applicativo che si occupa della gestione degli orchestratori, presenta più difetti che vantaggi. In quanto soluzione completamente "hosted" su un'infrastruttura di terzi, si viene sollevati dagli impegni di manutenzione e gestione del servizio, che però come riscontrato in ACM e Rancher, non si tratta di un overhead eccessivo in quanto si tratta di applicativo *ready-to-go* e se ne delega l'esecuzione e disponibilità alla piattaforma Kubernetes. Infine, in quanto servizio, presenta un costo mensile per gli utilizzi più comuni, che scala in base alla dimensione della propria rete di cluster.

- **Assenza di supporto per workload distribuiti**: Come evidenziato nelle sezioni precedenti all'analisi, la capacità di propagare workload in maniera distribuita su più cluster è una capability fondamentale per uno strumento multi-cloud. Purtroppo qui è totalmente assente, permettendo semplicemente di gestire Helm chart su singolo cluster.

A seguito dei grossi limiti riscontrati dallo strumento, questo non si è rilevato essere un **orchestratore di cluster distribuiti intesi come una singola entità o rete multi-cloud**, bensì questi cluster rimangono totalmente indipendenti l'uno dall'altro. Infine, il modello SaaS riscontrato, non sembra essere una soluzione efficace se comparata ai costi che sono richiesti da soluzioni self-hosted come Rancher e RHACM.

3.3.3 Use-cases

Se come soluzione multi-cloud si rivela altamente incompleta, P9MK potrebbe trovare utilizzi in contesti enterprise con altissime richieste di disponibilità di servizio, grazie alla forte SLA offerta. Si noti però che, se adottata come soluzione, questa deve essere valutata come interfaccia di gestione di cluster Kubernetes remoti, e quindi si limita al solo: provisioning rapido di cluster su public cloud, import di cluster privati e monitoring del loro stato. Grazie al monitoring-alerting integrato, non è necessaria un'installazione e configurazione a posteriori.
Per concludere quindi, **Platform9 Managed Kubernetes** è valutabile come soluzione solo in contesti dove:

- È richiesta una SLA ferrea.
- È necessario sollevare gli impegni di gestione e provisioning dello strumento a terzi.
- Si possono coprire i costi richiesti per il servizio.
- È necessario creare una moltitudine di cluster in poco tempo, con monitoring integrato.
- È necessario monitorare una moltitudine di cluster da una singola interfaccia centrale.

Capitolo 4

Use-case dimostrativo - Propagazione di uno stack EFK in contesto multi-cloud

A fini di confronto e analisi, in questo capitolo verranno valutati due approcci di risoluzione, multi-cloud e "canonico", di un problema pratico, riscontrato durante l'esperienza maturata all'interno di **Blue Reply**. Ne saranno evidenziati costi, requisiti e comparati i rispettivi pro e contro, al fine di valutare quale delle due soluzioni risulta essere vantaggiosa e perché. **Si noti che il problema reale è stato "esteso" a fini di multi-cloud, in quanto tutti i cluster coinvolti erano gestiti dallo stesso provider.**

4.1 Introduzione al problema

È in gestione al team DevOps un grosso numero di cluster AKS e AWS di un big cliente, il quale presenta la necessità di rendere attivo uno strumento di logging all'interno di ognuno di questi. Lo stack di logging richiesto è EFK, che prevede un'installazione tramite Helm charts da personalizzare tramite file values e ognuna di queste configurazioni però necessita parametri legati al singolo cluster, come per esempio l'hostname da esporre tramite risorsa Ingress.
I cluster a cui destinare questo stack sono nove, divisi in gruppi di tre per applicativo e ogni gruppo contiene un cluster di sviluppo (**SVIL**), User Acceptance Testing (**UAT**) e produzione (**PROD**). La suddivisione tra i vendor prevede che sei di questi cluster si trovino su Azure, mentre i restanti tre su AWS.

Per ciò che concerne gli strumenti e l'infrastruttura privata in possesso al team, questo ha accesso ha delle VM in esecuzione su un host vSphere, tra cui un server Jenkins. Di seguito una panoramica generale:

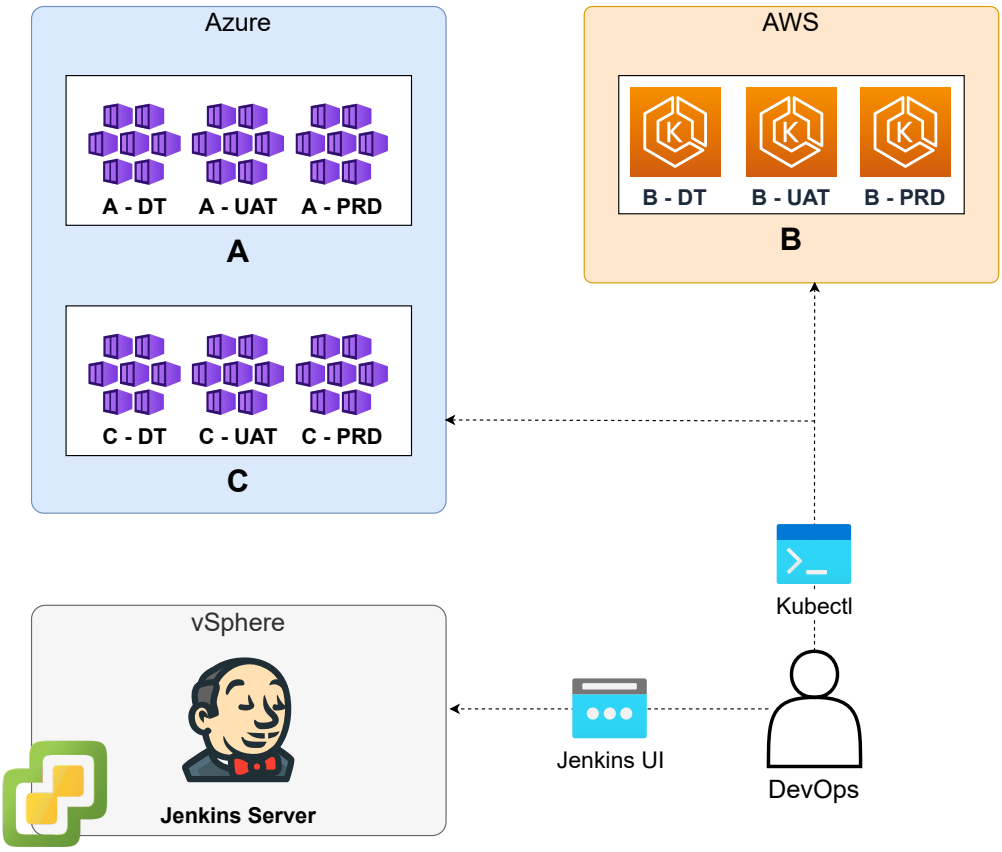

Figura 4.1: Panoramica generale dell'infrastruttura e topologia

La soluzione deve permettere di distribuire i vari helm d'installazione (*elasticsearch, fluentd, kibana, elasticsearch curator*) con possibilità di configurazione per-cluster e sopratutto, avendo predisposto un piano di azione con il cliente, la propagazione deve avvenire prima in DT e solo dopo un periodo di monitoring, propagata anche agli ambienti successivi.

4.2 Approccio canonico

L'approccio canonico condotto da **Blue Reply** è basato sull'utilizzo di pipeline Jenkins e di repository mantenute su BitBucket. Quest'ultime contengono tutti i principali file utilizzati dal server Jenkins per l'installazione dello stack, di seguito i più rilevanti[1]:

- **MakeFile**: MakeFile contenente le istruzioni helm per l'installazione dei rispettivi chart con i puntamenti ai rispettivi file values.yaml
- **JenkinsFile**: JenkinsFile per la definizione della relativa pipeline.
- **Values.yaml**: Rispettivi file values per i corrispettivi ambienti di sviluppo, UAT e produzione.

Una volta predisposta la repository, è necessario configurare il server Jenkins in quanto richiede:

- Installazione del plugin per la compatibilità con Kubernetes
- Inserimento e configurazione dei Kubeconfig di ogni cluster

Preparati repository e Jenkins server, è necessario lanciare una pipeline per i deployment su ogni cluster, in questo caso i tre di sviluppo, e monitorare lo stato dell'helm manualmente interfacciandosi via command-line sui tre cluster.
Eventuali modifiche o correzioni, richiedono la modifica del repository e la ri-esecuzione della pipeline che, non supportando l'approccio GitOps by-default, richiede che questa operazione sia manuale.
I punti principali che emergono da questa soluzione sono:

- Necessità di provisioning e configurazione di un server Jenkins.
- Necessità di preparazione di una repository di complessità media (può contenere diversi file tra cui JenkinsFile, MakeFile, values)
- Il monitoring dell'operazione è completamente manuale, e richiede l'intervente via command-line.
- Mantenimento costoso, qualsiasi modifica da replicare sui cluster richiede che la pipeline venga rilanciata. Incompatibile con approccio GitOps.
- Jenkins è ignaro che gli host di destinazione siano dei cluster K8s.
- Approccio GitOps non disponibile by-default. Integrabile previa configurazione workflow git repository + Jenkins server.

[1]Si noti che quella descritta non è l'unica soluzione applicabile tramite Jenkins

4.3 Approccio multi-cloud

L'approccio multi-cloud ideato per la propagazione di questo stack EFK, basa il suo funzionamento sulla creazione di una rete multi-cluster da gestire tramite Rancher. Ciò grazie al fatto che non si sta facendo uso di cluster OpenShift, permettendo quindi di risparmiare eventuali costi su licenze e risorse.
Per la gestione di una rete così piccola, non è necessario dedicare al Rancher server un cluster dedicato, bensì è possibile eseguirlo direttamente in un container Docker. La sua esecuzione la si può delegare o a una VM privata o, se fossero presenti problemi di connettività come evidenziati nei capitoli precedenti, a un container service su cloud pubblico.
Avendo predisposto il Rancher server tramite immagine Docker, è necessario effettuare l'import dei vari cluster e associare delle label che esprimano l'applicativo e l'ambiente, così da poterle sfruttare in fase di cluster selection.
Una volta creata la rete multi-cluster, è necessario il provisioning di una singola repository in accordo con le indicazione di Fleet. Una possibile soluzione potrebbe essere quella già esposta in **fig. 3.20**, ovvero sfruttare l'utilizzo di Helm charts esterni e Kustomize per la loro configurazione a seconda dell'ambiente di destinazione. In questo modo si ottiene una singola repository, che contiene solo i file essenziali quali: *fleet.yaml* e i vari *kustomization.yaml*.
Predisposta anche la repository, sarà necessario configurare il workload distribuito avendo cura d'indicare eventuale path, branch (utili per il watching GitOps) e le label relative all'ambiente di destinazione. Autonomamente Rancher si occuperà di propagare i chart e di fornire un feedback dalla sua interfaccia, evitando qualsiasi contatto diretto coi cluster di downstream.
Eventuali modifiche o correzioni, verranno immediatamente propagati sui cluster grazie all'approccio GitOps supportato da Rancher.
Di seguito, i punti principali della soluzione:

- Necessità di provisioning di un Rancher server, eventualmente su cloud pubblico.
- Creazione della rete multi-cluster, effettuando l'import dei cluster remoti.
- Necessità di preparazione di una singola repository di bassa complessità, ma richiede una pianificazione della sua struttura interna così da interagire correttamente con Fleet.
- Monitoring della propagazione direttamente da Rancher.
- L'approccio GitOps garantisce continua sincronizzazione dello stato dei cluster con quello della repository di riferimento.

4.4 Considerazione finali sulle soluzioni valutate

Entrambe le soluzioni proposte, permettono di raggiungere uno stack funzionante tramite Helm, ma si evince subito che:

- La prima soluzione canonica prevede un workflow adottabile per un contesto generico e non ad-hoc per Kubernetes. Questa sua adattabilità prevede l'assenza di alcuni comportamenti che invece si rivelano molto utili quando si lavora con ambienti Kubernetes.
- La seconda naturalmente è una soluzione che funziona solo per ambienti Kubernetes, superando gli ostacoli dovuti alla natura multi-cloud e questo si traduce anche in bassi costi d'implementazione e configurazione.

In termini di costi la prima soluzione presenta un overhead in termini di configurazione e provisioning di strumenti infrastrutturali maggiori rispetto alla seconda. Si evidenzi:

- La necessità di configurare e rendere operabile un Jenkins server.
- La necessità di scrivere e configurare Jenkinsfile in modo che la pipeline venga eseguita correttamente.
- Il monitoring dell'installazione prevede un intervento manuale sui cluster, aumentando i tempi e riducendo la QoL dell'operazione.

A differenza invece dell'approccio tramite Rancher, che nonostante condivida un costo in termini di provisioning dello strumento, riduce i costi e la complessità della procedura nella sua totalità, grazie al monitoring integrato e la necessità di organizzare la repository secondo una struttura ben precisa.
Nonostante l'utilizzo di Jenkins permetta di risparmiare in termini di provisioning dello strumento, in quanto già predisposto e funzionante, questo introduce poi degli overhead in varie fasi del processo di propagazione del workload. Infatti si consideri che Rancher lascio all'utente solo la configurazione del workload e preparazione del sorgente, mentre si occuperà automaticamente di selezionare i cluster di destinazione, applicare le configurazioni per-cluster all'applicativo da propagare (come nel caso di Helm chart, i corrispettivi values vengono applicati in completa autonomia) e, naturalmente, applicare le risorse per ogni cluster di destinazione; Jenkins invece prevede che tutti questi passaggi siano **descritti** dall'utente in uno o più file.
In conclusione, ciò che emerge dall'analisi di questo use-case pratico è il vantaggio principale di questi strumenti multi-cloud, ovvero la semplicità e la rapidità con quei questi prodotti possono essere portati a piena operabilità e la bontà della loro gestione di propagazione dei workload. Questo incentiva tanto l'adozione di

strumenti simili a favore di soluzioni pre-esistenti come quella presentata, poiché i costi presentati sono irrisori. Inoltre, data l'enorme diffusione di contesti Kubernetes, è necessaria l'adozione di strumenti che siano strutturati ad-hoc per interfacciarsi con ambienti di questo tipo.

Tabella 4.1: Riepilogo dei principali aspetti in comune delle soluzioni, con dettagli su pro e contro di ognuna

Aspetto	Rancher	Jenkins
Requisiti	Provisioning di un Rancher server. Data la dimensione della rete, possibile anche tramite Docker image.	Provisioning di un Jenkins server, anche se comunemente già presente.
Costi infrastrutturali	Provisioning dell'hub indifferente, data la possibilità di usare Docker. È richiesto invece un passaggio indispensabile di import dei cluster e labelling adeguato.	Provisioning e configurazione del Jenkins Server ed eventuali plugin.
Costi e complessità repository	Tramite Fleet, si riduce al minimo il numero dei file ed è possibile definire una logica di templating a seconda dell'ambiente di destinazione.	Necessario predisporre un JenkinsFile, eventuali file ausiliari e configurare la pipeline sul server.
Facilità di manutenzione	Complessità del repository minima e grazie all'approccio GitOps nativo, viene mantenuta sui cluster in maniera automatica.	Complessità del repository non indifferente, e l'approccio GitOps, se voluto, prevede una configurazione addizionale.
Monitoraggio dell'operazione	Ogni risorsa propagata viene mostrata, insieme al suo stato, direttamente nella web UI.	Jenkins non mostra altro che l'output dei task eseguiti. E' richiesto comunque un passaggio manuale tramite linea di comando su ogni cluster per verificarne l'effettivo stato post-installazione.

Capitolo 5

Conclusioni

Nei capitoli precedenti sono stati definiti i cosiddetti "*strumenti di gestione di orchestratori distribuiti*", strumenti atti a migliorare la QoL di quelle figure professionali che si devono interfacciare con più cluster distribuiti su diverse piattaforme, sia queste pubbliche che private. Dopo averne definite le funzionalità indispensabili, sono stati analizzati i principali prodotti leader nel settore, esplorandone le funzionalità principali, i punti di forza e criticità.
Sia Rancher Labs che Red Hat riescono a offrire una corposa offerta in termini di capabilities grazie al già forte background tecnologico di cui dispongono, ed è proprio l'interazione tra quest'ultimo e il prodotto che permette di offrire feature esclusive o performance stabili. Basti pensare alla piattaforma OpenShift per Red Hat o i già citati RKE e K3s che permettono a Rancher una disaster recovery efficiente. Si è riscontrata una particolare attenzione per aspetti come:

- Rapiditià e semplicità di installazione e configurazione del prodotto
- Configurabilità e stabilità della propagazione di workload distribuiti
- Alta compatibilità con piattaforme di cloud pubbliche, private e server bare-metal
- Observability centralizzata dell'intera rete multi-cloud

L'obiettivo di questi strumenti è divenire estremamente vantaggiosi rispetto alle soluzioni già adottate in contesti multi-cloud, riducendo al minimo i tempi di apprendimento riutilizzando tecnologie e framework già adottati, come le pipeline o gli Helm chart. Inoltre diventa sempre più importante evitare un'interazione point-to-point con ogni cluster della rete, garantendo invece gli stessi risultati ottenuti tramite command-line o interfacce per-cluster tramite interfacce centralizzate, che divengono veri e proprio "centri di comando" della propria rete multi-cloud.

5.1 Le differenze principali riscontrate tra Rancher e ACM

Si evidenzi in primis che gli strumenti posseggono una natura differente, e ciò ne definisce anche gli aspetti principali:

- **Red Hat Advanced Cluster Management for Kubernetes** è di fatto un Operator per la piattaforma **OpenShift**, la quale ne diviene requisito fondamentale e ne eredita tutti i principali, costi e requisiti.
- **Rancher** invece nasce come prodotto stand alone, completamente open-source, license-free e CNCF compliant, che tenta di porsi come alternativa "lightweight" nel mercato, divenendo adottabile anche in contesti enterprise.

5.1.1 Le piattaforme - OpenShift, RKE, K3s

E' necessario, per comprendere al meglio la differenza tra i due prodotti, anche le differenze tra le piattaforme che ne fanno da fondamenta, nonostante queste non siano il target principale dell'analisi.
OpenShift si presenta come una piattaforma Kubernetes altamente completa, solida e performante, che ha come target l'enterprise ed è proprio questo che ne definisce punti di forza e limitazione. Infatti la piattaforma prevede alte richieste hardware, licenze da acquistare per il supporto e una bassissima flessibilità che le consente di fornire performance e affidabilità altissime, sacrificando quindi adattabilità. La si può definire una soluzione "blackbox" al cui interno propone un set di feature (tra cui sicurezza, monitoring, autenticazione e autorizzazione) che risultano essere aspetti di grande interesse nell'ambiente enterprise.
RKE e K3s invece sono due distribuzioni open-source di Kubernetes, CNCF compliant il cui target principale è la riduzione di peso e consumi, per contesti più delicati come l'IoT. La prima ha lo scopo di distribuire Kubernetes in formato container, senza sacrificare performance, in questo modo se ne favorisce la distribuzione e provisioning, riducendo tempo e costi. La seconda invece ha lo scopo di distribuire Kubernetes anche in contesti di bassa disponibilità di risorse computazionali e fornendo supporto ARM, in questo modo se ne può effettuare il provisioning anche su IoT ed eventuali schede Arduino.

Tabella 5.1: Confronto tra piattaforme - OpenShift, RKE, K3s

Piattaforma	Pro	Contro
OpenShift	Alte performance a causa dello stack tecnologico (RHCOS) e alle alte richieste hardware. Presenta molti strumenti integrati di sicurezza, autenticazione e monitoring che espandono l'offerta di base.	Le alte performance si traducono in alti requisiti hardware, poca flessibilità e necessità di acquistare una licenza per ottenere supporto ufficiale.
RKE	Kubernetes completo e funzionante ma completamente containerizzato, ciò permette di facilitarne il provisioning, così come anche l'aggiornamento e la "disinstallazione". Se ne riducono i tempi e i costi di gestione, e risulta anche più semplice attuare politiche di backup, recovery e migration in quanto la piattaforma è interamente containerizzata. Inoltre la sua natura container lo rende adatto a processi di automation.	Difficile da gestire in caso di cluster con un enorme carico in termini di pod e traffico in/out.
K3s	Kubernetes ridotto al minimo, adatto all'edge e ottimizzato per ambienti ARM.	Difficile gestione in caso di carichi molto elevati, consigliata in quel caso la scalabilità orrizontale.

5.1.2 Costi e requisiti

La soluzione di Red Hat si presente altamente costosa, in quanto fa uso di OpenShift come requisito principale, e da questo eredita gli enormi requisiti hardware dei nodi. In aggiunta a ciò, prevede anche l'acquisto di licenze, rendendo il prodotto non adatto a tutti i contesti, ma solo a quelli fortemente enterprise.
Antitesi invece della natura free di Rancher che, in quanto prodotto open-source, non prevede alcun costo relativo a licenze e utilizzo. La totale assenza di vincoli relativi a piattaforme sottostanti, gli permette di essere eseguibile sia su Kubernetes che come immagine Docker su VM. In questo modo il prodotto offre flessibilità e possibilità di adozione in contesti a carenza di risorse come: sperimentazione, ricerca o IoT.

5.1.3 Installazione, configurazione e compatibilità

Entrambi gli strumenti puntano sulla facilità di installazione e dalla quasi assenza di parametri di configurazione, garantendo un comportamento di tipo "*plug'n play*". Non sono stati riscontrate problematiche relative al provisioning dello strumento, che si è sempre dimostrato operativo secondo le indicazioni delle documentazioni ufficiali. Si noti anche l'alta compatibilità con tutti i principali public cloud provider, siano queste piattaforme Kubernetes self-managed (modello IaaS tramite VMs) che managed, ma anche pieno supporto con **vSphere** per il cloud pubblico.

5.1.4 Gestione della rete multi-cluster

La possibilità di creare e manipolare reti multi-cluster e multi-cloud è un punto fondamentale per ogni buon strumento, che ne deve offrire le operazioni principali direttamente da singolo punto di contatto (hub centrale) senza interazione diretta con le piattaforme. Sotto questo aspetto, entrambi gli strumenti si rivelano essere completi e stabili, permettendo che il cluster venga creato in autonomia dal prodotto previa configurazioni delle credenziali cloud. Ciò permette anche di non doversi preoccupare di risorse aggiuntive (Load-balancer, proxy, firewall, etc.) da dover predisporre sul cloud pubblico affinchè il cluster sia operativo. Ciò è importante sopratutto per RHACM, che è vincolato a una piattaforma proprietaria che ha le proprie necessità in termini di rete, connettività e storage.
Purtroppo è proprio questa natura "vincolata" e **black-box** di ACM che lo rende poco flessibile in termini di connettività tra cluster. Non presenta problematiche se questa è composta da cloud pubblico ma, richiedendo molte risorse computazionali, spesso è conveniente usare cluster on-premise. Questi però, spesso protetti e isolati da NAT e proxy, potrebbero incontrare problemi nel raggiungere l'hub centrale e quindi deve essere necessaria un'attenta definizione della propria infrastruttura privata per i cluster OpenShift. Rancher anche qui si presenta come antitesi, offrendo altissima flessibilità grazie alle tante piattaforme Kubernetes che supporta, e per ognuna di queste esige diversi vincoli in termini di connettività. Grazie anche all'integrazione con RKE, permette di creare cluster facilmente in tantissimi ambienti, garantendo tramite Docker aggiornabilità e disaster recovery.

5.1.5 Gestione dei workload distribuiti

Secondo pilastro fondamentale in termini di capabilities è sicuramente la capacità di propagare su più cluster uno o più applicativi con efficacia. Su questo entrambi gli strumenti si sono rivelati efficaci, seppur ACM ha presentato criticità, sopratutto se si considera ciò che viene asserito nella documentazione di supporto. Se da un lato Rancher offre uno strumento molto potente per la progettazione di repository, secondo gerarchie di file e cartelle ben precisi, offrendo compatibilità anche per

Helm charts e Kustomize, ACM invece si presenta molto più acerbo, offrendo un supporto ad Helm chart senza presentare possibilità di configurazione rendendo di fatto inutilizzabile tale opzione. Si noti però che anche la sua logica di scanning di una repository presenta potenzialità non documentate che permette di ovviare l'assenza di un supporto diretto ad Helm e inoltre, a differenza della controparte open-source, presenta un paio di feature extra che possono rivelarsi utili in alcuni contesti.
Infine è stato valutato il monitoring della propagazione del workload, che è presente in entrambi gli strumenti ed evita interazione diretta, da parte dell'utente, con i cluster di downstream. Si evidenzi solamente che qui ACM ha presentato performance minori rispetto a Rancher.

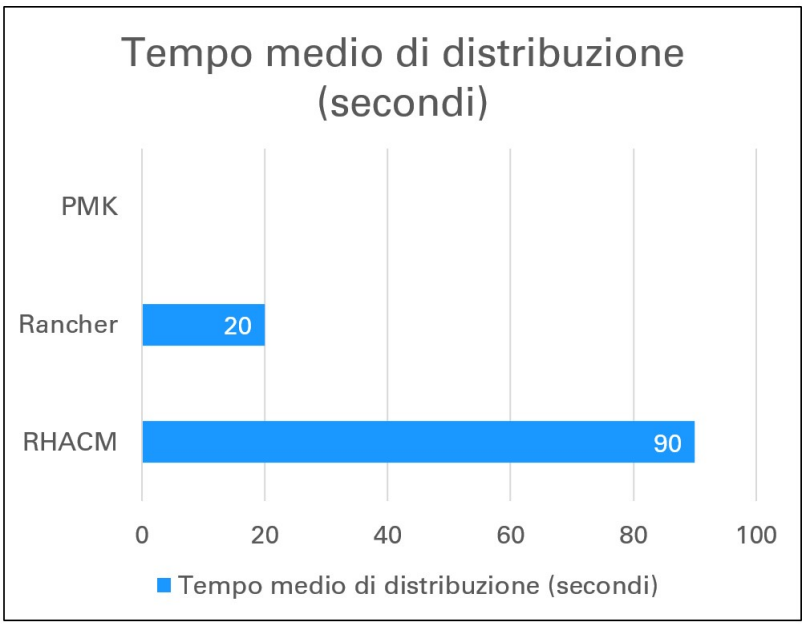

Figura 5.1: Comparazione tempi di propagazione di un'applicazione di test (Restic-Velero)

5.1.6 Supporto e documentazione

Si sono voluti evidenziare in questo documento anche gli aspetti relativi a completezza e precisione della documentazione a supporto dello strumento analizzato, che si ritengono di fondamentale importanza sia nelle fasi iniziali di studio dello strumento sia in caso di troubleshooting. Le documentazioni a supporto dei due strumenti presentano un divario in termini di qualità, è proprio quella relativa al prodotto Red Hat che presenta falle in alcune sezioni, sopratutto in merito a infrastruttura e connettività tra cluster, non chiarendo dettagli di fondamentale

importanza in fase di progettazione di una rete multi-cloud. Rancher ha invece permesso di evitare problemi durante il provisioning del'infrastruttura, già affrontati in ACM, grazie a una piena trasparenza in merito a connettività, rendendo ben chiari i requisiti indispensabili per la comunicazione tra i cluster.

Tabella 5.2: Confronto in termini di capabilities tra ACM e Rancher

Funzionalità	ACM	Rancher
Installazione, compatibilità e costi		
Requisiti d'installazione	Operator OpenShift da installare quindi solo minima versione di OpenShift	Modulabili sulla dimensione della rete da gestire. Installazione disponibile sia via Helm che Docker
Compatibilità con cloud vendor e piattaforme on-premise	Principali vendor e piattaforme, pubbliche e private	Principali vendor e piattaforme, pubbliche e private, e integrazione con K3s e RKE
Facilità d'installazione	Installazione basiliare tramite marketplace integrato della piattaforma OpenShift	Provisioning tramite Helm chart o Docker image
Costi (licenza, risorse hardware, etc.)	Alte richieste di hardware e licenze a pagamento per il supporto.	Richieste hardware minime molto basse e modulabili a seconda della dimensione della rete. License-free ma è presente anche l'opportunità di richiedere una licenza enterprise per supporto dedicato.
Possibilità di spegnere e accendere i nodi	A causa di OpenShift, la pratica non è attuabile se non attraverso dei workaround non ufficiali.	Basandosi su Kubernetes nativo, lo spegnimento, anche in piattaforme pubbliche, non presenta problematiche.
Gestione della rete multicluster		
Creazione, aggiornamento e distruzione di cluster remoti	Permette di creare, distruggere, importare e aggiornare cluster. La distruzione totale su piattaforme pubbliche non lascia residui.	Permette di creare, importare e rimuovere cluster. Non permette aggiornamento se non per cluster K3s o RKE, mentre la rimozione lascia residui sui cluster di downstream.

Import e detachment di cluster remoti	Permette di aggiungere un cluster fornendo all'utente un comando kubectl da lanciare sul cluster di destinazione. Il detachment potrebbe lasciare dei residui ma la documentazione a supporto fornisce linee guida per la loro pulizie	*Come ACM*
Connettività tra cluster	Il modello di connettività non è documentato a dovere e presenta poca flessibilità, risultando poco adattabile in contesti privati.	Modello di connettività altamente documentato e flessibile, variando nei suoi aspetti principali a seconda delle piattaforme utilizzate permettendo di adattarsi a qualsiasi infrastruttura.
Gestione dei workload distribuiti		
Compatibilità in termini di source	Compatibile con repository Git, Helm chart e object storage. La seconda opzione risulta però inutilizzabile causa la mancata possibilità di utilizzare dei valori custom.	Compatibile con Git repository tramite Fleet, che permette di utilizzare sia raw manifest, Kustomize che Helm chart sia locali che remoti.
Efficacia della procedura guidata	Semplice e chiara, con possibilità di programmare le propagazioni.	Semplice e chiara ma senza feature aggiuntive
Efficacia delle source	La più flessibile risulta la repository Git, anche se non ne vengono documentati le effettive potenzialità come quella di usarla per propagare helm charts.	Grazie a Fleet la git repository diventa un potente strumento che, tramite una corretta gerarchia di folder, permette di propagare applicativi con valori personalizzati per cluster.
Efficacia della CI/CD	Efficace e funzionante, il watching permette di accorgersi della modifica in poco tempo.	*Come ACM*
Monitoraggio dei progressi	Spesso fuori sincronia, poco aggiornato e non vengono riportato nel dettaglio eventuali errori dei cluster di downstream.	Propagazione sempre in sync e vengono riportati eventuali errori direttamente nell'hub.
Cluster selectors	Semplice, basato su labels.	*Come ACM*
Observability		

Efficacia dell'interfaccia	Chiara e minimale. Permette inoltre di accedere allo stato di salute di cluster e risorse dell'intera rete, filtrando eventualmente per namespace o label.	*Come ACM*
Disaster Recovery		
Backup-Recovery	Assenti per il singolo operator. Ereditati a livello di cluster dalla piattaforma OpenShift.	Presenti sia a livello di cluster solo se presente K3s o RKE sia a livello di hub tramite Rancher.
Migration dell'hub	Non disponibile	Disponibile tramite Rancher e permette di recuperare un backup memorizzato e usarlo come punto di ripristino su un nuovo nodo.
Documentazione		
Completezza e precisione	Generalmente discretamente dettagliata, senza troppa precisione a dettagli di livello infrastrutturale. Presenta però grosse mancanze in alcune sezioni come quelle relative alla connettività e all'utilizzo di helm chart come source.	Generalmente molto completa e precisa in tutte le sezioni. Sono approfonditi nel dettaglio anche aspetti di "core" come l'infrastruttura o la propagazione.

5.2 Use-cases applicativi

Si vogliono delineare infine, i possibili casi d'uso, in contesti naturalmente multi-cloud, in cui questi strumenti possono avere rilievo.
Per ciò che concerne Red Hat ACM, data la sua natura e a causa dei suoi alti requisiti, è una soluzione adottabile solo se si è in grado di soddisfarne le esigenze, tra cui la necessità di mantenere i cluster in funzione 24/24 7/7 in quanto non si prevede la possibilità di accendere e spegnere le macchine. Il costo pagato in requisiti però viene compensato dall'alta stabilità di OpenShift e dal suo pieno supporto al provisioning di componenti extra come stack logging, monitoring e ingress controller.
Rancher invece offre una soluzione il cui punto fondamentale è la flessibilità e l'alta scalabilità, permettendo di adattarsi ai diversi contesti applicativi, includendo

anche quelli più critici caratteristici di IoT. Il basarsi su Kubernetes gli permette di gestire reti multi-cluster fino all'ordine di migliaia di cluster, sia questi gestiti da terzi, su macchine proprie o containerizzati (RKE). L'alta flessibilità prevede anche di fornire diversi approcci infrastrutturali in fase di pianificazione della propria topologia, evitando quindi che, in caso sia già presente un'infrastruttura, questa debba essere riadattata in funzione dello strumento.
Entrambi gli use-case descritti prevedono però le stesse esigenze multi-cluster e multi-cloud, soddisfatte da entrambi gli strumenti:

- Singolo entrypoint di controllo da cui monitorare salute e risorse di ogni cluster
- Necessità di propagare workload sulla propria rete e configurarne comportamenti o parametri in funzione del cluster di destinazione
- Supporto ad automazione di provisioning di interi cluster su piattaforme pubbliche, private e server bare-metal

L'adozione di soluzioni simili, permette a figure professionali quali DevOps, IT Operations e sviluppatori, di ridurre notevolmente i tempi delle operazioni relative alla gestione di una moltitudine di cluster tramite singolo centro di controllo e così facendo, si ottiene anche un netto miglioramento della QoL. Questo, in contesti dove l'ordine dei cluster da gestire è dell'ordine delle centinaia o migliaia, si trasforma in aumento della produttività, riduzione dei tempi e rischi di errore, che invece vengono introdotti dalle *vecchie* soluzioni.

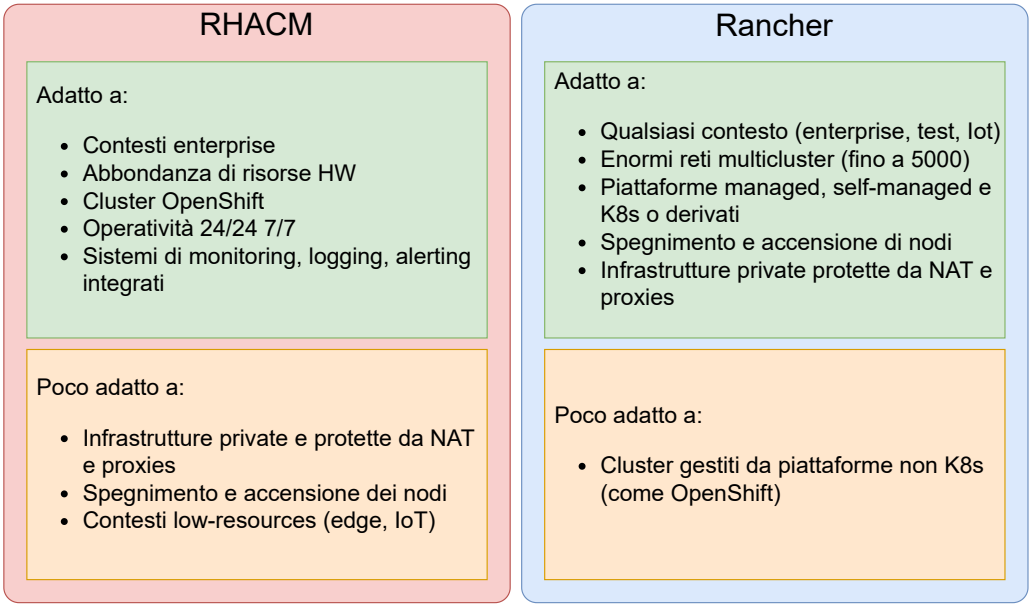

Figura 5.2: Riepilogo degli use-case per entrambi i prodotti. Si noti come la maggior flessibilità di Rancher gli permette di adattarsi alla maggior parte dei contesti.

Bibliografia

[1] Flexera. «Flexera 2021 - State of the Cloud Report». In: (2021) (cit. alle pp. i, 1, 3, 5).

[2] Gartner. «Magic Quadrant for Cloud Infrastructure and Platform Services». In: (2021) (cit. alle pp. 1, 2).

[3] Gartner. «The Edge Completes the Cloud: A Gartner Trend Insight Report». In: (2018) (cit. a p. 15).

[4] Forrester. «The Forrester Wave™: Multicloud Container Development Platforms, Q3 2020». In: (2020) (cit. alle pp. 21, 22).