POLITECNICO DI TORINO

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

Politecnico
di Torino

Tesi di Laurea Magistrale

Business Continuity e Disaster Recovery di applicazioni cloud native su piattaforme hybrid e multi-cloud

Relatore

Prof. Fulvio Giovanni RISSO

Tutor aziendale

Dott. Danilo ABBALDO

Candidato

Giacomo BRUSAMOLIN

Aprile 2022

Sommario

Negli ultimi anni molte aziende, in ambito Enterprise, si sono approcciate al mondo del cloud computing al fine di modernizzare le proprie infrastrutture ed applicazioni e proporre servizi appetibili e sempre up-to-date ai propri clienti. Tra i driver principali di questa scelta vi sono sicuramente la riduzione dei costi necessari per rendere disponibile un servizio e la flessibilità della gestione infrastrutturale sottostante. Per sfruttare a pieno le potenzialità del cloud, come ad esempio la scalabilità, il mondo Enterprise si sta concentrando nello sviluppo di applicazioni cloud native, costruite e pensate per architetture a microservizi e container. Questa nuova metodologia di sviluppo delle applicazioni, ed i concetti chiave su cui si basa, permette inoltre di aumentare l'affidabilità necessaria per la Business Continuity: un aspetto ritenuto fondamentale per le realtà Enterprise. Il paradigma infrastrutturale messo a disposizione dalle piattaforme cloud sfrutta tecnologie ed architetture diverse rispetto alla generazione precedente. Proprio per tale motivo si rende necessario studiare ed applicare nuove tecniche per poter effettuare un backup dei dati e delle risorse coinvolte. Le operazioni di backup e Disaster Recovery sono diventate un processo fondamentale per le aziende Enterprise perché, se utilizzate opportunamente, possono permettere la Business Continuity e minimizzare i tempi di downtime. Il lavoro di tesi analizza inizialmente i concetti di Disaster Recovery e Business Continuity, studiandone ed analizzandone le strategie, e successivamente approfondisce le metodologie e le soluzioni per il backup di applicazioni cloud native ed il loro ripristino in caso di "disastro" o disservizio. La soluzione tecnologica adottata tiene in considerazione anche l'aspetto economico e la necessità di un eventuale acquisto di licenze. Per quelle realtà che non hanno requisiti stringenti in termini di RPO e RTO, soluzioni anche a basso costo possono essere la scelta ideale. Lo studio prende in considerazione diversi scenari per il ripristino delle applicazioni, per esempio il ripristino all'interno dello stesso sito primario oppure in posizioni geografiche diverse. Le applicazioni cloud native considerate nel lavoro di tesi sono gestite da OpenShift, piattaforma di hybrid cloud di Red Hat, installata sui cloud provider AWS ed Azure. Lo strumento utilizzato per il salvataggio, backup e ripristino delle applicazioni è Velero.

Ringraziamenti

Ringrazio i miei genitori e le persone a me più care per avermi accompagnato e sostenuto in questo percorso di laurea magistrale.

Indice

Elenco delle tabelle

Elenco delle figure

Acronimi

VM
: Virtual Machine

VMM
: Virtual Machine Manager

OS
: Operative System

CLI
: Command Line Interface

IT
: Information Technology

IaaS
: Infrastructure as a Service

PaaS
: Platform as a Service

SaaS
: Software as a Service

RHCOS
: Red Hat Enterprise Linux CoreOS

OCP
: OpenShift Container Platform

AWS
: Amazon Web Services

PV
: Persistent Volume

PVC
: Persistent Volume Claim

CRD
: Custom Resource Definition

VPC
: Virtual Private Cloud

DR
: Disaster Recovery

RPO
: Recovery Point Objective

RTO
: Recovery Time Objective

WRT
: Work Recovery Time

MTD
: Maximum Tolerable Downtime

KPI
: Key Performance Indicator

BC
: Business Continuity

DRaaS
: Disaster Recovery as a Service

TTL
: Time To Live

Capitolo 1

Introduzione

L'adozione di un modello di business legato totalmente o parzialmente al cloud è ormai una delle scelte più comuni all'interno delle realtà aziendali. La tecnologia a microservizi, abbinata al cloud computing, è quella che ha riscosso un grande interesse in ambito enterprise, poiché permette flessibilità, scalabilità e allo stesso tempo influenza positivamente la business continuity, un aspetto ritenuto fondamentale nelle aziende.

Nonostante le nuove piattaforme basate sul paradigma a microservizi, come OpenShift, possano offrire servizi leggeri, scalabili e resilienti, non è possibile definirle anche come sistemi immuni da possibili perdite di dati o errori. Per cui è necessario studiare un sistema di backup per la protezione dei dati che sia in grado di soddisfare tutte le esigenze delle realtà aziendali. Poiché le tecnologie in questione sono diverse dalla generazione precedente, si rende necessario studiare ed applicare nuove tecniche per effettuare il backup.

Lo studio di tesi mira, in primo luogo, ad analizzare i piani di business continuity e disaster recovery in ambito enterprise per identificare le richieste ed evidenziare le differenze. Successivamente si analizzeranno le strategie adottabili per il backup e per il ripristino di applicazioni cloud native, evidenziandone le differenze con le tecnologie precedenti ed elencando tutti i componenti coinvolti. Dopo aver identificato le metodologie di disaster recovery si passerà allo studio di soluzioni a basso costo, valide per tutte quelle aziende che non hanno requisiti stringenti in termini di RPO e RTO.

Verranno presentati diversi scenari di disaster recovery applicati a soluzioni differenti, cioè dal recupero all'interno dello stesso cluster fino alla migrazione in posizioni geografiche diverse e su cloud provider differenti. Per ciascuno di essi saranno elencati i benefici e le limitazioni, nonché le tempistiche relative ai processi di backup e al ripristino dei servizi. In ultimo, ma non meno rilevante, ci si concentrerà sullo studio del prezzo, variabile determinante per la scelta della soluzione. Sarà quindi discusso il costo di una soluzione che apparentemente non sembra

essere d'aiuto al business dell'azienda, ma che invece è di fondamentale importanza. Infatti, in caso di "disastro" si possono avere conseguenze molto negative per il fatturato dell'azienda colpita, perciò è necessario progettare preventivamente un sistema di continuità del business.

Le applicazioni cloud native considerate nel lavoro di tesi sono gestite da OpenShift, piattaforma di hybrid cloud di Red Hat, installata sui cloud provider AWS ed Azure. Lo studio di queste tecnologie è stato utile per affrontare al meglio il problema del disaster recovery e per avvicinarsi di più alle reali necessità aziendali.

Le risorse economiche legate all'utilizzo delle infrastrutture cloud sono state offerte dall'azienda con cui è stato svolto questo lavoro di tesi: Blue Reply. Grazie a queste risorse è stato possibile sperimentare l'utilizzo dei principali cloud provider ed acquisire familiarità con la piattaforma enterprise utilizzata da molte realtà.

Capitolo 2

Contesto

2.1 Nuove tecnologie nel mondo enterprise

La sviluppo della tecnologia continua a crescere ogni anno e ormai tutte le realtà aziendali si affidano ad essa. Una delle grandi svolte dell'ultimo periodo è il continuo investimento e la crescita del cloud computing, che ha portato alla diffusione della tecnologia dei container superando la precedente basata sull'utilizzo esclusivo delle virtual machine.

2.1.1 VM vs container

Le macchine virtuali sono nate per eseguire software sopra un server fisico e quindi per emulare un sistema hardware. Esse sono avviate in modo indipendente dalle altre, creando un insieme unico di risorse non condivisibili formato da: sistema operativo, librerie e applicazioni. Ciò è possibile tramite la virtualizzazione resa disponibile dal *Hypervisor*, anche chiamato VMM, che emula tutte le risorse necessarie per ciascuna VM, in modo tale da consentire ai sistemi operativi eseguiti al loro interno di "credere" di poter accedere liberamente all'hardware sottostante.

Le VM sono un'ottima soluzione perché permettono la separazione delle applicazioni con librerie e sistemi operativi diversi e allo stesso tempo consentono di risparmiare con il numero di server fisici da installare. Questo approccio ha portato alla separazione dei concetti di server a disposizione e applicazioni eseguite al loro interno. Al contrario, le VM richiedono molte più risorse rispetto all'esecuzione del software direttamente sulla macchina con il proprio OS, rendendole così meno efficienti. Inoltre, sono presenti più problemi di privacy e sicurezza.

Rimanendo in tema di virtualizzazione, i container sono una soluzione più vantaggiosa che risolve i problemi riscontrati con le VM. Essi utilizzano un OS comune, come si può vedere nell'immagine 2.1, rendendo il sistema più efficiente e flessibile, grazie al quale sono più leggeri in termini di spazio occupato sul disco

e hanno meno overhead di virtualizzazione. I container consumano anche meno memoria permettendo l'allocazione dinamica, caratteristica non presente sulle macchine virtuali. Altri vantaggi sono la scalabilità e la portabilità poiché ogni server fisico può contenere centinaia di container, i quali possono essere facilmente spostati in caso di necessità e avviati rapidamente. Infine, essi applicano un isolamento logico virtualizzando tutte le risorse a livello di sistema operativo, permettendo di avere una *sandbox* isolata rispetto alle altre applicazioni.

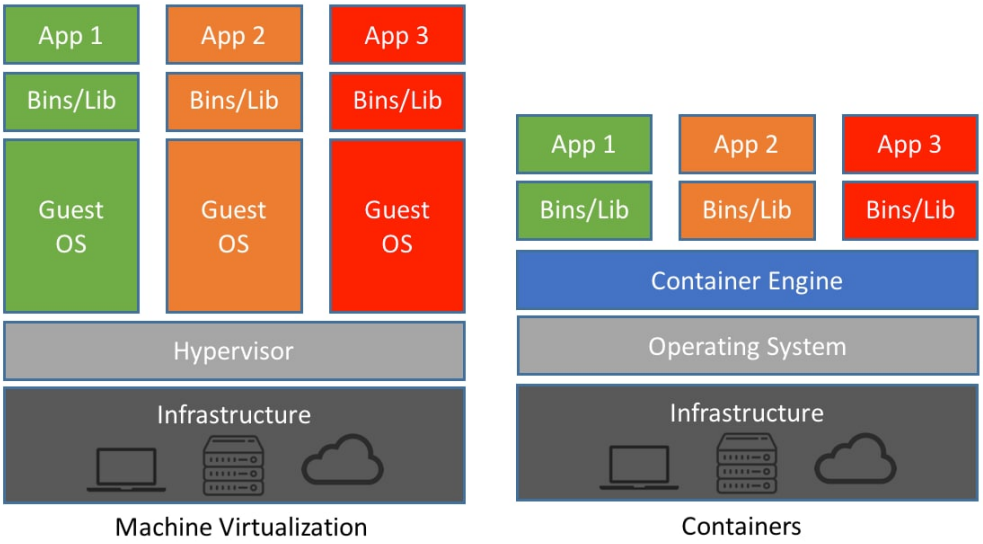

Figura 2.1: VM vs container [1]

2.2 Cloud computing

Il concetto di cloud computing è ormai noto a molte realtà aziendali, le quali sono passate in parte o interamente a gestire ed utilizzare questo nuovo paradigma di erogazione dei servizi. Il cloud computing si può quindi definire come la distribuzione di risorse informatiche accessibili tramite Internet. Queste risorse sono ad esempio servizi di calcolo, di archiviazione, server o database e hanno solitamente una tariffa basata sul consumo.

L'adozione del cloud è in continua crescita e, oggi, è una strategia che offre molte opportunità e vantaggi. Come documenta il *Four Trends Are Shaping the Future of Public Cloud 2021* di Gartner [2], l'adozione globale del cloud continuerà ad espandersi rapidamente, Gartner prevede che la spesa degli utenti finali per i servizi cloud pubblici raggiungerà i $396 miliardi nel 2021 e crescerà del 21,7% per raggiungere i $482 miliardi di dollari nel 2022.

I motivi per cui le aziende scelgono di utilizzare i servizi resi disponibili dal cloud sono tanti e di vario tipo, poiché ognuna di queste ha esigenze diverse. Di seguito sono elencati i principali vantaggi e le motivazioni per cui le realtà aziendali scelgono di utilizzarli.

Principali vantaggi

- **Velocità**: le risorse vengono allocate in pochi minuti dopo la richiesta, contrariamente a quanto accadrebbe utilizzando un server installato *on-premise*, che richiederebbe un'attesa elevata.
- **Scalabilità**: è uno dei vantaggi più interessanti e consente di modificare e ridimensionare le risorse, seguendo le proprie necessità in poco tempo e nella località geografica d'interesse.
- **Costo**: la comodità e la velocità di acquisire risorse non sono accompagnate da un costo maggiore. La creazione di un data center o di un server per la gestione di servizi ha costi molto maggiori e poco giustificabili anche nel lungo periodo, sia per l'evoluzione della tecnologia, sia per la crescita dell'azienda stessa. Nel cloud si pagano solo le risorse realmente utilizzate.
- **Elasticità**: grazie al provisioning rapido è possibile gestire i picchi di attività durante certi orari lavorativi e, grazie alla grande diffusione territoriale del cloud, espandere i servizi in altre zone geografiche.
- **Produttività**: i server *on-premise* devono essere installati, gestiti e mantenuti non solo fisicamente, ma anche tramite patch di sicurezza. Il cloud permette di eliminare i data center locali e consente all'azienda di focalizzarsi esclusivamente sui propri obiettivi.

- **Sito di disaster recovery**: il cloud può essere utilizzato anche all'interno di una strategia per il recupero a seguito di un disservizio presentatosi in una installazione *on-premise*.

2.2.1 Soluzioni

Ci sono diversi tipi di soluzioni che si possono adottare quando ci si approccia al cloud. Di seguito sono elencate le tre modalità di distribuzione dei servizi e sono spiegate le differenze ed i vantaggi.

Cloud pubblico

Il cloud pubblico appartiene ad un provider di terze parti, il quale si occupa di mantenere l'infrastruttura, l'hardware e il software. I servizi da esso erogati possono essere risorse di calcolo, di archiviazione o altro e sono gestibili tramite web browser o CLI. In questa modalità le risorse fisiche del fornitore sono condivise tra tutti gli utenti e gestite con limitazioni opportune.

Cloud privato

Questa modalità si contrappone alla precedente e si riferisce alle risorse cloud usate esclusivamente da una singola azienda. Il cloud privato si può trovare fisicamente nel data center dell'organizzazione oppure l'azienda può pagare un provider di terze parti che, a sua volta, fornisce lo spazio e l'alimentazione. Inoltre, il fornitore di servizi cloud può offrire anche le macchine e infrastruttura. Quest'ultimo caso si distingue da quello del cloud pubblico perché le risorse sono completamente dedicate, inclusa la rete che non è condivisa. Grazie al cloud privato l'azienda ha pieno controllo dei dati e si avvale di un'infrastruttura più sicura, ma va incontro ad una spesa maggiore.

Cloud ibrido

Come suggerisce la parola "ibrido", questa modalità è un misto tra le due precedentemente descritte. Il cloud ibrido permette di spostare dati e applicazioni tra una soluzione di cloud privata e una pubblica, rendendolo molto flessibile dal punto di vista della gestione del carico e della sicurezza.

Oggi il termine cloud privato è forviante perché può includere un ambiente situato all'interno di un provider di terze parti, per questo motivo la definizione di cloud ibrido non è sufficiente e richiede un elenco di caratteristiche. Esso deve quindi includere due o più ambienti, essere scalabile rapidamente, consentire di spostare le applicazioni tra gli ambienti ed integrare uno strumento di gestione unico. La caratteristica principale delle soluzioni ibride moderne è la portabilità

delle applicazioni eseguite negli ambienti. Essa richiede l'esecuzione dello stesso sistema operativo in ogni ambiente IT e di una piattaforma unificata per gestire tutti gli elementi [3]. Le applicazioni devono perciò essere create per tale scopo e si identificano con il nome di *cloud native applications*, di questo e della loro orchestrazione se ne parlerà nel capitolo 2.3.

È bene sottolineare la diffusione di quest'ultima soluzione cloud, il sondaggio *Flexera 2021 State of the Cloud Report* [4] evidenzia che il 92% delle aziende dispone di una strategia multi-cloud e l'80% di una strategia di cloud ibrido. Inoltre nella figura 2.2 si riporta che il 99% delle organizzazioni intervistate nel sondaggio utilizzano almeno un cloud pubblico o privato.

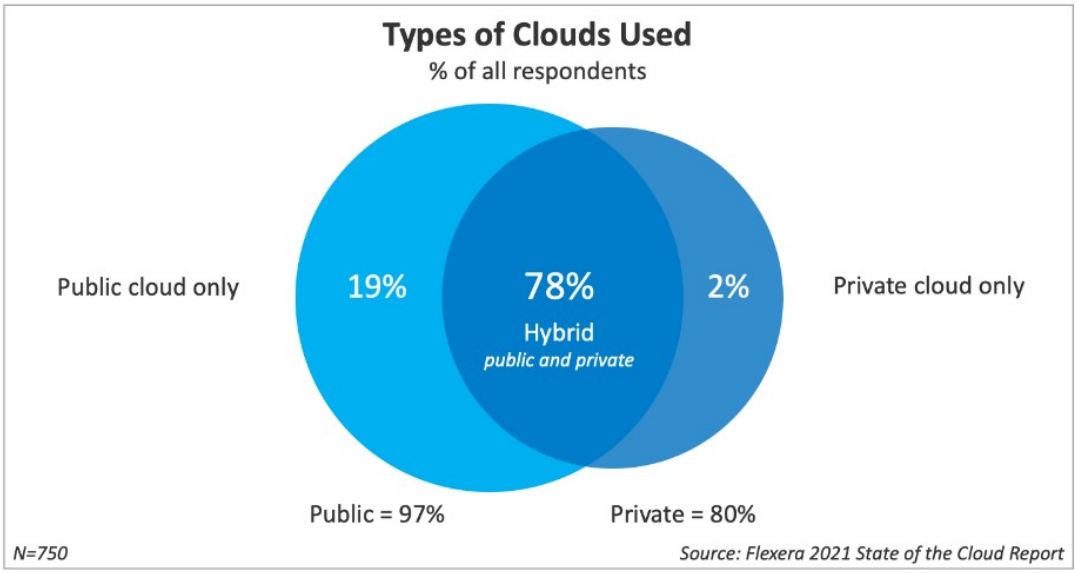

Figura 2.2: Tipi di cloud utilizzati dalle aziende [4]

2.2.2 Servizi

I cloud provider offrono servizi che si distinguono in tre principali modelli, ciascuno dei quali rappresenta un aspetto diverso del cloud computing e propone differenti livelli di gestione e controllo.

- **IaaS**: *Infrastructure as a Service* è l'insieme dei servizi che permette di affittare l'infrastruttura, quindi di poter gestire e creare VM, risorse di archiviazione e reti da un cloud provider. Esso offre il livello più alto di controllo sulle risorse ed è anche il più impegnativo dal punto di vista della manutenzione; inoltre permette la migrazione al cloud senza modificare le applicazioni.

- **Paas**: *Platform as a Service* è un tipo di servizio di più alto livello che consente al customer di concentrarsi di più sulla realizzazione delle applicazioni, senza dover occuparsi degli aspetti tecnici dell'infrastruttura, come il provisioning delle risorse, la manutenzione del software e le patch di sicurezza. Questo servizio rende astratto il sistema operativo affidando la sua gestione e configurazione al cloud provider.
- **SaaS**: *Software as a Service* offre un prodotto completo, gestito dal cloud provider; il customer non si deve preoccupare della manutenzione e di gestire l'infrastruttura. Il software viene consegnato con "chiavi in mano", senza doverlo sviluppare; infatti difficilmente è possibile richiedere modifiche personalizzate. Un esempio comune è un servizio di e-mail accessibile tramite web browser.

2.3 OpenShift

OpenShift è una piattaforma di orchestrazione di container basata su Kubernetes e sviluppata da Red Hat. È pensata per l'implementazione in ambienti di produzione poiché è venduta come un pacchetto completo, offre un valido supporto e aggiunge alcune funzionalità rispetto al core principale che è Kubernetes.

Kubernetes è un framework open-source creato da Google per gestire i container e automatizzare la distribuzione di applicazioni. Il sistema implementa la tecnica di sviluppo software a microservizi, cioè una tecnica di suddivisione dell'intera applicazione in componenti base; ciò porta innumerevoli vantaggi in termini di scalabilità e resilienza.

2.3.1 Architettura

Un cluster OpenShift è formato da un insieme di nodi, cioè delle macchine fisiche o virtuali che eseguono delle applicazioni all'interno di container. I nodi si distinguono in due categorie: nodi *worker* e nodi del *control plane*. I primi sono incaricati di eseguire i workload dell'utente, cioè le applicazioni richieste. I secondi, invece, gestiscono i nodi worker e tutto ciò che riguarda il sistema stesso.

Di seguito sono elencati i componenti di Kubernetes raffigurati nella figura 2.3, successivamente sono evidenziate le caratteristiche di OpenShift.

Componenti del control plane

Il *control plane* è caratterizzato da diversi componenti che svolgono la funzione di controllo su tutto il cluster. Solitamente, esso viene assegnato a più di un nodo per aumentare la tolleranza ai guasti e garantire l'alta disponibilità dei servizi offerti. Inoltre, le applicazioni non sono eseguite all'interno di questi nodi.

- **api-server**: si tratta del servizio che permette di accedere al control plane, rende disponibili le REST API di Kubernetes, processa le richieste API degli utenti ed aggiorna, di conseguenza, lo stato degli oggetti. Questo servizio si interfaccia con tutti i componenti ed è progettato per scalare orizzontalmente, cioè all'aumentare del numero di istanze.
- **kube-controller-manager**: è l'insieme dei processi che esegue un loop per controllare lo stato condiviso del cluster; essi hanno il compito di portare il cluster nello stato desiderato modificando quello corrente. Esistono diversi tipi di controller, ognuno dei quali si occupa di un oggetto del cluster.
- **cloud-controller-manager**: è un componente presente solo nelle installazioni su cloud providers, invece nei cluster on-premise è assente. Come il kube-controller-manager, esso esegue dei loop ma in questo caso i processi sono

specifici per il cloud a disposizione e quindi fortemente legati alle API del provider. I loop si possono occupare, ad esempio, del controllo dei nodi a disposizione o della gestione dell'infrastruttura, come la creazione e gestione dei load balancer.

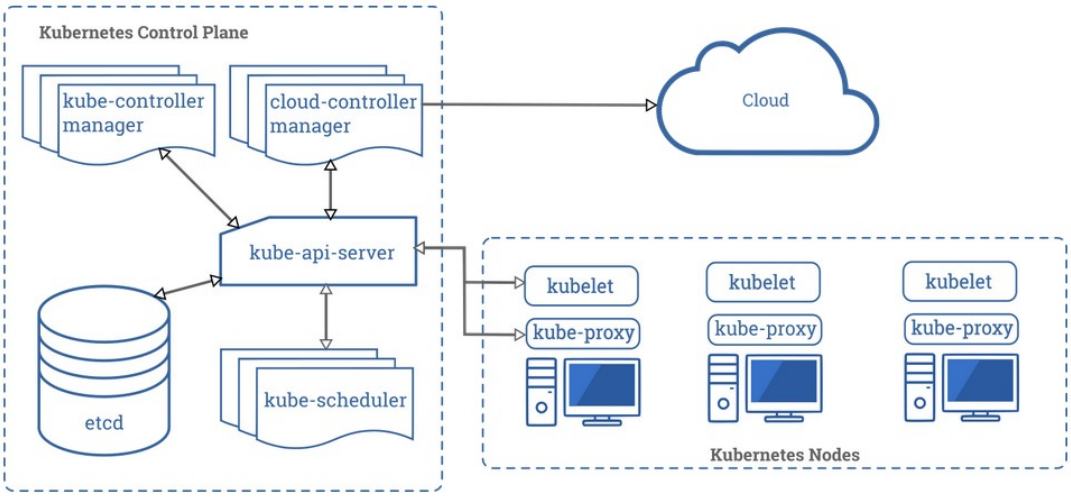

Figura 2.3: Diagramma di un cluster Kubernetes con i suoi componenti e relazioni. [5]

- **scheduler**: è il componente che ha il compito di controllare i nuovi container da creare e di assegnargli un nodo su cui farli eseguire. La scelta del nodo si basa su diversi fattori, tra cui la valutazione delle affinity e anti-affinity, delle risorse di calcolo disponibili e dei vincoli di policy.
- **etcd**: si tratta di un database ridondato e distribuito che utilizza il sistema per salvare ogni informazioni del cluster. Rappresenta lo stato generale di tutte le risorse in un preciso momento, per questo è in continuo aggiornamento.

Componenti dei nodi

Ogni nodo del cluster è caratterizzato da alcuni componenti che permettono l'esecuzione dei container e del loro environment.

- **kubelet**: è un agente responsabile della creazione e del monitoraggio di container, che assicura la loro esecuzione in un Pod e il loro funzionamento corretto.
- **kube-proxy**: è un proxy di rete che permette ai nodi di comunicare con gli altri nodi e con l'esterno del cluster. Ha il compito di gestire le regole di rete utilizzando le librerie del sistema operativo oppure in modo diretto.

- **Container runtime**: si tratta del software responsabile della gestione dei container, si occupa ad esempio dell'esecuzione e della terminazione di essi. OpenShift utilizza CRI-O, un'implementazione di *container runtime* che si integra strettamente al sistema operativo.

Implementazioni di Red Hat

Red Hat offre un pacchetto completo per consentire alle aziende di sviluppare le applicazioni con semplicità e consentendo loro di scalare in base alle necessità. OpenShift Container Platform è una piattaforma che gestisce deployment on-premise, su cloud ibrido e multi-cloud. Il sistema permette di gestire tutte le risorse del cluster e le applicazioni degli utenti tramite una pagina web (come mostra la figura 2.4) oppure da una custom CLI chiamata `oc`, dopo un'opportuna autenticazione.

Le informazioni riguardanti questa tecnologia derivano dalla documentazione ufficiale [6] e dall'esperienza personale.

Per quanto riguarda il control plane, oltre ai componenti di Kubernetes, sono eseguiti anche altri servizi di OCP. Si tratta di un **openshift-apiserver** che si occupa di gestire le risorse di OpenShift, un **openshift-controller-manager** che controlla le modifiche agli oggetti OpenShift nell'etcd e un servizio di autenticazione che gestisce utenti, gruppi e token con una API dedicata **openshift-oauth-apiserver**.

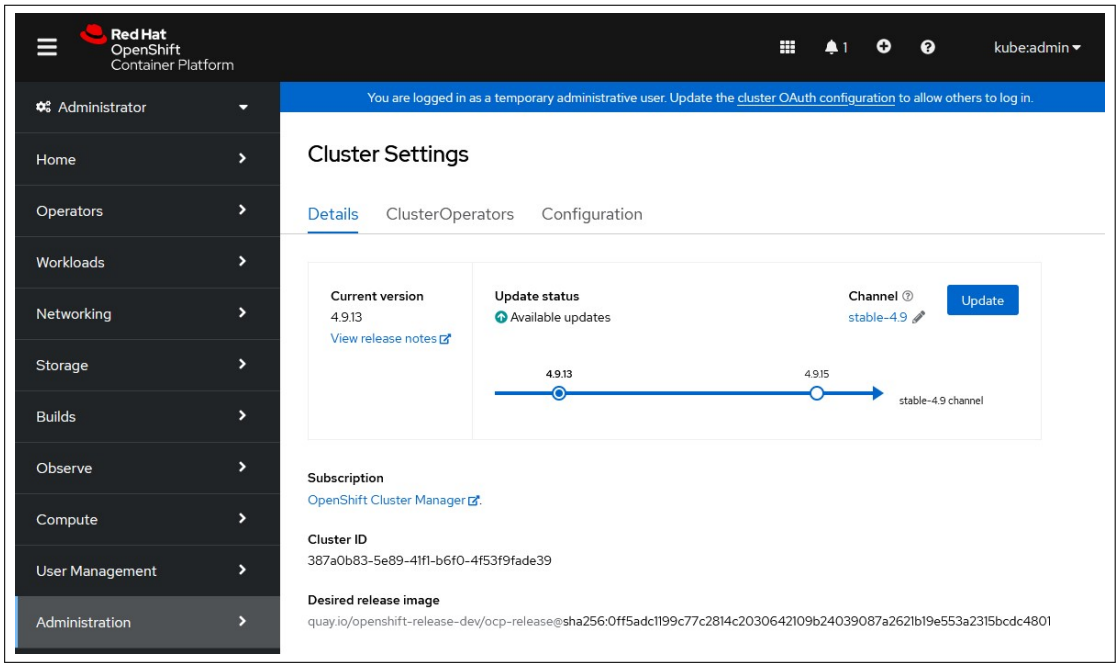

Figura 2.4: OCP da web browser: accesso eseguito con le credenziali di kube:admin

Ognuno di questi servizi ha il proprio operatore che ha il compito di gestire il lifecycle.

Gli **operatori** sono tra i componenti più importati di OCP, servono a distribuire e gestire i servizi per il control plane. Essi permettono di eseguire controlli sulla vita dei container, gestiscono gli aggiornamenti della piattaforma e possono anche essere utilizzati per offrire vantaggi alle applicazioni degli utenti. OpenShift mette a disposizione un'interfaccia web, chiamata *OperatorHub*, in cui gli amministratori del cluster possono cercare e installare operatori.

Ciascun nodo del cluster utilizza il sistema operativo enterprise **RHCOS** che è stato sviluppato da Red Hat unicamente per OpenShift. L'OS include il container runtime **CRI-O** che, rispetto ad altri, offre un set di funzionalità più ampio riducendo il rischio di attacchi informatici. CRI-O è un alternativa più leggera di Docker e gode di uno sviluppo interamente legato a Kubernetes.

OCP include anche un sistema di monitoraggio già preconfigurato, in cui sono inclusi una serie di alert che, in caso di problemi, notificano gli amministratori. Questo monitoraggio avviene grazie a **Prometheus**, creato da un Premetheus operator già incluso nel sistema. Per la visualizzazione delle statistiche e metriche è possibile configurare **Grafana**, anch'esso incluso. Gli amministratori del cluster, inoltre, possono decidere di abilitare il monitoraggio per i progetti definiti dagli utenti, in questo modo gli sviluppatori possono facilmente specificare quali Pod sono da monitorare e con quali metriche.

2.3.2 Come si crea un cluster

L'installazione del cluster avviene grazie ad un file di configurazione, chiamato *Ignition*, che viene utilizzato all'avvio dalle macchine. Questo file descrive il partizionamento dei dischi, la formattazione delle partizioni, la scrittura di file e la configurazione degli utenti. Per ogni macchina, *Ignition* prende l'immagine RHCOS e l'avvia con le configurazioni richieste. La fase di installazione è caratterizzata da una macchina di *bootstrap* che è avviata prima dei nodi e, alla fine del processo, viene eliminata. OCP si può installare in due modalità che l'utente può scegliere a seconda delle sue necessità.

- **Installer-provisioned**: si tratta di un'installazione guidata e consigliata da Red Hat che permette di avviare un cluster su un ambiente cloud con una configurazione minima. La gestione dell'infrastruttura è mantenuta da OpenShift.

- **User-provisioned**: è utilizzata quando si ha bisogno di più flessibilità e richiede un'infrastruttura esistente che deve essere mantenuta separatamente dall'utente.

L'installazione *Installer-provisioned* è eseguita tramite un programma rilasciato da Red Hat che si occupa della configurazione e creazione delle macchine necessarie. La procedura d'installazione di un cluster è composta da due fasi:

1. **Creazione guidata del file di configurazione**: si tratta della creazione del file *install-config.yaml* che può essere eseguita manualmente dall'utente oppure tramite una procedura guidata dal programma di installazione con il seguente comando:

   ```
   openshift-install create install-config --dir <directory>
   ```

2. **Avvio della procedura di installazione**: in questa fase è eseguito il comando per la creazione del cluster e di tutte le risorse che necessita sul cloud provider selezionato. Il comando richiede una cartella in cui salvare i file di output della procedura ed è possibile scegliere un livello di log da visualizzare:

   ```
   openshift-install create cluster --dir <directory> --log-level=info
   ```

Il file di configurazione *install-config.yaml* contiene diverse informazioni per la creazione del cluster, di seguito sono elecate le più rilevanti, per ulteriori approfondimenti si può fare riferimento alla guida ufficiale [6].

baseDomain: indica il dominio base per l'accesso al cluster e alle sue applicazioni.

compute: è la sezione in cui si specificano i dettagli dei nodi worker. É possibile scegliere l'architettura, le repliche e, opzionalmente, i dettagli della macchina virtuale da utilizzare e quanta memoria allocare.

controlPlane: è la sezione che riguarda le specifiche del controlPlane, molto simile a quella dei nodi worker. Si evidenzia che i requisiti per le macchine virtuali sono più stringenti rispetto a quelli dei nodi worker.

metadata: si possono aggiungere metadati del cluster, come il nome.

networking: si tratta della configurazione legata alla rete delle macchine (*machineNetwork*), dei Pod (*clusterNetwork*) e dei servizi (*serviceNetwork*).

platform: è la sezione che consente di aggiungere altri dettagli del cloud provider selezionato, come la regione di installazione o i *tag* da usare per le nuove risorse.

publish: indica se il cluster è da installare con un dominio pubblico (*External*) oppure privato (*Internal*).

pullSecret: è una stringa segreta, acquisibile dal proprio account Red Hat, che permette l'autenticazione per poter scaricare le immagini dei componenti di OCP.

sshKey: è richiesta almeno una chiave ssh per autenticare l'accesso alle macchine del cluster.

Di seguito è riportato un esempio di configurazione in YAML (formato per la serializzazione di dati tramite caratteri) per l'installazione di un cluster pubblico con tre nodi worker e tre control plane in modalità *installer-provisioned* su AWS. Inoltre, sono specificati i tipi di macchina da utilizzare e la dimensione della loro memoria persistente.

```yaml
apiVersion: v1
baseDomain: tesibrusamolin.xyz
compute:
- architecture: amd64
  hyperthreading: Enabled
  name: worker
  platform:
    aws:
      rootVolume:
        size: 100
      type: t3a.xlarge
  replicas: 3
controlPlane:
  architecture: amd64
  hyperthreading: Enabled
  name: master
  platform:
    aws:
      rootVolume:
        size: 100
      type: m5a.xlarge
  replicas: 3
metadata:
  creationTimestamp: null
  name: cluster1
networking:
  clusterNetwork:
  - cidr: 10.128.0.0/14
```

```
29      hostPrefix: 23
30    machineNetwork:
31    - cidr: 10.0.0.0/16
32    networkType: OpenShiftSDN
33    serviceNetwork:
34    - 172.30.0.0/16
35  platform:
36    aws:
37      region: eu-south-1
38      userTags:
39        owner: brusamolin
40        project: tesi
41  publish: External
42  pullSecret: ''
43  sshKey: |
44    ssh-rsa...
```

Per quanto riguarda l'installazione di un cluster privato, OpenShift non crea endpoint esterni, per cui il cluster è accessibile solo dalla rete interna e non da Internet. A differenza della procedura con dominio pubblico, il programma di installazione va eseguito in una macchina che ha accesso alle seguenti risorse:

- Rete privata sulla quale si vuole creare il cluster.
- Internet.
- Servizi API del cloud utilizzato.

Gli elementi elencati di seguito non sono richiesti o creati per l'installazione di un cluster privato:

- Sottoreti pubbliche.
- Load balancer pubblici.
- Una zona pubblica per il dominio utilizzato.

In questo caso, il programma di installazione ha bisogno di un *install-config* completo di sottoreti private da utilizzare. Tale procedura non crea tutte le risorse necessarie, contrariamente a quanto avviene nella procedura standard. Infatti, alcune risorse devono essere correttamente configurate prima dell'avvio della creazione del cluster; si tratta di:

- Gateway Internet

- Gateway NAT
- Sottoreti
- Tabelle di routing
- VPC
- Set opzioni DHCP
- Endpoint

Di seguito si riporta un esempio di *install-config* per la creazione di un cluster privato utilizzando tre sottoreti private di una VPC di Amazon Web Service.

```
apiVersion: v1
baseDomain: tesibrusamolin.local
compute:
- architecture: amd64
  hyperthreading: Enabled
  name: worker
  platform: {}
  replicas: 2
controlPlane:
  architecture: amd64
  hyperthreading: Enabled
  name: master
  platform: {}
  replicas: 3
metadata:
  creationTimestamp: null
  name: cluster1
networking:
  clusterNetwork:
  - cidr: 10.128.0.0/14
    hostPrefix: 23
  machineNetwork:
  - cidr: 10.0.0.0/16
  networkType: OpenShiftSDN
  serviceNetwork:
  - 172.30.0.0/16
platform:
```

```
28    aws:
29      region: eu-south-1
30      userTags:
31        owner: brusamolin
32        project: tesi
33      subnets:
34      - subnet-0621bb1f026ed4a42
35      - subnet-06b03daaba5aa6766
36      - subnet-0af9b700a2335a2f9
37  publish: Internal
38  pullSecret: ''
39  sshKey: |
40    ssh-rsa...
```

2.3.3 Gli oggetti del sistema

Come in Kubernetes, gli oggetti di OpenShift sono entità che identificano lo stato del sistema e dei suoi servizi e possono rappresentare le risorse di un'applicazione o delle sue policy. Ciascun oggetto è composto da una descrizione relativa alle specifiche richieste e allo stato corrente rilevato dal sistema.

Project

Il *project* è l'implementazione di Red Hat del namespace in Kubernetes. Esso offre la possibilità di dividere le applicazioni isolando gruppi di risorse all'interno di un unico cluster. Questo isolamento si può applicare solo ad alcuni oggetti, come Pod e Deployment, e non a quelli che identificano servizi per cluster intero, come PersistentVolume e StorageClass. I project si possono usare per ambienti con tanti utenti e gruppi di lavoro perché permettono molta flessibilità di configurazione dell'ambiente stesso. Non possono essere creati all'interno di altri project.

Pod

Il *Pod* è la più piccola unità di calcolo all'interno di un cluster. Include al suo interno uno o più container e definisce in che modo collegare gli archivi di dati e le risorse di rete. I container dello stesso Pod condividono le risorse di calcolo che vengono assegnate a quest'ultimo e sono in grado di comunicare liberamente tra di loro perché appartengono ad uno stesso contesto.

La definizione di un Pod contiene l'elenco dei container eseguiti, la loro immagine e altri dettagli di rete. Solitamente i Pod non vengono creati direttamente dall'utente ma sono gestiti da altri oggetti del sistema che permettono migliore gestione di essi.

Label

Una *label* è un'etichetta che viene legata ad un oggetto ed è caratterizzata da una coppia di chiave e valore (*key/value*). Viene utilizzata per identificare facilmente gruppi di risorse con caratteristiche in comune, ad esempio appartenenti allo stesso gruppo di lavoro oppure alla stessa applicazione. Sono un aiuto per l'utente e non aggiungono funzionalità al sistema.

ReplicaSet

Il *ReplicaSet* definisce quali e quanti Pod deve rendere disponibili, cioè creare o distruggere in base alle specifiche. Questo oggetto ha quindi lo scopo di tenere stabile il numero di Pod in esecuzione in un determinato momento. Se viene richiesta la creazione di un nuovo Pod, il ReplicaSet opportuno ha il compito di

gestire questo cambiamento; la stessa operazione viene eseguita quando un Pod termina il suo ciclo di vita in modo involontario. Il ReplicaSet identifica i Pod da gestire tramite l'utilizzo di Selector, cioè una Label specifica per questo scopo.

Deployment

Il *Deployment* si pone ad un livello più alto del ReplicaSet e, di solito, è l'oggetto che viene modificato dall'utente per gestire i Pod. Tramite il Deployment l'utente può scegliere l'immagine da utilizzare per l'applicazione ed il numero di Pod da creare. Ogni volta che viene aggiunto un nuovo Deployment si crea in automatico un ReplicaSet che andrà a gestire i Pod richiesti.

StatefulSet

L'oggetto *StatefulSet* è usato da OpenShift per gestire le applicazioni stateful. Come i Deployment, un StatefulSet gestisce la distribuzione e la creazione dei Pod interessati, ma garantisce anche il mantenimento di un'identità per ciascuno di essi. Questi Pod non sono intercambiabili: ognuno ha un identificatore che viene mantenuto anche dopo una terminazione e ricreazione della risorsa.

DaemonSet

Un *DaemonSet* si occupa di eseguire una copia di un Pod richiesto in ogni nodo. Questo oggetto viene utilizzato solo per specifici utilizzi, come l'esecuzione di servizi per il monitoraggio di ciascun nodo oppure per la collezione di log.

Service

Il Service viene utilizzato per legare logicamente un insieme di Pod, inserendoli nella stessa rete, in modo tale da applicare a tutti le stesse policy. Ogni Pod appena creato riceve un nuovo indirizzo IP che può variare, invece il Service garantisce un accesso stabile anche dopo la ricreazione di un Pod. Il Service permette di creare una mappa tra una porta di ingresso e una di uscita, chiamata *target*, scegliendo il protocollo opportuno. Esso viene utilizzato per rendere disponibile un'applicazione all'interno o all'esterno del cluster. Ci sono diversi tipi di questo oggetto.

- **ClusterIP**: serve a rendere disponibile il servizio all'interno del cluster, perciò i Pod legati ad esso saranno visibili solo dal cluster stesso. Questo è il tipo di Service di default.
- **NodePort**: serve a legare una porta ad ogni IP di ciascun nodo, cioè rende possibile l'accesso ad un Pod dall'esterno tramite l'IP del nodo e la porta che

è stata configurata nel Service, chiamata *NodePort*. L'accesso è disponibile tramite la seguente richiesta: *<NodeIP>:<NodePort>*.

- **LoadBalancer**: permette di utilizzare i load balancer disponibili nei cloud provider. Se si utilizza questo tipo di Service viene creato in automatico un NodePort ed un ClusterIP per poter raggiungere i Pod all'interno dei nodi del cluster.
- **ExternalName**: serve a legare il Service ad un record CNAME; viene restituito il record CNAME con il suo valore.

Route

L'oggetto Route è l'implementazione di Red Hat del *Ingress* di Kubernetes, permette di gestisce l'accesso esterno al cluster, tipicamente tramite HTTP. L'instradamento del traffico di rete è definito dalle regole presenti nella risorsa Route; una volta che i pacchetti sono entrati nel cluster, vengono distribuiti ai Pod tramite la risorsa di tipo Service. Anche il protocollo HTTPS è supportato tramite una connessione sicura gestita dai certificati all'interno di TLS. Inoltre OpenShift accetta anche la creazione di risorse di tipo Ingress che vengono automaticamente trasformate in Route.

ConfigMap

Il ConfigMap è un oggetto usato per archiviare dati non riservati, quindi non password o token. Al suo interno è possibile inserire parte di codice per la configurazione delle applicazioni, elenchi di variabili d'ambiente o argomenti della riga di comando. Tutte queste tipologie di ConfigMap sono utilizzabili dai Pod. Quest'oggetto richiede dati scritti sotto forma coppia chiave-valore. Lo scopo principale è mantenere i dati di configurazione separati dal codice dell'applicazione.

Secret

Un Secret è un oggetto simile al ConfigMap ma serve esclusivamente per memorizzare dati sensibili, come password, token o chiavi. L'utilizzo di un Secret è molto consigliato perché in questo modo non è necessario includere dati riservati nel codice dell'applicazione.

StorageClass

Uno StorageClass identifica un modo per descrivere il tipo di storage da utilizzare ed è modificabile dagli amministratori. I diversi tipi di classi possono distinguersi per i criteri di backup, qualità del servizio o velocità del tipo di storage. Ciascun

StorageClass ha un provisioner che determina quale plugin di volume viene utilizzato per il provisioning dei PersistenVolume; ci sono plugin per ogni tecnologia offerta dai cloud provider oppure on-premise.

PersistentVolume

Un PV è una risorsa che mette a disposizione una memoria su cui è possibile salvare dati. Si chiama persistente perché i dati salvati al suo interno non vengono distrutti alla terminazione di un Pod a cui è stato legato il PV in questione, perciò si può affermare che il PersistentVolume ha una vita indipendente dai Pod. Questa risorsa può essere creata direttamente facendo riferimento ad uno StorageClass oppure dinamicamente.

PersistentVolumeClaim

Il PVC è la richiesta, eseguita da un utente, per la creazione di uno storage in modo dinamico. Appena l'utente richiede la creazione di un Pod legato ad un determinato PVC, viene creata la risorsa persistente necessaria: si tratta di *dynamic provisioning*. Il PersistenVolumeClaim può includere specifiche sulla dimensione della memoria, sul tipo di StorageClass e sulla modalità di accesso.

CustomResourceDefinition

Come Kubernetes, OpenShift è molto configurabile e permette di estendere le sue funzionalità, ad esempio tramite le CustomResourceDefinition. La definizione di un oggetto CRD comporta la creazione di una nuova risorsa con un nome e uno schema personalizzato. Le API del sistema gestiscono l'archiviazione della risorsa personalizzata.

2.4 Disaster recovery

Il DR è il processo che rende possibile il recupero di dati in caso di disastro. Questa definizione, anche se apparentamene completa, non è sufficiente per definire bene il concetto; è necessario infatti descrivere cosa si intende per disastro e per recupero di dati.

Un disastro può presentarsi in varie forme. Di seguito sono elencate quelle identificate dal libro *Building Secure & Reliable Systems: Best Practices for Designing, Implementing and Maintaining Systems* [7].

- **Disastri naturali**, come terremoti, inondazioni e incendi. Possono recare diverse tipologie di problemi ad un sistema.
- **Disastri sull'infrastruttura**, come guasti ai componenti o configurazioni errate, non sono sempre facilmente diagnosticabili ed il loro impatto può variare.
- **Interruzioni di servizi o prodotti**, osservabili dai clienti.
- **Degrado dei servizi** in esecuzioni vicino ad una soglia di diverso genere. A volte sono difficili da identificare.
- **Un attaccante esterno**, che potrebbe ottenere un accesso non autorizzato per un periodo di tempo prolungato prima di essere identificato.
- **Divulgazione non autorizzata di dati sensibili**.
- **Vulnerabilità di sicurezza urgenti**, che richiedono l'applicazione immediata di patch e se non risolte tempestivamente aumentano il rischio di attacchi esterni.

La conoscenza di questi tipi di disastri aiuta a pianificare buone procedure, che a loro volta permettono di risolvere i disservizi.

Per quanto riguarda il recupero di dati si intende la possibilità parziale o totale di mantenere invariati i dati ad un istante prima del disastro. Il termine *dati* identifica tutte le informazioni salvate necessarie per l'utilizzo dei servizi, i servizi stessi e tutti i volumi persistenti su cui sono salvati informazioni di qualunque genere.

Nella figura 2.5 si possono analizzare le fasi generali che avvengono subito dopo un disastro; esse possono variare in base alle necessità. Inoltre, questa figura evidenzia l'importanza di un buon piano di disaster recevery per poter rispondere in maniera efficace al disastro.

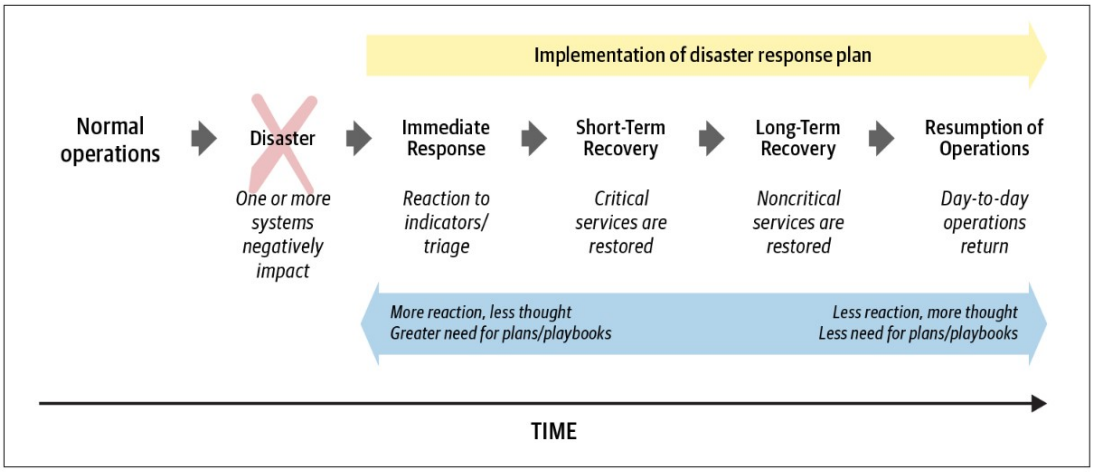

Figura 2.5: Fasi di risposta al disaster recovery [7].

2.4.1 Il piano di disaster recovery

Il *disaster recovery plan* è un piano di procedure (che possono essere scritte in un documento ufficiale) create appositamente per permettere di recuperare i dati e i servizi nel caso in cui si verificasse un disastro ad un'organizzazione. Questo piano include il DR e aggiunge tutto il contorno che permette all'azienda di poter portare con successo il ripristino in base alle proprie esigenze.

Quando si sviluppa una strategia di DR si devono affrontare diverse analisi. Alcune delle seguenti procedure sono state acquisite dal libro *Building Secure & Reliable Systems: Best Practices for Designing, Implementing and Maintaining Systems* [7].

- Analisi dei rischi e dei potenziali disastri che potrebbero interessare l'organizzazione o avere un impatto significativo.
- Identificazione di un team per il DR.
- Creazione di un piano e di un elenco di procedure dettagliate.
- Configurazione del sistema in modo appropriato.
- Configurazione del sistema di ripristino, se necessario.
- Prova ad applicazione delle procedure e dei sistemi configurati.
- Collezione di feedback dai test eseguiti e dalle valutazioni.
- Mantenimento di tutte le procedure e sistemi del piano di DR.

La valutazione dei rischi ha un ruolo fondamentale perché ogni piano di DR deve essere creato in base alle necessità e alla struttura dell'organizzazione. L'analisi deve identificare tutti i sistemi che operano per il mantenimento dell'azienda e le relative criticità, tutte le risorse, sia tecnologiche che umane, e tutti i possibili disastri che possono accadere. Dopo aver identificato un team di persone che si occupano del DR è necessario stabilire le tempistiche relative per il ripristino e i dati necessari. Per queste analisi si fa riferimento ad alcuni indici d'importanza cruciale per l'identificazione della tecnologia da utilizzare e del processo di DR da applicare [8].

RPO

Il *Recovery Point Objective* determina la quantità massima accettabile di perdita di dati misurata nel tempo. Questo parametro indica i dati sotto forma di tempo, più si richiede un tempo inferiore e meno saranno i dati persi. Ad esempio un'azienda potrebbe tollerare al massimo 30 minuti di dati; se invece richiede un RPO di 0 significa che non ci saranno perdite di dati dopo il ripristino.

RTO

Il *Recovery Time Objective* indica la quantità massima tollerabile per poter rendere di nuovo attivi tutti i servizi critici. Questo indice è strettamente correlato al tempo necessario che le procedure di DR richiedono per completare il ripristino; in questo periodo di tempo i servizi non saranno disponibili. Ogni organizzazione ha spesso più di un servizio attivo, perciò si è soliti stabilire un RTO per ogni applicazione, analizzando la valutazione dei rischi.

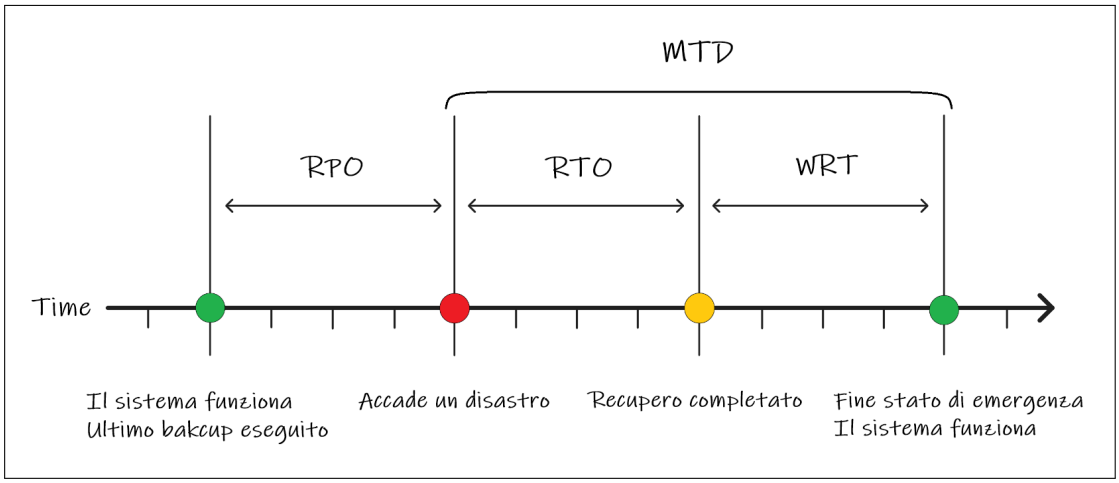

Figura 2.6: Indici KPI su base temporale

WRT

Il *Work Recovery Time* è il tempo massimo accettabile per la verifica di integrità del sistema e dei dati. In questo periodo di tempo, ad esempio, potrebbero essere in esecuzione dei controlli di database o verifiche sulla corretta ripresa dei servizi. Quando tutti i sistemi interessati dal disastro vengono ripristinati completamente e verificati, lo stato di emergenza terminerà con successo.

MTD

Il *Maximum Tolerable Downtime* indica il tempo massimo tollerabile di inattività di un servizio aziendale. Questo parametro si definisce come la somma di RTO e WRT. Il MTD deve essere definito per ciascun servizio di un'azienda e si basa sul suo tempo di interruzione massimo senza causare conseguenze inaccettabili.

2.5 Business continuity

Con il termine business continuity si identificano tutti i processi che una particolare azienda definisce per assicurare resilienza alla struttura e ai suoi sistemi e per individuare tutte le potenziali minacce [9]. Queste procedure hanno quindi come obiettivo la continua operatività dei servizi, anche in caso di incidenti o disastri, difendendo gli interessi e l'immagine dell'azienda. Per tali motivi la business continuity è di fondamentale importanza per le aziende che vogliono avere uno "0 downtime", preservare il fatturato e la reputazione aziendale. Inoltre, queste caratteristiche portano ad un maggior senso di sicurezza da parte dei dipendenti.

2.5.1 La gestione della business continuity

Il *Business continuity management* è un processo gestionale la cui funzione è identificare le possibili minacce per un'organizzazione, valutare quali potrebbero essere le conseguenze ed individuare le contromisure più adeguate [9]. In generale, quando si parla di *Business Continuity Management* ci si riferisce sia alla pianificazione che alla gestione delle procedure. La gestione della business continuity è regolata da una serie di norme ISO.

Una parte fondamentale della gestione è il *Business Continuity Plan* che viene stabilito al fine di concretizzare la richiesta di continuità operativa. Le strategie possono essere di vario tipo perché dipendono dalla dimensione dell'organizzazione e delle tecnologie adottate.
Di seguito sono riportate le fasi principali per la creazione di un piano di business continuity [10]:

1. **Identificare gli obiettivi del piano** che dipendono dalle aree aziendali coinvolte; possono essere più o meno stringenti in base alle esigenze del business.

2. **Formare i referenti** necessari per definire il piano e mantenerlo attivo.

3. **Identificare le funzioni critiche dell'azienda**, cioè quelle la cui assenza provocherebbe maggiori danni all'azienda.

4. **Eseguire un'analisi del rischio** per individuare le conseguenze di ogni tipo di evento in termini di: perdita di fatturato, perdita di dati, aumento di insoddisfazione dei clienti e danni alla reputazione dell'azienda.

5. **Creare un piano operativo** che include le strategie di prevenzione del rischio, la gestione del rischio e il superamento dell'emergenza fino al ripristino della continuità del business. Ad esempio, tra le strategie di prevenzione del rischio si può includere l'utilizzo di strumenti di backup di dati.

6. **Condurre una revisione periodica** è molto importante poiché l'azienda può cambiare dimensione e infrastruttura, pertanto si consiglia di aggiornare continuamente le procedure.

Disaster recevery e business continuity a confronto

Il concetto di BC può confondersi con quello di DR, anche se hanno fili logici in comune, sono due piani creati con scopi diversi. Un piano di business continuity prevede una continuità dei sistemi dell'azienda, anche durante il verificarsi di un disastro. Mentre il DR è specializzato nel recupero di dati e nella definizione di tempistiche per poterlo eseguire. Questi due concetti sono in parte legati: il disaster recovery si può definire come un sottoinsieme della business continuity; infatti in un piano di continuità del business si possono anche definire delle strategie per poter recuperare i dati in caso di grave disastro e poter ripristinare l'operatività dei servizi.

Disaster recovery plan	**Business continuity plan**
Procedure pianificate da utilizzare in caso di disastro	Mitigazione dei disservizi di risorse e funzioni aziendali
Prevede un'analisi del rischio	Prevede un'analisi del rischio
Valuta gli indici di RPO, RTO, WRT e MTD	Valuta l'impatto negativo di un disservizio per ogni funzione aziendale
Si occupa principalmente di infrastrutture IT	Non è un processo esclusivamente IT, prende in considerazione tutti gli aspetti del business
Si può considerare datacentrico	Si occupa di tutto ciò che riguarda il business
Recupera dati da un disastro	Garantisce continuità alle operazioni critiche
Può migliorare la business continuity	Può includere un piano di disaster recovery
I costi sono direttamente proporzionali a quanto saranno ambiziosi gli indici valutati	I costi dipendono dalle quantità di risorse e operazioni che sono state identificate come critiche per il business

Tabella 2.1: Confronto tra i piani di disaster recovery e business continuity

Se si progetta un buon sistema di DR è possibile, in caso di disastro, avere vantaggi anche dal punto di vista della continuità del servizio, nonostante non sia stato fatto un piano di business continuity. Invece, se il progetto di recovery è più semplice e con tempistiche molto alte, allora la continuità del business sarà sicuramente penalizzata in caso di disastro.

Il DR è datacentrico, può essere effettuato tramite il cloud computing o con altre risorse locali. Può utilizzare strategie di backup remoto a basso costo oppure includere una copia completa dell'infrastruttura in un altro luogo. Le aziende si possono confrontare con diverse organizzazioni o servizi che si occupano di disaster recovery applicabili a diverse soluzioni tecnologiche. Tutti i servizi di DR, però, sono concentrati sulla protezione dei dati e sul loro recupero, più o meno veloce.

La business continuity ha un focus più ampio perché cerca di rendere i servizi disponibili anche in caso di disastro. La BC non si occupa solo di casi estremi ma di tutti quei disservizi che possono portare a perdite di fatturato o dell'immagine dell'azienda. A differenza del DR non ci sono servizi disponibili che offrono un pacchetto completo, la business continuity si può ottenere seguendo tutte le linee guida delle norme ISO, creando e personalizzando le procedure per i requisiti dell'azienda.

La tabella 2.1 evidenzia le differenze tra il piano di disaster recovery e quello di business continuity.

Capitolo 3

Strategie per il backup di applicazioni cloud native

3.1 Introduzione

La diffusione e l'utilizzo del cloud è ormai presente in ogni realtà aziendale, così come le applicazioni in esecuzione, che non sono più solamente migrate in cloud, ma vengono sviluppate appositamente per la nuova tecnologia a microservizi.

Nonostante OpenShift permetta di avere applicazioni portabili, leggere, scalabili e resilienti, non è un sistema immune da possibili perdite di dati. Per cui è necessario, come per ogni soluzione tecnologica, studiare un sistema di backup per la protezione dei dati. Il nuovo modo di creare e gestire le applicazioni a microservizi ha dato origine a nuove sfide per la protezione dei dati in quanto non è possibile riutilizzare i vecchi sistemi di backup con il nuovo ecosistema cloud native.

3.2 Differenze con le tecnologie legacy

Il cloud native si distingue dalle tecnologie precedenti e allora stesso modo i sistemi di backup devono seguire i nuovi paradigmi delle applicazioni a microservizi.

I backup tradizionali sono spesso implementati a livello di macchina virtuale perché le applicazioni sono legate a ciascuna VM. Infatti si può avere una singola applicazione installata su una singola VM, in questo modo il backup delle virtual machine risulta efficace e chiaro. Tale sistema è chiamato snapshot e ha la caratteristica di preservare lo stato e i dati di una VM in un determinato momento. Dato che l'applicazione rimane in esecuzione potrebbe modificare dei file mentre il sistema di backup li sta ancora processando, per questo motivo sono state create diverse strategie che risolvono il problema.

Le applicazioni cloud native utilizzano i container che possono essere distrutti o ricreati in base alle necessità dalle piattaforme di orchestrazione. Si ricorda che un Pod può contenere uno o più container e ogni macchina virtuale ha in esecuzione tanti Pod. Inoltre un'applicazione è spesso distribuita su molte VM.

Se si considera un sistema a microservizi, il backup di una macchina virtuale potrebbe non racchiudere un'intera applicazione poiché questa potrebbe essere distribuita in diverse VM. Allo stesso tempo il backup considerato includerà molti altri container non richiesti che andranno ad influenzare la dimensione e le tempistiche di esecuzione dello snapshot.

Dopo queste considerazioni si può dedurre che i sistemi di backup tradizionali non sono più sufficienti per le applicazioni cloud native; ciò che non è cambiato sono i requisiti relativi al backup ed ai suoi obiettivi. Ad esempio, il libro *Kubernetes Backup & Recovery* [11] cita una delle regole in grado di affrontare efficacemente qualsiasi scenario: la regola di backup 3-2-1. Questo approccio aiuta ad identificare quanti file di backup bisogna avere e in quale luogo sono da archiviare. La regola definisce tre aspetti, illustrati anche nella figura 3.1:

- **3**: Possedere almeno tre copie dei dati per garantire che non ci sia un *single point of failure*.
- **2**: Conservare le copie su due supporti diversi, ad esempio un sistema di archiviazione a dischi solidi ed uno a nastro.
- **1**: Mantenere una copia del backup offsite, cioè lontano dal luogo dove i dati sono utilizzati dal servizio.

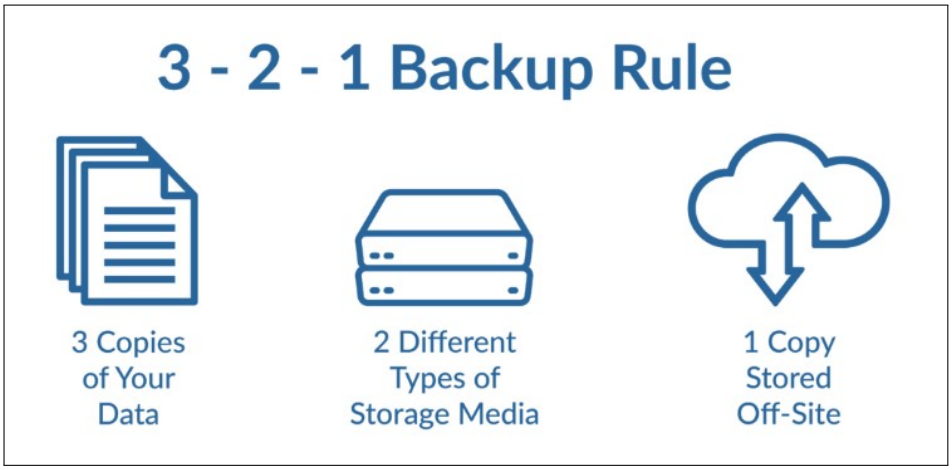

Figura 3.1: Regola di backup 3-2-1 [12]

3.2.1 Verso una soluzione

OpenShift è uno strumento potente in quanto in grado di mantenere le applicazioni in esecuzione anche con interruzioni parziali dell'infrastruttura. La tolleranza ai guasti è un grande punto di forza di questa tecnologia, ma è importante distinguere i concetti di resilienza e di protezione dei dati.

L'alta disponibilità (o *high availability*) e la replica delle risorse (o *replication*) sono due ottime caratteristiche di OpenShift, ma esse non garantiscono la protezione dei dati. C'è sempre il rischio di una perdita dati in qualunque tecnologia utilizzata, che può essere dovuta anche da un'azione umana volontaria o involontaria. In ultimo, l'installazione di OpenShift sul cloud non garantisce la resilienza dei dati, che potrebbero essere persi o corrotti.

Per risolvere il problema del backup nell'ecosistema delle applicazione cloud native, bisogna studiare una soluzione in grado di:

- capire e distinguere i container;
- riconoscere i progetti (namespace);
- focalizzarsi sulle applicazioni e non sull'infrastruttura;
- salvare i dati e le configurazioni delle applicazioni;
- identificare i diversi tipi di risorse;
- offrire un'ampia scelta sulla posizione dei dati di backup.

Poiché OpenShift è un orchestratore di container, anche il sistema di backup deve essere in grado di riconoscerli, anziché salvare tutto ciò che è contenuto in una VM. In questo modo è possibile salvare i singoli Pod o gruppi di Pod, identificandoli ad esempio tramite le *label*.

I progetti (namespace) sono essenziali per distinguere le diverse applicazioni in esecuzione o i vari gruppi di lavoro. Una soluzione di backup deve saper selezionare i progetti in modo tale da poterli salvare singolarmente.

3.3 Componenti coinvolti

Nella figura 3.2 si possono vedere alcuni dei principali componenti di Kubernetes, che connessi tra loro danno vita ad un'applicazione. Il compito di un sistema di backup non è banale perché ci sono tanti tipi di componenti e allo stesso tempo un'applicazione può essere formata da centinaia di essi.

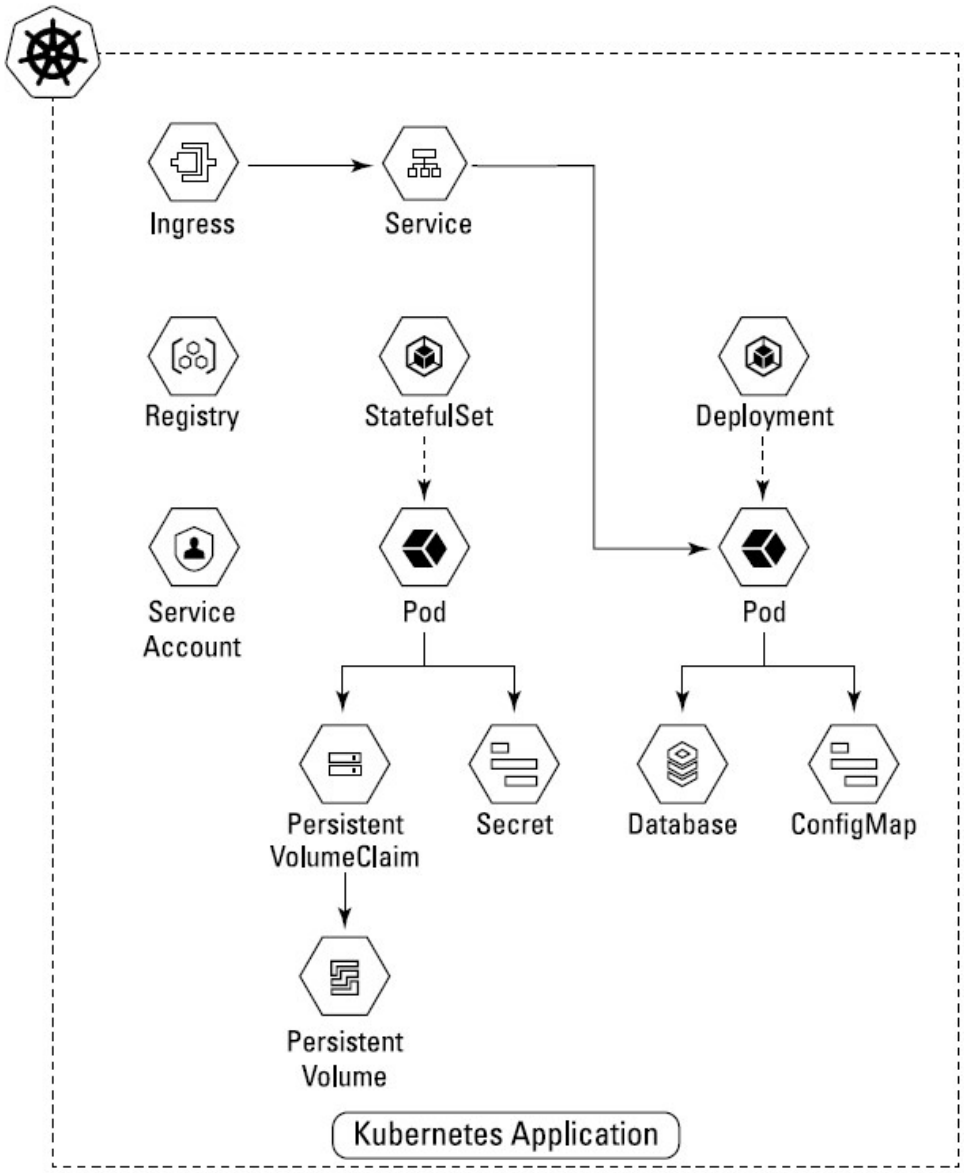

Figura 3.2: Esempio di applicazione Kubernetes con alcuni dei suoi componenti [11]

Il primo step da compiere per comprendere una soluzione di backup cloud native è la distinzione tra storage e applicazioni. I componenti all'interno di una tecnologia a microservizi si possono distinguere in due categorie identificabili tramite questi termini: risorse e dati.

3.3.1 Risorse

In questa tesi si definisce "risorsa" un oggetto presente all'interno di un cluster. É possibile creare le risorse tramite i *manifest file*, cioè dei file contenenti le specifiche di un oggetto Kubernetes attraverso una lista di *key:value* in formato YAML o JSON. Inoltre, è possibile anche visualizzare lo stato delle risorse già create nei formati precedentemente descritti. Si può notare come la figura 3.2 racchiuda tutte risorse.
Un esempio di risorsa è il Deployment di una semplice applicazione, di seguito viene riportata la sua descrizione:

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: hello-openshift
spec:
  replicas: 1
  selector:
    matchLabels:
      app: hello-openshift
  template:
    metadata:
      labels:
        app: hello-openshift
    spec:
      containers:
      - name: hello-openshift
        image: openshift/hello-openshift:latest
        ports:
        - containerPort: 80
```

Il *manifest file* appena descritto è salvato con il nome `hello-openshift.yaml` ed è inviato al cluster con il seguente comando:

```
oc apply -f hello-openshift.yaml
```

OpenShift provvederà alla creazione della risorsa richiesta e aggiungerà:

- I parametri omessi con i loro valori di default. Alcuni di essi sono utilizzati per l'identificazione della risorsa (es: *uid*), altri corrispondono a caratteristiche aggiuntive. (es: *imagePullPolicy*).
- Una sezione all'interno della descrizione, chiamata *status*, in cui è specificato lo stato della risorsa in ogni istante.

```
status:
  availableReplicas: 1
  replicas: 1
```

- Alcune risorse aggiuntive per poter rendere disponibile il Deployment, ad esempio i Pod necessari e un ReplicaSet.

I tipi di oggetti descritti nel capitolo 2.3.3 sono alcuni esempi di risorse. In un cluster possono coesistere centinaia di tipi diversi, inoltre è possibile crearne altri tramite le CustomResourceDefinition.

Alcune risorse richiedono particolare attenzione:

- PV e PVC non contengono i dati ma solo i riferimenti di come raggiungere un determinato volume persistente, ciò giustifica la necessità di considerare i dati come un'entità separata.
- Machine, MachineConfig e Node descrivono dei componenti dell'infrastruttura e sono unici nel cluster su cui sono definiti.

3.3.2 Dati

In questa tesi è utilizzato il termine "dati" per indicare il contenuto di un PersistentVolume. Così facendo si può distinguere chiaramente tutto ciò che riguarda la configurazione delle applicazioni ed i loro dati che vengono salvati in modo persistente.

In un cluster ci sono anche altri tipi di dati, essi però non sono considerati come tali, si tratta delle risorse di ConfigMap e Secret. Queste infatti non indicano un componente esterno su cui salvare le informazioni, ma le contengono direttamente al loro interno, perciò sono considerate risorse e non dati. Un oggetto di tipo ConfigMap archivia informazioni, come il codice, per configurare applicazioni, elenchi di variabili d'ambiente o argomenti della riga di comando; mentre un Secret contiene al suo interno dati sensibili.

Un utente del cluster, ad esempio uno sviluppatore, per poter utilizzare uno storage persistente nella sua applicazione deve ricorrere alla creazione di un PVC. Un PersistentVolumeClaim è un tipo di risorsa specifica per un namespace; invece, un PersistentVolume è condiviso nel cluster perché ognuno di essi può essere utilizzato da qualsiasi progetto. A seguito del legame tra un PV ed un PVC, il PersistentVolume in questione non potrà essere più legato ad altri PVC. Questa distinzione aiuta anche a separare i ruoli e le responsabilità: gli sviluppatori possono usare il PVC per richiedere risorse PV senza avere una conoscenza specifica dell'infrastruttura di storage sottostante; mentre la gestione dei PersistentVolume è affidata all'amministratore del cluster. Una descrizione grafica di questo processo è rappresentata in figura 3.3.

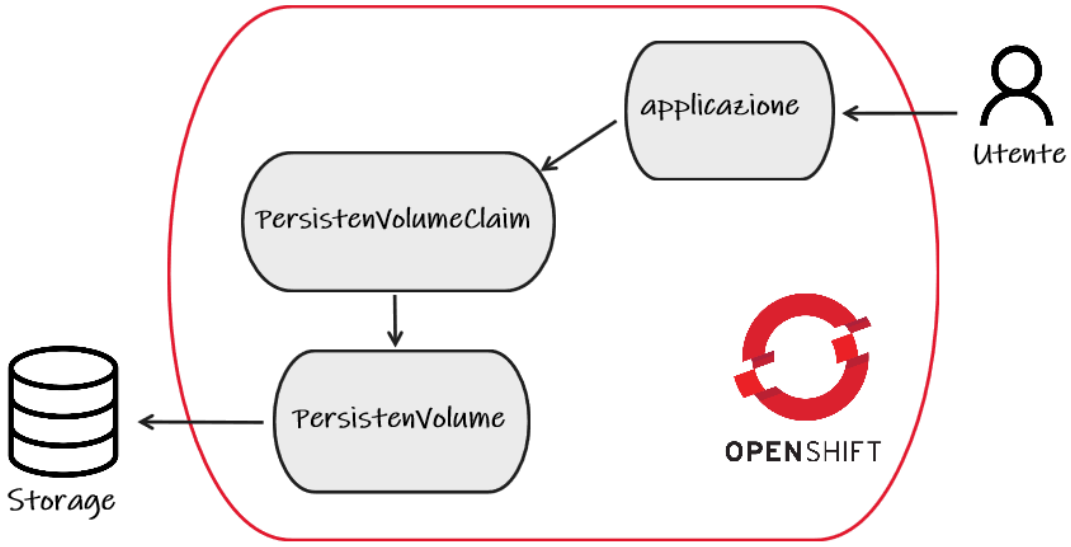

Figura 3.3: Processo logico per l'utilizzo di uno storage persistente

Un PV può essere creato in modo statico dall'amministratore o in modo dinamico utilizzando un oggetto StorageClass. Queste due modalità permettono allo sviluppatore di poter usare una risorsa di storage già definita oppure crearla appositamente con le caratteristiche specificate, ad esempio con la dimensione richiesta.

Lo storage persistente può essere quindi utilizzato dai Pod tramite una richiesta PVC. Il cluster ispeziona la richiesta per trovare il volume associato e legarlo al Pod richiedente.

I due criteri utilizzabili per la ricerca o per la creazione di un PV sono: la modalità di accesso e la dimensione. La modalità di accesso dipende dalla tecnologia utilizzata e può essere anche ridotta di capacità dall'amministratore; il PVC può

richiederne una specifica. Per entrambi i criteri il cluster potrebbe concedere di più di quanto richiesto, ma mai di meno; sia per quanto riguarda la dimensione, sia per le modalità di accesso elencate più avanti.
Di seguito è riportato un esempio di PVC:

```
1  apiVersion: v1
2  kind: PersistentVolumeClaim
3  metadata:
4    name: example-pv-claim
5  spec:
6    storageClassName: gp2
7    accessModes:
8      - ReadWriteOnce
9    resources:
10     requests:
11       storage: 3Gi
```

Questo PVC, di nome *example-pv-claim*, richiede un volume persistente di 3 GiB tramite lo StorageClass *gp2*. Quest'ultima caratteristica specifica la tecnologia che verrà utilizzata, in tal caso il nome suggerisce un tipo di Elastic Block Store di AWS. La specifica di StorageClassName è una richiesta opzionale che può essere omessa se è presente una risorsa di default. Viene inoltre richiesto una modalità di accesso specifica di tipo *ReadWriteOnce*.

Le modalità di accesso ad un PV sono di tre tipi:

- *ReadWriteOnce*: il volume può essere "montato" in lettura e scrittura da un singolo nodo. Questa modalità può anche consentire a più Pod di accedere al volume quando essi sono in esecuzione sullo stesso nodo.
- *ReadOnlyMany*: il volume può essere "montato" in sola lettura da tanti nodi.
- *ReadWriteMany*: il volume può essere "montato" in lettura e scrittura da tanti nodi.

Una volta creato il PVC, è possibile legare il volume persistente ad un Pod specificando la richiesta all'interno del suo *manifest file*. Di seguito è riportato un esempio di Pod che richiede un volume tramite il PVC *example-pv-claim*.

```
1  kind: Pod
2  apiVersion: v1
```

```
3  metadata:
4    name: example-pv-pod
5  spec:
6    containers:
7      - name: myfrontend
8        image: dockerfile/nginx
9        volumeMounts:
10       - mountPath: "/var/www/html"
11         name: example-pv
12   volumes:
13     - name: example-pv
14       persistentVolumeClaim:
15         claimName: example-pv-claim
```

I volumi persistenti possono essere di diverso tipo, si distinguono per la tecnologia utilizzata ed alcuni si affidano alle risorse dei cloud provider. Ora sono elencati alcuni volumi supportati da OpenShift [6].

Dischi forniti dai cloud provider

Si possono utilizzare i tipi di archiviazione che offrono i cloud provider, accessibili direttamente dalle macchine virtuali. Il servizio è un sistema scalabile ad alte prestazioni che è progettato per essere fruibile direttamente dalle VM del cloud provider. Esso offre la possibilità di scegliere il sistema fisico su cui salvare i dati in base alle proprie necessità, ad esempio su dischi rigidi oppure solidi. Inoltre, alcuni cloud provider permettono anche di scegliere le specifiche sulla velocità effettiva, sulle operazioni di I/O al secondo e sulla dimensione massima allocabile.
Ci sono però delle restrizioni importanti durante l'utilizzo del servizio:

- I nodi su cui sono in esecuzione i Pod, che richiedono questo tipo di archiviazione dati, devono essere istanze di VM appartenenti allo stesso cloud provider.
- Le istanze devono oltretutto trovarsi nella stessa regione e zona di disponibilità del volume richiesto.
- Il tipo di volume supporta una singola istanza e non è possibile collegare più Pod ad esso, anche se si seleziona l'accesso in sola lettura [1].

[1] L'archiviazione offerta da Google consente l'utilizzo dello stesso volume a più Pod contemporaneamente in sola lettura

I cloud provider supportati da OpenShift che offrono questo tipo di servizio sono AWS, Azure e Google; i relativi servizi che propongono si chiamano in ordine: AWS Elastic Block Store, Azure Disk e GCE Persistent Disk.

NFS

Il Network File System è un protocollo di rete che consente ad una macchina virtuale di accedere ad un disco remoto.

Un volume NFS consente di collegare un'archiviazione di rete NFS esistente con un Pod. Questo tipo di volume permette l'accesso in lettura e scrittura contemporaneamente da più Pod.

Azure File

Azure File è un servizio di archiviazione offerto da Azure che permette di condividere facilmente i dati tra container accessibili con i protocolli NFS o SMB. Questo tipo di volume permette di condividere i dati tra più nodi e Pod contemporaneamente, anche in modalità di lettura e scrittura. Inoltre è possibile scegliere l'archiviazione tra dischi rigidi o solidi ad alte prestazioni.

iSCSI

Un volume iSCSI consente di "montare" un volume esistente con la tecnologia *SCSI over IP* nel Pod desiderato. Il protocollo iSCSI permette di inviare comandi a sistemi di memoria virtuali collegati attraverso la rete e consente di utilizzare un disco come se fosse legato direttamente alla macchina, anche se è situato su un dispositivo di archiviazione remoto. I protocolli di rete utilizzati sono TCP/IP.

I dati presenti in un volume iSCSI possono essere condivisi tra i Pod; infatti questo tipo di volume permette l'accesso contemporaneo in sola lettura da parte di tanti Pod. Tale caratteristica consente di precompilare un volume con i dati necessari e successivamente utilizzarlo in parallelo da tutti i Pod che erogano un servizio. Tuttavia, i volumi iSCSI possono essere montati in modalità di lettura e scrittura da un singolo consumatore, perciò non è ammesso l'accesso in scrittura simultaneo [13].

Fibre Channel

Il Fibre Channel è una tecnologia di archiviazione dati che, a differenza dello iSCSI, utilizza un protocollo dedicato ed una nuova interfaccia in grado di combinare gli aspetti sia della tecnologia SCSI che di quella Ethernet.

Questa tecnologia permette l'utilizzo di un volume Fibre Channel all'interno di un cluster condiviso tra diversi Pod in modalità sola lettura, ma non permette la scrittura simultanea.

HostPath e Local

Un volume HostPath rende disponibile un file o una cartella dal nodo host ad un Pod. Questo tipo di volume presenta molti rischi per la sicurezza e non è consigliato il suo utilizzo perché lega il nodo al Pod.

Un volume Local rappresenta un dispositivo di archiviazione locale come ad esempio un disco, una partizione o una cartella. In questo caso i volumi diventano portatili rispetto al tipo precedente, ma hanno molte limitazioni in quanto sono soggetti alla disponibilità del nodo sottostante.

Red Hat OpenShift Data Foundation

Red Hat OpenShift Data Foundation è uno storage persistente *software-defined* integrato e ottimizzato per OCP [14]. Si basa sulla tecnologia Ceph ed essendo integrato pianamente con OpenShift è possibile installarlo on-premise o nel cloud utilizzando i nodi del cluster.

Si tratta di un tipo di archiviazione *container-native* e supporta tre tipi di storage:

- **Archiviazione di file** condivisi per l'integrazione e l'aggregazione continua dei dati. I file sono salvati all'interno di cartelle e sottocartelle ed è necessario conoscere il percorso per poter accederci.
- **Archiviazione a blocchi** utilizzabile ad esempio per database. Essa divide i file in blocchi e li archivia come elementi separati.
- **Archiviazione ad oggetti** per backup o dati multimediali. I dati sono frammentati in unità distinte chiamate oggetti e salvati in un repository.

3.3.3 Etcd

Etcd è un sistema di archiviazione di tipo chiave-valore utilizzato da OpenShift per salvare lo stato di tutti gli oggetti del cluster [6].

L'etcd è progettato per resistere ai guasti perché è installato su più macchine, garantendo così l'alta disponibilità; infatti può ripristinarsi in automatico se una macchina dovesse essere indisponibile per un periodo temporaneo. Questa caratteristica viene garantita con alcune limitazioni e tollera al massimo $(N - 1) / 2$ guasti permanenti in un cluster con N membri. Con guasto permanente si intende un errore a livello hardware o altro che causa la completa perdita di funzionalità.

Nel caso in cui il cluster perdesse più di $(N - 1) / 2$ membri, l'etcd smetterà di funzionare e non riceverà nuovi aggiornamenti.

Red Hat consiglia di eseguire un backup dell'etcd per poter ripristinare un cluster in caso di disastro che può essere dovuto alla perdita di nodi oppure alla cancellazione di qualcosa di critico da parte di un amministratore. La funzione di backup è integrata nell'etcd che tramite un comando apposito crea un file snapshot contenente tutte le informazioni sugli oggetti del cluster. Inoltre, è possibile anche crittografare questo file per proteggere i dati sensibili.

Per eseguire il backup su OpenShift bisogna collegarsi ad una macchina del control plane tramite il comando:

```
oc debug node/<node-name>
```

ed eseguire lo script `cluster-backup.sh`, il quale ha il compito di creare due file:

- *static_kuberesources_<datetimestamp>.tar.gz* che contiene le risorse per i pod statici di tipo kube-apiserver, kube-controller-manager, kube-scheduler ed etcd.
- *snapshot_<datetimestamp>.db* che è lo snapshot dell'etcd.

Red Hat consiglia di eseguire il backup regolarmente e ogni volta che si aggiorna il cluster perché è possibile avviare un ripristino solo se la versione del backup corrisponde a quella del cluster. Inoltre è buona norma copiare il backup fuori dal nodo in cui è stato salvato. Di seguito è riportato un esempio di script che mantiene due copie di backup sul nodo e due copie in locale (cioè sulla macchina sulla quale viene eseguito). Questo script può essere aggiunto a *crontab*, con cui è possibile farlo eseguire automaticamente una volta a settimana.

```
#!/bin/bash
ssh core@<node_name> "sudo rm -r ./assets/old-backup/*"
ssh core@<node_name> "sudo mv ./assets/backup/* ./assets/old-backup"
ssh core@<node_name> "sudo /usr/local/bin/cluster-backup.sh ./assets/backup"
ssh core@<node_name> "sudo chown -R core:core ./assets"
rm -r cluster1-etcdbkp/old-backup/*
mv cluster1-etcdbkp/backup/* cluster1-etcdbkp/old-backup
scp -r core@<node_name>:./assets/backup/* cluster1-etcdbkp/backup
```

Si può estrarre l'elenco dei nodi appartenenti al control plane eseguendo questo comando:

```
oc get no -o jsonpath='{.items[?(@.spec.taints[0].key=="node-role
↪  .kubernetes.io/master")].status.addresses[?(@.type=="Hostname"
↪  )].address}'
```

Il restore dell'etcd si può applicare come forma di disaster recovery quando si presenta una di queste situazioni:

- Il cluster ha perso la maggior parte del control plane raggiungendo quindi il numero massimo di macchine non disponibili.
- Un amministratore ha eliminato qualcosa di critico che non può essere ripristinato automaticamente.

Se invece è ancora possibile riceve dei dati tramite i server API del cluster, allora l'etcd è disponibile ed è necessario risolvere il problema in altri modi, ad esempio riavviando tutte le istanze dei server API.

Il ripristino dell'etcd ha come conseguenza il ritorno indietro nel tempo del cluster e ciò può influire negativamente sulle risorse che pensano di trovare dei file che invece non sono più presenti. Questo è uno dei motivi per cui non sempre può andare a buon fine questo tipo di DR e quindi potrebbe richiedere delle azioni manuali.
I requisiti imposti da Red Hat sul ripristino dell'etcd sono:

- Una macchina del control plane funzionante da usare come host di ripristino.
- L'accesso SSH a questa macchina.
- I due file di backup dell'etcd devo essere presenti all'interno dell'host utilizzato per il ripristino.

Per la procedura di ripristino si può seguire la guida di riferimento [6]. In sintesi si tratta di eseguire lo script `cluster-restore.sh` presente all'interno di una macchina del control plane disponibile e successivamente richiedere una nuova distribuzione delle risorse di etcd, kubeapiserver, kubecontrollermanager e kubescheduler.

Questi due comandi sono stati creati per verificare che tutti i Pod in esecuzione sul cluster siano disponibili e senza errori.

```
oc get po -A -o jsonpath='{range .items[*]}[{.metadata.namespace}
↪  {"\t"}{.metadata.name}{"\t"}{.status.containerStatuses[*].state}
↪  ]{"\n"}{end}'| grep -v running | grep -v terminated
```

```
oc get po -A -o jsonpath='{range .items[*]}[{.metadata.namespace}
↪  {"\t"}{.metadata.name}{"\t"}{.status.phase}]{"\n"}{end}' | grep
↪  -v Running | grep -v Succeeded
```

Infine, si sottolinea che questo tipo di disaster recovery è strettamente legato alle risorse, non ripristina in alcun modo i dati ed è un metodo "distruttivo" che porta ha complicazioni da gestire successivamente, sopratutto se dopo l'ultimo backup effettuato ci sono state modifiche rilevanti al cluster. Per questi motivi si ritiene il backup dell'etcd necessario ma non sufficiente per la maggior parte delle richieste dei piani di DR aziendali.

3.3.4 Infrastruttura

Per un sistema di backup conoscere l'infrastruttura del cluster non è di fondamentale importanza, perché è esattamente una della caratteristiche che valorizza OpenShift, cioè la portabilità delle applicazioni e la separazione dal contesto fisico. Il backup di un'applicazione a microservizi si concentra sul salvataggio dei dati e delle configurazioni di risorse e non sulla macchina fisica o VM su cui è eseguita. Ciò evidenzia come OpenShift possa essere una piattaforma di cloud ibrido; grazie alla portabilità e alla migrazione dei servizi, il contesto fisico sul quale è installato il cluster è considerato in secondo piano.

Tutte queste riflessioni non sono sufficienti per poter trascurare completamente l'infrastruttura; infatti, durante la fase di progettazione di un piano di DR, il numero di nodi di un cluster e le loro disponibilità "giocano" un ruolo importante sulla possibilità di schedulazione dei Pod. Se si progetta di migrare dei servizi su un cluster con meno risorse di calcolo, i Pod potrebbero rimanere in stato di Pending senza poter essere eseguiti su un nodo.

Un'altra considerazione riguarda le installazioni di cluster in modalità *Installer-provisioned* e *User-provisioned* nelle quali si deve scegliere il tipo di macchine o VM da allocare. OpenShift mette delle limitazioni sul tipo di virtual machine e propone un elenco di istanze supportate. Queste si caratterizzano per l'architettura utilizzata e per il tipo di processore.
Dunque, all'interno di due cluster con lo stesso numero di nodi e lo stesso numero di vCPU si possono avere delle capacità di calcolo diverse. Per questo motivo, è necessario tenerne conto durante la creazione di un piano di DR che include una migrazione di servizi in un cluster diverso.

Una funzionalità esterna ad OpenShift è la possibilità di chiedere la creazione di un *load balancer*, cioè un sistema di bilanciamento del carico, al cloud provider su cui è installato il cluster. La richiesta avviene tramite l'applicazione di una risorsa *Service* di tipo *LoadBalancer*, mentre la creazione effettiva avviene in modo asincrono.
Questo componente non fa parte dell'infrastruttura di OpenShift ma, poiché è identificato da una risorsa, può essere sottoposto a backup tramite essa. Ci sono però delle limitazioni per quanto riguarda la sua migrazione in cloud provider diversi:

- È possibile specificare un *loadBalancerIP* per richiedere un IP specifico; questa opzione però non è supportata da tutti i cloud provider, perciò può essere rifiutata e in questo caso sarà utilizzato un IP temporaneo. Altri provider, per poter utilizzare questa funzione, richiedono preventivamente la creazione di una risorsa per allocare in modo statico un IP pubblico.
- Un'altra specifica che è possibile richiedere è il *loadBalancerSourceRanges*, si tratta di un metodo per limitare il traffico che accetta solo gli intervalli di indirizzi IP desiderati. Anch'esso è ignorato se il cloud provider non supporta questa opzione.
- Il protocollo richiesto deve essere supportato dal cloud provider, ad esempio non è detto che UDP lo sia da tutti i fornitori di servizi cloud.

Un altro componente richiesto al cloud provider è il tipo di *load balancer* legato alla risorsa *IngressController*. In AWS è possibile richiedere un *Classic Load Balancer* oppure un *Network Load Balancer*, questa distinzione, per ora, non è disponibile su altri cloud provider. Quindi, se il piano di DR necessita una migrazione di un cluster con un *IngressController* di questo tipo, allora bisogna evitare di spostare questa risorsa.

Un ultimo aspetto legato alla migrazione riguarda il tipo di archiviazione utilizzato; infatti un cluster a microservizi permette di richiedere la creazione di volumi al cloud provider utilizzando un StorageClass che risulta essere diverso per ogni fornitore di servizi cloud. Di conseguenza, anche se OpenShift si definisce come una piattaforma, la separazione dall'infrastruttura non è così marcata ed occorre considerare tutti gli aspetti in un piano di disaster recovery.

3.4 Metodologie

Alla creazione di un processo di disaster recovery è necessario trovare una soluzione per il backup delle risorse cloud. Oltre alla tecnologia da applicare, è fondamentale stabilire la metodologia opportuna in base alle necessità dell'organizzazione.

Per metodologia si intende il modo in cui viene applicato un piano di DR, quindi le modalità di backup e le procedure per il ripristino. Ciò influenza direttamente le tempistiche per il recupero dei servizi e il costo del sistema adottato.

Si possono identificare tre modalità principali per la disposizione di un secondo sito dedicato al DR: backup a freddo, tiepido e caldo [15]. La figura 3.4, oltre a metterle a confronto, evidenzia come il costo ed il tempo per il recupero siano strettamente legati. La freccia nera identifica i sistemi tradizionali, mentre quella rossa i sistemi di DR progettati sul cloud. Si può notare come quest'ultimi siano più convenienti e con un tempo di recupero inferiore.

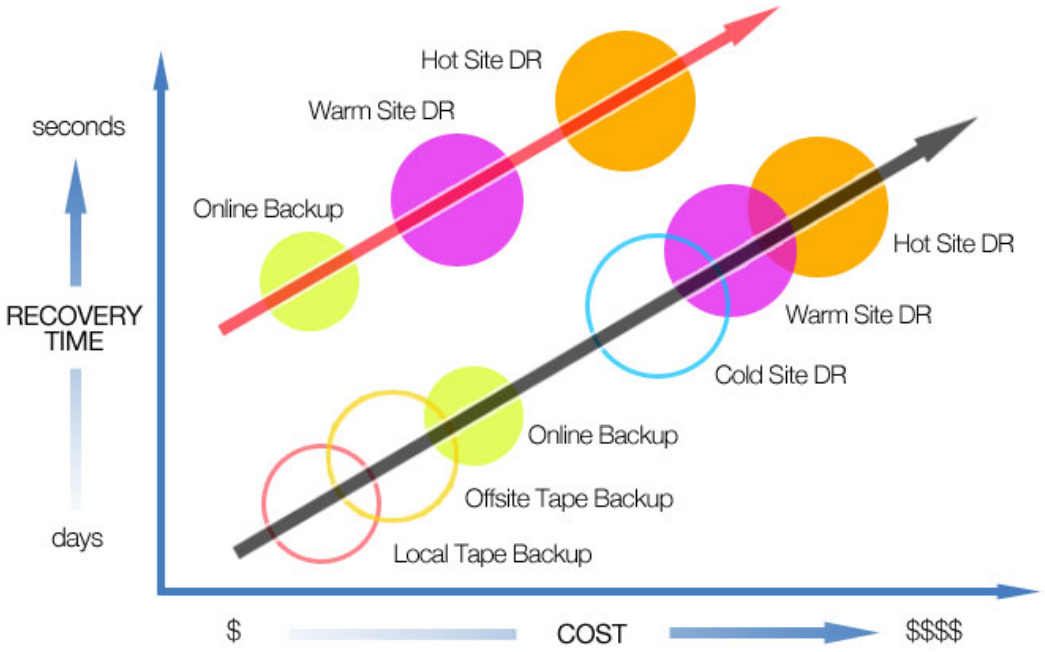

Figura 3.4: Trade-off tra costo e tempo di recupero [16]

Le metodologie citate nel paragrafo precedente si possono applicare anche su ambienti cloud native, di seguito sono elencate le differenze, i loro vantaggi ed il loro legame con le soluzioni di tipo sincrono ed asincrono.

3.4.1 Backup a freddo

Il backup a freddo, in inglese *cold backup*, è il livello più basso ed economico che permette di progettare un sito secondario incaricato del recupero dei servizi in caso di disastro. Con questo termine si fa riferimento ad un sistema spento o non allocato, che quindi ha bisogno di un tempo elevato per poter essere disponibile ad offrire un servizio. Se si applica questa modalità ad un cluster a microservizi significa che le macchine virtuali situate all'interno di un cloud provider possono, ad esempio, essere già allocate e pronte all'avvio.

Utilizzando tale modalità si può parlare di *asynchronous backup & restore* perché il ripristino dei servizi si basa sul recupero da un backup effettuato in precedenza ed avviene in un secondo momento.

Essendo la modalità più economica, è anche la più lenta ad eseguire la fase di ripristino; infatti, se il piano di business continuity richiede un'alta disponibilità, essa non potrà essere utilizzata con i sistemi *core* di un'organizzazione.

3.4.2 Backup a tiepido

Il *warm backup* identifica una modalità intermedia in cui il sistema di recupero è già allocato e pronto all'uso e potrebbe avere già in parte una configurazione del servizio da erogare. In uno scenario cloud native, dunque, si potrebbe gestire un cluster spento in un sito secondario ma già configurato e pronto ad essere eseguito. Inoltre è possibile applicare una soluzione di ripristino periodico che accende il cluster e aggiorna i dati e la loro configurazione dagli ultimi backup disponibili. Ciò consentirebbe di ridurre il tempo di recupero dal disastro e garantire un indice di RTO migliore.

Anche una soluzione di questo livello utilizza il modello asincrono e permette di avere un servizio attivo in poco tempo con un costo maggiore, dovuto all'allocazione permanente delle risorse e della loro configurazione.

3.4.3 Backup a caldo

Il backup a caldo (*hot backup*) consente di avere un sistema già configurato, attivo e pronto all'uso. Esso è in grado di ottenere un recupero molto veloce, garantisce poche perdite di dati ed è correlato ad un costo maggiore del servizio. In una strategia di DR a microservizi è richiesta la configurazione di un cluster secondario ed un sistema che possa sincronizzare i dati dal sito principale a quello di disaster recovery. Con questa soluzione è possibile avere, oltre ad un basso tempo di recupero, anche uno zero RPO, cioè nessuna perdita di dati tra il disastro ed il ripristino dei servizi. Pertanto, si può garantire una notevole continuità del business.

È possibile applicare un backup a caldo con un modello sincrono che permette di tenere sincronizzati i servizi, le configurazioni delle applicazioni ed i dati tra i due cluster: primario e secondario. Questo sistema solitamente non richiede un'archiviazione esterna per il backup, ma necessita che i due siti non siano troppo distanti per motivi di latenza. Infatti è importante sottolineare come tale soluzione sia applicabile solo con notevoli limitazioni geografiche, le quali possono variare in base alla rete e all'applicazione da proteggere. La figura 3.5 illustra un esempio di sistema sincrono gestito da uno strumento enterprise chiamato Portworx nella quale è possibile notare che anche tutto l'etcd è replicato. In questo modo tramite una piattaforma di gestione è possibile attivare il cluster di DR senza che venga percepito alcun cambiamento da parte dell'utente.

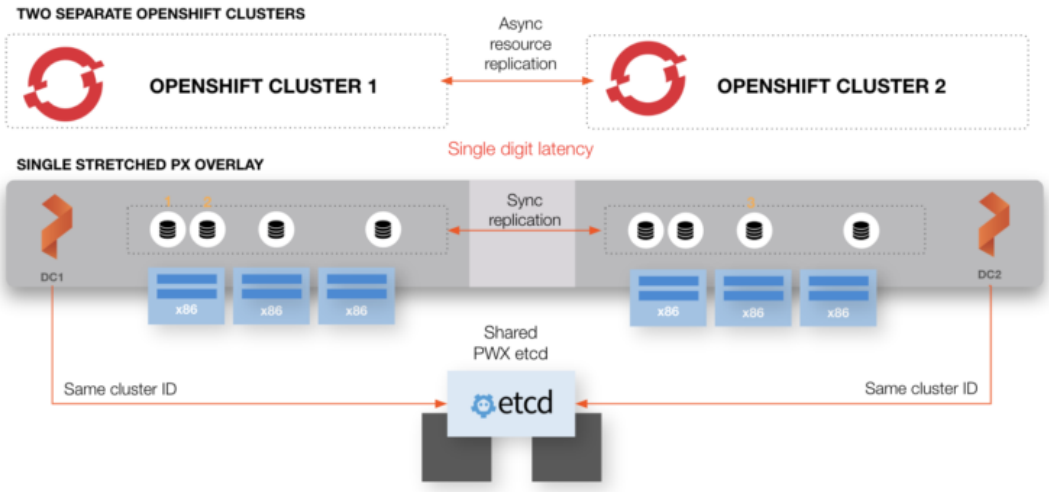

Figura 3.5: Esempio di backup a caldo sincrono con un cluster di DR [17]

3.5 Soluzioni disponibili

Un sistema di backup per applicazioni cloud native deve essere in grado di riconoscere e salvare tutte le risorse necessarie e deve riuscire a ripristinarle in modo trasparente, cioè l'utente non deve accorgersi se è stata cambiata infrastruttura o posizione geografica. Come è stato elencato nel capitolo 3.3 ci sono diversi concetti da prendere in considerazione, ma in sintesi si possono sintetizzare in due: risorse e dati.

Una buona soluzione di backup deve poter essere flessibile permettendo di salvare una selezione precisa di risorse, identificate da label o namespace, e allo stesso tempo deve poter anche permettere di migrare tutto il cluster. Inoltre, essa deve racchiudere il backup in un'unica unità permettendo il ripristino dell'applicazione da un backup desiderato.

3.5.1 Strumenti cloud native

Negli ultimi anni sono nati diversi strumenti che propongono soluzioni di backup di applicazioni a microservizi e che possono essere integrate nei piani di disaster recovery. Queste soluzioni si presentano come strumenti più o meno integrati al sistema di orchestrazione utilizzato, sono progetti open source o enterprise e richiedono un amministratore che li configuri.

Gli strumenti cloud native disponibili sono sistemi flessibili e utilizzabili in tutti i tipi di installazioni, dai cluster on-premise a quelli sul cloud. Offrono tutti la possibilità di selezionare le risorse da salvare e permettono la migrazione dell'interno cluster.

Dove salvare il backup

Un argomento critico di questi sistemi è la selezione della posizione dei file di backup. Gli strumenti cloud native propongono diverse soluzioni che l'amministratore può scegliere e configurare in base alle proprie necessità.

Per il tipo di utilizzo *backup & restore* è necessario selezionare dove si vuole salvare i file di backup, in genere ciò avviene attraverso alcuni servizi proposti dai cloud provider. Un esempio è il sistema di archiviazione ad oggetti che permette di accedere ai dati in modo veloce, sicuro ed accessibile da qualunque cluster che abbia l'accesso ad Internet. Il tipo di archiviazione ad oggetti è perfetto per l'utilizzo di file di backup ed è un servizio scalabile e personalizzabile. Inoltre, è possibile scegliere la posizione geografica di archiviazione permettendo di rispettare alcuni requisiti per la privacy e la latenza di accesso. La figura 3.6 propone una configurazione di backup all'interno di un cluster Kubernetes gestito da AWS, chiamato Elastic Kubernetes Service.

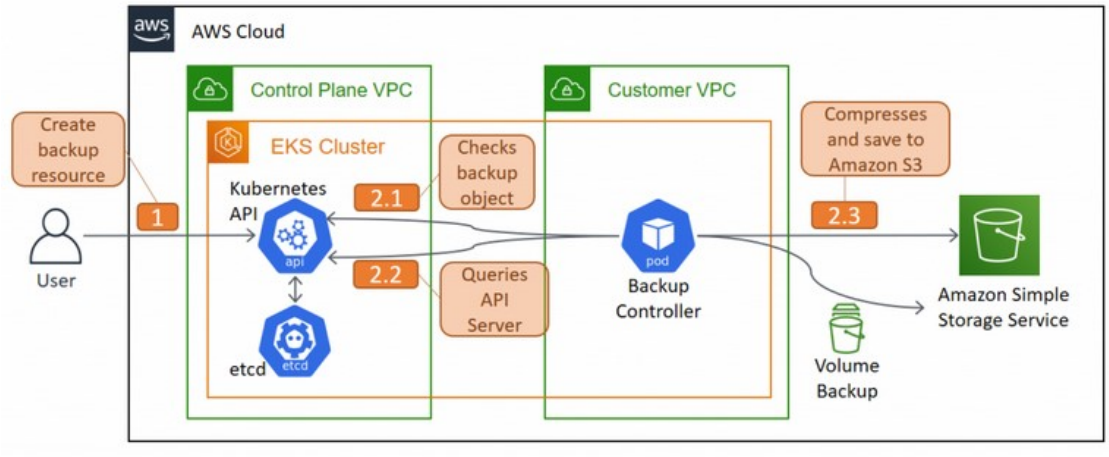

Figura 3.6: Esempio di uno strumento per il backup su Kubernetes [18]

Altre posizioni di salvataggio possono includere sistemi proposti dal cloud provider stesso che offrono un sistema di backup per le risorse di tipo Persistent Volume. I dischi sono direttamente gestiti già dal cloud provider in questione che può proporre un servizio per il backup dell'intera risorsa di archiviazione.

3.5.2 DRaaS

Di seguito si cita una definizione in inglese chiara e completa [19]:

"Disaster recovery as a service (DRaaS) is a cloud-based solution that third-party cloud providers offer to small and large businesses to ensure data protection, limit downtime, and shorten Recovery Point Objectives (RPOs) when a disaster happens"

Il servizio di DR è scelto dalle aziende che non voglio dedicare risorse per la creazione ed il mantenimento di un sistema di backup. Il contratto che i fornitori propongono è personalizzato in base alla dimensione dei dati, dei servizi da proteggere ed agli accordi stabiliti, come i tempi di RTO e RPO. Di conseguenza il fornitore del servizio è responsabile del processo di recupero dal disastro che può essere eseguito tramite una strategia basata su backup nel cloud oppure con una replica completa dei servizi e dati.

In ambienti cloud native anche questo tipo di soluzione può essere considerata nel piano di DR aziendale. Oltre a fornitori che propongono esclusivamente servizi di recupero da disastri e strategie di backup, ci sono anche i cloud provider che negli ultimi anni si sono evoluti anche su questi aspetti. Quindi, un'azienda che gestisce un cluster sul cloud, può sfruttare le offerte che il provider propone. Ad

esempio, Azure offre il servizio *Azure Site Recovery* che si occupa di replicare le macchine virtuali selezionate in un'altra regione, così facendo all'accadere di un disastro è possibile utilizzare il sito secondario. Tutta la gestione della replica è gestita da Azure e non influisce sui carichi di del sito principale.

Tramite il DraaS è possibile pagare un servizio che rispecchia le necessità dell'azienda e garantisce un recupero dei sistemi in tempi stabiliti dal contratto di *Service level agreement*.

Capitolo 4

Scenario preso in esame

4.1 Introduzione

In questa tesi si è preso in considerazione la tipologia di *backup & restore* perché permette una strategia di DR efficace e allo stesso tempo accessibile a tutte le aziende che hanno necessità di proteggere i dati del loro cluster. Oltre alla creazione di un backup, è possibile usufruire di questo servizio per migrare applicazioni verso altri cluster o per ripristinare i dati ad uno stato precedente.
I cloud provider utilizzati sono: AWS e Azure.

Si ricorda inoltre che è possibile implementare un sistema di backup su strategie di DR a freddo e a tiepido, rendendo quindi molto ampio lo spettro di organizzazioni che possono beneficiare di questo studio.

In questo capitolo tratterà dei tipi di applicazioni selezionate; successivamente sarà presentata la soluzione tecnologica adottata per far fronte al problema del DR su cluster OpenShift. A seguire si mostreranno tutte le prove effettuate con le relative tempistiche, ciò porterà verso una scelta corretta del sistema di DR da adottare, in base alle necessità di ciascuna azienda. In ultimo, si analizzerà il costo della soluzione in base alla quantità di dati da sottoporre a backup.

4.2 Applicazioni cloud native considerate

In un ambiente a microservizi è possibile eseguire applicazioni di vario genere, sviluppate appositamente per un contesto scalabile. Anche se le applicazioni possono avere scopi diversi si possono tutte condurre a due tipologie: stateless e stateful. Nello studio di questa tesi sono state considerate entrambe per far fronte a tutti i possibili scenari di DR.

Di seguito sono presentate le due tipologie e le applicazioni di riferimento scelte per lo studio.

4.2.1 Stateless

Un'applicazione stateless si considera come in una condizione di isolamento, dove non esistono dati memorizzati o riferimenti passati [20]. Un esempio di questo tipo può essere un server che rende disponibile una pagina web o una ricerca online.

Nello studio di tesi è stata usata un'applicazione di esempio sviluppata da Google e citata all'interno della documentazione di Kubernetes [21]. Il Deployment che richiede la creazione di questa applicazione è applicato all'interno di un progetto di nome *hello* ed è così definito:

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    app.kubernetes.io/name: load-balancer-example
  name: hello-world
spec:
  replicas: 5
  selector:
    matchLabels:
      app.kubernetes.io/name: load-balancer-example
  template:
    metadata:
      labels:
        app.kubernetes.io/name: load-balancer-example
    spec:
      containers:
      - image: gcr.io/google-samples/node-hello:1.0
        name: hello-world
        ports:
        - containerPort: 8080
```

Successivamente si invia il comando per la creazione del servizio di tipo LoadBalancer:

```
oc expose deploy hello-world --type=LoadBalancer --name=my-service
```

il quale richiederà al cloud provider una risorsa per poter raggiungere l'applicazione dall'esterno.

4.2.2 Stateful

Le applicazioni *stateful* necessitano di uno spazio di archiviazione su cui salvare lo stato corrente o modificare dei dati specifici come tabelle di database. Esse rappresentano la maggior parte delle applicazioni utilizzate in contesti enterprise.

Di questo argomento ne parla chiaramente Red Hat di cui si citano le parole [20].

"Inizialmente i container erano progettati per essere stateless, uno stato adatto alle loro caratteristiche intrinseche di portabilità e flessibilità. Alla crescente diffusione del loro impiego è corrisposta una maggiore containerizzazione delle app stateful esistenti, con la riprogettazione e la creazione di nuovi pacchetti per agevolare l'esecuzione da container."

Pertanto, in un ambiente a microservizi, ogni volta che un'applicazione necessita di utilizzare un volume persistente utilizza un approccio stateful.

mysql

Lo scenario stateful considerato in questo studio utilizza un'applicazione di database a singola istanza alla quale si è collegato un PersistentVolume. Questo esempio viene utilizzato all'interno della documentazione di Kubernetes [22].

Nel sottocapitolo 3.3.2 sono elencati i diversi tipi di PV, in questa tesi si è deciso di considerare i dischi offerti dal cloud provider perché sono la soluzione più comune ed economica per l'applicazione considerata.

Il cluster utilizzato è stato installato in modalità *installer-provisioned* su AWS e successivamente anche su Azure. Dopo aver creato un progetto di nome *mysql*, è stato applicato un *manifest file* per una risorsa PVC:

```
1  apiVersion: v1
2  kind: PersistentVolumeClaim
3  metadata:
4    name: mysql-pv-claim
5  spec:
6    storageClassName: gp2
7    accessModes:
8      - ReadWriteOnce
9    resources:
10     requests:
11       storage: 100Gi
```

Il PersistentVolumeClaim appena descritto utilizza uno StorageClass di nome *gp2*. Esso indica il metodo utilizzato per la creazione dei volumi, in particolare è stato

usato il servizio Elastic Block Store offerto dal provider. Esso permette di avere un'archiviazione a blocchi legata direttamente alle macchine virtuali del cluster. Lo StorageClass è stato configurato per richiedere ad AWS la creazione dinamica di volumi di tipo gp2 che identificano dischi SSD a utilizzo generico con un buon rapporto prezzo-prestazioni.
La dimensione richiesta per la creazione del volume potrà variare a seconda degli scenari presi in considerazione, nel *manifest file* qui descritto è stato riportato un PVC di 100 GiB.

La creazione del Pod è richiesta da un Deployment che si occupa di specificare il volume persistente da collegare; è inoltre applicata una risorsa di tipo Service con la porta usata dal Pod.
Di seguito si riporta il *manifest file* utilizzato:

```
1  apiVersion: v1
2  kind: Service
3  metadata:
4    name: mysql
5  spec:
6    ports:
7    - port: 3306
8    selector:
9      app: mysql
10   clusterIP: None
11 ---
12 apiVersion: apps/v1
13 kind: Deployment
14 metadata:
15   name: mysql
16 spec:
17   selector:
18     matchLabels:
19       app: mysql
20   strategy:
21     type: Recreate
22   template:
23     metadata:
24       labels:
25         app: mysql
26     spec:
27       containers:
28       - image: mysql:5.6
```

```
29          name: mysql
30          env:
31            # Use secret in real usage
32          - name: MYSQL_ROOT_PASSWORD
33            value: password
34          ports:
35          - containerPort: 3306
36            name: mysql
37          volumeMounts:
38          - name: mysql-persistent-storage
39            mountPath: /var/lib/mysql
40        volumes:
41        - name: mysql-persistent-storage
42          persistentVolumeClaim:
43            claimName: mysql-pv-claim
```

Come si può notare alla riga 33, la password per il database è stata inserita in chiaro direttamente sul *manifest file* tramite le variabili d'ambiente. In uno scenario reale è consigliato l'utilizzo di un Secret per archiviare correttamente il dato.
Un'altra considerazione riguarda la versione di *mysql*, si tratta della 5.6 che utilizza il percorso

```
/var/lib/mysql
```

per accedere al volume persistente.

4.3 Soluzione tecnologica: Velero

Velero è uno strumento open source capace di salvare e ripristinare risorse di Kubernetes e volumi di dati [23]. Dato che si può utilizzare su tutte le piattaforme Kubernetes, è possibile usufruire delle sue potenzialità su ambienti on-premise e sul cloud.
Velero permette di:

- Creare backup di applicazioni cloud native e ripristinarle in caso di errori o cancellazioni accidentali di file.
- Creare backup di tutte le risorse del cluster.
- Migrare le risorse di un cluster in altri cluster.
- Replicare un sistema di produzione in uno di sviluppo, o viceversa, su namespace o cluster diversi.

Questo strumento è in grado di ridurre i tempi di ripristino in caso di perdita dell'infrastruttura, perdita di dati oppure interruzioni di servizi. Esso consente la portabilità del cluster migrando facilmente le risorse da un cluster all'altro con una discreta personalizzazione.
Velero si presenta con:

- un server in esecuzione sul cluster;
- un client utilizzabile da CLI.

Grazie al client installato in locale è possibile gestire tutto il sistema, come inviare una nuova richiesta di backup o vedere lo stato del ripristino in corso.

Il client di Velero riesce a dialogare con il server grazie all'utilizzo del *kubeconfig*, un file che racchiude le informazioni sul cluster.
Il *kubeconfig* viene creato in automatico dopo l'installazione del cluster e contiene:

- l'indirizzo API del cluster;
- il certificato del cluster;
- un contesto;
- un utente *admin*.

Di seguito si riporta la sua struttura, utile nel caso di aggiunta di un utente specifico (ad esempio tramite token) a cui sono state assegnate le autorizzazioni necessarie per l'utilizzo di Velero.

```yaml
apiVersion: v1
clusters:
- cluster:
    certificate-authority-data: ...
    server: https://...
  name: cluster1
contexts:
- context:
    cluster: cluster1
    user: velero-user
  name: user1
current-context: user1
kind: Config
preferences: {}
users:
- name: velero-user
  user:
    token: ...
```

Il file *kubeconfig* viene identificato in automatico da Velero se è salvato con il nome `config` all'interno della cartella `$HOME/.kube`. In alternativa, è possibile impostare la variabile d'ambiente `KUBECONFIG` con il percorso del file desiderato oppure utilizzare il flag `--kubeconfig`.

Velero offre diverse funzioni, ad esempio la creazione di un backup, di un ripristino o di un backup programmato. Ognuna di queste funzioni è una risorsa personalizzata e definita, all'interno di etcd, da una Custom Resource Definition di Kubernetes. Di seguito sono elencati i tipi di CRD creati durante l'installazione dello strumento sul cluster.

```
Backups
BackupStorageLocations
DeleteBackupRequests
DownloadRequests
PodVolumeBackups
PodVolumeRestores
ResticRepositories
Restores
Schedules
ServerStatusRequests
VolumeSnapshotLocations
```

Nei successivi sottocapitoli saranno trattate le principali funzioni di Velero con le quali si è potuto sperimentare e analizzare diverse strategie di disaster recovery.

4.3.1 Backup

La risorsa *Backup* identifica un unico backup richiesto dall'utente, contiene tutte le informazioni necessarie per la sua identificazione e lo stato attuale, cioè se è in progresso o se è stato completato.
La richiesta di creazione di un backup può essere inviata tramite il comando

```
velero backup create <backup-name>
```

che scatena, in ordine, le seguenti azioni:

1. Il client Velero effettua una chiamata al server API di Kubernetes per la creazione di un oggetto di tipo Backup.
2. Il BackupController rileva il nuovo oggetto Backup appena creato e lo convalida.
3. Il BackupController avvia il processo di backup e raccoglie i dati richiesti interrogando il server API per le risorse.
4. Il BackupController effettua una chiamata al servizio di archiviazione ad oggetti per caricare il file di backup.
5. Se sono presenti delle risorse di tipo PersistentVolume, il BackupController si occupa di chiedere al cloud provider, tramite richieste API, la creazione di uno snapshot del disco. Questo argomento sarà approfondito all'interno dei sottocapitoli 4.3.5 e 4.3.6.

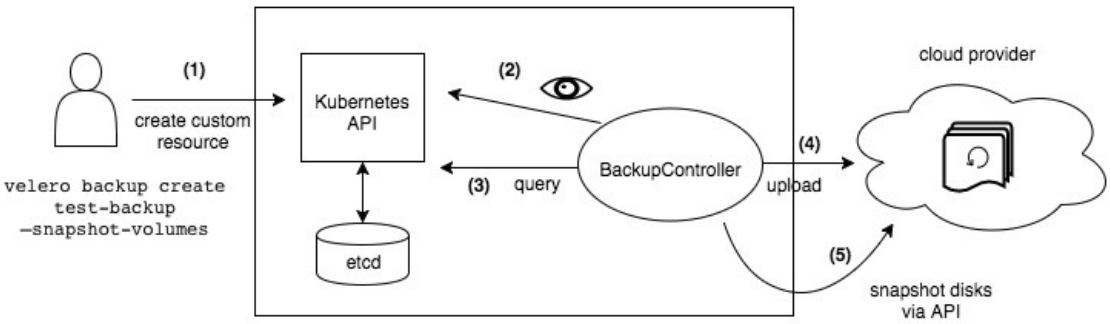

Figura 4.1: Schema per la creazione di un backup tramite Velero [23]

È possibile specificare diversi filtri, in base alle proprie esigenze, nel momento in cui si invia la richiesta di un nuovo backup.

- `--include-namespaces` o `--include-resources`: sono filtri utilizzati per includere solo i namespace o risorse desiderate. Ad esempio, includendo solo un namespace, tutte le risorse appartenenti ad altri namespace saranno escluse; lo stesso esempio si può applicare sui tipi di risorse. Inoltre, è possibile selezionare più di un elemento separandoli attraverso le virgole.
- `--exclude-namespaces` o `--exclude-resources`: queste opzioni si comportano in modo complementare rispetto alle precedenti, perciò il backup conterrà tutto tranne i namespace o le risorse escluse.
- `--include-cluster-resources`: è un filtro che offre la possibilità di selezionare o deselezionare tutte le risorse che non appartengono ad alcun namespace; esse possono far parte della configurazione del cluster oppure possono far parte di una o più applicazioni.
- `--selector`: include tutte le risorse che corrispondono alla label selezionata. Si specifica con la notazione `<key>=<value>`.
- *velero.io/exclude-from-backup=true*: tutte le risorse con questa label non sono incluse nel backup, anche se contengono un ulteriore label che corrisponde al *selector* desiderato.

Inoltre, è possibile specificare un TTL (Time To Live) aggiungendo il flag:

`--ttl <duration>`.

Il periodo di conservazione del backup può essere scritto in *ore, minuti, secondi*, ad esempio `24h0m0s`; se non specificato il valore di default è 30 giorni. Nel caso in cui il backup non sia più valido, Velero procede a rimuovere sia la risorsa di backup, sia il file di backup presente all'interno dell'archivio ad oggetti.

Quando si esegue un backup, è possibile specificare uno o più comandi da eseguire all'interno di un container di un Pod. I comandi possono essere eseguiti prima o dopo l'esecuzione del backup, si chiamano *pre hooks* e *post hooks*. Esistono due modi per specificare i comandi: direttamente sul Pod oppure nelle specifiche di backup.
Questa caratteristica può essere usata per bloccare la scrittura su disco di alcuni sistemi di log oppure per bloccare il file system al fine di garantire che le operazioni di I/O del disco in sospeso siano state completate prima di eseguire un backup.

Velero offre anche la possibilità di scegliere il backup delle risorse di uno specifico tipo in un ordine desiderato, utilizzando l'opzione `--ordered-resources`.
I nomi delle risorse sono separati da virgole e sono nel formato:

`<namespace/nome-risorsa>`

Un esempio di utilizzo è il seguente:

```
--ordered-resources 'pods=ns1/pod1;persistentvolumes=pv4,pv8'
```

In questa tesi sarà utilizzato principalmente il filtro per selezionare il progetto d'interesse, cioè *hello* nel caso di applicazione stateless e *mysql* per quella stateful. Di seguito è riportata la risorsa per una richiesta di backup sull'applicazione stateless.

```yaml
apiVersion: velero.io/v1
kind: Backup
metadata:
  name: hello-backup
  namespace: velero
spec:
  defaultVolumesToRestic: false
  hooks: {}
  includedNamespaces:
  - hello
  metadata: {}
  storageLocation: default
  ttl: 720h0m0s
  volumeSnapshotLocations:
  - default
```

Questa risorsa è creata in automatico se si esegue il seguente comando:

```
velero backup create hello -include-namespaces=hello
```

4.3.2 Schedule

Velero permette anche di programmare un backup tramite la risorsa Schedule che consente di specificare ogni quanto tempo eseguirlo oppure selezionare un orario fisso. La programmazione del backup avviene specificando un'espressione *cron*. Di seguito si riporta un esempio di comando per inviare una richiesta di backup ogni giorno alle ore 3:00.

```
velero schedule create schedule-hello --schedule="0 3 * * *"
```

La risorsa di tipo Schedule, rispetto a quella di Backup, aggiunge solo una specifica per la programmazione *cron*, il resto della struttura rimane invariata.

4.3.3 Restore

Il processo di recupero delle applicazioni avviene nel momento della creazione di una risorsa di tipo Restore specificando il backup da cui si vuole avviare la procedura di ripristino. È possibile filtrare le risorse da ripristinare utilizzando gli stessi filtri del Backup spiegati nel sottocapitolo 4.3.1, perciò si ha molta flessibilità sulle strategie di recupero dati. Inoltre si possono recuperare le risorse all'interno dello stesso cluster su namespace differenti tramite l'opzione `--namespace-mappings`.
La procedura di ripristino aggiunge a tutte le risorse recuperate la seguente *label*:

```
velero.io/restore-name:  <restore-name>
```

Durante la procedura di recupero alcune risorse sono escluse di default, ad esempio quelle che identificano i nodi del sistema, tutti gli eventi e le risorse stesse di Velero. Per quanto riguarda le risorse Service, Velero elimina di default i NodePort assegnati automaticamente, in questo modo il cluster assegna un nuovo NodePort al Service appena ripristinato. Per preservare il NodePort è necessario utilizzare l'opzione `--preserve-nodeports`, ma è responsabilità dell'utente pervenire una collisione con una porta già utilizzata.

In questa tesi la richiesta per la creazione di un ripristino dell'applicazione stateless è inviata tramite il seguente comando:

```
velero restore create hello-restore --from-backup hello
```

4.3.4 BackupStorageLocation

I backup delle risorse creati da Velero sono salvati all'interno di sistemi di archiviazione ad oggetti offerti dai cloud provider. La risorsa BackupStorageLocation si occupa di identificare questi sistemi specificando i dettagli richiesti dal cloud provider in uso.

Grazie ai Plugin di Velero è possibile aggiungere il supporto a diversi cloud provider, alcuni di essi sono anche supportati in modo ufficiale. La definizione di una nuova posizione di salvataggio può essere specificata in fase di installazione oppure successivamente dopo aver aggiunto il Plugin necessario. I dettagli su come aggiungere il bucket si possono consultare all'interno della documentazione del Plugin scelto.

Una caratteristica comune a tutti i BackupStorageLocation è la possibilità di specificare la modalità di accesso: ReadWrite o ReadOnly. In questo modo quando si avvia una procedura di ripristino, si può prevenire la scrittura di un eventuale backup, sul bucket in uso, selezionando la modalità opportuna.

In questo elaborato è stato utilizzato un bucket S3 di AWS creato nella regione *eu-south-1*, cioè nel datacenter di Milano ed un bucket di Azure situato nella regione *germanywestcentral*.

4.3.5 VolumeSnapshotLocation

I volumi persistenti di un cluster sono salvati tramite un sistema chiamato snapshot, offerto direttamente dal cloud provider utilizzato. All'invio del comando per un nuovo backup, Velero chiede la creazione di uno snapshot per ciascuno dei volumi creati all'interno del cloud provider. Se è stata configurata una risorsa di nome VolumeSnapshotLocation, specificando la regione in cui richiedere gli snapshot, Velero può interagire con il fornitore di servizi cloud, altrimenti non sarà possibile salvare i PV.

In questa tesi è stato utilizzato il sistema di snapshot offerto da AWS e Azure; questo tema sarà approfondito nei sottocapitoli 4.4.2 e 4.4.4.

4.3.6 Restic

Velero ha integrato un supporto per un altro metodo di backup che utilizza uno strumento open source chiamato Restic. Attraverso Restic è possibile salvare tutte le risorse di Kubernetes in un unico posto, includendo anche i volumi persistenti; si tratta del sistema di archiviazione ad oggetti. Questa modalità è un'aggiunta rispetto al metodo di default che usa il sistema di snapshot; il suo utilizzo è richiesto alla creazione di un backup.

La funzionalità di Restic deve essere abilitata in fase di installazione; si tratta di un DaemonSet che crea un Pod su ogni nodo worker responsabile dell'esecuzione del software open source. Velero ha tre CRD legate all'utilizzo di Restic:

- **ResticRepository**: rappresenta il ciclo di vita del repository Restic e viene creata all'inizializzazione di ogni nuovo repository. Quando si richiede la creazione di un nuovo backup con Restic, Velero crea una risorsa di tipo ResticRepository per ogni namespace coinvolto, se non è già presente. Il controller per questa risorsa si occupa della gestione del repository ed esegue i comandi di inizializzazione, controllo e pulizia periodica.
- **PodVolumeBackup**: rappresenta un backup di un volume in un Pod tramite Restic. Questa risorsa è creata da Velero durante la fase di backup quando trova un Pod associato ad un PV. Ogni nodo del cluster esegue un controller in un DaemonSet che gestisce le risorse di tipo PodVolumeBackup per i Pod situati nel nodo corrente. Il controller ha il compito di eseguire i comandi per creare il backup dei dati.
- **PodVolumeRestore**: rappresenta un ripristino di un volume in un Pod con Restic. Questa risorsa è creata da Velero quando, durante la fase di ripristino, trova un volume a cui è associato un backup con Restic. Il controller, che viene eseguito su ogni nodo, si occupa di eseguire i comandi di ripristino per recuperare i dati del volume del Pod.

Con il comando:

```
velero restic repo get
```

è possibile vedere le informazioni dei repository configurati. Ad esempio una risorsa di tipo ResticRepository, responsabile del progetto *mysql*, è la seguente:

```yaml
apiVersion: velero.io/v1
kind: ResticRepository
metadata:
  name: mysql-default-rb4n8
  namespace: velero
spec:
  backupStorageLocation: default
  maintenanceFrequency: 168h0m0s
  resticIdentifier:
  ↪ s3:s3-eu-south-1.amazonaws.com/st-velero-aws-eu-south-1/restic/mysql
  volumeNamespace: mysql
status:
  lastMaintenanceTime: "2022-02-17T10:26:58Z"
  phase: Ready
```

Questa risorsa specifica il BackupStorageLocation sul quale verranno salvati i dati, il namespace coinvolto e la frequenza di esecuzione della manutenzione che corrisponde alla pulizia del repository.

La documentazione di Velero [23] suggerisce di utilizzare Restic per quei volumi che non supportano la funzionalità di snapshot, ad esempio EFS, AzureFile, NFS, emptyDir e local. In questa tesi Restic è stato utilizzato anche per i PV creati suoi cloud provider, argomento che sarà trattato nel sottocapitolo 4.4.3.

4.4 Backup e recupero nella stessa regione

In questa tesi si sono presi in considerazione diversi scenari applicabili a diverse situazioni di disaster recovery. In questo primo sottocapitolo si tratterà del backup e del ripristino di applicazioni all'interno della stessa regione di un cloud provider.

I fornitori di servizi cloud offrono risorse in diverse posizioni geografiche, chiamate regioni, che identificano un data center. Non tutte le regioni offrono gli stessi servizi e non tutti i tipi di risorse sono legati esclusivamente ad una regione. Solo i grandi cloud provider hanno regioni in tutto il mondo. AWS propone una regione in Italia, situata a Milano, mentre Azure ad oggi non è presente sul territorio italiano.

Nel caso in cui il disastro sia recuperabile all'interno dello stesso sito primario allora si possono identificare diverse strategie di DR, le quali saranno presentate e discusse. Inoltre, sarà analizzato e calcolato il tempo necessario per poter eseguire il backup e il ripristino tramite il sistema di snapshot; successivamente i risultati ottenuti saranno confrontati con Restic. Le prove effettuate saranno applicate alle applicazioni stateless e stateful descritte nei sottocapitoli 4.2.2 e 4.2.1. Infine si analizzerà le potenzialità del backup incrementale e delle zone di disponibilità all'interno della stessa regione.

In questo sottocapitolo si utilizza una installazione *installer-provisioned* di OpenShift aggiornata alla versione 4.9.9, con tre nodi di control plane con le seguenti caratteristiche (una parte del file *install-config.yaml*):

```
controlPlane:
  architecture: amd64
  hyperthreading: Enabled
  name: master
  platform:
    aws:
      rootVolume:
        size: 100
      type: m5a.xlarge
  replicas: 3
```

e due nodi worker con queste specifiche:

```
compute:
- architecture: amd64
  hyperthreading: Enabled
  name: worker
```

```
platform:
  aws:
    rootVolume:
      size: 100
    type: t3a.xlarge
replicas: 2
```

Le macchine virtuali *m5a.xlarge* e *t3a.xlarge* offrono entrambe 4 vCPU e 16 GiB di memoria, ma si distinguono per la fatturazione dei costi. Questa scelta è stata decisa insieme all'azienda con cui ho collaborato.

La piattaforma di orchestrazione di container sarà utilizzata all'interno della regione *eu-south-1* (Milano) di AWS. Mentre, la soluzione tecnologia per il salvataggio e il recupero dei servizi sarà Velero in versione 1.7.1.

4.4.1 DR di applicazioni stateless

Le applicazioni stateless sono le più semplici da includere in un piano di disaster recovery. Esse consistono solo in risorse e si identificano con i *manifest file*. I backup di Velero permettono di salvare tutti i tipi di risorse all'interno di un sistema di archiviazione ad oggetti esterno.

Applicazione hello

La prima prova è legata il salvataggio dell'applicazione *hello* (definita nel sottocapitolo 4.2.1), la quale ha impiegato 15 s. Il backup è stato eseguito tre volte per avere un riscontro sui valori e il risultato è stato consistente in tutte le prove. Una volta eliminato il namespace all'interno del cluster, è stato richiesto il recupero dell'applicazione, il quale ha impiegato solo 2 s. I comandi eseguiti per questa prima prova di DR sono stati i seguenti:

```
velero backup create hello --include-namespaces=hello
oc delete namespace hello
velero restore create hello-restore --from-backup hello
```

Il tempo totale di esecuzione è stato rilevato attraverso i dettagli delle risorse di tipo Backup e Restore che includono la data e ora di inizio e fine esecuzione. In ultimo si evidenzia che in questo backup sono state incluse 66 risorse in totale.

È importante sottolineare che il recupero eseguito da Velero applica solo i *manifest file* al sistema, quindi per avere un'applicazione attiva e disponibile è necessario un ulteriore tempo aggiuntivo T_{agg} che include:

1. Lo scaricamento delle immagini necessarie per i container (se non sono già presenti all'interno del cluster).
2. La creazione di tutti i Pod necessari.
3. La creazione e la distribuzione del LoadBalancer richiesto al cloud provider.

Dopo queste considerazioni è stato possibile creare una formula per stimare il tempo di recupero di applicazioni stateless:

$$T_{r\_tot} = T_{velero} + T_{agg} \tag{4.1}$$

Lo scenario stateless preso in esame evidenzia come i primi due punti non influiscano particolarmente sul tempo totale. Durante l'esecuzione delle prove, si è misurato il tempo che il cluster ha impegnato per rendere i Pod in stato di *Running*, si tratta di un valore intorno ai 5 s, che non è consistente, ma è sempre stato inferiore ai 20 s. Mentre, per quanto riguarda il terzo punto, si sono misurati dei tempi molto più alti, nell'ordine dei minuti e poco consistenti, dai 2 ai 15 min. Questo comportamento è legato probabilmente alla distribuzione del DNS di AWS restituito dal LoadBalancer.

Tutto il cluster

Una seconda prova stateless ha coinvolto il salvataggio di tutto il cluster per poter verificare il tempo richiesto da Velero per salvare grandi quantità di oggetti. Il backup richiesto ha escluso solo le risorse di tipo *events* con la seguente opzione:

```
--exclude-resources=events,events.events.k8s.io
```

Il risultato è stato un salvataggio di 6626 risorse in 54 s.

Queste considerazioni evidenziano come il backup di risorse sia relativamente veloce rendendo quindi le applicazioni stateless facilmente ripristinabili all'interno dello stesso cluster.

4.4.2 DR con il sistema di snapshot

I volumi delle applicazioni stateful possono essere salvati con Velero tramite il sistema di snapshot offerto dai cloud provider. Per verificare le potenzialità e i limiti di questo sistema di salvataggio si è deciso di provare a chiedere il backup di volumi con diverse dimensioni. È stato utilizzato Velero 1.7.1 con il plugin di AWS in versione 1.3.0.

Nello specifico si è deciso di utilizzare l'applicazione stateful *mysql* presentata nel sottocapitolo 4.2.2 con un volume persistente, richiesto tramite il PVC, di dimensioni pari a: 1, 10, 50, 100 GiB.

Prima di elencare i risultati è necessario fare alcune considerazioni. In primo luogo, un disco Elastic Block Store di AWS ha specifiche diverse in base alla dimensione allocata; nella tabella 4.1 sono elencate le specifiche delle dimensioni considerate.

GiB	**Tipo**	**IOPS**
1	gp2	100
10	gp2	100
50	gp2	150
100	gp2	300

Tabella 4.1: Tipi di volumi utilizzati su AWS

Inoltre, il controller di Velero non interagisce con il cloud provider per sapere lo stato del backup del volume persistente; infatti tutti i backup richiesti, con i tipi di volumi presentati precedentemente, risultano essere completati sempre dopo soli 15 s. Per questo motivo è stato necessario trovare un modo per poter sapere il tempo di completamento dello snapshot. Si è deciso di creare uno script bash che interroga il cloud ogni secondo, tramite la sua CLI, per sapere se esiste uno snapshot che abbia una data compatibile con la richiesta e se questo sia completato. In questo modo è stato possibile ricavare un tempo totale corretto per il backup preso in considerazione. Di seguito viene riportato lo script utilizzato:

```
#!/bin/bash
start=`date +%s`
bkp_name="bkp-snapshot-test"
velero create backup ${bkp_name} --include-namespaces=mysql
hour_now=`date +"%T" --date='1 hour ago'`
date_now=`date +"%F" --date='1 hour ago'`
date="${date_now}T${hour_now}"
snap=`aws ec2 describe-snapshots --owner-ids 760012763089 --query
↪  "Snapshots[?(StartTime>='${date}')].[SnapshotId]" --output text`
while [ "$snap" = "" ]
do
  sleep 1
  snap=`aws ec2 describe-snapshots --owner-ids 760012763089 --query
  ↪  "Snapshots[?(StartTime>='${date}')].[SnapshotId]" --output text`
done
```

```
int=`date +%s`
echo
echo "snapshot started after $((int-start)) seconds"
snap=`aws ec2 describe-snapshots --owner-ids 760012763089 --filters
↪  Name=status,Values=completed --query
↪  "Snapshots[?(StartTime>='${date}')].[SnapshotId]" --output text`
while [ "$snap" = "" ]
do
  sleep 1
  snap=`aws ec2 describe-snapshots --owner-ids 760012763089 --filters
  ↪  Name=status,Values=completed --query
  ↪  "Snapshots[?(StartTime>='${date}')].[SnapshotId]" --output text`
done
end=`date +%s`
velero describe backup ${bkp_name}
echo
echo "snapshot took $((end-int)) seconds, while total backup took
↪  $((end-start)) seconds"
```

Il grafico 4.2 riporta le tempistiche rilevate al variare della dimensione del volume persistente. L'asse delle ascisse rappresentata la dimensione del volume in GiB, mentre sulle ordinate è indicato il tempo totale in secondi. Il volume in questione è stato creato della dimensione richiesta senza ulteriori modifiche, perciò contiene solo i file base di mysql che occupano una dimensione di circa 115 MiB).

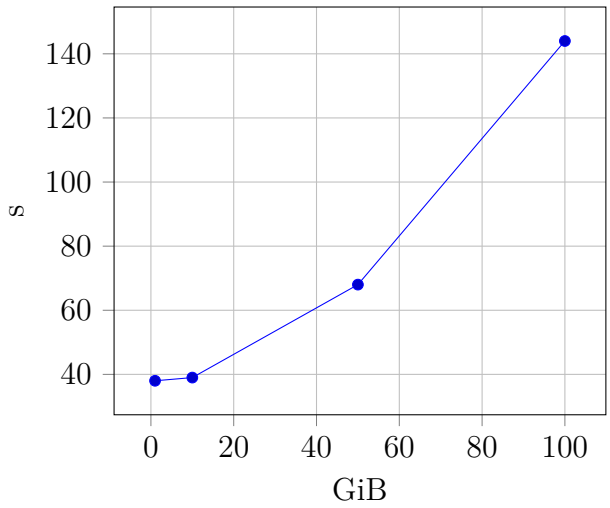

Figura 4.2: Tempistiche di backup per volumi vuoti tramite snapshot

Successivamente sono stati richiesti altri backup considerando gli stessi scenari ma con i volumi pieni di dati. Nello nello specifico sono stati considerati:

(a) Un volume con all'interno un unico grande file che occupa tutto lo spazio disponibile. Questo scenario ha lo scopo di simulare i database o tutte quelle applicazioni che utilizzano file di grandi dimensioni.

(b) Un volume interamente occupato da tanti piccoli file per verificare le prestazioni del sistema di backup anche su questo tipo di scenario. In particolare si è deciso di utilizzare file di dimensione 4 MB.

I due tipi di file sono stati creati generando byte casuali, di seguito sono riportati i comandi utilizzati per i volumi di dimensione pari a 1 GiB:

(a) `dd if=/dev/urandom of=1GB bs=4MB count=262 status=progress`

(b)
```
for i in {001..262}; do dd if=/dev/urandom of=file$i bs=4MB
↪  count=1 > /dev/null 2>&1; done
```

I risultati sono riportati nella figura 4.3. In particole, il primo grafico 4.3i confronta un volume vuoto con uno pieno, cioè lo scenario (a); il secondo grafico 4.3ii evidenzia l'impatto dei file di piccole dimensioni (b).

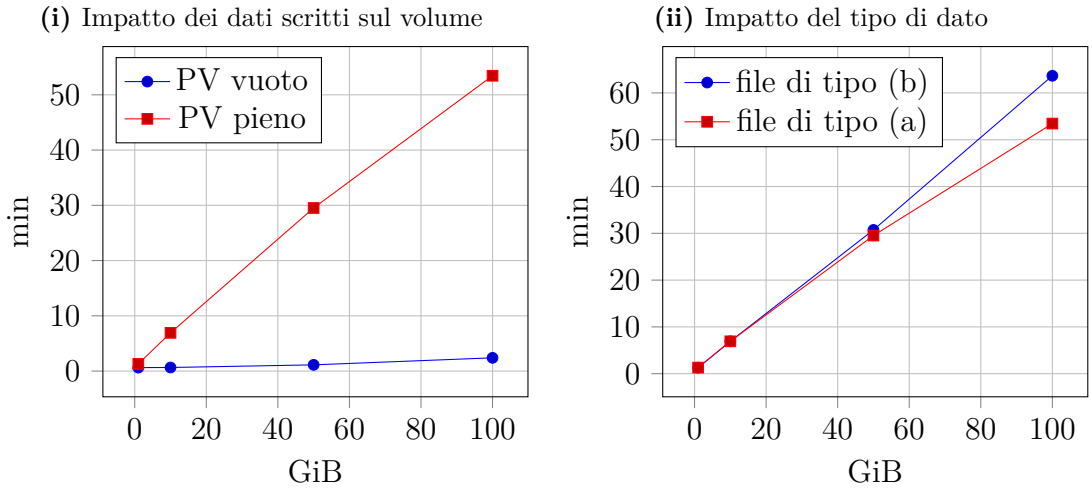

Figura 4.3: Backup di applicazione stateful tramite snapshot

Grazie alle tempistiche degli snapshot, misurati dallo script, e al totale dello spazio occupato all'interno dal volume si è calcolata la velocità media di elaborazione dei dati. In questo calcolo è stato preso in considerazione tutto il tempo necessario per il completamento dello snapshot, perciò potrebbe includere un eventuale

inizializzazione per il sistema di salvataggio del cloud provider. È stato possibile misurare lo spazio totale occupato del volume eseguendo il seguente comando all'interno del Pod di mysql:

```
du -s -b /var/lib/mysql
```

Nella figura 4.4 si riporta il grafico delle prestazioni calcolate per i due scenari considerati. Si può notare il valore anomalo del volume da 1 GiB che potrebbe confermare l'ipotesi dell'inizializzazione per il sistema di snapshot poiché ha più impatto su un volume di piccole dimensioni e con un tempo totale ristretto.

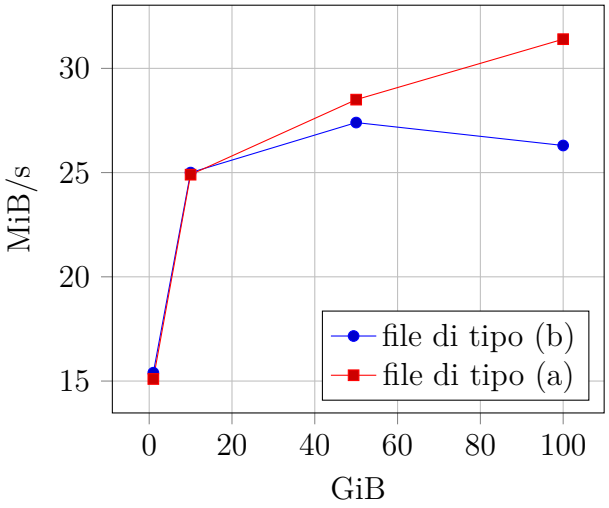

Figura 4.4: Throughput calcolato per il sistema di snapshot

In tutti gli scenari considerati per il calcolo dei tempi di backup, dopo aver eliminato il namespace dell'applicazione, è stato effettuato anche una simulazione di recupero dei dati all'interno dello stesso cluster. Tutte le prove sono risultate essere quasi immediate con un tempo di soli 3 s, che include sia le risorse di OpenShift sia il volume pieno di dati creato su AWS. Per questo motivo il sistema di snapshot del cloud provider probabilmente non elimina effettivamente il volume a cui è legato uno snapshot, anche se ne è richiesta la cancellazione.
Poiché il ripristino dei volumi è immediato, la formula 4.1 a pagina 65 è valida anche per le applicazioni stateful con il sistema di snapshot dei cloud provider.

4.4.3 DR con Restic

Velero offre la possibilità di utilizzare Restic, un programma moderno per il backup di dati. La documentazione di Velero consiglia di utilizzarlo con quei volumi che non supportano la funzionalità di snapshot. In questa tesi è stato utilizzato con gli stessi PV creati su AWS e quindi con gli stessi scenari presi in considerazione nel

sottocapitolo precedente 4.4.2, per evidenziare eventuali differenze, potenzialità o limitazioni. Le versioni dei software e la configurazione del cluster sono rimaste invariate rispetto alle prove precedenti.
La funzione di Restic è selezionabile quando si richiede un nuovo backup con l'opzione:

```
--default-volumes-to-restic
```

Se si utilizza Restic, la misurazione del tempo necessario per il backup viene inclusa direttamente nella risorse di tipo Backup di Velero, perciò non è stato necessario l'utilizzo di alcuno script. La procedura utilizzata per l'acquisizione dei tempi necessari è simile a quella usata con le applicazioni stateless, ma aggiunge una pulizia al repository alla fine di ogni prova. Infatti, a differenza degli snapshot, quando si richiede l'eliminazione di un backup, Restic non cancella i file salvati sull'archiviazione ad oggetti, ma è necessario forzarne la pulizia del repository. L'esecuzione del comando `restic prune` è gestita dal controller Restic ed è possibile selezionare la frequenza di manutenzione all'interno della risorsa ResticRepository.

Il tempo necessario per il backup del volume vuoto di *mysql* risulta essere consistente con tutte le dimensioni del volumi, si tratta di soli 18 s, decisamente inferiore al sistema di snapshot, vedi figura 4.2.

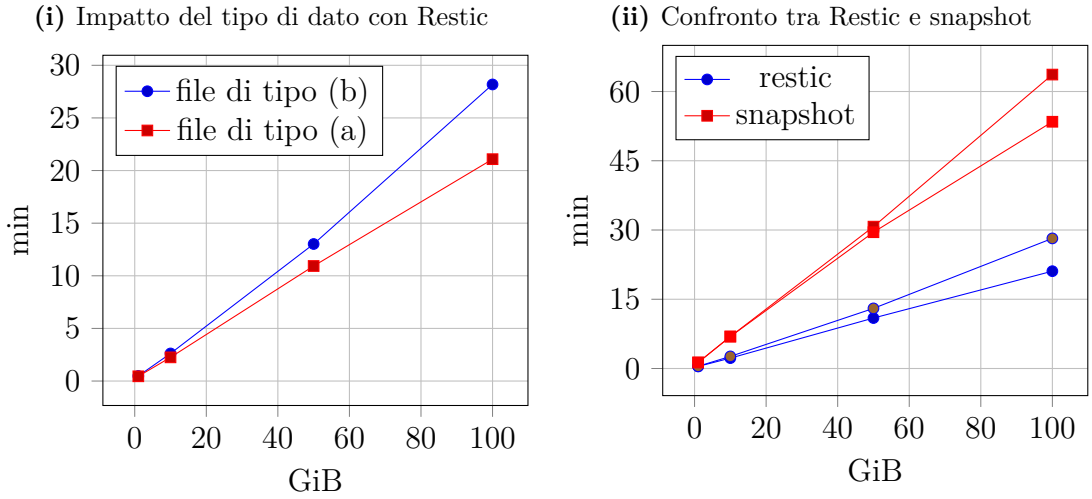

Figura 4.5: Backup di applicazione stateful con Restic e con il sistema di snapshot

Il confronto tra i due diversi tipi di file (a) e (b) scritti sul volume ha riportato una differenza più marcata rispetto a quella riscontrata con il sistema di snapshot. Il grafico 4.5i mostra il tempo necessario al variare della dimensione del volume sui due scenari presi in considerazione. Considerando la dimensione di 100 GiB si evidenzia un incremento di tempo del 33 %. Il secondo grafico 4.5ii confronta i

risultati del metodo Restic e di quello a snapshot, si può notare come il primo sia circa 2.5 volte più veloce del secondo.

Un secondo aspetto interessante è la variazione della velocità al variare della dimensione del volume: i risultati evidenziano una netta differenza tra i due scenari, il grafico è riportato in figura 4.6. I file di grandi dimensioni garantiscono una prestazione stabile di circa 80 MiB/s. Mentre il secondo scenario, oltre a mostrare prestazioni inferiori, evidenzia un throughput che decresce all'aumentare del numero di file utilizzati.

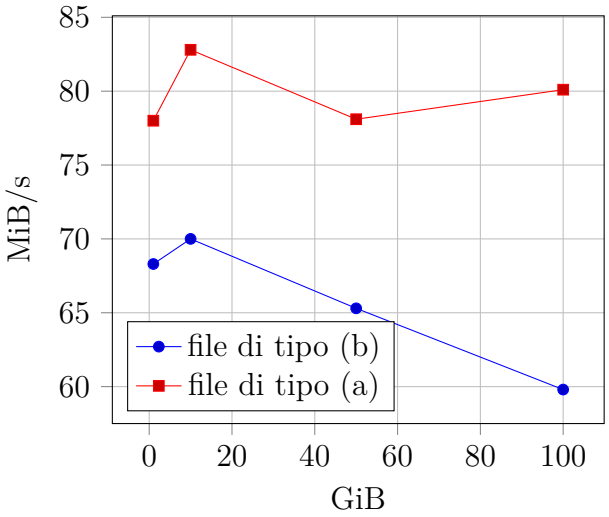

Figura 4.6: Throughput calcolato con Restic

Su ciascuna prova, dopo aver cancellato il namespace corrispondente, è stato eseguito anche il recupero dell'applicazione stateful. Con entrambi i tipi di volume considerati si sono registrate delle tempistiche migliori rispetto al backup eseguito precedentemente. Se si prende in considerazione il PV da 100 GiB, si evidenzia una riduzione di tempo di circa 29 % per lo scenario (a) e di 43 % per lo scenario (b). Nella figura 4.7 sono riportati i grafici che confrontano i tempi necessari per il completamento del backup e per il ripristino di tutti i dati.

In ultimo si evidenzia una limitazione legata alla funzione Restic sperimentata durante l'esecuzione delle prove. Infatti, se si richiede un ripristino di un volume con lo spazio completamente utilizzato, allora questo fallirà. Il motivo è dettagliato nei log, in sintesi si tratta di spazio insufficiente per la creazione di file temporanei, necessari a Restic per completare il processo di ripristino. Per questo motivo in tutte le prove effettuate è stato lasciato circa 4 MiB di spazio libero. La limitazione appena descritta non ha però influenzato in nessun modo sulla veridicità dei dati poiché le velocità di throughput sono state calcolate sulla dimensione dei dati scritti nei volumi.

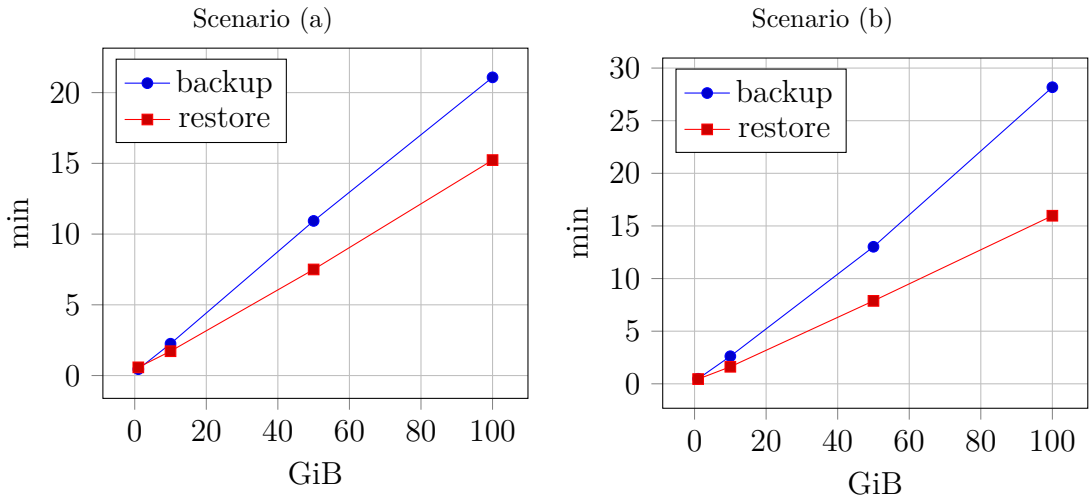

Figura 4.7: Backup e recupero di applicazione stateful con Restic

Scalabilità

Il backup eseguito tramite Restic richiede l'allocazione e l'utilizzo di risorse del cluster, perciò le applicazione possono avere meno potenza di calcolo a disposizione. Per evitare questo problema si può ricorrere all'utilizzo della funzionalità di Schedule. Inoltre, l'utilizzo di risorse può essere gestito attraverso le funzionalità di OpenShift, ereditate da Kubernetes, chiamate CPU request/limit e il corrispettivo per la memoria. Di default, il Daemonset di Restic ha un limite preconfigurato di 1 CPU e 1 GiB ma è possibile aumentarlo in caso di necessità. In ultimo si ricorda che il consumo di risorse necessario per le operazioni di Restic è strettamente legato al nodo su cui sono presenti Pod con volumi.

Poiché Restic viene eseguito all'interno del cluster si ha più controllo sul suo funzionamento; questa caratteristica si contrappone al sistema di snapshot, dove invece si può valutare solo il risultato come un modello a scatola nera.

Dopo queste considerazione si è deciso di verificare la scalabilità di questo metodo, per valutarne le potenzialità o eventuali limitazioni. Si è deciso di fissare la dimensione del volume a 50 GiB e di variare i limiti del Daemonset di Restic. Le specifiche di request e limit sono modificabili direttamente sulla risorsa tramite il comando `oc edit` oppure `oc patch` come indica la documentazione di Velero [23].

Per questo tipo di prova è stato creato un nuovo cluster perché quello corrente non aveva le risorse necessarie. Si è deciso di provare ad aumentare le risorse richieste fino a 4 vCPU e, poiché il cluster precedentemente creato utilizzava nodi da 4 vCPU, non è stato possibile allocare sullo stesso nodo tutte le risorse necessarie. Questa limitazione è giustificata dalla presenza di altri Pod sullo stesso nodo in cui

è presente Restic, come ad esempio quelli del sistema di OpenShift. Il nuovo cluster OpenShift creato è in versione 4.9.11 con 1 nodo di control plane (m5a.xlarge) e 2 worker di tipo t3a.2xlarge, cioè con 8 vCPU.

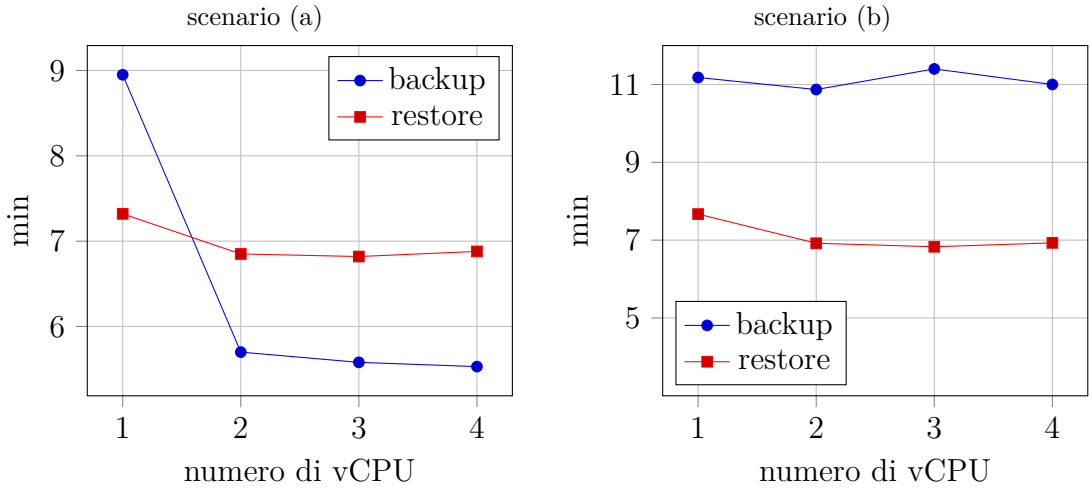

Figura 4.8: Scalabilità di Restic su un volume pieno di dati

La figura 4.8 illustra i due scenari considerati su cui sono state cambiate le configurazione di cpu request e limit. Per semplicità si è deciso di cambiare entrambe le specifiche per verificare se è possibile allocare le risorse necessarie sui nodi worker. La memoria è stata aumentata solo a 2 GiB poiché risulta non essere utilizzata in modo aggressivo in tutti gli scenari (circa 300 MiB durante i backup e 600 MiB nella fase di restore).

Durante l'esecuzione delle prove si sono analizzate le statistiche di effettivo utilizzo delle risorse allocate e non vengono utilizzate pienamente, proprio come suggeriscono i risultati. In particolare, nello scenario (b) Restic utilizza circa 1 vCPU per il backup e 1.5 vCPU per il restore; mentre nel primo scenario si ha un aumento di risorse utilizzate ma è sempre inferiore a 2 vCPU. Inoltre, la veridicità dei risultati è stata confermata da un'ulteriore prova con un volume di maggiori prestazioni. Infatti, è stato utilizzato un volume di 500 GiB con 1500IOPS e non si sono registrati miglioramenti rispetto allo scenario precedente.

Un secondo aspetto che è stato considerato è la scalabilità orizzontale, riferita all'aumentare del numero di volumi su più nodi. Purtroppo, al momento della stesura di questo elaborato, Velero non implementa una strategia per l'esecuzione in parallelo di più backup. Infatti, se si richiedono due backup, il secondo viene messo in coda e verrà eseguito successivamente. Un ultima prova è stata eseguita creando due applicazioni stateful (mysql) sui due nodi worker a disposizione, cioè un Pod su ciascuno, all'interno dello stesso progetto. Anche in questo caso l'esecuzione di

Restic non viene eseguita in parallelo, prima viene eseguito il backup del volume sul primo nodo e poi sul secondo. Questo risultato si presenta come una limitazione importante e su questo il team del progetto open source è al lavoro: è aperta una *issue* su Github [24].

4.4.4 Backup incrementale

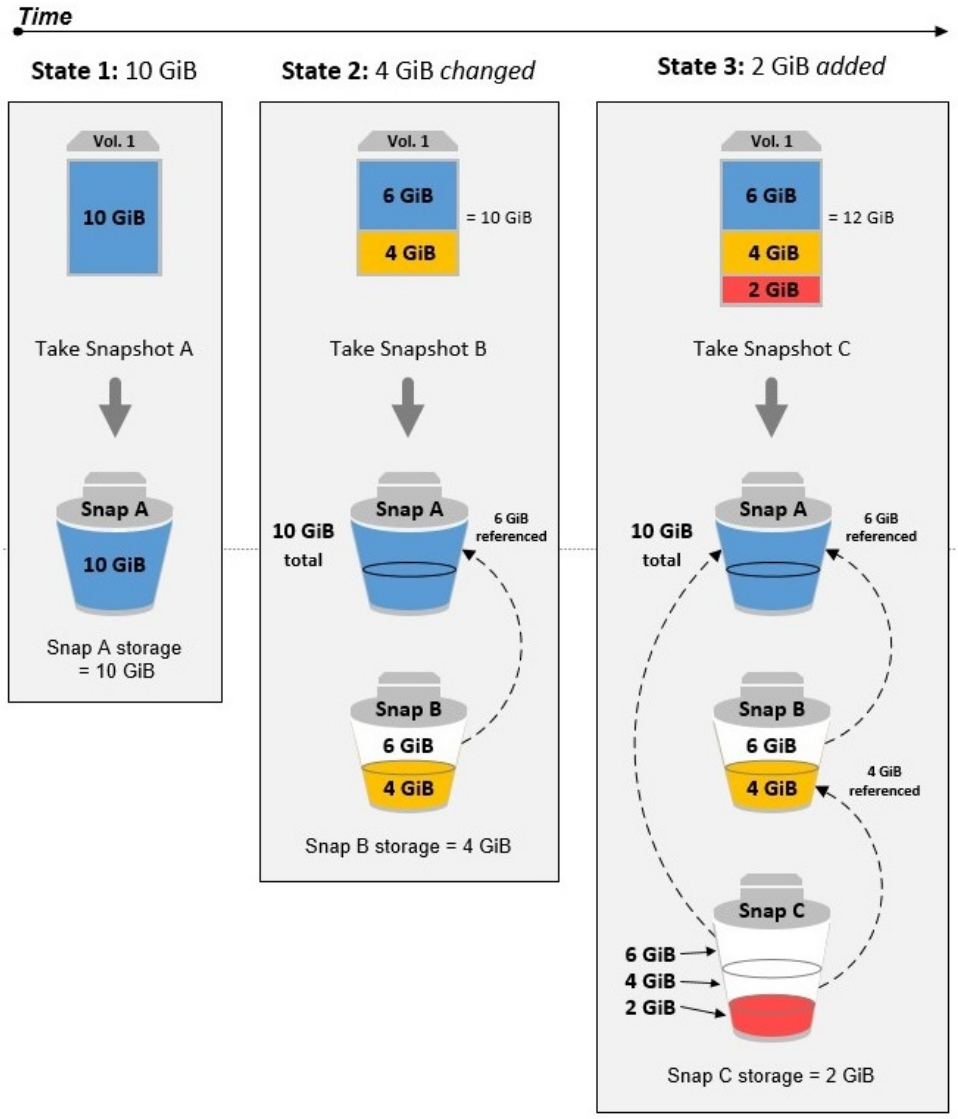

Figura 4.9: Come funziona il backup incrementale dei volumi su AWS [25]

Il sistema di snapshot è implementato completamente dal cloud provider ed è offerto come una scatola nera in cui non è possibile sapere il suo funzionamento interno. Le uniche informazioni che le documentazioni dei provider propongono sono le spiegazioni legate alla tariffazione e il supporto del backup incrementale.

Il backup incrementale viene offerto da diversi provider ed è molto utile perché promette un risparmio di tempo, spazio e denaro. Ad esempio, il sistema di Amazon Elastic Block Store, salva solo i blocchi che sono stati modificati dopo lo snapshot più recente e ciascuno snapshot contiene tutte le informazioni necessarie per ripristinare i dati su un nuovo volume. La figura 4.9 evidenzia due scenari: il primo (stato 2) mostra il comportamento dopo la modifica di una parte del volume; nello stato 3, invece, sono aggiunti ulteriori dati al volume. Come si può notare, i dati già salvati in backup precedenti sono solo referenziati dagli snapshot successivi.

Per verificare la sua funzionalità sono state eseguite alcune prove, utilizzando il cluster con 3 nodi di control plane precedentemente descritto 4.4. Si tratta di un confronto tra un backup completo di un volume da 50 GiB e di uno su cui è stato aggiunto un file da circa 10 GiB prima di ogni nuova richiesta di backup. Questa prova è stata eseguita anche con Restic e il loro risultato è riportato in figura 4.10. Come si può notare, il backup incrementale funziona bene per entrambi i sistemi e i tempi registrati sono corretti e prevedibili.

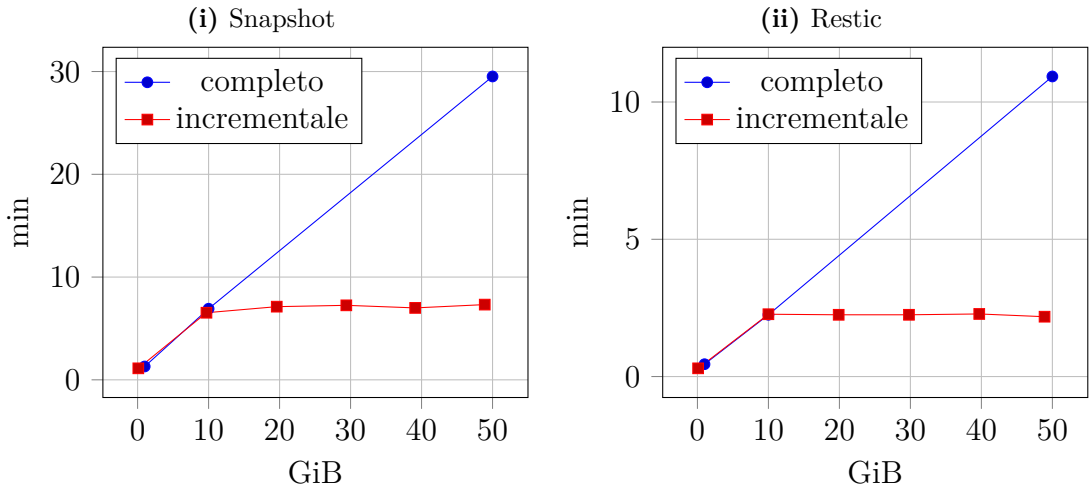

Figura 4.10: Backup incrementale su volume da 50 GiB

Nel primo scenario ci si è concentrati sull'aggiunta di file, mentre per quanto riguarda la modifica di questi è stata stabilita una seconda prova. Si è deciso di creare un backup di un volume da 10 GiB con un file di grande dimensione al suo interno; dopodiché si è modificata una piccola parte di questo file con il programma `hexedit` che permette di visualizzare e modificare in formato esadecimale. Dopo

aver modificato una piccola parte (due numeri esadecimali) del file da circa 10 GiB, si è richiesto un secondo backup per verificare il tempo richiesto metodo utilizzato. Il sistema di snapshot non ha riportato miglioramenti in termini di tempo, perciò probabilmente AWS non controlla le differenze dei file ma crea un backup su tutto ciò che è stato modificato o aggiunto e referenzia il resto con i backup precedenti. Invece Restic ha riportato un risultato interessante, cioè un tempo inferiore alla metà del primo backup. Inoltre, lo spazio occupato all'interno dell'archiviazione ad oggetti è aumentato di soli 1 MiB, perciò questo sistema è molto più efficiente in termini di spazio e tempo richiesto per completare il backup. Se l'applicazione utilizzata è, ad esempio, un database che modifica in piccola parte file di grandi dimensioni, allora Restic è la soluzione da considerare.

4.5 Migrazione

Nello studio di tesi sono stati anche considerati altri scenari in ambienti multi cluster, cioè utilizzando due cluster installati su infrastrutture diverse, e multi-cloud, cioè usando più cloud provider. Nello specifico si sono installati cluster su AWS ed Azure e le prove sono state eseguite per evidenziare limitazioni e tempistiche di migrazione.

La migrazione è un procedimento fondamentale in diversi piani di disaster recovery poiché permette di spostare le applicazioni e i servizi in un secondo luogo con un'infrastruttura diversa, in questo modo è possibile evitare qualunque tipo di problema capitatosi al cluster primario. Questo metodo di DR richiede dunque che ci sia un secondo cluster già installato e pronto all'uso, altrimenti è necessario crearne uno al momento del disastro con evidenti impatti sui tempi di recupero. Si è discusso di queste tematiche nel sottocapitolo 3.4. Gli scenari studiati in questa tesi sono applicabili su DR a freddo e a tiepido.

In questo sottocapitolo sono descritte le prove eseguite per migrare le applicazioni, inizialmente all'interno della stessa regione ma su cluster diversi, poi su altre regioni ed infine su cluster installati in altri cloud provider.

Le applicazioni utilizzate nei diversi scenari sono le stesse del sottocapitolo precedente, installate su un cluster OpenShift in versione 4.9.11 su AWS *eu-south-1* con tre nodi di control plane (m5a.xlarge) e due worker (t3a.xlarge). Su Azure è stato utilizzata la regione *germanywestcentral*, i dettagli verranno spiegati nel sottocapitolo corrispondente alla migrazione su altri cloud provider 4.5.3. La versione di Velero utilizzata per tutte le prove è sempre la stessa: 1.7.1.

4.5.1 Verso altre zone di disponibilità

Le zone di disponibilità, in inglese *availability zones*, sono delle aree ben distinte all'interno di un data center e hanno come scopo principale l'alta disponibilità e la tolleranza ai guasti [26]. Ogni zona risulta essere un data center indipendente per quanto riguarda le risorse; infatti la rete e l'alimentazione sono ridondati e separati dalle altre zone. Tutte le zone di disponibilità, all'interno di una regione, sono spesso edifici fisici diversi ma interconnessi tra di loro con dei link ad alto throughput e bassa latenza, la figura 4.11 illustra un esempio per una regione di Azure. Quest'ultimo aspetto è fondamentale perché permette di avere una replica sincrona fra le zone di disponibilità consentendo di aumentare la tolleranza ai guasti per le aziende che utilizzando il cloud.

Il processo di installazione di OpenShift considera le zone di disponibilità dei cloud provider e cerca di trarne benefici installando le repliche del control plane su zone distinte. Infatti la documentazione di OpenShift [6] consiglia di utilizzare

3 repliche di control plane, sarà poi compito dell'installer andare a richiedere l'allocazione delle macchine virtuali nelle zone opportune.

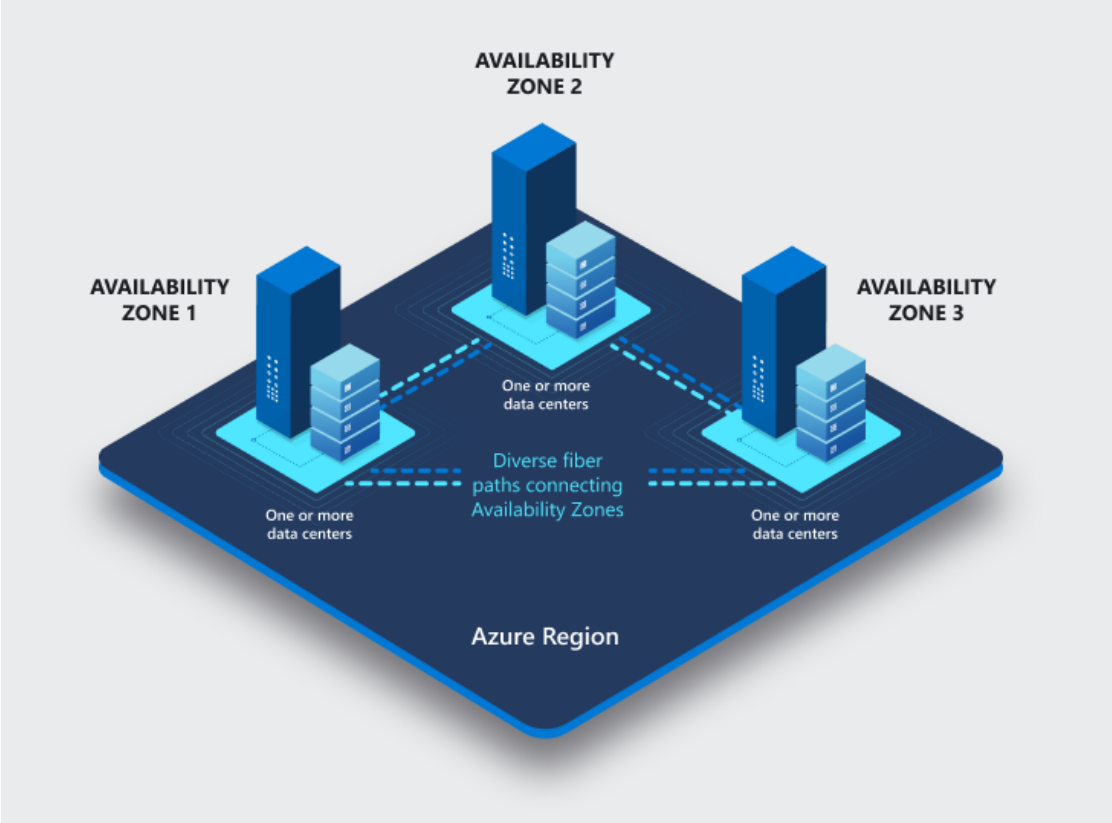

Figura 4.11: Zone di disponibilità in una regione di Azure [27]

Durante lo studio di tesi si è ritenuto opportuno approfondire l'aspetto delle zone di disponibilità perché, oltre ad aumentare la resilienza delle applicazioni, può anche essere utilizzato per uno scenario multi cluster. Il motivo è legato alla posizione fisica che è distinta tra le diverse zone, perciò si possono avere cluster installati in zone diverse della stessa regione ma con un solo control plane ciascuno. Con questa configurazione se un'azienda ha necessità di spostare le applicazioni tra i cluster è necessario usare una strategia di migrazione, perciò si sono identificati degli scenari con le tecnologie e applicazioni usate nel sottocapitolo 4.4.

Si è deciso di usare due cluster OpenShift 4.9.11 con 1 control plane ciascuno, uno installato sulla zone di disponibilità *eu-south-1a* di AWS e uno in *eu-south-1b*, entrambi con 2 worker. I tipi di macchine virtuali richiesti sono gli stessi descritti nel sottocapitolo 4.4.

Gli scenari considerati utilizzano la tecnologia di Velero in versione 1.7.1 e sono così descritti:

1. Due cluster installati nella stessa zona di disponibilità (*eu-south-1a*).
2. Due cluster creati su zone di disponibilità diverse della stessa regione (*eu-south-1a* e *eu-south-1b*).

In entrambi gli scenari è stata provata la migrazione di un'applicazione stateless e una stateful utilizzando le due tecnologie già descritte nel sottocapitolo precedente: sistema di snapshot del cloud provider e Restic. Di seguito sono riportati i commenti per entrambi gli scenari:

1. Considerando una sola zona di disponibilità non si sono riscontrati problemi o limitazioni legati al ripristino all'interno di un secondo cluster. Perciò si può affermare che il ripristino di risorse e dati avviene senza problemi come se si usasse lo stesso cluster, sia con il metodo a snapshot, sia con Restic.
2. Per quanto riguarda l'utilizzo di due zone di disponibilità diverse si è evidenziato un risultato inaspettato. Il sistema di snapshot non risulta funzionare, probabilmente perché Velero non è in grado di selezionare la zona per il ripristino del volume. Infatti un volume Elastic Block Store è utilizzabile solo dalle istanze di macchine virtuali nella zona su cui è stato creato. Mentre si è registrato un risultato positivo con Restic poiché permette la migrazione di risorse e dati in modo semplice e veloce. Inoltre non si sono riscontrati tempi diversi dal ripristino utilizzando un unico cluster, sia per quanto riguarda i volumi con dati di grande dimensione, sia per quelli pieni di piccoli file (4 MB). Nello specifico è stato usato un volume di 50 GiB.

4.5.2 Verso altre regioni

La migrazione delle risorse e dai dati in cluster secondari, situati in luoghi geograficamente lontani, è uno dei metodi più comuni all'interno dei piani di disaster recovery. Infatti, la distanza fisica di dati o servizi è una delle caratteristiche fondamentali per diminuire la probabilità di un disastro che a sua volta influenza negativamente la business continuity. In questa sezione ci si concentra sul DR e quindi sul backup e ripristino, in caso di necessità, di applicazioni su cluster secondari.

In questo sottocapitolo sono stati considerati scenari di migrazione utilizzando uno stesso cloud provider, nello specifico AWS. I più grandi fornitori di servizi cloud offrono la possibilità di eseguire le applicazioni e i servizi nella zona geografica di proprio interesse che può essere scelta secondo le necessità dell'azienda. La caratteristica di avere a disposizione diverse regioni sui cui eseguire i servizi porta a dei vantaggi per il DR. Infatti è possibile adottare strategie di backup a freddo o a tiepido su un cluster secondario che si trovano in una zona geografica diversa da quello principale.

Nello studio di tesi si è deciso di utilizzare un cluster primario nella regione *eu-south-1* e uno secondario in *eu-west-3*, cioè nel data center situato a Parigi. In entrambe le regioni si è utilizzano OpenShift in versione 4.9.11 e Velero 1.7.1. Le applicazioni considerate sono le stesse delle sezioni precedenti e descritte nei sottocapitoli 4.2.1 e 4.2.2; per quanto riguarda il *mysql* si è deciso di utilizzare un volume da 50 GiB. Lo scopo delle prove effettuate è stato quello di verificare se lo strumento tecnologico scelto permette di eseguire il DR su cluster geograficamente distanti, esaminando tempistiche ed eventuali limitazioni.

Il sistema di snapshot, gestito da Velero, non permette il recupero dei volumi su altre regioni che non siano quelle su cui è stato eseguito il backup. Dato che AWS offre la possibilità di copiare gli snapshot in altre regioni, è stata richiesta la copia dello snapshot, creato da Velero, nella regione *eu-west-3* con lo scopo di provare a recuperare i dati anche in quella regione con questa modalità. Dopo aver aggiunto il bucket S3, correttamente configurato su Velero, nel cluster di Parigi, non è stato possibile recuperare il volume dallo snapshot copiato precedentemente. Dopo un attenta analisi si è accurato che Velero non riusciva a identificare lo snapshot poiché in fase di copia AWS ha modificato il suo codice identificativo. Per risolvere questo problema si è deciso di creare manualmente il volume dallo snapshot copiato e successivamente legarlo alla piattaforma di OpenShift creando una risorsa dedicata. Dunque, prima di richiedere il recupero dell'applicazione è necessario creare la risorsa di tipo PVC e il PV legato ad esso aggiungendo questa specifica per selezionare l'ID del volume creato manualmente:

```
spec:
  awsElasticBlockStore:
    volumeID: 'aws://eu-west-3a/vol-0d943a748ff7fb2be'
```

Una volta creato il PV, è possibile richiedere il recupero dell'applicazione escludendo le risorse dei volumi con questo comando:

```
velero restore create mysql-cluster2 --from-backup=mysql
↪  --exclude-resources=pv,pvc
```

Al fine di misurare il tempo richiesto da AWS per il trasferimento dello snapshot tra due diverse regioni è stato creato uno script bash in cui è necessario inserire ogni volta l'identificativo dello snapshot da copiare. Questo si occupa di inviare la richiesta della copia e controlla ogni secondo se questa è stata completata; infine stampa a video il tempo totale impiegato. Di seguito è riportato lo script:

```
#!/bin/bash
snap_id="snap-0f668190c05cbec64"
```

```
hour_now=`date +"%T" --date='1 hour ago'`
date_now=`date +"%F" --date='1 hour ago'`
date="${date_now}T${hour_now}"
start=`date +%s`
aws ec2 copy-snapshot --region eu-west-3 --source-region eu-south-1
↪  --source-snapshot-id ${snap_id} --output text
snap=`aws ec2 describe-snapshots --owner-ids 760012763089 --filters
↪  Name=status,Values=completed --query
↪  "Snapshots[?(StartTime>='${date}')].[SnapshotId]" --output text`
while [ "$snap" = "" ]
do
  sleep 1
  snap=`aws ec2 describe-snapshots --owner-ids 760012763089 --filters
  ↪  Name=status,Values=completed --query
  ↪  "Snapshots[?(StartTime>='${date}')].[SnapshotId]" --output text`
done
end=`date +%s`
echo "copy snapshot with id ${snap_id} took $((end-start)) seconds"
```

I tempi necessari per il ripristino tramite il sistema di snapshot sono quindi modellabili tramite questa formula:

$$T_{r\_tot} = T_{copia} + T_{volume} + T_{velero} + T_{agg} \tag{4.2}$$

dove:

- T_{copia} è il tempo necessario per la copia dello snapshot nella regione su cui si vuole eseguire il ripristino.
- T_{volume} si riferisce alla creazione del volume dallo snapshot e delle risorse in OpenShift di tipo PV e PVC.
- T_{velero} indica il tempo richiesto da Velero per il ripristino delle risorse necessarie.
- T_{agg} è il tempo aggiuntivo, cioè lo stesso descritto per la formula precedente 4.1.

Per quanto riguarda il metodo di DR con Restic, invece, non è necessario eseguire alcuna copia e il ripristino tramite Velero funziona semplicemente acquisendo tutte le informazioni necessarie dall'archiviazione ad oggetti utilizzata. Questo scenario è paragonabile a quello con l'applicazione stateless, poiché sono coinvolte solo le informazioni all'interno del bucket S3. La formula valida per il calcolo del

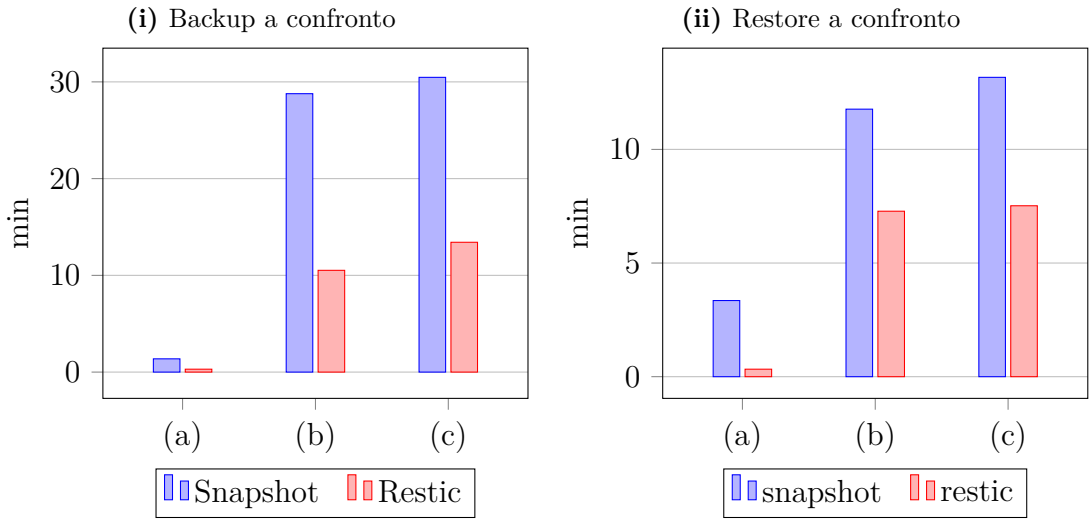

Figura 4.12: Tempistiche di backup a Milano e recupero a Parigi con AWS

ripristino in questi scenari (Restic e applicazioni stateless) rimane quella iniziale, vedi 4.1. Inoltre durante il ripristino delle applicazioni stateless non sono stati rilevati rallentamenti rispetto al recupero dei servizi all'interno dello stesso cluster.

Il grafico 4.12i mostra i tempi registrati per effettuare un backup nella regione *eu-south-1* di AWS (Milano). Si sono riportate le differenze tra le due modalità proposte da Velero (snapshot e Restic) al fine di confrontarle anche per il processo di ripristino. Nel grafico 4.12ii, invece, sono riportati i risultati legati al recupero dell'applicazione stateful nella regione *eu-west-3* (Parigi).
I tre scenari considerati indicano un diverso utilizzo del volume su AWS gestito dall'applicazione stateful e sono qui descritti:

(a) Volume vuoto, cioè con solo i file di *mysql* che occupano circa 115 MiB.

(b) Volume con all'interno un unico grande file che occupa tutto lo spazio disponibile.

(c) Volume interamente occupato da tanti piccoli file. In particolare si è deciso di utilizzare file di dimensione 4 MB.

La creazione dei file avviene con i comandi descritti nel sottocapitolo 4.4.2.

Come si può notare dai dati riportati, i tempi per il DR, su una regione diversa da quella del sito principale, sono migliori con il metodo Restic. Per analizzare meglio il confronto tra le due tecnologie si sono creati due grafici ad hoc. Il primo 4.13i confronta il backup e il ripristino con snapshot e Restic in un unico grafico a barre. Il secondo 4.13ii mostra il tempo necessario per il completamento dell'operazione

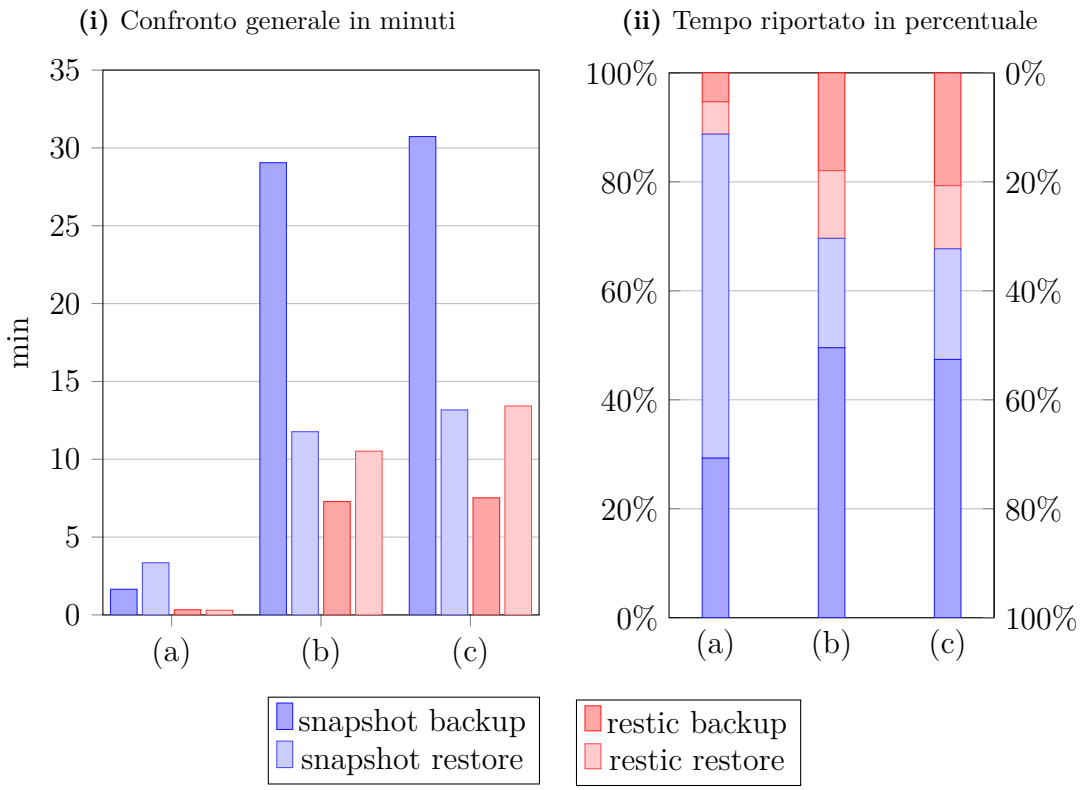

Figura 4.13: Confronto tra snapshot e Restic per la migrazione verso altre regioni

rispetto alle altre con i valori descritti in percentuale sul totale dei quattro tipi di dati. Quest'ultimo grafico vuole evidenziare le differenze di ciascuna operazione di backup e ripristino con i due metodi utilizzati ed è possibile leggerlo prendendo come riferimento l'asse delle ordinate di sinistra per focalizzarsi sugli snapshot o quello di destra per iniziare il confronto con Restic. I colori con tonalità sul blu sono legati al metodo con snapshot, mentre il rosso identifica Restic.

4.5.3 Verso altri cloud provider

Un ultimo scenario considerato per i piani di disaster recovery su cluster a microservizi è la migrazione verso cloud provider diversi da quello utilizzato nel sito principale. Infatti questa strategia è quella che isola di più l'area del guasto con il sito di recupero: oltre ad essere distante geograficamente, non sono presenti servizi condivisi come quelli tra due regioni dello stesso cloud provider. In questo modo è possibile intraprendere soluzioni di DR a freddo e a tiepido molto affidabili poiché considerano molti più scenari di disastro dai quali si è protetti, aumentando dunque la resilienza per la business continuity.

Al fine di analizzare uno scenario multi-cloud, è stato deciso di usufruire dei servizi di AWS ed Azure. Sono quindi stati creati due cluster OpenShift in versione 4.9.11 formati da 1 control plane e 2 nodi worker. Per cercare di utilizzare risorse dello stesso tipo è stato deciso di utilizzare le macchine virtuali *m5a.xlarge* nella regione *eu-south-1* di AWS, mentre su Azure sono state richieste le VM *Standard_D4s_v3* nella regione *germanywestcentral*. Entrambi i tipi di macchine virtuali hanno 4 vCPU e 16 GiB di memoria.
Si è deciso di utilizzare un volume di dimensione pari a 50 GiB l'applicazione stateful di *mysql*. Inoltre, sono stati creati due bucket nelle stesse regioni dei cluster per permettere a Velero, in versione 1.7.1, di salvare i dati sia su AWS che su Azure. I nomi dei bucket sono stati dati seguendo le convenzioni consigliate dai cloud provider:

- `st-velero-aws-eu-south-1`
- `st-velero-azure-germanywestcentral-1`

Per poter ripristinare con successo le applicazioni stateful in un cloud provider diverso è stato necessario aggiungere una informazione a Velero, cioè il tipo di storage utilizzato per il provisioning dei volumi. Infatti se si utilizzano Persistent Volume tramite i dischi offerti dai cloud provider, è necessario configurare correttamente un StorageClass opportuno. Questo processo avviene in automatico tramite l'installazione del cluster, ma la risorsa StorageClass è sicuramente diversa tra cloud provider differenti. Dunque, se si vuole migrare un PV in un contesto multi-cloud è necessario aggiungere, con nomenclatura chiave-valore, una risorsa di tipo ConfigMap nel progetto di Velero specificando come chiave il nome del vecchio StorageClass e come valore quello nuovo. Di seguito è riportato il manifest file di tipo ConfigMap utilizzato per la migrazione da AWS ad Azure.

```
1  apiVersion: v1
2  kind: ConfigMap
3  metadata:
4    name: change-storage-class-config
5    namespace: velero
6    labels:
7      velero.io/plugin-config: ""
8      velero.io/change-storage-class: RestoreItemAction
9  data:
10   gp2: managed-premium
```

Si sono identificati quattro diversi scenari, che evidenziano due strategie di migrazione; in questo modo è stato possibile analizzare diverse configurazioni di DR e verificare l'effettiva funzionalità della soluzione tecnologica scelta.

1. Migrazione da AWS ad Azure:
   - 1.1 utilizzando il bucket situato nella regione di partenza (AWS);
   - 1.2 utilizzando il bucket creato nella regione di destinazione (Azure).
2. Migrazione da Azure ad AWS
   - 2.1 utilizzando il bucket situato nella regione di partenza (Azure);
   - 2.2 utilizzando il bucket creato nella regione di destinazione (AWS).

I risultati sono interessanti e diversi dalle aspettative; essi sono stati riportati nelle figure 4.14 per quanto riguarda la migrazione da AWS ad Azure e 4.15 per la migrazione nel verso opposto. In tutti gli scenari sono stati considerati i servizi utilizzati anche nei sottocapitoli precedenti, in particolare l'applicazione stateless definita in 4.2.1 e quella stateful 4.2.2.

L'applicazione stateless non ha dato problemi durante la fase di ripristino e anche i tempi sono come quelli discussi nel sottocapitolo 4.4.1, perciò si può utilizzare la formula 4.1 per stimare il tempo di recupero.

Per quanto riguarda il DR dell'applicazione stateful si sono riportati i grafici contenteti solo i risultati più significativi, cioè con un volume di dati pieno di file; in particolare utilizzando i due tipi di file descritti nel sottocapitolo precedente: (b) e (c).

Negli scenari considerati si è voluto confrontare i tempi per eseguire backup e recupero dati utilizzando archiviazioni ad oggetti differenti, cioè offerti da fornitori diversi e in due posizioni geografiche diverse. Diversamente da quanto si poteva immaginare, la distanza geografica sembra che non abbia influenzato questo tipo di prova poiché sono registrate differenze dell'ordine di pochi punti percentuali, in media circa 3 %, con eccezione per i risultati riportati nel grafico 4.14i.

Si sono registrate delle anomalie per quando riguarda il backup effettuato sul cluster AWS verso il bucket di Azure. Tramite l'utilizzo del volume con file di tipo (b) la differenza rilevata può essere legata esclusivamente al diverso tipo di bucket; infatti il Blob di Azure non ha mai registrato una velocità di scrittura superiore a 36 MiB/s e quindi un tempo inferiore a 23 min. Analizzando i tempi con i file di tipo (c) si può stilare una seconda ipotesi, più realistica, legata alla funzionalità del bucket; infatti utilizzando file di dimensione 4 MB la tecnologia di archiviazione non usa l'HTBB [28]. La tecnologia di *High-Throughput Block Blob* è stata implementata da Azure per aumentare la velocità di scrittura e gestione delle richieste di PUT ed è abilitata di default sul tipo di archiviazione utilizzato,

cioè Blob Storage. Questa tecnologia, però, funziona solo per richieste PUT di dimensione superiore a 4 MiB e quindi non è attiva con lo scenario considerato.

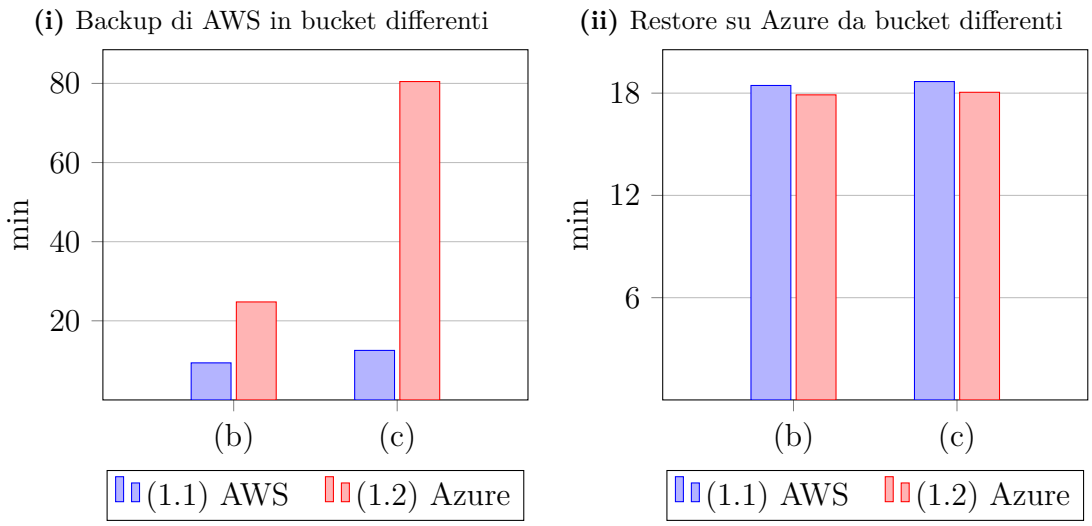

Figura 4.14: Migrazione da AWS ad Azure

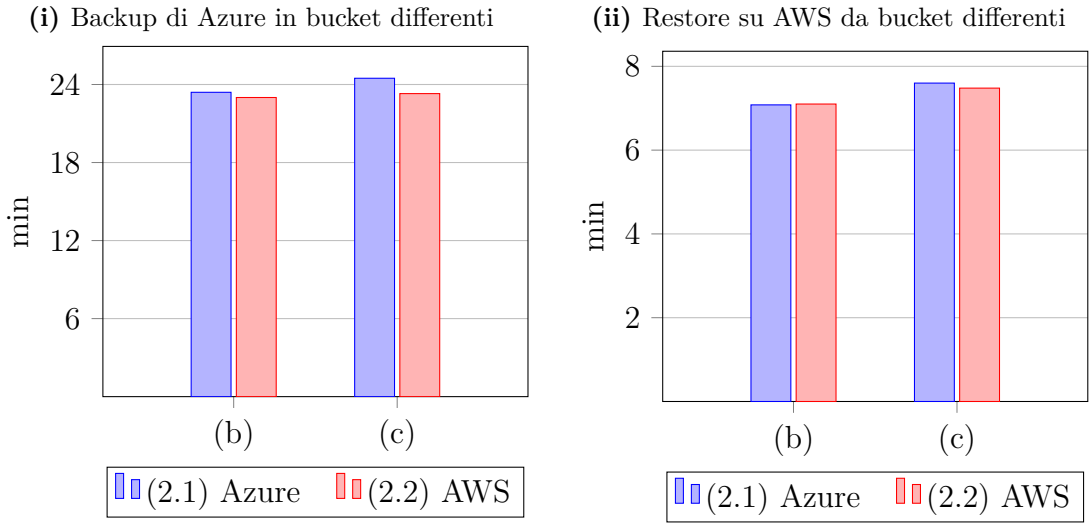

Figura 4.15: Migrazione da Azure ad AWS

Un ulteriore commento è legato alle velocità di scrittura e lettura sui bucket aggregando i risultati per i due cluster utilizzati. Si può notare come il cluster OpenShift installato su AWS riesca a scrivere e leggere circa 2.5 volte più veloce rispetto a quello creato su Azure. Questo dato lo si può acquisire dai tempi di

backup e restore registrati con il bucket di AWS, ma anche dalle tempistiche del recupero tramite il bucket di Azure. Mentre il backup tramite quest'ultima archiviazione ad oggetti risulta essere diverso e verrà commentato di seguito.
Se si analizzano i tempi di backup con il bucket di Azure si può notare un comportamento insolito, cioè tempi molto alti legati al cluster AWS, allo stesso tempo è stato anche l'unico test in cui le risorse del cluster sono state utilizzate meno. Si è registrato infatti un utilizzo di vCPU di circa 0.4 durante il backup del il volume di tipo (b) e di 0.2 vCPU per il (c). Se si esegue un calcolo per ipotizzare un backup con un utilizzo della vCPU al 100 %, si possono trovare dei tempi molto vicini a quelli misurati con il bucket di AWS. Per questo motivo, si è potuto constatare che le prove sono state eseguite con risorse diverse a livello di prestazioni, perciò i dati acquisiti non possono essere utilizzati in modo generalizzato.

In ultimo, si può affermare che i dati registrati in questi tipi di scenari sono soggetti a variazione da tante incognite, tra cui le prestazioni delle VM utilizzate, la velocità dei dischi per i volumi e le capacità legate al sistema di archiviazione ad oggetti utilizzato, oltre che alla posizione geografica. Pertanto si è potuto solo commentare le prove eseguite in ambiente multi-cloud, senza però acquisire dei tempi generalizzabili.

4.6 Costi

Dopo aver analizzato diverse soluzioni per il DR e individuato le loro prestazioni e limitazioni è necessario concludere l'analisi valutandone il costo. In ogni realtà aziendale il costo delle soluzioni proposte ha una grande influenza sulle scelte decisionali, sopratutto quando si parla di soluzioni simili. Questa tesi si focalizza sull'utilizzo di uno strumento open source per eliminare i costi di eventuali licenze, quindi saranno analizzati i prezzi dei vari scenari considerando solo le risorse cloud richiesti da essi.

Le soluzioni intraprese hanno mostrato l'utilizzo di un sistema di snapshot offerto dal cloud provider e da un'archiviazione ad oggetti, chiamata S3 su AWS e Blob storage su Azure. Di seguito, sono discusse queste due tipologie di soluzione, dal punto di vista economico e, successivamente, sono fornite delle formule che modellano i costi del loro utilizzo.

4.6.1 Velero con Restic

Velero ha il compito di salvare le risorse OpenShift su un'archiviazione ad oggetti che può essere utilizzata anche per i dati dei volumi tramite Restic.

In questo sottocapitolo è analizzato il costo di soluzioni di backup che includono l'utilizzo esclusivo dell'archiviazione ad oggetti, quindi per applicazioni stateless oppure stateful tramite l'utilizzo di Restic. In particolare, sono stati studiati i bucket di AWS, chiamati S3, e i Blob storage di Azure.

Il costo di implementazione di Velero non è rilevante poiché si tratta di uno strumento open source. Pertanto, l'analisi dei costi tratta i dati salvati all'interno dei servizi cloud e la loro interazione, cioè le richieste di accesso e il trasferimento dei dati. I prezzi dei servizi dei cloud provider sono definiti in dollari, per semplicità e precisione si è deciso di considerarli senza eseguire la conversione in euro. Sono qui elencate le risorse che necessitano di un pagamento e sono state considerate ai fini del calcolo dei costi:

1. Archiviazione ad oggetti utilizzata per i dati dei backup.
2. Richieste necessarie per interagire con il servizio di archiviazione.
3. Trasferimento dei dati verso altre regioni dello stesso fornitore di servizi.
4. Trasferimento dei dati verso Internet.

Di seguito sono spiegati i costi in modo schematico seguendo l'ordine appena descritto.

1. L'utilizzo dell'archiviazione ad oggetti AWS S3 Standard in *eu-south-1* ha un costo di 0,024 USD per GiB al mese [29], poi vengono applicati alcuni sconti per dimensioni di dati superiori a 50 TiB. Su Azure lo stesso servizio nella regione *germanywestcentral* costa 0,0196 USD per GiB al mese [30], ma è consigliato utilizzare il servizio offerto con ridondanza geografica in una regione secondaria al prezzo di 0,0392 USD per GiB al mese.

2. Un secondo aspetto da considerare è legato alle richieste effettuate per inserire e leggere dati sui bucket. Infatti i cloud provider fatturano anche il numero di interazioni con il servizio, che sono suddivise per tipologie. AWS richiede 0,0053 USD ogni 1000 richieste di tipo PUT, COPY, POST e LIST e 0,0004 USD per tutte le altre. Azure propone 0,0054 USD ogni 1000 richieste di scrittura, questo prezzo sale a 0,0108 USD se si sceglie il sistema con ridondanza geografica; le letture rimangono invariate a 0,00043 ogni 1000 richieste.

3. Il trasferimento dei dati in entrata verso il data center del provider non viene contabilizzato, sia per AWS che per Azure, mentre per il flusso verso l'esterno è a pagamento. I due cloud provider hanno assegnato un prezzo al traffico in uscita verso le altre regioni, si tratta di 0,02 USD per GiB, inoltre Azure aumenta il costo se la regione di destinazione si trova al di fuori dell'Europa e del Nord America.

4. I due cloud provider offrono i primi dati di trasferimento verso Internet gratuiti ogni mese (AWS 100 GiB, Azure 5 GiB), successivamente il costo varia in base al totale del traffico mensile: dai 0,09 USD al GiB ai 0,04 USD.

Di conseguenza, il costo totale mensile legato all'utilizzo del sistema di archiviazione ad oggetti è dato dalla somma di tutti i costi appena elencati:

$$C_{oggetti} = C_{dati} + C_{richieste} + C_{regioni} + C_{Internet} \tag{4.3}$$

Ogni elemento di questa formula deve essere calcolato con il prezzo del cloud provider considerato. Inoltre, tutti i costi sono frutto di un semplice conteggio di risorse utilizzate moltiplicando il prezzo dichiarato dal fornitore, ad eccezione del primo elemento in cui è necessario suddividere il mese in ore e calcolare il costo totale in base ai dati occupati in ogni ora. Per chiarezza si riporta un esempio.

Supponiamo di eseguire un backup di un volume con 100 GiB di dati occupati in AWS S3, dopo 15 giorni si esegue un secondo backup di un altro volume con 150 GiB. Ipotizzando che il mese in questione abbia 31 giorni, il consumo totale di spazio sarà la moltiplicazione tra i dati salvati per le ore di utilizzo, quindi:

$100\,\text{GiB} \cdot 15 \cdot 24\,\text{h} + 150\,\text{GiB} \cdot 16 \cdot 24\,\text{h} = 93{,}600\,\text{B h} \cdot 2^{30}$

Successivamente il totale viene diviso per il numero di ore del mese e moltiplicato per il costo mensile al GiB (ad esempio per AWS è pari a 0.024 USD):

$93{,}600\,\mathrm{B\,h} \cdot 2^{30} \cdot C_{storage}/744\,\mathrm{h} = 3.02$ USD

Per semplificare il calcolo del consumo di spazio al mese nell'archiviazione ad oggetti è stata ideata una formula in cui vengono considerati tutti i periodi in cui la dimensione occupata cambia. Si tratta di una sommatoria di N periodi:

$$C_{dati} = C \cdot (\sum_{k=1}^{N} D_k \cdot T_k)/H_{mese} \tag{4.4}$$

dove:

- C è il costo mensile al GiB per i dati occupati all'interno dell'archiviazione ad oggetti;
- D_k è la dimensione dello spazio utilizzano nel periodo considerato in GiB;
- T_k è il tempo in cui sono salvati i dati del periodo k, esplicitato in ore;
- H_{mese} sono le ore totali del mese considerato.

Le ultime considerazioni riguardano gli scenari identificati durante lo studio di tesi. Infatti, il costo può variare in base al tipo di applicazione e di DR scelto. Il backup di applicazioni stateless occupa pochi dati, per cui in caso d'implementazione di migrazioni in siti diversi il costo per il salvataggio e per il trasferimento sarà ridotto. Per quanto riguarda le applicazioni stateful, il costo della soluzione può cambiare se si sceglie di implementare un DR in locale o in un sito secondario. Pertanto, per soluzioni che includono migrazioni all'interno dello stesso cloud provider si applicheranno costi per il trasferimento dei dati. Se invece si considera uno scenario multi-cloud, allora i costi di trasferimento saranno maggiori. Inoltre i prezzi legati al trasferimento possono essere ancora maggiorati se si utilizza un bucket per il backup situato fuori dalla regione su cui è installato il cluster OpenShift.

Esempio

Di seguito viene proposto un esempio di strategia per il backup di un'applicazione stateful in un cluster installato su AWS *eu-south-1*. Si considera un backup giornaliero di un volume con ritenzione di 7 giorni e con 100 GiB di dati occupati. Ogni giorno sono modificati 10 GiB. Il backup, eseguito con Restic, è salvato su un bucket della stessa regione. Inoltre si aggiunge anche un ulteriore backup settimanale su un bucket in un'altra regione (*eu-west-3*) con ritenzione di 4 settimane. In questo caso i costi sono:

- C_{dati}: si può calcolare in modo abbastanza preciso, dividendo un ipotetico mese di 31 giorni nei periodi necessari al calcolo e utilizzando la formula 4.4. Per il primo backup la sommatoria dei dati salvati con le ore associate risulta essere 114,000 GiB h, mentre per il secondo è opportuno aggiungere una considerazione iniziale. Se i 10 GiB di dati modificati ogni giorno si ipotizzano essere circa 50 GiB per l'intera settimana, il backup incrementale dovrà considerare quei dati in più rispetto a quelli già salvati, perciò il calcolo dei dati per le ore risulta essere pari a 135,600 GiB h. In entrambe le regioni i backup hanno un costo di 0.024 USD al GiB, quindi il totale corrispondente è 3.68 USD e 4.37 USD.

  Questo calcolo è valido per il primo mese, per i successivi si considerano i dati occupati che sono stati raggiunti a regime, perciò 160 GiB per il primo backup e 250 GiB per il secondo, con un costo corrispondente di 3.84 USD e 6 USD.

- $C_{richieste}$: purtroppo è difficile prevedere le richieste inviate, in base all'esperienza maturata con gli scenari utilizzati, si può stimare circa 0.5 USD.

- $C_{regioni}$: si considera solo il secondo backup che si compone di un primo trasferimento di 100 GiB e di quattro successivi per i backup incrementali. Dunque, si consumano 300 GiB in trasferimento dati ogni mese verso la regione di Parigi al costo di 0.02 USD al GiB, per un totale di 6 USD.

- $C_{Internet}$: in questa soluzione si utilizza un solo cloud provider e non si avrà un trasferimento di dati periodico verso altri bucket appartenenti a fornitori diversi.

In totale la soluzione di esempio costerà $3.68 + 4.37 + 0.5 + 6 = 14.55$ USD per il primo mese e $3.84 + 6 + 0.5 + 6 = 16.34$ USD per i mesi successivi.

4.6.2 Velero con snapshot

La funzionalità di backup, chiamata snapshot, è offerta dai cloud provider come un servizio completo a livello tecnologico e consente di salvare i dati dei volumi desiderati. Per analizzare i costi di questa soluzione sono state consultate le documentazioni dei cloud provider utilizzati ed i loro sistemi per il calcolo dei prezzi, in particolare su AWS [29] e Azure [30]. I prezzi dei servizi dei cloud provider sono definiti in dollari, per semplicità e precisione si è deciso di considerarli senza eseguire la conversione in euro.

AWS definisce il costo degli snapshot per GiB di dati archiviati nel volume su cui è stato richiesto il backup, quindi non si considera nella fatturazione la dimensione

del volume stesso, ma solo i dati occupati. In particolare, il prezzo su AWS è 0.0525 USD per GiB al mese di dati archiviati, mentre su Azure è pari a 0.0500 USD.

Velero può richiedere il backup del PV tramite il sistema di snapshot, però i dati delle risorse sono salvati in un'archiviazione ad oggetti. Per questo motivo il calcolo dei costi in tale modalità deve tenere conto non solo del servizio di snapshot, ma anche di tutti gli elementi discussi nel sottocapitolo precedente 4.6.1. La formula per il costo totale della soluzione si compone di due parti: quello relativo all'archiviazione ad oggetti e quello relativo al servizio di snapshot.

$$C_{tot} = Coggetti + Csnapshot \tag{4.5}$$

Il costo degli snapshot è calcolato solamente dai dati presenti sui volumi e poiché il backup rimane nella stessa zona di disponibilità non risultano esserci altri elementi di pagamento aggiuntivi. Invece, se la soluzione di DR richiede il trasferimento degli snapshot in altre regioni, allora questo sarà fatturato in base ai dati trasferiti tra le regioni. Successivamente, dopo la copia, sono applicate le tariffe standard degli snapshot per l'archiviazione nella regione di destinazione.

Di seguito è riportata la formula ideata per il calcolo del costo totale legato agli snapshot. In particolare è stata considerata una sommatoria di tutti i volumi considerando la dimensione dei dati occupati e i dati aggiunti per ogni nuova richiesta di backup.

$$C_{snapshot} = \sum_{v=1}^{N} D_v \cdot C + (S_v - 1) \cdot I_v \cdot C \tag{4.6}$$

I valori utilizzati nella sommatoria sono così definiti:

- C: costo al GiB al mese del servizio di snapshot.
- N: numero totale di volumi.
- D_v: dimensione dei dati occupati all'interno dello specifico volume.
- S_v: numero di snapshot al mese. Non si intende gli snapshot eseguiti, ma quelli mantenuti in totale in un mese.
- I_v: dimensione di incremento, cioè il numero in GiB di dati che sono modificati o aggiunti in media tra due richieste di backup.

Per quanto riguarda il processo di recupero dati, non è stato analizzato il costo poiché è un evento raro e non periodico, perciò non ha molta influenza a livello economico. Si vuole solo evidenziare che il prezzo per il recupero eventuale dei dati dipende molto dalla strategia di DR adottata. Se si considera solo il processo di ripristino, allora il costo sarà legato solo alle risorse e ai dati replicati.

Esempio

Qui si vuole riproporre la soluzione di esempio descritta alla fine del sottocapitolo 4.6.1 per stimare un costo utilizzando il sistema di snapshot.

Per semplicità di lettura viene riportata in sintesi la strategia di backup adottata. Viene richiesto un backup giornaliero di un volume con ritenzione di 7 giorni e con 100 GiB di dati occupati e ogni giorno vengono modificati in media 10 GiB. Si aggiunge un secondo backup settimanale, con ritenzione di 4 settimane, salvato nella regione di Parigi, perciò si deve richiedere una copia dello snapshot in una seconda regione. In questo scenario i costi per l'utilizzo dell'archiviazione ad oggetti, legato al backup delle risorse, sono così distribuiti:

- C_{dati}: i dati sono legati esclusivamente alle risorse e perciò la loro dimensione è molto ridotta, si stima essere circa 80 KiB per ogni richiesta di backup (ad esempio, un backup di tutte le risorse del cluster utilizzato occupa uno spazio di 15 MiB). La sommatoria dei dati salvati per ciascun periodo risulta pari a 376,320 KiB h per il primo backup ed a 181,440 KiB h per il secondo. Perciò il costo totale mensile è irrisorio e pari a $0.00001 + 0.000005$ USD.

  La stessa valutazione può essere fatta per i mesi successivi; infatti il costo rimane sempre inferiore a 0.01 USD.

- $C_{richieste}$: è difficile prevedere le richieste inviate, ma dato che i dati inviati sono davvero pochi rispetto all'utilizzo di Restic si può escludere questo costo dal totale.

- $C_{regioni}$: i dati inviati alla regione di Parigi sono legati solo alle risorse per il secondo backup che esegue cinque trasferimenti di dati, perciò in totale si hanno 400 KiB. Il costo è molto inferiore a 0.01 USD al mese, perciò non verrà considerato.

- $C_{Internet}$: in questa soluzione si utilizza un solo cloud provider, perciò non si avrà un trasferimento di dati periodico verso altri bucket appartenenti a fornitori diversi.

Invece, per quanto riguarda il sistema di snapshot i costi sono legati ad un unico volume. Di seguito sono stabiliti i numeri da applicare alla formula degli snapshot per il primo backup:

- D_1: i dati del volume occupati sono 100 GiB.

- S_1: il numero di snapshot mantenuti dal sistema ogni mese è pari a 7.

- I_1: la dimensione di incremento tra due richieste di backup è uguale a 10 GiB, cioè i dati modificati ogni giorno.

Il costo per i backup successivi al primo risulta essere 8.4 USD. Per sapere il prezzo del primo backup si può utilizzare la formula 4.4 che permette di calcolare i dati utilizzati dividendo i periodi tra ogni richiesta di backup. Il risultato è un prezzo uguale a 8.04 USD.

Per il secondo backup si può implementare una soluzione che consente solo la copia dello snapshot nella regione desiderata, lasciando a Velero il compito di salvare le risorse nell'archiviazione ad oggetti di Parigi senza includere i PV. Il trasferimento dei dati di Velero e degli snapshot verso la seconda regione risulta essere circa di 300 GiB se AWS non invia due volte gli stessi blocchi, come avviene per i backup incrementali; altrimenti, il totale dei dati trasferiti aumenta a 500 GiB. Il prezzo totale per le due ipotesi corrisponde a 6 USD oppure 10 USD.
Dopo la copia dello snapshot sono applicati i costi per il salvataggio dei dati in quella regione, perciò si deve calcolare una seconda volta la formula 4.6 con i seguenti elementi:

- D_1: i dati del volume occupati sono 100 GiB.
- S_1: il numero di snapshot mantenuti dal sistema ogni mese è pari a 4.
- I_1: la dimensione di incremento tra due richieste di backup è uguale a 50 GiB, cioè i dati modificati ogni settimana.

Il costo degli snapshot nella seconda regione è di 0.053 USD per GiB, perciò il totale per il salvataggio dei dati al mese risulta essere 13.25 USD, mentre per il primo mese è di soli 9.66 USD. Nel caso in cui vengano salvati anche i blocchi uguali (la seconda ipotesi precedentemente spiegata), allora il costo non terrà conto degli incrementi e il prezzo al mese sarà di 19.72 USD.

Il costo totale per l'implementazione della strategia di backup proposta è di $8.04 + 6 + 9.66 = 23.7$ USD per il primo mese e di $8.4 + 6 + 13.25 = 27.65$ USD per i successivi. Mentre se AWS non supporta la copia differenziale (come per il backup), il prezzo sale a $8.4 + 10 + 19.72 = 37.76$. Dopo un test effettuato si è riscontrato che il tempo di trasferimento di un backup eseguito in modo incrementale è stato molto inferiore a quello completo. A seguito di tale considerazione si può ipotizzare che AWS copi solo i blocchi di dati non presenti all'interno di altri snapshot salvati nella regione di destinazione.

4.7 Valutazioni su Velero e strumenti alternativi

La soluzione tecnologica scelta in questo studio di tesi è stata Velero poiché si presenta come uno strumento che permette tutto ciò di cui si ha bisogno per eseguire un backup su cluster OpenShift. Dopo le diverse soluzioni adottate, ed un considerevole approfondimento sulle sue capacità, sono state evidenziate diverse limitazioni, le quali devono essere considerate nel momento di adozione di tale soluzione. Infatti, esse possono essere limitanti oppure superflue in base alle necessità dell'azienda, al piano di DR che stabilisce e al tipo di applicazioni eseguite.

Di seguito sono elencati i limiti di Velero emersi durante l'utilizzo di questo strumento:

- **Scalabilità verticale**: è stato instaurato un test apposito per verificare questa caratteristica con Restic. Il risultato non è stato molto apprezzato poiché si sono evidenziati miglioramenti solo con 2 vCPU e in certe condizioni specifiche. Per quanto riguarda il processo di recupero dati non si sono registrati tempi particolarmente migliori. Inoltre il backup tramite snapshot è gestito dal cloud provider, quindi non è possibile modificarne le specifiche per aumentarne le prestazioni.

  In ultimo, è necessario evidenziare che il backup tramite Restic esegue la scansione di ciascun file in un singolo thread, perciò la ricerca di file duplicati o differenze tra il backup precedente richiede un tempo elevato oltre ad essere poco scalabile verticalmente.

- **Scalabilità orizzontale**: è legata all'aumentare del numero di volumi e di nodi del cluster. Sono state eseguite alcune prove per verificarne l'efficacia, dato che è un parametro ritenuto fondamentale per la tecnologia moderna. Purtroppo Velero non implementa una strategia per l'esecuzione in parallelo dei processi, questi sono messi in coda ed eseguiti in modo sequenziale. Questo aspetto è in fase di sviluppo e se implementato dalla community potrebbe portare molto benefici.

  Un secondo aspetto della scalabilità orizzontale si riferisce al numero di nodi, cioè la possibilità di eseguire in contemporanea il backup di più volumi legati ad istanze di VM diverse. Anche questo scenario è stato deludente poiché i due PV sono salvati in modo sequenziale anche se potenzialmente il DaemonSet di Restic sarebbe stato in grado di eseguirli in parallelo senza conflitti.

- **Snapshot legato alla zona di disponibilità**: il sistema di snapshot offerto dai cloud provider risulta essere molto efficace durante la fase di recupero del volume, purtroppo questa metodologia è utilizzabile solo sui DR che pianificano

il recupero all'interno della stessa zona di disponibilità della regione. Si è provato a spostare manualmente lo snapshot su altre regioni ma ciò non è implementato direttamente su Velero ed è un po' "macchinoso".

- **Snapshot verso altri cloud provider**: la copia degli snapshot è utilizzabile solo all'interno dello stesso fornitore di servizi cloud, perciò se il DR ha un'ottica multi-cloud è necessario utilizzare univocamente Restic.

- **Utilizzo di diverse posizioni per il backup**: purtroppo Velero non permette di inviare i dati di un unico backup a più di un backup-location, perciò se il DR richiede ridondanza dei backup è necessario creare due risorse di tipo Schedule che salvano i dati nei bucket desiderati.

- **Risorse già presenti in fase di recupero**: Velero in versione 1.7.1, durante la fase di ripristino, non sostituisce le risorse già presenti e non permette di decidere il suo comportamento nei loro confronti, ovvero se mantenerle o sostituirle.

- **Numero di nodi inferiore per il recupero**: si è provato migrare un'applicazione che richiedesse più risorse di quante ne avesse disponibili il cluster di destinazione: in questo caso, il recupero è andato a buon fine ma alcuni Pod sono rimasti in stato di Pending.

- **Crittografia parziale**: Restic rende sicuri i file del backup eseguendo una crittografia di tipo AES-256 in counter mode (CTR), ma Velero utilizza una chiave univoca e statica per la gestione dei file, perciò chiunque ha accesso al bucket potrebbe riuscire a leggere i dati. Ragione per cui è necessario gestire in modo appropriato le autorizzazioni del bucket.

- **Versioni diverse**: la documentazione di Velero consiglia di usare la stessa versione del cluster e dello strumento di backup all'interno di uno scenario di migrazione. Durante la tesi è stato sperimentato con successo anche il ripristino di applicazioni su versioni di Kubernetes diverse dall'origine. Il problema di utilizzare cluster in versione diversa è dovuto principalmente alle *apiVersion* delle risorse e alla loro retrocompatibilità. Per questo motivo, Velero propone un'opzione, ancora in sviluppo, che consente di mitigare e ridurre i problemi di compatibilità, di nome *EnableAPIGroupVersions*. Si consiglia, inoltre, di verificare sempre le differenze tra le versioni dei due cluster tramite i changelog di Kubernetes.

Questo elenco di limitazioni appena descritto non vuole essere un modo per rifiutare Velero come soluzione tecnologica, ma è utile sottolineare i compromessi e i limiti che possono essere accettati o meno in base alle esigenze dell'azienda. Nel

complesso, Velero si può valutare come uno strumento che permette di eseguire diversi scenari di DR con successo su cluster a microservizi e anche in ottica multi-cloud, perciò si può definire potente e versatile.

Infine, si vuole sottolineare nuovamente come la tecnologia open source sia stata scelta sia per un aspetto economico sia per la tipologia di DR studiata, cioè legata ai backup a tiepido ed a freddo. Per quanto riguarda i DR a caldo o soluzioni che offrono anche un supporto tecnico è necessario orientarsi sulle offerte enterprise. Di seguito si citano alcuni strumenti interessanti e attraverso i quali si potrebbe continuare questo studio.

TrilioVault è una piattaforma di protezione dati per cluster Kubernetes utilizzabile in ambienti di cloud pubblico e ibrido [31]. Si tratta di uno strumento molto simile a Velero ma sotto alcuni aspetti migliori, ad esempio si può impedire agli utenti di cancellare un backup per un certo periodo di tempo. Questa caratteristica evita un'eliminazione involontaria dei dati anche da parte di un *admin* oppure di un attaccante esterno. Dal punto di vista della sicurezza, offre anche la Multi-Factor Authentication per proteggere ulteriormente l'accesso ai dati. A differenza di Velero, questa piattaforma permette il salvataggio contemporaneo dello stesso backup su diverse posizioni di archiviazione e offre anche una piattaforma web per l'interazione e la gestione dei backup.

Portworx PX-Backup è un'altra piattaforma enterprise che consente il backup ed il ripristino di applicazioni cloud native [32]. Si presenta con un'interfaccia web evoluta ed intuitiva che permette di interagire con il servizio e gestire comodamente tutti i backup. Offre tutto ciò che è stato sperimentato con Velero e aggiunge alcune funzionalità come la gestione delle regole per selezionare in modo veloce e preciso le risorse da includere nei backup. Una peculiarità è la possibilità di replicare in modo sincrono un cluster in un'altra posizione geografica relativamente vicina, applicando così strategie di DR a caldo che richiedono uno zero RPO.

Capitolo 5

Conclusioni

In questa tesi sono stati analizzati diversi scenari che affrontano il problema del disaster recovery di cluster a microservizi. Le diverse strategie adottate sono state sviluppate e modellate in base alle esigenze richieste delle aziende moderne, cioè in ottica multi e hybrid cloud senza dimenticare i costi necessari per l'implementazione. Dopo aver analizzato il funzionamento dei piani di business continuity aziendali e dei backup di applicazioni cloud native, si sono approfondite diverse strategie adottabili tramite lo strumento open source Velero.

Sono state proposte ed analizzate diverse soluzioni che hanno permesso di stabilirne le potenzialità, tempistiche e differenze di ciascuna. I risultati sono positivi secondo alcuni punti di vista e più deludenti secondo altri.

Per quanto concerne le applicazioni stateless, queste sono molto veloci da salvare e da ripristinare, inoltre il loro backup richiede poco spazio di archiviazione; per tale motivo è possibile stimare un basso RTO ed è stata modellata una formula che ne aiuta il calcolo. Il problema riscontrato con l'applicazione di prova è legato alla creazione e distribuzione del LoadBalancer richiesto al provider che ha registrato un tempo variabile e non prevedibile.

Se un'azienda utilizza servizi con dati persistenti, allora, i tempi di backup cambiano e crescono linearmente all'aumentare della dimensione occupata. Invece, i tempi richiesti dal ripristino dell'applicazione stateful dipendono strettamente dalla tecnologia adottata e dal tipo di file presenti nel volume. Infatti, se si utilizza il sistema di snapshot si possono registrare tempistiche molto basse e prevedibili, ma questa tecnica è utilizzabile solo all'interno della stessa zona del cloud provider. Mentre la soluzione implementata con Restic consente anche il ripristino dell'applicazioni in regioni diverse o in cloud provider differenti. Le tempistiche sono più elevate, seppur prevedibili, ma inferiori a quelle richieste dal processo di backup. Se all'interno del volume sono presenti solo file di piccole dimensioni, circa 4 MB ciascuno, si sono registrati peggioramenti dal 3 % al 30 % sul tempo di backup e ripristino in entrambe le strategie utilizzate.

In un contesto enterprise, i risultati ricavati dalle prove di backup sono un buon indizio per lo studio di una soluzione di DR da applicare in base al RPO richiesto. Invece, per quanto riguarda il recupero, i tempi necessari per il calcolo dell'indice di RTO sono soggetti alla variazione di molte incognite, tra cui le prestazioni delle VM utilizzate, la velocità dei dischi per i volumi, le capacità legate al sistema di archiviazione ad oggetti ed il cloud provider utilizzato. Seppur ci siano così tante incognite, si può stimare un tempo di RTO in base alla dimensione dei dati da salvare e verificarlo dopo una prova di recupero dati.

In ultimo, si sono analizzati i costi delle soluzioni adottate e si sono stabilite delle formule generali per permettere un calcolo veloce ed efficace del prezzo mensile. Inoltre, tramite un esempio di configurazione, è stato evidenziato come la soluzione con Restic sia più economica e flessibile. Questa valutazione, però, non identifica Velero utilizzato insieme a Restic come soluzione adatta a tutti. Infatti il recupero dei dati è molto più lento rispetto al sistema di snapshot implementato dal cloud provider, che a sua volta è più costoso di circa 2 volte.

Nonostante Velero abbia molte limitazioni può essere considerato come una soluzione economica e sicura di disaster recovery di tipo asincrono, applicabile a molti cluster di medie/piccole dimensioni. Si vuole nuovamente evidenziare il limite sulla scalabilità, ritenuto molto importante per le applicazioni moderne. Nel caso in cui, in futuro, la community di Velero implementerà questa caratteristica, allora lo strumento open source potrà essere ritenuto molto valido per la maggior parte delle necessità aziendali.

Bibliografia

[1] Doug Jones. *Containers vs. Virtual Machines (VMs): What's the Difference?* 2018. URL: https://www.netapp.com/media/Screen-Shot-2018-03-20-at-9.24.09-AM_tcm19-56643.png (cit. a p. 4).

[2] Gartner. *Four Trends Are Shaping the Future of Public Cloud*. Ago. 2021. URL: https://www.gartner.com/en/newsroom/press-releases/2021-08-02-gartner-says-four-trends-are-shaping-the-future-of-public-cloud (cit. a p. 5).

[3] Red Hat. *Cos'è il cloud ibrido?* Mar. 2018. URL: https://www.redhat.com/it/topics/cloud-computing/what-is-hybrid-cloud (cit. a p. 7).

[4] Flexera. *Flexera 2021 State of the Cloud Report*. 2021. URL: https://resources.flexera.com/web/pdf/report-cm-state-of-the-cloud-2021.pdf (cit. a p. 7).

[5] *I componenti di Kubernetes*. 2020. URL: https://kubernetes.io/it/docs/concepts/overview/components (cit. a p. 10).

[6] Red Hat. *OpenShift Container Platform 4.9 Documentation*. Gen. 2022. URL: https://docs.openshift.com/container-platform/4.9 (cit. alle pp. 11, 13, 37, 39, 41, 77).

[7] Heather Adkins, Betsy Beyer, Paul Blankinship, Piotr Lewandowski, Ana Oprea e Adam Stubblefield. *Building Secure & Reliable Systems: Best Practices for Designing, Implementing and Maintaining Systems*. 1005 Gravenstein Highway North, Sebastopol, CA 95472: O'Reilly Media, Inc., 2020 (cit. alle pp. 22, 23).

[8] Marek.Z. *RPO, RTO, WRT, MTD... WTH?!* 2013. URL: https://defaultreasoning.com/2013/12/10/rpo-rto-wrt-mtdwth (cit. a p. 24).

[9] *Business continuity: la definizione di un concetto chiave nell'era digitale*. Feb. 2022. URL: https://www.bucap.it/news/approfondimenti-tematici/gestione-del-magazzino/definizione-business-continuity-concetto-digitale.htm (cit. a p. 26).

[10] Leonardo La Polla. *Business Continuity Plan: perché è importante?* Feb. 2022. URL: https://www.extrasys.it/it/red/business-continuity-plan (cit. a p. 26).

[11] Steve Kaelble. *Kubernetes Backup & Recovery*. 111 River St.Hoboken, NJ 07030-5774: John Wiley & Sons, Inc., 2021 (cit. alle pp. 30, 32).

[12] Kevin Hinterlong. *3-2-1 Backup Rule*. Gen. 2021. URL: https://edercasellait.com/3-2-1-backup-rule (cit. a p. 30).

[13] *kubernetes volumes*. Feb. 2022. URL: https://kubernetes.io/docs/concepts/storage/volumes (cit. a p. 38).

[14] *Red Hat OpenShift Data Foundation*. Feb. 2022. URL: https://cloud.redhat.com/products/container-storage/ (cit. a p. 39).

[15] *What is the difference between a cold, warm and hot disaster recovery site?* Feb. 2022. URL: https://www.otava.com/blog/what-is-the-difference-between-a-cold-warm-and-hot-disaster-recovery-site (cit. a p. 44).

[16] *Disaster Recovery in Cloud Computing*. Feb. 2022. URL: https://www.otava.com/reference/the-benefits-of-disaster-recovery-in-cloud-computing (cit. a p. 44).

[17] Michael Ferranti. *How to achieve Disaster Recovery (DR) for Red Hat OpenShift*. Mag. 2020. URL: https://portworx.com/blog/openshift-disaster-recovery (cit. a p. 46).

[18] Aaron Miller e Federica Ciuffo. *Backup and restore your Amazon EKS cluster resources using Velero*. Dic. 2021. URL: https://aws.amazon.com/it/blogs/containers/backup-and-restore-your-amazon-eks-cluster-resources-using-velero (cit. a p. 48).

[19] *What is DRaaS? (Disaster recovery as a service)*. Feb. 2022. URL: https://www.acronis.com/en-us/articles/draas (cit. a p. 48).

[20] Red Hat. *Stateful e stateless*. Mar. 2020. URL: https://www.redhat.com/it/topics/cloud-native-apps/stateful-vs-stateless (cit. alle pp. 51, 52).

[21] *Stateless Applications*. Giu. 2021. URL: https://kubernetes.io/docs/tutorials/stateless-application (cit. a p. 51).

[22] *Run a Single-Instance Stateful Application*. Feb. 2021. URL: https://kubernetes.io/docs/tasks/run-application/run-single-instance-stateful-application (cit. a p. 52).

[23] *Velero documentation*. Feb. 2022. URL: https://velero.io/docs/v1.7 (cit. alle pp. 55, 57, 62, 72).

[24] *Support running multiple velero backups/restores concurrently*. Mar. 2022. URL: https://github.com/vmware-tanzu/velero/issues/487 (cit. a p. 74).

[25] *Amazon EBS snapshots*. Mar. 2022. URL: https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/EBSSnapshots.html (cit. a p. 74).

[26] *Regioni e zone di disponibilità*. Mar. 2022. URL: https://aws.amazon.com/it/about-aws/global-infrastructure/regions_az (cit. a p. 77).

[27] *Aree e zone di disponibilità*. Mar. 2022. URL: https://docs.microsoft.com/it-it/azure/availability-zones/az-overview (cit. a p. 78).

[28] *High-Throughput with Azure Blob Storage*. Mar. 2019. URL: https://azure.microsoft.com/it-it/blog/high-throughput-with-azure-blob-storage (cit. a p. 85).

[29] *Prezzi AWS*. Mar. 2022. URL: https://aws.amazon.com/it/pricing (cit. alle pp. 89, 91).

[30] *Prezzi di Azure*. Mar. 2022. URL: https://azure.microsoft.com/it-it/pricing (cit. alle pp. 89, 91).

[31] *TrilioVault for Kubernetes*. Mar. 2022. URL: https://trilio.io/products/triliovault-kubernetes (cit. a p. 97).

[32] *Portworx PX-Backup*. Mar. 2022. URL: https://portworx.com/products/px-backup (cit. a p. 97).