ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Scuola di Ingegneria e Architettura
Dipartimento Informatica – Scienza e Ingegneria
Corso di Laurea Magistrale in Ingegneria Informatica

PROGETTAZIONE DI UN CLUSTER SUL CLOUD CON IL FRAMEWORK HPC

Tesi di Laurea in Tecnologie e Sistemi per la Gestione di Basi di Dati e Big Data M

Laureando
Michele Nigro

Relatore
Chiar.mo Dott. Marco Patella

Correlatore
Gianpaolo Gottardi

III Sessione
ANNO ACCADEMICO 2018-2019

INDICE

INTRODUZIONE

Obiettivo di questo lavoro di tesi è consentire migrazione completa sul cloud e pieno controllo dell'infrastruttura di calcolo distribuito fornita da Prometeia, totalmente basata sulle tecnologie Microsoft. In particolare, le tecnologie utilizzate per la migrazione richiedono l'uso della piattaforma cloud Microsoft Azure: attraverso il servizio IaaS (in cui le risorse sono offerte a livello di calcolo, *storage* e rete) da essa fornito è possibile configurare un insieme di macchine virtuali in qualità di risorse di calcolo, messe a disposizione per l'elaborazione concorrente di flussi di lavoro e contesti applicativi appositamente progettati. Fornendo un ambiente esecutivo di questo tipo, al cliente vengono offerte risorse computazionali IaaS gestite come se fossero PaaS (ovvero senza alcuna gestione infrastrutturale e di configurazione) a tutti gli effetti: l'utente, infatti, non deve preoccuparsi di alcuna impostazione manuale delle macchine virtuali disponibili, in quanto questo compito è delegato al modello di automazione di cui Azure dispone.

La struttura di questa tesi è organizzata in modo da offrire innanzitutto una panoramica di Prometeia, azienda presso la quale è stato svolto il tirocinio che ha prodotto i risultati qui illustrati e il cui interesse per le tecnologie più avanzate ha consentito la realizzazione di questo progetto. I due capitoli successivi illustrano, invece, un quadro approfondito degli strumenti usati per l'attuazione del lavoro descritto. In particolare, nel Capitolo 2 sono mostrati Microsoft Azure e Windows Powershell, mentre il Capitolo 3 verte sul framework HPC realizzato da Microsoft. Il Capitolo 4 presenta i dettagli tecnici dell'implementazione apportata dall'attività svolta in tutti i suoi passi tramite i mezzi identificati nei capitoli precedenti. Infine, il Capitolo 5 ha l'obiettivo di mostrare i risultati conclusivi e i vantaggi apportati dalla soluzione progettata.

La prima fase di sviluppo del progetto, pertanto, è consistita nella realizzazione dell'intera infrastruttura dedicata. Usando i servizi di Azure per la creazione di risorse (macchine virtuali, *networking*, archiviazione), è possibile automatizzare l'installazione di un insieme (*cluster*) di server dedicati al calcolo distribuito: a partire dagli applicativi sviluppati da Prometeia, quindi, verrà illustrato l'intero processo di configurazione di una rete di nodi di calcolo, ciascuno adibito all'esecuzione di una

funzione specifica. Azure, infatti, consente di applicare la distribuzione delle risorse necessarie con un'unica operazione, mediante quello che viene definito modello di risorse di Azure Resource Manager: si tratta di un file in formato .json che, con una sintassi dichiarativa appositamente studiata, stabilisce le istruzioni da eseguire per la realizzazione dell'infrastruttura. Il punto di partenza, in questo senso, è stato individuato nel progetto condiviso da Sunbin Zhu (ingegnere presso Microsoft) tramite la piattaforma Github. A partire da qui, pertanto, lo sviluppo della soluzione personalizzata è stata frutto di un lavoro di modifica delle risorse specificate, adattate allo specifico contesto di Prometeia, e nell'esecuzione di una serie di operazioni conclusive attuate tramite la progettazione di script Powershell.

L'orchestrazione delle risorse computazionali per lo scenario di calcolo distribuito richiesto necessita dell'installazione del software HPC Pack fornito da Microsoft: si tratta di uno strumento estremamente flessibile che gestisce un insieme di macchine (non necessariamente virtuali), in modo da schedulare e amministrare il contesto di esecuzione di un'applicazione distribuita. Ciò ha lo scopo di organizzare le risorse e ridurre drasticamente i tempi richiesti per l'esecuzione delle operazioni che, come nel caso preparato da Prometeia, richiedono l'elaborazione di moli di dati incredibilmente grandi e che quindi, in scenari non distribuiti, possono richiedere tempi anche molto lunghi.

Dopo che l'infrastruttura di calcolo è stata correttamente messa in piedi e configurata, essa è pronta per essere utilizzata: in questo modo il cliente dispone di un insieme di risorse di calcolo dedicate unicamente alle sue necessità, e che è possibile eliminare in qualsiasi momento. Il fulcro della soluzione proposta in questo lavoro, infatti, si basa sull'abbattimento dei costi derivanti dalle risorse di calcolo: passare da soluzioni locali, in cui le macchine sono fisicamente gestite dall'utente, a scenari cloud comporta un notevole risparmio economico, oltre a evitare lunghe configurazioni e aggiornamenti. Grazie a Azure il cliente può distribuire la propria infrastruttura personale solo quando necessaria e disfarla quando non più richiesta.

La seconda fase di questo progetto fa di questo paradigma il suo punto di forza e, grazie allo sviluppo di un'applicazione .NET e alle API offerte da HPC Pack, permette di modellare il *cluster* realizzato nel passo precedente a seconda dell'uso. Il codice

implementato nell'interfaccia dell'applicativo client di ERMAS di Prometeia consiste in una serie di funzioni tramite le quali l'utente ha piena visibilità delle risorse di calcolo disponibili e totale libertà di aggiungerne o rimuoverne la quantità desiderata.

1. PROMETEIA

La realizzazione di questa attività di tesi è stata resa possibile grazie alla disponibilità e all'interesse nel progetto mostrati da Prometeia. Il costante aggiornamento e la ricerca delle tecnologie più innovative hanno permesso a Prometeia di affermarsi tra le principali imprese in Europa nel settore di *Risk* e *Wealth Management*.

Fondata nel 1974 a Bologna da un gruppo di giovani docenti universitari come istituto indipendente di ricerca economica, Prometeia offre dal 1981 servizi di analisi per imprese e intermediari finanziari integrando, a partire dagli anni '90, ricerca, consulenza ed elaborazione di sistemi software all'avanguardia. Opera per conto di oltre 300 enti finanziari tra Europa, Medio Oriente e Africa in 20 paesi diversi, supportati da figure professionali specializzate quali consulenti, analisti, economisti e sviluppatori software. La rapida crescita sul mercato ha permesso a Prometeia di espandersi e dislocarsi in diverse sedi in Italia e fuori: sebbene il polo principale sia a Bologna, infatti, vanta sedi a Milano, Roma, Londra, Mosca, Istanbul e Il Cairo.

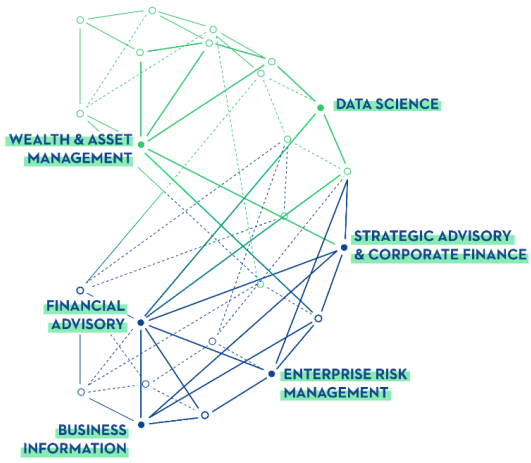

Fig. 1: Ambiti di ricerca di Prometeia

Tra i principali settori di ricerca, Prometeia si occupa di **Enterprise Risk Management**, prefigurandosi come punto di riferimento per l'innovazione tecnologica nell'ambito dei servizi finanziari. Si tratta di un servizio di consulenza che permette di rilevare il rischio aziendale e agire nel modo più efficace per raggiungere i traguardi prefissati. In ambito aziendale si parla di Enterprise Risk Management quando è necessario effettuare un'analisi approfondita di tutti i possibili scenari di rischio, allo scopo di contenerli e di massimizzare il valore creato.

L'utilizzo estensivo di metriche di rischio e di *decision making* consente l'uso di moderni approcci di valutazione e analisi dei rischi. La modellizzazione del rischio, destinata a diverse categorie di clienti, permette l'elaborazione di strategie di gestione e approcci atti a massimizzare il valore dell'impresa: ciò avviene attraverso l'uso di scenari macroeconomici alternativi e di stress implementabili all'interno dei sistemi software Prometeia. La gestione dei rischi, infatti, deve tenere conto della correlazione di una vasta gamma di fattori di credito, liquidità e rischio di mercato, in modo da elaborare gli scenari di attenuazione necessari.

Allo stesso modo, viene dedicata una cura particolare sulle soluzioni informatiche offerte: oltre a soddisfare le esigenze legate al business, infatti, Prometeia si avvale di strumenti informatici avanzati per fornire consulenza in ambito *Risk & Finance* in maniera ottimale (ERMAS, ECAPro). [1]

Mediante l'uso di strumenti all'avanguardia, Prometeia garantisce la qualità delle *delivery* dalla concezione iniziale fino alla realizzazione finale dei sistemi, per l'intero ciclo di progetto. Inoltre, viene fatto largo uso di modelli di dati e processi atti a garantire gestione e qualità dei dati conformi ai principi BCBS 239 (Basel Committee on Banking Supervision), garantendo gli standard più elevati in termini di aggregazione dei dati di rischio e *reporting*. Dal punto di vista tecnologico, è particolarmente rilevante l'attenzione dedicata alle moderne procedure di gestione di **Big Data**, necessarie data l'enorme mole di dati gestita da Prometeia, spesso molto differenti in termini di natura e sorgenti. Ciò va di pari passo con lo sviluppo di strategie legate al calcolo distribuito, adottando avanzate soluzioni di *grid computing*.

Gli strumenti di *Data Science* hanno un ruolo altrettanto significativo: l'impiego di *Machine Learning* e Intelligenza Artificiale permettono di consolidare le soluzioni di

Enterprise Risk Management. Ciò è dovuto all'utilizzo di dati di ogni tipo e più o meno strutturati, come flussi di denaro e transazioni in generale, per creare un quadro di valutazione più completo su cui effettuare le dovute analisi. È fondamentale estrapolare informazioni anche da sorgenti esterne (quotidiani, *blog*, *social media*) attraverso l'uso di strumenti automatizzati di intelligenza artificiale, al fine di elaborare analisi predittive e misure di contenimento del rischio attraverso tecniche di *Machine Learning*. A ciò si aggiunge l'impiego di *Anomaly Detection*, ossia rilevamento di situazioni "anomale" all'interno dei dati raccolti, applicabile ai portafogli dei clienti, dati di mercato e esposizione al rischio.

Prometeia è inoltre in grado di fornire soluzioni innovative in ambito **Wealth and Asset Management** mediante servizi di investimento e assicurativi costantemente al passo con regolamentazioni ed esigenze in perenne cambiamento. Si tratta di un servizio di consulenza in grado di fornire assistenza sia riguardo i bisogni del cliente, mettendo insieme diversi servizi finanziari, sia circa la gestione dei risparmi di privati e società attraverso operazioni finanziarie e non finanziarie, sulla base delle necessità del cliente stesso.

Pertanto, i clienti possono usufruire di un'importante consulenza nella trasformazione dei processi e nel *change management*, tramite pianificazione strategica degli investimenti. Altrettanto rilevante è il servizio fornito nel disegno e nella realizzazione di programmi di *compliance*, realizzato in modo da adeguarsi costantemente ai requisiti del *business*. Pertanto, una costante analisi di mercato permette di avere un quadro generale della domanda e degli sviluppi del mercato stesso, specialmente in ambito di investimenti e assicurazioni: è possibile, dunque, realizzare previsioni macroeconomiche e, di conseguenza, scelte a lungo termine sulla base di predizioni di mercato, analisi dello scenario competitivo, dei trend e delle strategie di *marketing*. Proprio per questo, Prometeia dispone di modelli e algoritmi proprietari in grado di ottimizzare la gestione dei prodotti assicurativi e di stimarne il potenziale commerciale presente e futuro. La realizzazione di strumenti così potenti necessita di tecnologie di pari importanza, per permettere di stabilire le decisioni migliori in ogni momento, con strumenti di valutazione all'avanguardia. Ciò richiede pertanto l'analisi di dati sempre differenti, provenienti dai clienti e dall'esterno, per ottenere studio dei bisogni e *management report* approfonditi e costantemente aggiornati. [1]

Un importante servizio di consulenza finanziaria è fornita da **Prometeia Advisor SIM**. Il suo compito principale consiste nella previsione dei mercati finanziari, e quindi della realizzazione di strategie di *asset allocation*. Ciò è possibile grazie alla valutazione di un insieme di fattori quali la volatilità dei mercati, l'analisi dei portafogli e la conseguente previsione di redditività e di rischi legati a possibili perdite. Queste operazioni permettono anche di tracciare i mutamenti dei mercati, in modo da gestire le decisioni a breve termine e fornire consulenza flessibile e personalizzabile in base alle esigenze del cliente. Gli strumenti forniti vengono analizzati utilizzando un insieme di misure quantitative delle *performance* (ISP).

Attraverso l'analisi degli scenari bancari-assicurativi e di fondi pensione nordeuropei, Prometeia offre soluzioni altamente adattabili studiando flussi attesi, patrimoni e impegni e fornendo indicatori necessari a contenere i rischi e migliorare le prestazioni. Il supporto fornito ai clienti è attuato attraverso l'uso di un database proprietario, che può avvalersi di oltre 400 contributori a titolo gratuito: ciò garantisce l'ottimizzazione degli strumenti di cui si serve la consulenza circa la selezione di gestori e prodotti gestiti. Sulla base dei dati raccolti, pertanto, la selezione è supportata da analisi sia quantitative che qualitative, attraverso l'impiego di algoritmi proprietari. In queste operazioni, Prometeia fornisce un costante supporto al *Transition Management*. Prometeia Advisor SIM, inoltre, assiste i processi di investimento in ogni loro passo: dallo studio di fattibilità, basato sulle peculiarità dell'investitore, alla realizzazione del modello di implementazione, dall'avvio del comparto al suo sviluppo nel tempo. [1]

Infine, un settore di pari rilevanza in cui Prometeia opera riguarda la **ricerca economico-finanziaria**, rafforzata dalle relazioni con gli scenari accademici, finanziari e industriali. Le attività di ricerca riguardano sia il mondo della macroeconomia, permettendo predizioni circa gli andamenti dello scenario italiano e internazionale, che l'ambito microeconomico, analizzando situazioni familiari di reddito e indebitamento, mercato del lavoro e atteggiamenti delle imprese. A tal proposito, Prometeia dispone di Economic Scenario Service, uno strumento capace di gestire gli scenari di mercato al fine di modellare i processi decisionali interni. Questo servizio consente la definizione di scenari alternativi personalizzati, in base ai quali sono calcolati i relativi fattori di rischio e quindi l'impatto sulle condizioni patrimoniali e di liquidità. Nel settore della ricerca, Prometeia contribuisce al Progetto Link,

finanziato dalle Nazioni Unite, e al progetto EUROFRAME (European Forecasting and Research Association for the Macro-Economy), che si prefigge lo scopo di supportare la politica economica dell'Unione Europea con analisi e previsioni di mercato.

Gli strumenti di analisi di Prometeia consentono di avere un quadro generale dei vari settori produttivi nel territorio italiano e all'estero, individuando possibili partner e regioni con maggiore domanda. Un ulteriore supporto definisce i più vantaggiosi tra i potenziali clienti e fornitori, oltre a identificare i principali concorrenti nei settori più disparati e a definire le direttive principali a sostegno dell'esportazione, permettendo l'espansione anche nei mercati esteri.

Tra i numerosi riconoscimenti e i partner illustri che Prometeia può vantare, allo scopo di questa attività di tesi è rilevante citare la *partnership* con Microsoft: i rapporti con l'azienda americana riguardano in particolar modo le tecnologie di *grid computing* e High Performance Computing (**HPC**), Analytics Performance Systems (**APS**) e sviluppo di applicazioni mediante l'uso di .NET Framework. I due principali applicativi di Prometeia, **ERMAS** e **PFTPro**, ne confermano la leadership nei rispettivi ambiti di utilizzo, in particolare grazie all'ottima compatibilità riscontrata con le tecnologie Microsoft. [1]

1.1 PFTPro

La piattaforma **PFTPro** consente di combinare le sue funzionalità per supportare la consulenza fornita al cliente al fine di migliorare l'assistenza alle soluzioni per il *Wealth Management*. In particolare, gli *engine* finanziari sono disaccoppiati rispetto al *front-end* in modo da rendere la struttura applicativa modulare e più facilmente integrabile all'interno di sistemi già esistenti. I principali servizi offerti si occupano di:

- *Financial Advisory* (analisi del patrimonio finanziario, *risk management*, analisi di portafoglio, simulazioni e scenari "*what-if*");
- *Wealth Planning* per analisi delle ricchezze del cliente e pianificazione in base alle sue esigenze (analisi dei bisogni, pianificazione fiscale e pensionistica, *lending*);

- *Insurance Advisory*, ossia attività di consulenza agli intermediari assicurativi. Ciò permette di notificare al cliente i possibili scenari di rischio, di mettere in pratica un'analisi delle sue esigenze di protezione e di monitorare la consulenza fornita;
- *Corporate Advisory*, per garantire rapporti più stretti tra banche e imprese, valutando le prestazioni dei clienti, dei loro *competitor* nel micro-settore, degli scenari "*what-if*" e dei bisogni dell'impresa;
- *Data Analysis Service* mediante l'elaborazione costantemente aggiornata di fattori di rischio riguardo mercati, credito e liquidità: fornisce, pertanto, numerosi indicatori di *risk* e *financial analytics*, processi di produzione affidabili ed efficaci;
- *Asset Management* altamente flessibile e competitivo, realizzato per rilevare le occasioni di *business* e investimento sul mercato, fornisce i dati e i parametri necessari al cliente per espandere i propri interessi sul territorio. Ciò avviene mediante sondaggi, ricerche, studio delle informazioni (flussi di denaro, fondi e prospetti più interessanti). [1]

1.2 ERMAS

ERMAS è uno dei prodotti più interessanti e promettenti di Prometeia. Dal momento che buona parte delle risorse è dedicata all'*Enterprise Risk Management*, un gran numero di professionisti agisce per il suo miglioramento costante. Si tratta di una piattaforma in grado di fornire:

- *Balance Sheet Management*,
- *Credit Risk Analysis*
- *Regulatory & Accounting Compliance*
- *Credit Decision Management*
- *Performance Management & Control*

Ciò che ERMAS offre riguarda, in generale, il supporto alla gestione dei dati e la generazione periodica di *report*, modellando i dati secondo le necessità del cliente attraverso l'uso di *dashboard* estremamente intuitive.

Il *Balance Sheet Management* permette di gestire e analizzare i rischi di mercato e di liquidità, i *funding plan* e la proiezione del margine di interesse, al fine di ottenere strategie di bilancio ottimali. Ciò avviene attraverso simulazioni interattive di scenari di mercato, "*what-if*" *analysis* e previsioni di bilancio, modellate per supportare una pianificazione strategica e l'ottimizzazione del portafoglio. ERMAS permette un approccio proattivo delle banche, quantificando la loro liquidità sia a breve termine che nel medio-lungo periodo. È possibile, infatti, realizzare *report* in maniera automatica, fornendo i punti di vista necessari a comprendere le dinamiche sottostanti, pur mantenendo una buona adattabilità agli scenari in uso.

Allo scopo di identificare e mitigare i rischi è necessario provvedere all'attuazione approcci di simulazione realistica, tenendo conto di una visione d'insieme che ERMAS è in grado di realizzare considerando i rapporti tra rischi di mercato, credito e liquidità: tutto questo ottimizzando anche i costi e le prestazioni computazionali, grazie all'impiego del calcolo distribuito. Dunque, è possibile mantenere in costante aggiornamento gli indicatori di rischio di vario tipo, conservando una gestione dei dati in piena conformità con i principi BCBS e GDPR. Il modulo di Data Management di ERMAS è compatibile con l'infrastruttura di Extract, Transform, Load (ETL) di tutte le banche, definendo strumenti per caricamento, *storage* e sincronizzazione dei dati, con annesso aggiornamento dei *report* correlati. [1]

Fig. 2: Analisi degli scenari di ERMAS

L'*engine* di ERMAS supporta l'allocazione di costi e ricavi in ogni livello organizzativo, trasferendo i rischi alle unità competenti per ridurre i rischi in questione. Talvolta può essere necessario effettuare numerose simulazioni in questo senso, e tener traccia di alcuni scenari per uso futuro.

La soluzione di ERMAS consente totale libertà al cliente, provvedendo anche agli strumenti necessari per definire fattori di rischio altamente configurabili mediante l'uso di librerie matematiche apposite. Il modulo *Funds Transfer Pricing*, quindi, permette di definire l'allocazione dei rischi, la definizione di prezzi di prodotti esterni e l'attribuzione di margini, stabilendo le direttive per incrementare i redditi delle banche.

ERMAS si occupa, inoltre, di analisi del rischio di credito, con scenari macroeconomici e valutazioni "*what-if*", oltre a tecniche consolidate come il metodo Monte Carlo. Sulla base dei modelli di rischio, pertanto, Prometeia fornisce un approccio intuitivo al *credit decision making* sulla base di *workflow*, *data management* e analisi di rischio e portafoglio. Un particolare impegno di Prometeia si riflette nel costante rinnovamento in favore dei criteri più aggiornati, secondo i principi IFRS 9 (parametri di calcolo del costo ammortizzato e rettifiche creditizie) e IFRS 13 (simulazione avanzata per il calcolo del *fair-value*). Analogamente, è possibile determinare valori di rischio di mercato, di credito, di concentrazione secondo i trattati del comitato di Basilea e dell'EBA.

Un particolare punto di forza di ERMAS è strettamente legato all'uso che viene fatto dei dati che i clienti forniscono: si fa, infatti, largo impiego di *Analytical Database* con costanti qualità e integrità garantite attraverso strategie di trasformazione e pulizia dei dati. [1]

ERMAS dispone di un'interfaccia estremamente *user-friendly* che, mediante le varie opzioni di simulazioni "*real time*", produce output adattabili alle esigenze: è possibile, infatti, analizzare i dati secondo le più disparate dimensioni, in modo da esplorarli con delle *facility drag-and-drop* appositamente disegnate, generando *report* facili sia da produrre che da comprendere. Dal punto di vista tecnologico e implementativo, negli ultimi anni la sfida di ERMAS riguarda l'incredibile ammontare di informazioni da

processare e distribuire, con vincoli di tempo e di precisione estremamente rigidi per far fronte anche alle situazioni di maggiore stress.

Punto di forza di ERMAS, come già citato, è senza dubbio la partnership con le tecnologie Microsoft: questa piattaforma, infatti, trae il meglio dai sistemi operativi Windows ed è sviluppata usando il .NET Framework. Fa largo impiego, inoltre, del linguaggio Microsoft C# e dei database forniti da Microsoft SQL Server. Il calcolo distribuito, vantaggio essenziale per il corretto funzionamento di ERMAS, richiede l'uso di HPC e della piattaforma cloud Microsoft Azure. Lo stretto rapporto con il colosso americano è testimoniato dall'uso di molti altri strumenti avanzati da esso forniti, come MS SQL Parallel Data Warehouse (elevata scalabilità orizzontale e dettaglio di analisi grazie all'efficienza dell'elaborazione parallela) e strumenti di *reporting*.

Il sistema di ERMAS è estremamente modulare: la struttura organizzata in livelli consente di monitorare e migliorare le prestazioni di ogni sua parte senza dover modificare l'implementazione del resto del framework:

- **DQM & ETL** per l'estrazione e l'elaborazione dei dati;
- **Data Storage** formato da database MSSQL o Parallel Data Warehouse di Microsoft. In particolare, queste strutture consente la compatibilità con le moderne tecnologie di **Hadoop**;
- **SIMPro**, un ambiente di elaborazione parallela basata su HPC;
- **BI & Reporting** per l'analisi OLAP e *pivoting*;
- **Microsoft Active Directory** per l'autenticazione nel dominio.

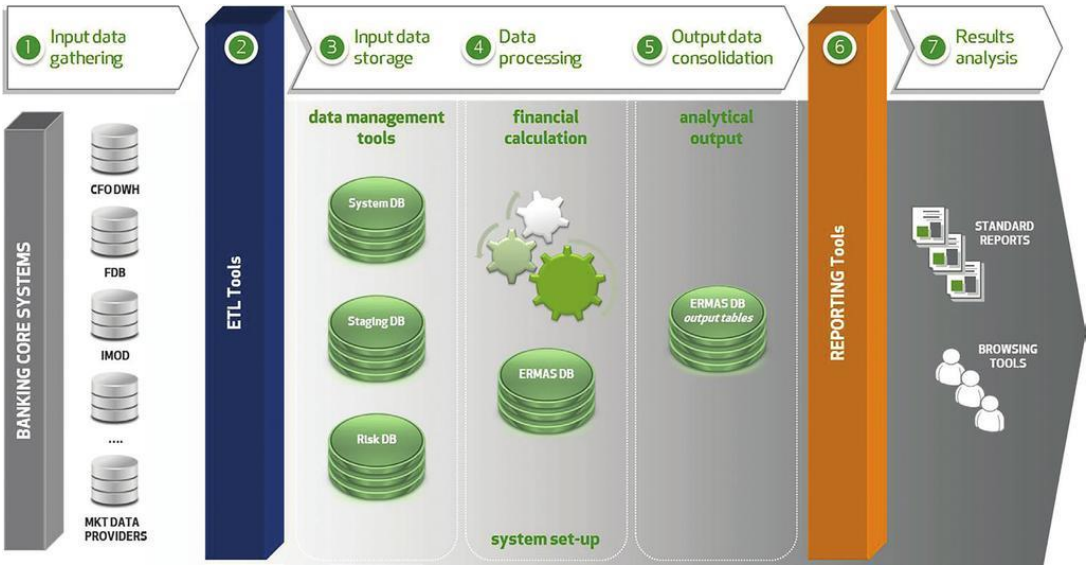

Fig. 3: Flusso di esecuzione dell'analisi di ERMAS

L'applicativo client di ERMAS fornisce all'utente i mezzi per configurare e invocare processi sia interattivi, a basso sforzo computazionale e con requisiti *real time*, che *batch*, ossia che necessitano di esecuzione solitamente notturna per concedere il tempo della preparazione dell'output. I *job*, ossia i processi di calcolo, sono modulari e altamente personalizzabili: è possibile selezionare, per ogni *job* che viene creato dall'utente, i dati di input, le operazioni cui essi saranno sottoposti, il formato dell'output e i parametri di configurazione. A tal proposito, i risultati finali possono essere prodotti, tra le altre opzioni, sotto forma di tabelle Excel o di file .csv. Il *workflow* di un *job* ERMAS si compone di alcune fasi fondamentali:

- una fase iniziale, in cui vengono effettuati i controlli di coerenza e di preparazione dei database di input;
- una prima fase di esecuzione, ossia l'elaborazione principale dei *task* distribuiti attraverso HPC. In base alle esigenze, le strutture dati possono richiedere l'impiego di database Microsoft SQL Server, Parallel Data Warehouse o Hadoop File System (soluzione Big Data);
- fase finale di sincronizzazione tra i processi e organizzazione dell'output parziale;
- fase di *strategy distribution*, analoga alla fase iniziale;

- fase di *strategy execution*, analoga alla fase di esecuzione;
- fase conclusiva, nella quale avvengono una o più aggregazioni dei dati di output delle fasi precedenti, in modo da produrre *report* adatti alle richieste modellate dal cliente. Ciò può avvenire, analogamente, attraverso SQL Server, Data Warehouse o Spark. [1]

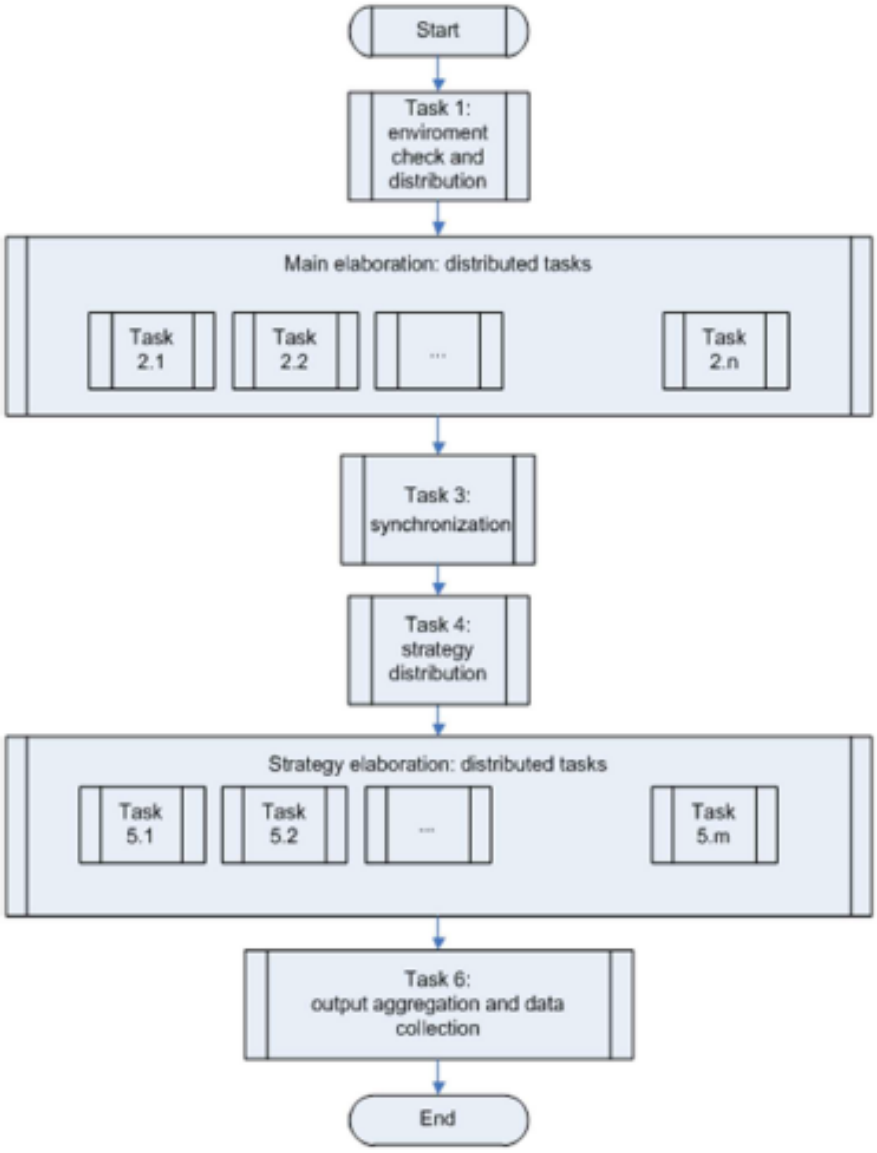

Fig. 4: modello computazionale di ERMAS

Allo stato attuale, l'elaborazione dei sistemi ERMAS è delegata a server fisici dedicati all'elaborazione del calcolo parallelo secondo gli schemi rappresentati. Questo scenario comporta diversi svantaggi: innanzitutto l'acquisto di un *cluster* di server fisici ha un costo notevole, che va incrementando se si considera anche la manutenzione e la gestione di eventuali guasti. Inoltre, queste macchine hanno bisogno di una configurazione manuale al momento della prima installazione (impostazioni di rete, di Active Directory, installazione di HPC e del software applicativo) e di costante aggiornamento. Tali operazioni possono essere facilmente evitate e, in gran parte, automatizzate mediante l'uso di script. Un altro grosso svantaggio nell'uso di server dedicati risiede nel fatto che le risorse di calcolo di ERMAS sono, di solito, impiegate per periodi relativamente brevi: la computazione è di tipo *batch* e i clienti ne richiedono l'elaborazione in momenti ben definiti, come gli ultimi giorni di ogni mese. Risulta evidente che pagare per dei server che, per gran parte del tempo, non sono impiegati è un dispendio di risorse anche economiche. Questa operazione costituisce la prima parte del progetto di tesi presentato.

Spostare le proprie risorse sul cloud è una soluzione fortemente consigliata, dal momento che l'utente non deve più occuparsi della manutenzione delle risorse fisiche a disposizione. Inoltre, soluzioni del genere consentono di controllare le risorse di calcolo gestite: per esempio, Microsoft Azure permette di arrestare le macchine virtuali create se non ne è richiesto l'uso e riavviarle quando è invece necessario che esse siano attive. Sulla base di questo principio prende forma la seconda parte del progetto: avere risorse distribuite sul cloud produce vantaggi economici non indifferenti, ma una miglioria ulteriore può essere apportata. Per questo è stato progettato un servizio (che verrà implementato all'interno di ERMAS arricchendo l'interfaccia utente) che permette di organizzare le risorse computazionali in modo da aumentare o ridurre il numero di macchine virtuali di Azure attive a seconda dell'utilizzo necessario. Infatti, i costi possono essere ottimizzati qualora alcune di queste risorse siano mantenute operative ma inutilizzate, impattando sulle tariffe previste dal servizio di Azure.

2. MICROSOFT AZURE

2.1 CLOUD COMPUTING

Alla base delle tecnologie impiegate per realizzare questo progetto di tesi c'è sicuramente il concetto di *cloud computing*. Si intende, con questo termine, raggruppare un insieme molto ampio e vasto di servizi forniti *on demand* attraverso la rete, in cui le risorse (*storage*, calcolo, *networking*) vengono offerte sulla base delle necessità del cliente.

Il NIST (*National Institute of Standards and Technologies*) definisce il cloud come:

> *Cloud Computing is a model of enabling convenient, on-demand network access to a shared pool of configurable computing resources (e.g. networks, servers, storage, applications, and service) that can be rapidly provisioned and released with minimal management effort or service provider interaction.*

Per implementare soluzioni di questo tipo, le imprese che forniscono il servizio di cloud mettono a disposizione un insieme di risorse condivise sufficientemente ampio e coordinato da supportare la domanda nella sua interezza. Il bisogno del cliente, perciò, può riguardare applicazioni a cui accedere tramite il Web o server fisici da impiegare su richiesta. Un *cloud provider* deve assicurare una serie di garanzie, riguardanti soprattutto prestazioni, scalabilità, affidabilità e sicurezza.

Il vantaggio principale dell'impiego di tecnologie così avanzate è economico: esso, infatti, presuppone una riduzione sensibile dei costi di manutenzione dell'hardware, in quanto le infrastrutture possedute non necessitano più di manutenzione e aggiornamenti ad hoc. Inoltre, le piattaforme cloud offrono servizi potenziati in termini di sicurezza e gestione di *disaster recovery*, evitando agli sviluppatori compiti altrimenti estremamente gravosi. Ciò è reso possibile soprattutto facendo uso di avanzate tecniche di replicazione, evitando che guasti sulle singole macchine fisiche causino perdita di dati e rallentamenti nei servizi offerti. I server impiegati, infatti, sono spesso distribuiti in *data center* sparsi per il mondo coordinati in maniera estremamente rapida e affidabile. Una conseguenza naturale di questi modelli riguarda lo *scale-out* del *pool* di risorse utilizzate: se in passato l'infrastruttura era totalmente

centralizzata e le risorse erano statiche e non ampliabili (spesso causa di colli di bottiglia), ora si preferiscono configurazioni più dinamiche e adattabili all'uso. In questo modo, pertanto, è venuta meno la necessità di acquistare e manutenere macchinari sempre più potenti e costosi in favore di *cluster* di computer meno potenti e più facilmente sostituibili. Il costo del servizio cloud diventa, quindi, totalmente orientato al *pay-per-use*: chi acquisisce un determinato *pool* di risorse, infatti, paga unicamente per ciò che impiega, con costi calcolati sia in base al tempo di utilizzo che alla quantità di mezzi richiesti. Dall'altra parte, i fornitori di questo servizio devono assicurare un insieme di requisiti misurabili attraverso i cosiddetti *Quality of Service*, che possono essere definiti sia qualitativamente che quantitativamente. [2] [3] [4] [5]

La quantità di servizi che il cloud computing mette a disposizione è enorme e diversificata, poiché numerose sono le esigenze degli sviluppatori che ne fanno un largo utilizzo. In particolare, se ne possono definire tre macrocategorie:

- **Software as a Service** (SaaS) è un modello in cui le applicazioni sono ospitate per i clienti che dispongono di un accesso al Web. Il vantaggio principale di un approccio SaaS è che non sono più richiesti aggiornamenti negli applicativi client e nelle infrastrutture utilizzate, in quanto tutto quanto è gestito dal lato server. Inoltre, queste soluzioni permettono di offrire disponibilità maggiore, grazie all'ampiezza di banda allocata. Esempi di SaaS sono le Google Apps (Gmail, Google Calendar), servizi di messaggistica, *social network* e Microsoft Windows Live (Messenger, Hotmail).
- **Platform as a Service** (PaaS) fornisce piattaforme software per l'esecuzione remota, definendo sistemi in grado di interagire tra di loro. Le infrastrutture hardware sono quindi totalmente gestite dal *provider* (supporto di rete, database, sistemi operativi). La piattaforma comprende, per esempio, API, librerie e *test environment*. Google Maps rappresenta un valido esempio di PaaS.
- **Infrastructure as a Service** (IaaS) provvede alle istanze di macchina necessarie agli sviluppatori. Si tratta quindi di server dedicati di cui essi hanno il totale controllo con approcci di *load balancing* più o meno raffinati. La gestione delle soluzioni di *storage*, installazione dei sistemi operativi e

supporto software, pertanto, è demandata al cliente, mentre il *provider* fornisce le macchine (fisiche e virtuali) mediante l'uso di virtualizzazione in ambienti isolati in cui l'utente può operare. Amazon Elastic Compute Cloud (EC2), per esempio, può essere identificato come servizio IaaS.

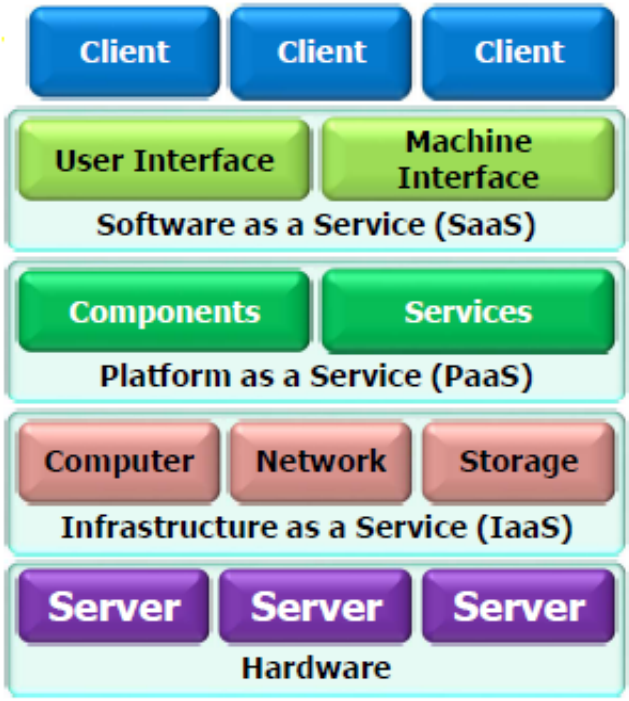

Fig. 5: Stack delle tecnologie IaaS, Paas, SaaS

Questi tre concetti, comunque, non demarcano in maniera forte i servizi offerti: esistono anche versioni ibride modellate in base alle richieste del cliente: si parla, per esempio, di *Data as a Service* (DaaS) e *Hardware as a Service* (HaaS). [2] [3] [6]

2.2 ORGANIZZAZIONE DI MICROSOFT AZURE

Microsoft Azure è la piattaforma cloud fornita da Microsoft e rilasciata nel 2010. I servizi che offre riguardano le più disparate categorie: *computing* (macchine virtuali, *app services*, *hosting* di siti web), *mobile*, *Machine Learning*, *storage*, IoT,

networking, *streaming*, *Business Intelligence* e microservizi. I settori di applicazione possono variare dai servizi finanziari al *retail*, dalla produzione al settore energetico.

Grazie a Azure si dispone di una quantità pressoché illimitata di risorse utilizzabili per qualsiasi scopo, con un sistema di costi ottimizzato per l'uso: gli addebiti riguardano soprattutto le risorse di *storage* (con un costo in GB/mese), di calcolo (RAM e vCPU) e di banda. L'elevata disponibilità è rafforzata dalla scalabilità globale delle risorse, realizzata attraverso la presenza *data center* sparsi in 55 aree in tutto il mondo. Questa piattaforma, inoltre, fornisce diversi servizi di cloud (privato, pubblico e ibrido) garantendo interoperabilità con numerosi servizi interni ed esterni a Microsoft.

In particolare, **Fabric Controller** rappresenta il cuore di Azure: si tratta di un gestore dei *cluster*, replicato in ogni *data center* Microsoft con compiti di *load balancing*, *switch* e aggiornamento e coordinamento delle risorse gestite.

Il funzionamento di Azure richiede una struttura gerarchica per la gestione degli account degli utenti che ne usufruiscono:

- se si opera in contesti aziendali, come in quasi tutti gli scenari, i servizi cloud di Microsoft sono gestiti dall'**organizzazione** in questione, che raggruppa in sé una o più sottoscrizioni;
- le **sottoscrizioni** sono a tutti gli effetti dei contratti, stipulati con Microsoft per impiegare le piattaforme cloud offerte sulla base di costi variabili dipendenti dall'uso: i servizi SaaS sono fatturati a seconda della licenza dell'utente, mentre se si tratta di PaaS e IaaS il costo è correlato alle risorse consumate;
- le **licenze** permettono ai singoli utenti di usufruire dei servizi SaaS addebitando costi fissi alla sottoscrizione in uso. Le licenze non sono utilizzate, invece, in scenari PaaS e IaaS, in quanto esse sono integrate nei costi addebitati: se si usano macchine virtuali, per esempio, le immagini dei sistemi operativi e i software installati hanno un prezzo integrabile nella *tariffa pay-per-use*;
- per utilizzare i servizi inclusi in una sottoscrizione su Azure, è necessario disporre di un account utente censito da un *tenant* di **Azure Active Directory** (Azure AD);
- il **tenant**, infine, non è altro che un'istanza di Azure AD che incorpora account e gruppi gestiti. [4] [7] [8]

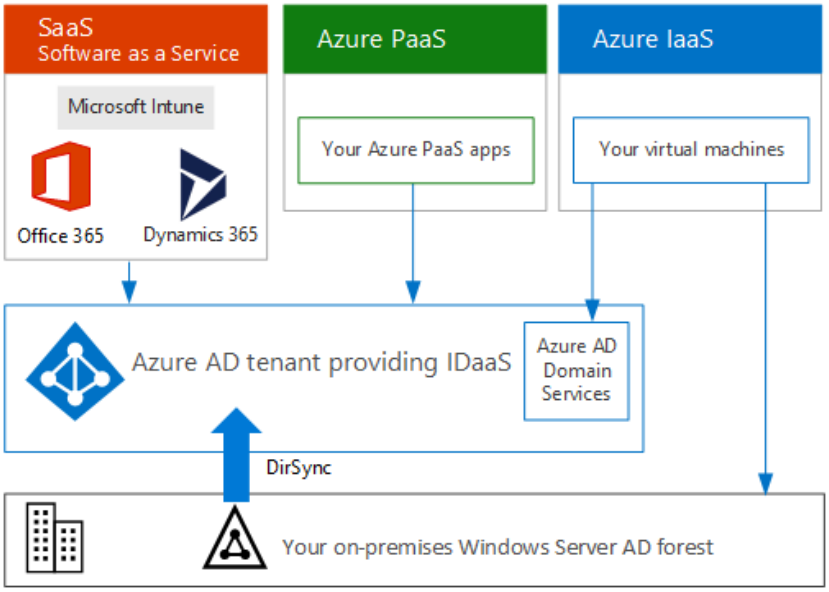

Fig. 6: Struttura del contesto di Azure Active Directory

Azure Active Directory è un servizio di *directory multitenant* che gestisce le identità e gli accessi sul cloud in maniera sicura. Azure AD può essere usato in maniera complementare con *Windows Server Active Directory* (orientata verso le soluzioni *on-premise*) tramite la registrazione dei vari utenti e gruppi anche sul cloud, o come un servizio del tutto autonomo. Come già detto, un'istanza di Azure AD definisce un *tenant*, isolato rispetto agli altri servizi, in modo da evitare che i dati su esso presenti siano in qualche modo accessibili ad altri *tenant* o utenti che non dispongono dei permessi necessari. In particolare, Azure AD permette la registrazione e l'autenticazione degli utenti con un controllo *role-based* e un efficace monitoraggio dell'utilizzo delle applicazioni e della sicurezza. La gestione di questo strumento di autenticazione è affidata agli amministratori IT aziendali e agli sviluppatori di app, per lavorare con credenziali predefinite.

All'interno di una sottoscrizione, inoltre, è possibile definire uno o più gruppi di risorse (**Resource Group**). In breve, si tratta di contenitori logici che includono le risorse gestite cosicché che condividano lo stesso ciclo di vita. In questo modo è possibile distribuire, eliminare e amministrare gli oggetti creati in maniera più rapida. La gestione delle risorse è mansione del cosiddetto **Azure Resource Manager**, attraverso

il quale si può operare sulle singole risorse e controllarne gli accessi. Previa autenticazione, è possibile comunicare con esso per organizzare servizi, reti, macchine virtuali e database: ciò può essere realizzato attraverso l'interfaccia fornita dal portale di Azure, i comandi (chiamati *cmdlet*) di **Azure Powershell**, **Azure CLI** e **REST API**. Le risorse possono appartenere a un solo *Resource Group*, ma è possibile effettuare una migrazione da un gruppo all'altro. Inoltre, si può abilitare la comunicazione di risorse locate in aree differenti o inserite in gruppi di risorse distinti.

In questo progetto lo strumento utilizzato per creare le istanze necessarie è il **modello di risorse** Azure Resource Manager. Ciò, in collaborazione con il modulo Azure Powershell, fornisce un meccanismo di automazione molto potente: come risultato, si ottiene un'infrastruttura estremamente flessibile, configurabile *on demand*, attivabile e disattivabile in tempo reale. Questa situazione consente di ottimizzare i costi sia rispetto alla soluzione *on-premise*, in cui i costi di manutenzione e aggiornamento delle macchine gravano sull'organizzazione, che rispetto alle soluzioni cloud realizzate mediante l'interfaccia del portale Azure, che richiede la creazione e gestione manuale delle risorse gestite. [7] [9] [10] [11]

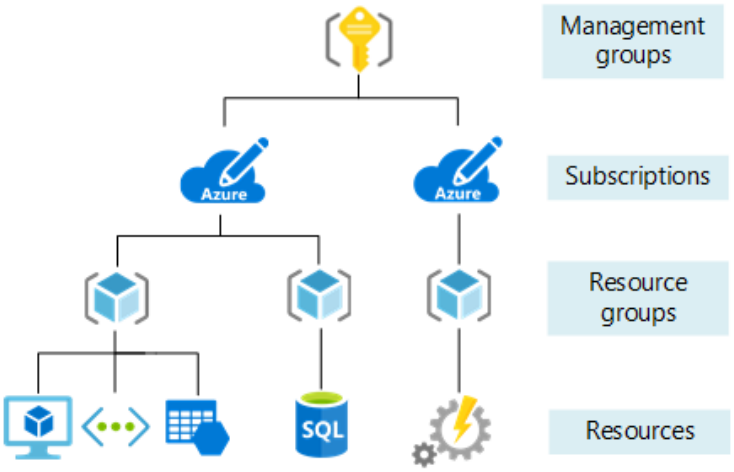

Fig. 7: Gerarchia delle risorse Azure

2.2.1 AZURE AD APPLICATION

Un cenno, innanzitutto, va dedicato al sistema di gestione delle **applicazioni** che Azure Active Directory mette a disposizione, con un metodo di identificazione per app sia cloud che locali. Ciò risulta utile soprattutto in ambito aziendale, dove i dipendenti devono accedere a numerose applicazioni da diversi luoghi e dispositivi: Azure AD, pertanto, offre un punto di autenticazione centralizzato per l'accesso alle applicazioni secondo l'approccio **Single Sign-On** (SSO). Dal punto di vista dell'utente, ciò migliora l'accesso raccogliendo le applicazioni e le password necessarie all'interno di un singolo pannello di accesso. Anche dal punto di vista amministrativo, SSO aggira il lungo inserimento dei singoli account utente all'interno di ciascuna applicazione mappando in maniera automatica le credenziali gestite per tutte le applicazioni. Queste ultime, in particolare, possono appartenere al modello interno di Azure AD, che dispone nativamente di autenticazione SSO, ma possono essere gestite anche per vie esterne e personalizzate: in tal caso l'autenticazione va integrata come un servizio aggiuntivo.

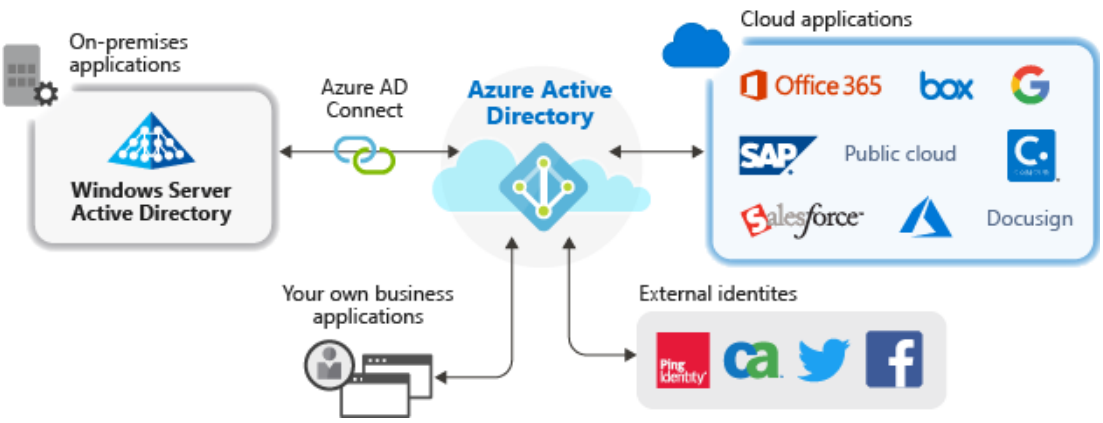

Fig. 8: Contesto di Azure AD per l'uso di applicazioni sul cloud

La creazione di un'applicazione all'interno dell'Azure Active Directory può essere combinata con l'uso di **HPC Pack** fornito da Microsoft, in modo da permettere una configurazione dinamica del *cluster* di macchine virtuali. Come si vedrà, gli strumenti

forniti da HPC Pack consentono creazione ed eliminazione di risorse di calcolo create sul cloud di Azure, a seconda delle esigenze. [7]

2.2.2 AZURE RESOURCE MANAGER

L'infrastruttura per il calcolo distribuito su cui Prometeia fa leva per soddisfare le richieste di numerosi clienti si compone principalmente di un insieme di macchine virtuali realizzabili attraverso il servizio IaaS di Azure, coordinate attraverso l'uso di Microsoft HPC Pack per la distribuzione e l'esecuzione del calcolo parallelo, in modo da figurare a tutti gli effetti come un servizio PaaS.

Il primo passo svolto per automatizzare la realizzazione e il ridimensionamento di questa infrastruttura consiste nella realizzazione di un modello di Azure Resource Management: si tratta di un file in formato .json che, attraverso l'uso di una sintassi dichiarativa, consente di definire le risorse necessarie all'uso, distribuirle sul cloud e aggiornarle. L'impiego di questi file, noti come **ARM Template**, evita la noiosa e ripetitiva fase di configurazione manuale (creazione di macchine virtuali, dimensionamento dello *storage*, gestione delle infrastrutture di rete). Tale operazione permette, dunque, di definire una sola volta le risorse di cui si necessita e di crearle ogni volta che se ne richiede l'utilizzo. Ciò risulta estremamente utile in fase di test quando, per esempio, si nota che il modello distribuito non soddisfa i propri bisogni: in tal caso è possibile rimuovere le risorse create (singolarmente, o eliminando l'intero Resource Group) oppure cambiare il *template* creato e distribuirlo nuovamente. Quest'ultimo caso, in particolare, opera sulle risorse già create apportando le modifiche che si riflettono sul modello creato.

Inoltre, Azure Resource Manager definisce un controllo degli accessi basato sui ruoli (RBAC) alle risorse. Come si vedrà, all'interno di un ARM *template* è possibile anche parametrizzare campi come nomi delle risorse e stabilire delle **dipendenze** tra risorse specifiche: ciò deve essere specificato all'interno del *template*, nella definizione dell'oggetto stesso. In questo modo, la creazione di una risorsa non avviene finché tutte le dipendenze non saranno state risolte. [7] [12]

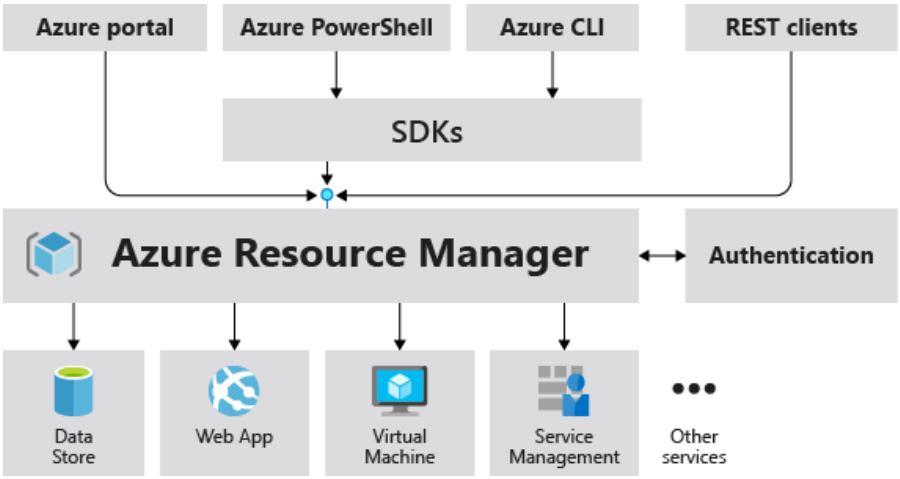

Fig. 9: Accesso e gestione di Azure Resource Manager

Usando ARM *template* la distribuzione delle risorse è molto più rapida rispetto all'utilizzo di *script* Powershell o Azure CLI (*Command-Line Interface*), in quanto l'operazione non è intesa come una sequenza di comandi e viene svolta secondo il massimo grado di parallelismo disponibile. È possibile definire anche **estensioni** per le proprie risorse, ossia delle configurazioni aggiuntive vere e proprie: ciò permette di aggirare la fase di settaggio manuale per portarle nello stato desiderato. In questo modo i comandi Powershell e l'interfaccia Azure si rendono necessari solo se si intende gestire le risorse in tempo reale (per esempio, avviare e arrestare una macchina virtuale, modificare password, permessi e server DNS). Il modello creato, inoltre, può anche essere esportato da un gruppo di risorse già esistente. Se l'infrastruttura è stata creata tramite il portale Azure, infatti, scaricare il *template* (automaticamente generabile) per poter ripristinare il modello attuato può essere una buona soluzione.

Un vantaggio importante nel *deployment* di modelli di Azure Resource Manager consiste nella convalida che Azure stesso effettua prima di avviare l'esecuzione, per verificare che tutte le risorse siano configurate correttamente: nomi e campi impostati correttamente, dipendenze strutturate e acicliche). Ovviamente, situazioni di errore possono comunque verificarsi: un caso particolare può essere la presenza di riferimenti non validi in tempo reale, per esempio errori di connessione o nel download di alcuni file. È anche possibile innestare *template* più piccoli uno dentro l'altro, organizzandoli in modo tale che l'uno richiami gli altri: questo aumenta la portabilità del modello,

consentendo la definizione di sottoinsiemi di risorse modulari più facilmente assemblabili. In alcuni casi, inoltre, è utile creare ARM *template* separati: disaccoppiando la definizione dei parametri da utilizzare dalle risorse effettive.

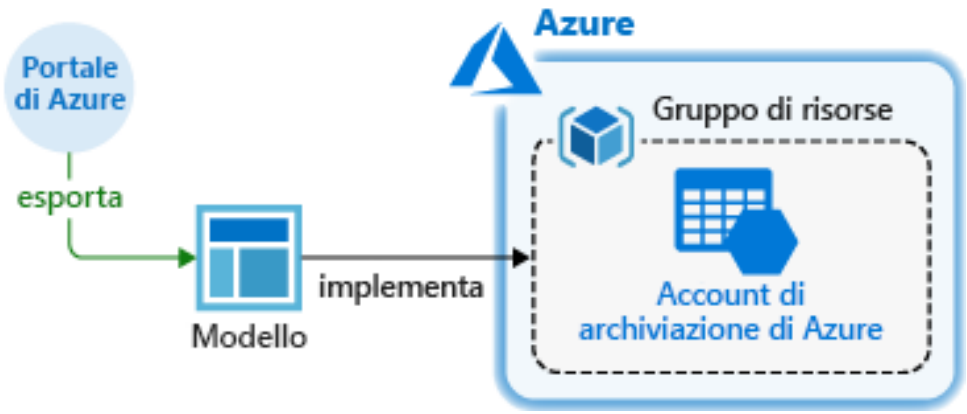

Fig. 10: Schema di distribuzione delle risorse su Azure

Si intende ora presentare lo strumento dell'ARM *template* nel dettaglio, mostrandone la struttura generale e, in seguito, specificandone le risorse utilizzate in questo progetto di tesi (definizione e struttura all'interno del modello stesso). Queste risorse, in particolare, consentono la realizzazione di un *cluster* di macchine virtuali all'interno di una rete privata. [7] [12]

Il modello richiede, innanzitutto, dei campi globali:

- *$schema* rappresenta il percorso dello schema JSON che indica la versione del linguaggio del modello;
- *contentVersion* è usato per indicare la versione del modello, organizzabile in modo da indicare esplicitamente l'ARM *template* più aggiornato e quindi su cui lavorare. Solitamente si procede con valori incrementali di versione;
- *apiProfile*, ossia la versione delle API per i tipi di risorse. Questo campo non è obbligatorio, ma semplifica la gestione delle API all'interno delle altre risorse gestite.

Per quanto riguarda la struttura principale del modello, essa si compone di cinque sezioni:

- **Parametri** (facoltativi), che rappresentano i valori usati nella distribuzione delle risorse. Essi permettono di personalizzare il modello, in modo che sia portabile in ambienti differenti. Possono essere di diversi tipi (stringhe, interi, oggetti) con una struttura ben definita;
- **Variabili** (facoltative) ossia valori che, al pari dei parametri, sono impiegati per semplificare la definizione delle risorse. Spesso vengono costruite a partire dai parametri (o da altre variabili) mediante l'uso di riferimenti. Anche le variabili possono avere una propria struttura, realizzata in modo da essere utilizzata dalle risorse;
- **Funzioni** (facoltative). Esse definiscono funzioni personalizzate che possono essere impiegate nella realizzazione del modello. Al loro interno vengono definiti dei valori di input e di output, in modo da organizzare in maniera chiara gli oggetti da esse restituiti. Esse saranno utilizzate nella definizione delle risorse. Tali funzioni hanno però dei vincoli: non possono accedere alle variabili e non possono invocare altre funzioni definite dall'utente. Differiscono dalle funzioni di default, che sono invece insite nel motore del modello. Le funzioni standard sono molto varie e permettono operazioni di controllo (*equals*, *length*, *min*, *max*, *contains*, *less*, *greater*), operatori logici (*and*, *or*, *if*) e funzioni numeriche (operatori algebrici). È possibile usare anche funzioni che operano sulle stringhe, per esempio concatenamento e conversione, e sulle risorse stesse: a partire da una risorsa, una sottoscrizione o un gruppo di risorse è possibile estrapolare informazioni come indirizzi web, ID e nomi di vario genere;
- **Risorse**, il cuore dell'ARM *template*, indicano cosa andrà distribuito o aggiornato all'interno del modello. Esse presentano alcuni campi fondamentali come tipo di risorsa, versione delle API, nome, locazione, *tag* e dipendenze. Le proprietà rimanenti sono strettamente connesse al tipo di risorsa che si intende definire. È possibile stabilire anche campi facoltativi come *condition* (ossia la verifica di una condizione, solitamente implementata attraverso una funzione, che definisce se una risorsa andrà creata o meno), e *copy*.

Quest'ultimo comando permette di definire il numero di risorse di un certo tipo che devono essere create, evitando di definire ognuna separatamente. Se si fa uso di *copy*, è utile impiegare la funzione *copyIndex* che, a partire da un valore predefinito, compie un'iterazione sulla proprietà indicata per crearne nuove istanze a seconda del numero specificato (per esempio, aiuta nella definizione dei nomi delle risorse replicate);

- **Output** (opzionale), raccoglie i valori che dovranno essere restituiti dalla distribuzione: solitamente fornisce campi restituiti dalle risorse gestite.

```
{
  "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",
  "contentVersion": "",
  "apiProfile": "",
  "parameters": {  },
  "variables": {  },
  "functions": [  ],
  "resources": [  ],
  "outputs": {  }
}
```

Fig. 11: Modello base di un ARM Template

Un campo comune a tutte le componenti dell'ARM *template* riguarda l'uso di commenti e metadati allo scopo di rendere più chiaro e leggibile il modello realizzato. [7] [9]

2.2.3 AZURE STORAGE

Primo elemento per lavorare su Azure è il suo servizio di ***storage***, realizzato con ben definite caratteristiche di scalabilità, persistenza e ridondanza. Azure, infatti, si occupa di gestire la manutenzione e le criticità per l'utente, garantendo capacità di *storage* oltre i 500 TB per utente. Esso si compone di quattro servizi fondamentali: *File Storage*, *Table Storage*, *Blob Storage* e *Queue Storage*, che supportano prestazioni *Standard* o *Premium* (queste ultime usano solo dischi SSD, con costi maggiori). **Azure Storage** è utile in numerosi scenari come applicazioni che forniscono dati

multimediali, applicazioni mediche, *gaming* e *social network*. Gli addebiti sono calcolati solo in base alla capacità di memoria richiesta e al traffico in rete.

La persistenza dei dati è ottenuta sfruttando la vasta quantità di macchine a disposizione, attraverso replicazione *on-demand* su diversi livelli:

- **Locally-Redundant Storage** (LRS) copre il livello di replicazione più basso ed economico, realizzando tre copie delle informazioni all'interno dello stesso *data center* secondo politiche efficienti di *fault tolerance* e di coordinamento. Questo scenario offre la replicazione più veloce, dato che le operazioni richieste avvengono tutte all'interno della stessa regione;
- **Zone-Redundant Storage** (ZRS) opera in maniera asincrona realizzando tre copie dei dati in altrettanti *data center* all'interno della stessa regione, oltre a tre repliche nel *data center* originale che ospita le informazioni. Questo modello presenta degli svantaggi, non supportando politiche di *logging* e misurazione di prestazioni. In caso di *disaster recovery*, alcune informazioni potrebbero essere perse o recuperate con maggiore latenza, poiché esse non sono distribuite in ogni *data center* utilizzato; tuttavia offre maggiore persistenza e disponibilità rispetto al modello LRS;
- **Geo-Redundant Storage** (GRS) effettua replicazione totale dei dati in due regioni differenti, in maniera sincrona nella prima, asincrona nella seconda (garantendo maggiore latenza). Ovviamente questo modello permette il massimo grado di disponibilità e persistenza anche in caso di *failure*, ma ciò ha un costo sia economico che di latenza nel recupero dei dati. Inoltre, in caso di *failure* del primo *data center*, si potrebbe avere perdita dei dati che non sono stati aggiornati nella seconda replica;
- **Read-Access Geo-Redundant Storage** (RA-GRS) segue le stesse politiche di GRS, ma il secondo *data center* utilizzato fornisce solo operazioni di lettura (nessuna perdita dei dati, quindi) finché la prima replica non viene correttamente ripristinata.

Come menzionato in precedenza, Azure fornisce due tipi di prestazioni: il modello Standard offre *storage* su dischi HDD ed è ampiamente soddisfacente per carichi di lavoro in cui la latenza non è un requisito stringente: il suo vantaggio principale

riguarda i costi contenuti. Uno *storage* Premium, al contrario, è ideale per applicazioni con latenza minima e utilizza dischi SSD, con prestazioni e costi più elevati.

Perché il servizio Azure Storage sia attivo, è richiesta all'utente la creazione di uno **Storage Account**, ossia un account di archiviazione che conterrà tutti i dati necessari. Esso può appartenere a diverse categorie: **General-purpose V1** è il modello più datato e meno esoso e non supporta le politiche più moderne di accesso ai dati. **Blob storage** e **General-purpose V2**, invece, offrono il più avanzato modello di *storage* ottimizzando gli accessi e riducendo i costi.

Una caratteristica particolarmente efficace di questo servizio è il cosiddetto *access tier*, che consente di definire in fase iniziale l'utilizzo che si intende fare dei dati all'interno del proprio Storage Account, in modo da abbattere i costi:

- **Hot**, il più efficiente per dati per cui sono pianificate frequenti operazioni di lettura e scrittura. I costi di accesso agli stessi sono abbattuti, a discapito di quelli per l'archiviazione dei dati;
- **Cool**, utile per dati cui si accede molto di rado (per esempio, con cadenza mensile). Questo modello prevede costi minori per lo *storage* ma maggiori per gli accessi. Risulta vantaggioso se, per esempio, si lavora con *backup* e contenuti aggiornati molto di rado;
- **Archive**, è il livello più basso, con costi minimi in archiviazione e massimi per accedervi. [7] [9] [13] [14] [15]

I dati caricati in uno Storage Account sono crittografati lato server, e l'accesso agli stessi può essere gestito a livello di Active Directory (privato), mediante condivisione della chiave o con accesso pubblico. I costi per lo *storage*, dunque, sono determinati dall'area geografica in cui sono locati, dalla capacità richiesta, dal traffico in rete e dalla frequenza degli accessi.

Come accennato, Azure Storage supporta quattro tipi di oggetti:

- **Blob Storage** (*Binary Large Objects*) è il modello più classico di dati (file HTML, immagini, Excel, testo). Essi sono identificati tramite URL e sono, quindi, accessibili con le REST API o con le librerie fornite da Azure SDK. La struttura di base prevede la creazione di *container* (simili a cartelle) all'interno

dello Storage Account, in cui inserire i *blob* necessari. Non è possibile organizzare i *container* in gerarchia, ma ognuno di essi può a sua volta contenere nuove cartelle. A seconda delle dimensioni, i *blob* possono essere raggruppati in *block blob*, *page blob* e *append blob*;

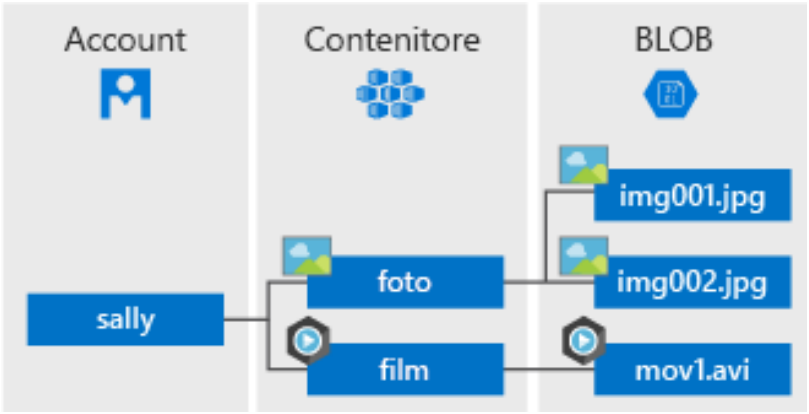

Fig. 12: Organizzazione dei BLOB

- **File Storage** estende al cloud il concetto di *file share*, contenendo dati cui le macchine virtuali create necessitano di accedere;
- **Table Storage** offre un servizio scalabile per dati NoSQL per archiviare informazioni semistrutturate e non adattabili a database relazionali. È possibile definire indici e compiere *query* sufficientemente semplici su questi dati. Lo storage consiste nella creazione di record sotto forma di coppie chiave-valore. In particolare, ognuno degli elementi salvati dispone di una chiave di partizione (che ne identifica la macchina su cui è salvato, se richiesto), una chiave primaria e un *timestamp*;
- **Queue Storage** è usato per archiviare e recuperare messaggi (di dimensioni non maggiori a 64 KB), garantendo ottimo *throughput* e latenza minima. Tale servizio è utile quando si vogliano elaborare grandi quantità di messaggi organizzati in code in maniera asincrona, con politiche FIFO (*First In, First Out*). Questo tipo di dati è progettato in modo da evitare la loro perdita finché essi non sono stati effettivamente processati dall'applicazione che ne usufruisce.

Tutti questi dati, quindi, possono essere raggruppati sotto un'unica risorsa di Storage Account, condividendo lo stesso ciclo di vita. In particolare, in questo progetto di tesi è utilizzato il modello di Blob Storage. [7] [9] [13] [14] [15]

2.2.4 AZURE KEY VAULT

Affinché l'infrastruttura di calcolo distribuito offerto da Prometeia possa essere distribuita sul cloud di Azure, un'altra risorsa essenziale è il **Key Vault**, ossia un gestore sicuro per certificati, chiavi e segreti richiesti dall'applicazione che si desidera realizzare. In particolare, esso sarà impiegato per creare e gestire i certificati richiesti dall'infrastruttura HPC sottostante.

In questo modo gli sviluppatori sono sollevati dal compito di organizzazione dei segreti, affidandolo a un servizio centralizzato sul cloud ed evitandone l'amministrazione nel codice applicativo. Gli oggetti presenti nel Key Vault sono accessibili tramite URI e custoditi con algoritmi standard di protezione e autenticazione basata su Azure AD. È consentito creare segreti direttamente all'interno del Key Vault (mediante Azure CLI, Powershell o interfaccia) oppure inserirvi certificati preventivamente costruiti. [7] [16]

2.2.5 AZURE VIRTUAL NETWORK

Ovviamente, la gestione di un *cluster* di macchine virtuali e servizi presenti su Azure richiede la configurazione di una rete che consenta la comunicazione tra i vari attori dell'infrastruttura. In breve, è necessario garantire l'esistenza di una rete virtuale che connetta le istanze create e ne metta in pratica politiche di protezione adeguate. Una **Azure Virtual Network** può essere paragonata, sotto molti punti di vista, a una rete fisica.

Un vincolo importante è rappresentato dalle istanze che possono essere contenute in una rete virtuale, che devono appartenere alla stessa regione e alla stessa sottoscrizione. Comunque, questo vincolo può essere superato attraverso la creazione di risorse esterne per interfacciare reti differenti (definendo *gateway* di rete per

connessione *Point-to-Point*, *Point-to-Site*, *Site-to-Site*). Strategie di questo tipo sono utili anche per soluzioni ibride dove, per esempio, parte dell'infrastruttura è locata sul cloud e parte è mantenuta *on-premise*.

In fase di configurazione è necessario identificare alcune proprietà della rete virtuale che si intende creare. Innanzitutto, bisogna specificare uno spazio di indirizzi IP privati, cosicché ognuna delle risorse che vi appartengono siano univocamente identificabili: se, per esempio, si usa uno spazio di indirizzi 10.0.0.0/16, 16 bit dell'indirizzo IP privato sono dedicati all'indirizzamento degli *host* nella rete. Una Virtual Network, inoltre, è organizzata in **Subnet**: ovvero la rete viene partizionata in sottoreti ognuna con il proprio spazio di indirizzi dedicato. Come tutte le risorse di Azure, inoltre, è richiesta l'area in cui la rete deve essere creata e la sottoscrizione relativa.

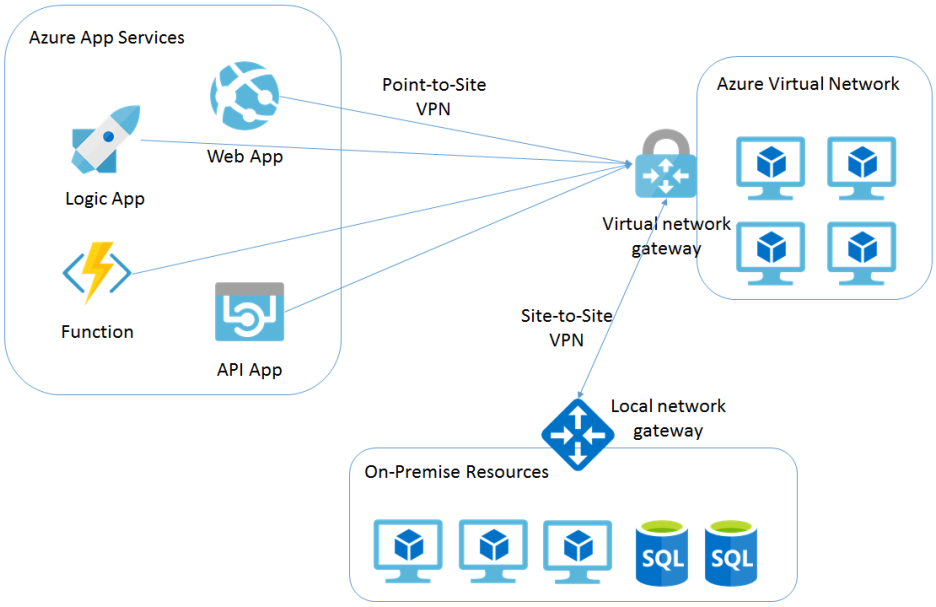

Fig. 13: Modello di Virtual Network

La gestione delle Virtual Network offerte da Azure è un servizio totalmente gratuito. I costi in tal senso riguardano solo le istanze che vi appartengono applicando le rispettive tariffe. Affinché la connessione sia abilitata in una rete costituita da macchine virtuali, esse necessitano di una serie di risorse aggiuntive: gruppi di regole

di sicurezza, per controllare il traffico in ingresso e in uscita (come un comune *firewall*), interfacce di rete virtuale, per consentire la comunicazione nel *network* creato, indirizzi IP pubblici e nomi DNS. Questi ultimi, in particolare, sono facoltativi. [7] [9] [10] [12] [17]

Per la gestione delle proprietà di connettività di una rete virtuale, è bene corredarla di un gruppo di regole di sicurezza. È richiesta, pertanto, la creazione di un **Network Security Group**. Esso contiene il controllo del traffico in ingresso e in uscita della Virtual Network che gestisce le interfacce di rete correlate. In questo modo si ottiene un controllo a grana fine nei flussi da e per le macchine virtuali e le sottoreti, definendo esplicitamente norme che consentono o rifiutano il traffico in rete. Le regole del Network Security Group sono basate sul controllo di indirizzi IP e porte sorgenti e di destinazione del traffico. Ovviamente la loro definizione non è obbligatoria, ma fortemente consigliata per evitare spiacevoli intrusioni nella rete configurata.

La creazione di una regola richiede la definizione di alcuni parametri: il protocollo in questione (TCP, UDP o ICMP), indirizzo IP e porta (o range di porte) da gestire, direzione del traffico di rete e, ovviamente, se si intende consentire o impedire la comunicazione. Altro parametro importante è la priorità della regola, compresa tra 100 e 4096. il Network Security Group è organizzato in modo da elaborare prima le regole con valore più basso: quando arriva un pacchetto di dati, il traffico è sottoposto alle regole in ordine di priorità, finché non se ne trova una che sia compatibile con il caso in questione.

Ci sono, inoltre, dei limiti nell'uso di un Network Security Group. Essi riguardano sia il numero di regole totali da avere all'interno di una sottoscrizione, che nel numero di gruppi di sicurezza legati a una rete virtuale: è possibile associarne soltanto uno per ciascuna Virtual Network creata. [7] [9] [12]

Affinché le macchine virtuali create su Azure possano comunicare in rete, esse necessitano di un indirizzo IP (per accedere a Internet) e di una interfaccia di rete. Quest'ultima, presente su Azure con il nome di **Network Interface** (NIC), si occupa dell'accesso delle risorse nella propria rete virtuale. Non è obbligatorio assegnarne una sola a ciascuna macchina virtuale, anzi è possibile configurarne multiple in base alla dimensione della stessa e al numero di reti cui deve appartenere.

Quando a una macchina virtuale viene assegnata un'interfaccia di rete, essa ottiene un indirizzo IP privato utilizzabile nella rete di appartenenza: è essenziale che questo indirizzo IP appartenga a una sottorete a essa collegata. È comunque possibile assegnare una Network Interface per ognuna delle sottoreti presenti. Una caratteristica sicuramente molto utile quando si opera all'interno di una Virtual Network è la definizione dei **DNS Server**. Questi possono essere specificati in ognuna delle interfacce di rete create, ma è possibile anche non assegnarne: in tal caso essi saranno ereditati dalle proprietà della rete virtuale stessa.

Le risorse su Azure possono interfacciarsi anche con la rete pubblica. Come per qualsiasi altro dispositivo connesso alla rete, perché questa operazione sia attuabile, è necessario un **indirizzo IP pubblico**. Si può assegnare un indirizzo a diverse risorse quali macchine virtuali, Load Balancer e Gateway VPN di Azure. In scenari più complessi si può comunicare sulla rete anche senza un indirizzo IP pubblico assegnato, se si opera in un contesto di *load balancing* in cui l'orchestratore possiede un indirizzo pubblico.

Questo tipo di istanza, in particolare, può essere associato a una sola risorsa ed è allocabile sia in maniera statica che dinamica. Nel primo caso l'indirizzo IP viene mantenuto per l'intero ciclo di vita della risorsa, anche se questa viene arrestata, fino alla sua cancellazione. Se si opta per un'allocazione dinamica, invece, l'indirizzo IP viene liberato in fasi di inutilizzo e quindi dissociato; al suo riavvio, pertanto, verrà assegnato un nuovo IP differente dal precedente. La scelta dell'uno o dell'altro caso dipende fortemente dallo scenario che si intende realizzare. [7] [9]

2.2.6 AZURE VIRTUAL MACHINE

Una parte molto corposa e rilevante delle tecnologie IaaS offerte da Azure riguarda la gestione delle macchine virtuali (**Azure Virtual Machine**). Alla base del concetto di macchina virtuale c'è la **virtualizzazione**, tecnica in realtà abbastanza datata ma le cui potenzialità sono tuttora sfruttate appieno. Il suo principale vantaggio è la possibilità di disaccoppiare logicamente le risorse fisiche di un server dalle capacità virtuali. Senza entrare troppo nel dettaglio, la virtualizzazione consente l'installazione di un

sistema operativo e l'esecuzione di applicazioni su hardware virtuale, ossia su infrastrutture simulate con un livello superiore di astrazione rispetto alle risorse fisiche su cui questo modello poggia.

Una macchina virtuale può essere definita, quindi, come un ambiente emulato su un'architettura fisica, e dispone di risorse virtuali quali dischi virtuali, RAM virtuale (**vRAM**) e CPU virtuale (**vCPU**). La sua utilità riguarda principalmente la creazione di piattaforme *on-demand* per gestire ed elaborare grossi carichi di lavoro sul cloud. Queste risorse possono essere gestite, configurate e monitorate tramite diversi strumenti. È possibile anche stabilire regole di connettività tra macchine virtuali o interfacciarle con il web.

A differenza delle tecnologie PaaS, la creazione di istanze di macchine virtuali richiede al cliente la totale gestione di aggiornamenti, installazione di software e di *patch*. Ciò è realizzabile manualmente, accedendo alla macchina virtuale con comandi remoti Powershell o mediante **Remote Desktop Protocol** (RDP). Come si vedrà, comunque, è possibile automatizzare processi di questo tipo attraverso le estensioni fornite dal Resource Manager di Azure. In base alle esigenze, si può ridimensionare la propria macchina virtuale, aggiungendo o rimuovendo capacità sia computazionale che di *storage*. Per di più i dati salvati sui dischi delle macchine virtuali sono resi permanenti da Azure, in modo da non essere cancellati nel caso in cui si voglia riavviare o arrestare un'istanza gestita.

In fase di creazione di una *Virtual Machine* bisogna specificare alcuni parametri: oltre alle informazioni comuni a tutte le risorse su Azure (nome, regione di archiviazione, ecc.) è essenziale fornire le dimensioni della VM (vCPU, vRAM), le risorse a essa correlate (indirizzi IP pubblici, interfacce di rete, dischi di archiviazione) e il sistema operativo ospitato.

In tal senso, Azure offre una vasta gamma di scelta in base alle esigenze e configurabili comunque anche dopo la creazione della risorsa. Le dimensioni di una macchina virtuale dipendono dal carico di lavoro pianificato e influiscono sulla potenza computazionale e sulla memoria. Dimensioni differenti, pertanto, generano costi molto diversi tra di loro. [7] [9] [10] [11]

L’utilizzo di **Availability Set** (set di disponibilità) garantisce che una o più macchine virtuali siano replicate e tolleranti ai guasti. Questa è una configurazione statica, ossia stabilita in fase di creazione della risorsa e non modificabile in seguito. Inoltre, è possibile gestire un numero sufficientemente ampio di macchine virtuali in parallelo, attraverso l’uso di **VM Scale Set**: si tratta di uno strumento che raggruppa e organizza contemporaneamente un insieme di Virtual Machine gestendone la scalabilità orizzontale sulla base di metriche ben definite (utilizzo di CPU, traffico di rete, numero di operazioni su disco al secondo).

Una componente altrettanto significativa nel calcolo dei costi per una macchina virtuale su Azure dipende dal sistema operativo ospitato. La scelta comprende numerose immagini, per le quali il prezzo calcolato su base oraria comprende anche le licenze che esse offrono di default: per quanto riguarda Windows le versioni disponibili comprendono Windows 10 Pro, Windows Server (2012, 2016, 2019) e numerose versioni di SQL Server. Tuttavia, la compatibilità è estesa anche alla creazione di macchine virtuali Linux, permettendo di ospitare distribuzioni più o meno diffuse come Ubuntu Server, Red Hat Enterprise, SUSE Linux Enterprise Server, Debian e CentOS. I sistemi operativi Linux sono di solito i più economici, trattandosi di istanze che non richiedono alcun addebito legato a licenze software. Infine, una terza opzione riguarda la disponibilità a usare delle immagini personalizzate, definite dall’utente in fase di creazione.

L’attività di una Azure Virtual Machine attraversa diversi **stati**, a partire dalla sua creazione fino alla sua eliminazione. Dopo una iniziale fase di avvio, essa è operativa e pronta all’uso (**Running**). Per l’intero tempo di attività, essa permane in questo stato. Talvolta può essere utile arrestare questa risorsa attraverso le API di PowerOff oppure dall’interno del sistema operativo, facendola transizionare verso lo stato di **Stopped**. In alternativa, se la macchina virtuale viene arrestata attraverso le API di Azure (portale Azure, Powershell, Azure CLI) essa passerà invece allo stato di **Deallocated**. La differenza tra questi due stati è notevole: se una VM è Stopped disporrà ancora dell’hardware a essa dedicato, continuando a pesare sulla fattura del cliente; se la si porta in stato Deallocated, al contrario, essa viene privata delle risorse finora dedicate, mettendo in pausa anche il costo per il suo utilizzo. Questa è la soluzione più adatta nel caso in cui non si voglia impiegare una macchina virtuale per periodi molto lunghi.

Le opzioni di pagamento tra cui è possibile scegliere dipendono dalla flessibilità che l'utente vuole per le proprie istanze di macchina virtuale:

- Un'opzione **Pay-As-You-Go** (PAYG) consente di pagare solo per il tempo e le risorse che sono utilizzate; in questo modo, per interrompere l'erogazione di un servizio, è sufficiente deallocare (o eliminare) le risorse di cui si dispone. Questa soluzione è ideale per applicazioni con carichi di lavoro non uniformi e predicibili, ma anche per ambienti molto piccoli o che vengono impiegati *una tantum*;
- Un'opzione di **istanze riservate**, in cui le macchine virtuali vengono mantenute per periodi molto lunghi (da uno fino a tre anni). Una via di questo tipo è preferibile se l'applicazione gestita richiede attività predicibile e continua, anche per periodi molto lunghi. Se si pianifica di mantenere in stato di attività le istanze per molto tempo, questo espediente risulta più economico di PAYG. [7] [9] [10] [11]

Le macchine virtuali di Azure necessitano di uno spazio di archiviazione dedicato, per poter simulare del tutto l'esecuzione di un sistema operativo. In questo, **Azure Managed Disks** consente l'uso di dischi di archiviazione gestiti, trattati come dei dischi fisici ma organizzati in un ambiente virtualizzato.

Questi dischi possono differire in prestazioni, e offrono quindi diverse capacità (32 GB, 64 GB, 128 GB, 256 GB, 512 GB, 1 TB, 2 TB, 4 TB) e soprattutto varie tipologie di archiviazione: da **HDD Standard**, la più economica e meno performante, passando per **SSD Standard** e **SSD Premium**, fino alla più costosa **Ultra**. I dischi gestiti sono stati creati per garantire disponibilità quasi totale, grazie alla creazione di tre repliche dei dati, e ottima tolleranza ai guasti. Come detto, il numero di dischi collegabili a una macchina virtuale dipende dalla dimensione della stessa; all'interno di una singola sottoscrizione, inoltre, è consentito creare fino a 50 000 dischi.

Dal punto di vista della sicurezza, Azure consente due tipi di crittografia: lato server, crittografando i dati inattivi, e *Disk Encryption*, che cifra sia i dischi del sistema operativo che quelli usati dalle macchine virtuali. In entrambi i casi, comunque, è possibile gestire le chiavi di crittografia oppure delegare questo compito ad Azure Key Vault.

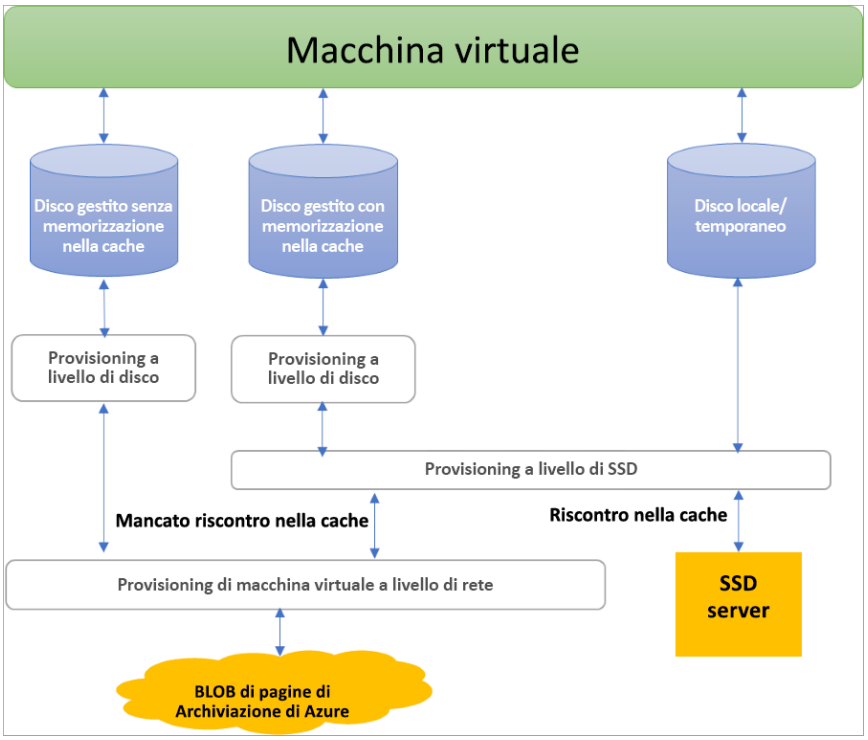

Fig. 14: Organizzazione dei dischi di una VM

Esistono, in generale, tre ruoli per i dischi in Azure: il **disco dati** è collegato alla macchina virtuale ed è utilizzato per archiviare i dati dell'applicazione (o altri dati necessari). Il **disco del sistema operativo**, necessario per ogni macchina virtuale, dispone del sistema operativo preinstallato. Infine, è utile l'impiego di **dischi temporanei** che offrono *storage* a breve termine. Questi dati, essendo presenti all'interno di un disco non gestito, non sono permanenti e potrebbero essere persi in fase di manutenzione o aggiornamento della macchina virtuale.

In caso di arresti anomali del sistema di un disco gestito è possibile effettuare delle copie di sola lettura (*snapshot*) che ne garantiscono il *backup*. Operazione analoga riguarda l'acquisizione di un'immagine personalizzata di un sistema operativo, realizzabile a partire da una macchina virtuale (che, per esempio, si intende deallocare). Questa strategia è utile se, in seguito, si intende ricreare la stessa istanza includendo anche i dischi gestiti. In questi casi, i dati acquisiti sono importati all'interno di uno Storage Account. [7] [11]

Gli strumenti di automazione di Azure riguardano in particolar modo le macchine virtuali. Per la loro configurazione, infatti, viene offerto un potente mezzo di configurazione e pianificazione. Per esempio, basti pensare al **Desired State Configuration** (DSC) di Powershell, che permette di portare la propria macchina virtuale nello stato desiderato in fase di *deployment* (installazione di pacchetti, autorizzazioni, ecc.).

La strategia con cui tali operazioni, più o meno invasive, vengono eseguite su una o più istanze di macchine virtuali prende il nome di **Estensione**. Attraverso la definizione di una serie di parametri (molti dei quali dipendono dalla specifica estensione che si intende applicare), è possibile personalizzare l'istanza di VM creata. Oltre alle estensioni DSC sono disponibili numerose altre configurazioni: alcune, fornite nativamente da Microsoft, introducono strumenti di sicurezza, monitoraggio e gestione del servizio di Active Directory; altre, personalizzate direttamente dall'utente, consistono nell'esecuzione di script.

Questo progetto di tesi prevede l'uso di entrambe le soluzioni ma soprattutto l'uso di script Powershell. Gli script specificati all'interno di un'estensione devono essere reperibili online e ne richiedono, dunque, l'URI (che si tratti di un file presente sul web o in uno Storage Account di Azure). Ovviamente, il concetto di estensione è valido anche per set di scalabilità, ossia è applicabile anche a gruppi arbitrariamente ampi di macchine virtuali gestite con gli stessi criteri.

Come per tutte le risorse Azure, anche per le estensioni è possibile specificare delle dipendenze all'interno dell'ARM *template*: banalmente, l'estensione di una risorsa è vincolata alla creazione della stessa, ma è possibile anche creare estensioni che dipendano da altre. Da notare, infine, che per una macchina virtuale è possibile eseguire solo uno script personalizzato e che quindi estensioni multiple non possono essere applicate in parallelo alla stessa risorsa. [7] [12] [18]

Un ultimo artificio impiegato relativamente all'Azure Resource Manager riguarda l'uso di **modelli di distribuzione innestati**: inserire un modello all'interno di un altro, infatti, permette di gestire le risorse in maniera consequenziale. Quando un modello innestato viene applicato, esso può essere eseguito con **modalità completa** (andando a eliminare tutte le risorse nel gruppo che in esso non sono specificate) oppure

incrementale (lasciando immutate le risorse non citate o applicandovi degli aggiornamenti, per esempio estensioni, dove specificato). Quest'ultimo caso, in particolare, ritorna molto utile allo scenario descritto in questa tesi. [7]

```
{
  "$schema": "https://schema.management.azure.com/schemas/2019-04-01/deploymentTemplate.json#",
  "contentVersion": "1.0.0.0",
  "parameters": {},
  "variables": {},
  "resources": [
    {
      "name": "nestedTemplate1",
      "apiVersion": "2017-05-10",
      "type": "Microsoft.Resources/deployments",
      "properties": {
        "mode": "Incremental",
        "template": {
          <nested-template-syntax>
        }
      }
    }
  ],
  "outputs": {
  }
}
```

Fig. 15: Schema di un template innestato

2.2.7 SQL IN AZURE

In molti scenari in cui l'infrastruttura si compone di risorse organizzate da Azure è buona prassi migrare anche l'aspetto di gestione dei database sul cloud. In questo modo i dati sono archiviati con elevate disponibilità e prestazioni. In particolare, i database SQL su Azure possono essere gestiti in tre maniere differenti.

Se si utilizza un **singolo database**, esso è totalmente gestito e isolato, e le risorse che vi vengono dedicate possono essere arbitrariamente modellate anche dopo la sua creazione. Comunque, si tratta di una soluzione per ambienti estremamente piccoli e con accessi prevedibili. [7] [9] [14]

Se, al contrario, sono richiesti più database, le soluzioni possono essere diverse. Un ***pool* elastico** consiste in un insieme di database SQL che fornisce una gestione rapida delle dimensioni del server, le cui risorse sono condivise tra i database a un prezzo specifico. Attraverso il *pool* elastico l'allocazione della capacità computazionale è modellata sulla base delle necessità correnti. Particolarmente adattabile allo sviluppo di applicazioni SaaS, questo modello consente vantaggi in caso di uso medio basso di risorse, con picchi poco frequenti. [7] [9] [14]

In alternativa, Azure offre la possibilità di creare un **server di database SQL**. Si tratta di un'astrazione logica che funge da *broker* per comunicare con i database, siano essi singoli o in *pool*. La creazione di un server SQL è necessaria per ospitare i database singoli che ne faranno parte; inoltre essi apparterranno alla stessa regione del server cui sono legati. Una risorsa di questo tipo può contenere al proprio interno database, data warehouse e *pool* elastici, e necessita della creazione di un'identità amministratrice, con la possibilità di implementare anche gli strumenti di autenticazione forniti da Azure Active Directory. Questi modelli offrono spazio di archiviazione fino a 100 TB per ogni database. [7] [9] [14]

Ulteriore opzione, in fase di migrazione nel cloud, è l'**istanza gestita SQL**: essa consiste in una raccolta di database con un gruppo di risorse pronte all'uso. Si tratta di un modello intermedio rispetto a quelli elencati in precedenza, in quanto offre funzionalità aggiuntive. È comparabile a un'istanza di Windows SQL Server e dispone di una rete virtuale per far fronte ai problemi di sicurezza, offrendo compatibilità quasi totale con SQL Server locale. La peculiarità delle istanze gestite SQL riguarda la migrazione verso il cloud garantendo un ottimo margine di continuità con la soluzione *on-premise*, pur continuando a garantire le funzionalità dei modelli PaaS (aggiornamento continuo, disponibilità e backup automatici). Sono resi disponibili 8 TB in totale per i database gestiti. [7] [9] [14]

Infine, per casi di applicazioni che necessitano di un accesso a livello di sistema operativo, Azure offre un modello IaaS di **Macchine Virtuali SQL**. In questo caso la compatibilità con le soluzioni *on-premise* è pressoché totale, garantendo migrazione rapida al cloud. Questo specifico caso fornisce un controllo completo sull'istanza SQL Server. I costi relativi sono gli stessi di una macchina virtuale; se si dispone di una

licenza SQL Server, inoltre, è possibile utilizzarla evitando di pagarne una nuova. Inoltre, proprio in quanto Virtual Machine, si può organizzare l'allocazione e la deallocazione delle risorse a essa dedicate. A differenza dei modelli precedenti la manutenzione, l'applicazione di *patch*, la configurazione del servizio e dei *backup* e l'organizzazione dei permessi sono in mano all'utente che gestisce la macchina virtuale. Il dimensionamento di una macchina virtuale può arrivare fino a 256 TB di spazio di archiviazione.

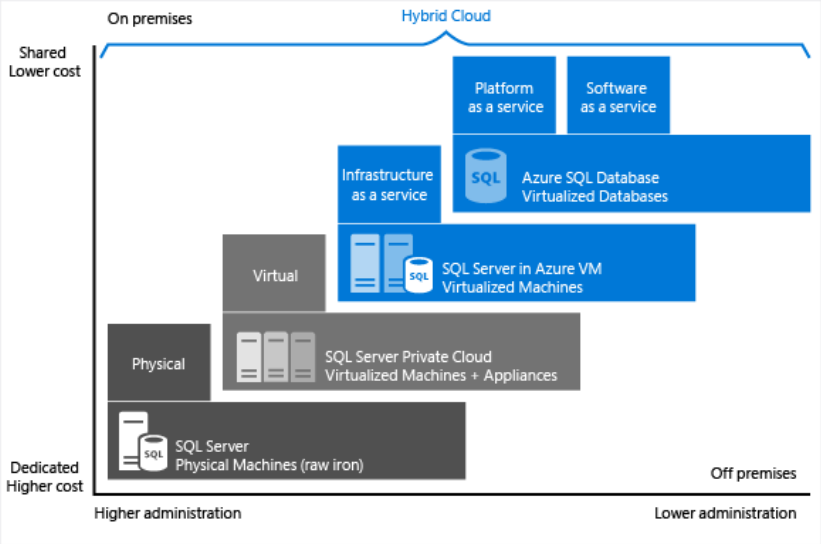

Fig. 16: Rapporto costi- amministrazione di SQL Server su Azure

La scelta di una di queste implementazioni dei database SQL dipende fortemente dallo scenario di cui si dispone:

- Dal punto di vista amministrativo, le soluzioni PaaS evitano onerose operazioni di gestione del database; di contro, anche le operazioni e gli script che vi si possono applicare sono ridotte. Basti pensare che il CLR (*Common Language Runtime*) non è supportato da database singoli e *pool* elastici;
- Le soluzioni PaaS richiedono delle modifiche per attuare la migrazione verso il cloud; questa esigenza è molto contenuta per le istanze gestite e assente per le macchine virtuali SQL;

- I database SQL e le istanze gestite addebitano costi ristretti a parità di spazio occupato dai database. Una macchina virtuale SQL, d'altro canto, ha prezzi maggiori (tenendo conto anche della licenza fornita da Microsoft) ma offre una ottima flessibilità, dovuta alla possibilità di deallocare le risorse dedicate quando non necessarie, anche per lunghi periodi. Per i modelli PaaS, invece, questo non è possibile, e le risorse a essi destinate non possono essere rimosse per l'intero ciclo di vita dei database. Questo si traduce in un dispendio economico anche notevole se si pianifica di impiegare questi database per periodi abbastanza ristretti. [7] [9] [14]

2.3 POWERSHELL

La configurazione delle macchine virtuali appartenenti all'infrastruttura creata in questo progetto è stata studiata in modo da essere totalmente automatizzata. Come visto, ciò è reso possibile attraverso l'impiego delle estensioni delle macchine virtuali di Azure che, una volta creata la risorsa in questione, permettono operazioni ampiamente personalizzabili che non richiedano l'intervento manuale dell'utente, anche su larga scala. In particolare, per raggiungere questo obiettivo si fa uso pervasivo di **Windows Powershell**, sia per quanto riguarda l'esecuzione di **script** che per l'attuazione di **Desired State Configuration**.

Windows Powershell è uno strumento estremamente potente per la gestione di pressoché qualsiasi cosa all'interno della propria macchina (fisica o virtuale che sia), implementando funzionalità di *networking*, installazione, sviluppo software e controllo dei file. Può essere usato come prompt interattivo oppure come ambiente di *scripting*: in quest'ultimo caso, in particolare, si può progettare una sequenza di comandi che annullino (o riducano al minimo) l'intervento dell'utente in fase di esecuzione.

Powershell si basa su CLR .NET Framework e sulle API da esso fornite. Grazie a ciò, è possibile installare o creare da zero nuovi moduli, ossia componenti software che contengono i cosiddetti **cmdlet** (comandi singoli non nativi incorporabili nella *shell*).

Per esempio, moduli possono riguardare l'Active Directory, la gestione delle risorse su Azure, la creazione di certificati e la configurazione di un SQL Server.

Se usato in maniera adeguata, Powershell consente di effettuare qualsiasi operazione che, altrimenti, richiederebbe l'impiego di interfacce utente, prolungando i tempi di preparazione di ambienti specifici. Buona parte della capacità di utilizzo di Powershell consiste nell'uso che si fa dei *cmdlet*, dell'input che essi ricevono e dell'output restituito: lo *scripting* di Powershell ne organizza la gestione mediante oggetti che l'utente può modellare a seconda delle proprie necessità (che sia visualizzato su schermo, salvato in una variabile o utilizzato in pipeline più o meno complesse).

Per andare incontro alle esigenze degli sviluppatori, Windows offre un ambiente chiamato **Integrated Scripting Environment** (ISE), rendendo possibile eseguire comandi e *script*, sottoponendoli a *debug* in un ambiente isolato. Di seguito saranno illustrati i moduli e le piattaforme Powershell che l'infrastruttura qui presentata utilizza. [7]

2.3.1 DESIRED STATE CONFIGURATION

Desired State Configuration fa uso di una sintassi dichiarativa che offre pieno controllo dei sistemi utilizzati. DSC consente l'installazione di software o la configurazione di server nello stato desiderato. La realizzazione di una configurazione di questo tipo si compone di tre parti:

- un blocco di **configurazione**, che definisce le istanze delle risorse: esplicita, per esempio, i nodi (macchine fisiche e virtuali) da configurare, le proprietà che verranno usate e i comandi (con sintassi Powershell) di preparazione all'ambiente;
- un blocco di **risorse**, ossia la parte attiva di DSC. Si tratta di oggetti che comprendono delle proprietà configurabili e funzioni di *script* Powershell. Esse, dunque, contengono il codice necessario a portare e mantenere i nodi gestiti nello stato desiderato;
- un blocco di **gestione** di configurazione locale, cioè il motore di DSC per abilitare l'interoperabilità tra risorse e configurazioni. Esso viene eseguito in

ognuno dei nodi coinvolti e definisce modalità e frequenza di aggiornamento e di polling per verificare che lo stato corrente sia corretto (gestendo anche l'eventuale ripristino). [7] [12]

2.3.2 AZURE AD

La gestione di Azure Active Directory mediante *cmdlet* di Powershell è affidata al modulo **AzureAD**. È possibile, per esempio, configurare le applicazioni presenti, crearne nuove e modificare le impostazioni di *Single Sign-On*. In più, si possono impostare i settaggi delle *directory*, delle *policy* e la gestione dei certificati e delle *Certification Authority* dell'Azure AD. Un ulteriore insieme di comandi molto vario, infine, amministra le utenze sotto ogni aspetto: credenziali, gruppi, ruoli e dispositivi collegati. [7]

2.3.3 AZ POWERSHELL

Il più recente modulo Powershell per la gestione di tutto ciò che ha a che fare con Azure (sottoscrizioni, risorse e gruppi di risorse) è stato rilasciato a dicembre 2018 e prende il nome di **Az**. Esso è stato ideato per fornire comandi più stabili, intuitivi e interoperabili rispetto al predecessore AzureRM. In realtà, il modulo Az si compone di una spropositata varietà di sotto-moduli. Di questi, ognuno si occupa di un particolare aspetto di Azure, e consentono di realizzare qualsiasi cosa sia possibile effettuare anche attraverso l'interfaccia del portale Azure. Quelli legati alla realizzazione dell'infrastruttura di macchine virtuali su HPC comprendono prevalentemente il modulo **Az.Resources** (per la creazione, eliminazione gestione di gruppi di risorse, oltre che per il *deployment* di ARM *template*); il modulo **Az.Compute**, invece, si occupa dell'organizzazione delle macchine virtuali e delle operazioni correlate (avvio, deallocazione, riavvio, ridimensionamento). Altri moduli rilevanti sono **Az.KeyVault**, per la manutenzione di segreti e certificati, e **Az.Storage** per le varie soluzioni di archiviazione dei dati. Nel Capitolo 3 saranno approfonditi i *cmdlet* appartenenti ai vari moduli di cui si fa uso.

Innanzitutto, l'accesso a Azure abilita la gestione diretta delle risorse; ciò avviene tramite il comando **Connect-AzAccount,** parte del modulo Az.Accounts. I parametri di cui esso necessita sono facoltativi: possono specificare ID del *tenant*, della sottoscrizione, contesto e tipo di autenticazione. In particolare, la tecnica più elementare e diffusa richiede la definizione di un oggetto di tipo System.Management.Automation.PSCredential (composto da mail e password dell'utente Azure), ma è possibile gestire anche altre modalità, tramite credenziali di Azure Active Directory, un file di impostazioni o un certificato. [7] [15] [19] [20]

3. MICROSOFT HPC PACK

Finora sono stati illustrati gli strumenti adibiti alla preparazione logica dell'infrastruttura messa a disposizione da ERMAS. Comunque, il cuore della struttura applicativa risiede nel calcolo distribuito e, in particolar modo, nell'uso di **Microsoft HPC Pack**. HPC (*High-Performance Computing*) è uno strumento di automazione e coordinamento delle risorse di calcolo di cui si dispone, elaborato per garantire flessibilità massima e al contempo orchestrazione delle istanze che appartengono al sistema gestito. Il suo obiettivo principale riguarda il processamento di quantità di dati molto grandi con prestazioni ottimizzate, e trova applicazione soprattutto in *Machine Learning*, ricerca scientifica e scenari di simulazione. Un *cluster* HPC, di solito, è costituito da numerosi server di potenza computazionale non troppo elevata con capacità di sincronizzazione, ed è orchestrato da uno o più nodi dedicati con compito di scheduling: si preferisce una rete di computer di complessità intermedia, facilmente sostituibili in caso di guasti, piuttosto che pochi calcolatori estremamente potenti e costosi, difficilmente rimpiazzabili.

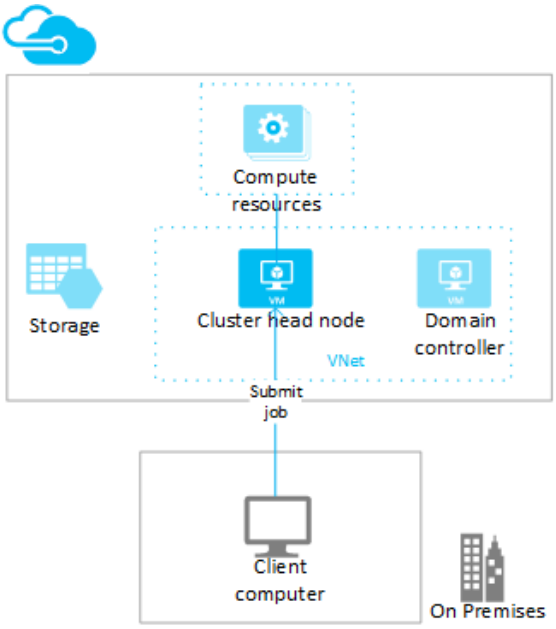

Fig. 17: Modello di HPC su cloud

Come si vedrà, HPC Pack è compatibile con le soluzioni *on-premise* (in cui i nodi di calcolo sono macchine fisicamente a disposizione dell'utente), ma il rendimento diventa esponenzialmente maggiore se si opera su Azure attraverso l'uso di macchine virtuali. La flessibilità nel loro dimensionamento, infatti, consente un design ottimizzato della gestione delle risorse di calcolo, modellabili anche in tempo reale in base all'utilizzo. Il costante controllo del loro impiego consiste in un potente strumento di ampliamento e riduzione delle risorse di calcolo al fine di ottimizzare i costi. È palese che l'utilizzo di tecnologie IaaS e PaaS di Azure offre soluzioni più adattabili rispetto all'impiego di macchine fisiche a disposizione dello sviluppatore, che richiedono manutenzione e aggiornamenti manuali e scarsa possibilità di modellare le risorse disponibili.

In particolare, le istanze di macchina virtuale più pertinenti alle tecnologie di High-Performance Computing sono quelle appartenenti alle *A-series*, per l'ottima capacità di *networking* e latenza molto contenuta, alle *N-series*, per applicazioni a intensivo uso di calcolo e capacità grafiche (*deep learning*, *gaming*, virtualizzazione) e alle *H-series*, nativamente adattabili a HPC. Tali macchine virtuali possono ospitare istanze di Windows Server 2012 e 2016, oltre che server Linux. Esistono diverse versioni di HPC (2008, 2012, 2016, 2019). In questo caso ci si focalizzerà su quello attualmente più usato, HPC Pack 2016. [7] [14] [21]

3.1 RUOLI E CONFIGURAZIONI IN HPC

Innanzitutto, è bene chiarire la struttura delle macchine appartenenti al cluster HPC. Il ruolo di orchestrazione, comunicazione con i nodi di calcolo e distribuzione dell'esecuzione delle applicazioni è delegata a uno o più **Head Node**. Questi non hanno il compito di elaborazione della logica applicativa ma si occupano di operazioni di più alto livello: verificano lo stato di avanzamento dell'esecuzione, gestiscono il *cluster*, monitorano lo stato di utilizzo, il bilanciamento del carico e il *polling* alle istanze gestite. Solitamente, il numero di Head Node è molto contenuto e in stretto rapporto con il numero di nodi di calcolo. Questi ultimi prendono il nome di **Compute Node**: si tratta delle istanze che si occupano dell'effettivo carico di lavoro da eseguire. Il loro compito, dunque, è limitato all'esecuzione dei comandi a essi delegati sotto

forma di *job* (solitamente si tratta di eseguire delle applicazioni appositamente realizzate, con un ben determinato insieme di dati di input). Il vero punto di forza di HPC, pertanto, consiste nella realizzazione di applicazioni che non tengano conto delle risorse di calcolo sottostanti, in quanto la distribuzione delle operazioni sulle varie istanze di macchina (fisica o virtuale) viene delegata agli Head Node in maniera del tutto trasparente.

HPC Pack supporta anche l'utilizzo di macchine remote che ospitano **Microsoft SQL Server**. Esso usa cinque database differenti dedicati alle informazioni circa **job scheduling**, **gestione del cluster**, **reportistica**, **diagnostica** e **monitoraggio**. Questi database possono risiedere nell'Head Node (impostazione di default in soluzioni con un singolo Head Node) oppure in un server remoto. Quest'ultima configurazione migliora l'efficienza del *cluster*, alleggerendo il carico di lavoro dell'Head Node. La scelta può ricadere su macchine SQL Server oppure, in soluzioni basate su Azure, su istanze di *pool* elastico, istanze gestite o macchine virtuali.

HPC Pack definisce in maniera chiara una separazione nei ruoli delle macchine coinvolte nel cluster e, quindi, nelle loro funzionalità.

- Gli **Head Node** si occupano dello *scheduling* dei *job* e delle richieste SOA (Service Oriented Architecture) delle applicazioni client, comunicando con i Broker Node;
- I **Broker Node** ricevono le richieste SOA e le distribuiscono ai nodi che ospitano il servizio desiderato; dunque, raccolgono le risposte e le inoltrano all'applicazione client:
- I **Workstation Node** si occupano di ospitare servizi SOA, ma possono anche eseguire *job*;
- I **Windows Azure Worker Node** eseguono i *job* richiesti e possono ospitare servizi SOA;
- Gli **Unmanaged Server Node** sono istanze che possono essere usate occasionalmente per eseguire *job* o per fornire servizi di *networking* e *file sharing*.

Una considerazione importante riguarda il ruolo dell'Head Node che di default può fungere anche da Broker e Compute Node, se portato in stato Online. Lo stesso può

essere fatto per i Broker Node, che possono gestire anche il ruolo di Compute Node in scenari di alta disponibilità. [7]

I nodi HPC Pack, qualsiasi sia il loro ruolo, sono descritti da *State* e *Health*, grazie al monitoraggio costante cui sono sottoposti. Il **Node State** definisce lo stato di *deployment* e la sua disponibilità: un nodo è **Online** se può accettare ed eseguire *job*. Pertanto, lo *scheduling* di un *job* si occupa di distribuire l'esecuzione tra i soli nodi Online. Questo stato può essere configurato dall'amministratore del Cluster mediante l'Head Node. Lo stato **Offline**, al contrario, specifica che un nodo non può eseguire *job* e, nel caso di Broker Node, che non può gestire sessioni SOA. Lo stato **Unknown** indica che un nodo non è parte del *cluster* o che si è verificato un fallimento. Gli stati di **Provisioning**, **Removing**, **Stopping**, **Starting** e **Draining** sono transitori. **Rejected** indica che un nodo è stato rifiutato dall'amministratore del *cluster*, mentre **Not-Deployed** (valido solo per gli Azure Node) specifica che il nodo in questione è stato definito, ma l'istanza relativa non è stata ancora creata su Azure.

Il **Node Health**, invece, descrive lo stato di *warning* o errore del servizio HPC su un nodo: l'assenza di fallimenti è indicata dallo stato **OK**; al contrario, si può incorrere in **Warning** (test diagnostici falliti) o **Error** (nodo non raggiungibile, rifiutato o *deployment* fallito). Se un nodo non appartiene al *cluster* o non è ancora stato creato (Azure Node) è marcato come **Unapproved**, mentre **Transitional** rappresenta una fase transitoria (avviato, modificato, o portato in stato Online/Offline).

I *cluster* che ospitano HPC possono essere realizzati in accordo con diverse topologie supportate nativamente dal sistema:

- **Compute Node isolati in una rete privata**: il traffico da e verso i nodi di calcolo passa attraverso l'Head Node. In questo scenario i Compute Node non sono direttamente accessibili dall'utente, e le risorse (database e file server) non sono direttamente visibili dai nodi di calcolo. Situazioni di questo tipo potrebbero tradursi in colli di bottiglia;
- **Tutti i nodi in una rete privata e in una rete *enterprise*, compreso l'Head Node**: il traffico, a differenza dello scenario precedente, può essere indirizzato direttamente verso i nodi di calcolo. Inoltre, questi ultimi sono accessibili all'utente;

- **Compute Node isolati in due reti, una privata e una legata all'applicazione**: questo modello offre migliori prestazioni, separando il *Node Management* del *cluster* dalla logica applicativa (accesso ai dati). Anche in questo caso, tuttavia, i nodi di calcolo non sono direttamente accessibili all'utente;
- **Tutti i nodi connessi sulle reti *enterprise,* privata e applicativa**: vengono messi insieme i vantaggi dei modelli precedenti, consentendo comunicazione diretta con i Compute Node tramite la rete *enterprise*, accessibilità dell'utente a essi e gestione del *cluster* in una rete dedicata;
- **Tutti i nodi in un'unica *enterprise network***: il traffico *enterprise*, applicativo e del *cluster* è instradato in un'unica rete. [7]

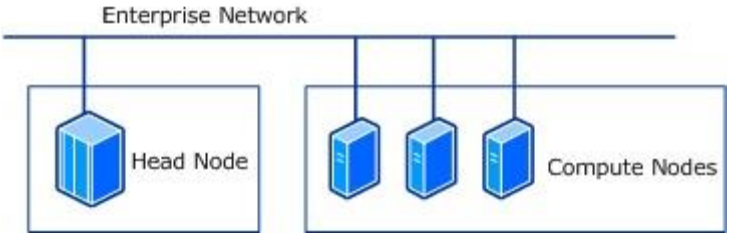

Fig. 18: Topologia di comunicazione HPC su enterprise network

3.2 SOLUZIONI DI HPC

HPC Pack offre tre modelli principali di *deployment* a seconda del tipo di nodi di cui si dispone, del tutto trasversali alla topologia scelta, alla logica applicativa e al numero di macchine impiegate. Innanzitutto, è possibile disporre di un proprio ***cluster on-premise***, secondo l'architettura e il *networking* desiderato. Ovviamente in questo caso la manutenzione e l'aggiornamento delle macchine sono totalmente a carico dell'utente. Il controllo del *cluster*, invece, è in mano al dominio Active Directory: le macchine *on-premise* che si intende aggiungere al *cluster* HPC devono essere parte del dominio. Perciò, prima dell'installazione di HPC, è necessario effettuare le operazioni utili all'ingresso nell'AD. A seconda dell'organizzazione aziendale, può essere preferibile creare un dominio a parte per il *cluster* (politiche di connessione e di gestione differenti). L'impiego di un dominio garantisce un sistema fortemente basato

sui ruoli e sul controllo degli accessi: l'installazione di HPC su un Head Node, per esempio, richiede che l'utente corrente sia un amministratore del dominio. La comunicazione nel *cluster*, infine, necessita di certificati: uno per l'Head Node stesso, l'altro per gli altri nodi. [7] [14] [21]

Una strategia intermedia consiste nell'uso di un'**architettura HPC ibrida**, ottenuta combinando le macchine *on-premise* di cui l'utente dispone con le tecnologie cloud di Azure: rispetto al modello precedente, un approccio ibrido garantisce maggiore flessibilità nell'uso di risorse computazionali. Utilizzare nodi di calcolo Azure, oltre a evitare lunghe procedure di manutenzione, semplifica il processo di adattamento delle dimensioni del *cluster* alle esigenze correnti. Una politica ibrida consente di migrare sul cloud risorse di vario tipo quali l'Head Node, i Compute Node e i Set di Scalabilità. Inoltre, alcune operazioni aggiuntive sono richieste: dal momento che le risorse computazionali distribuite su Azure necessitano di comunicare con la rete *on-premise*, vanno modificate diverse impostazioni di connessione. Per creare un collegamento tra la rete *on-premise* e quella virtuale (cui appartengono le risorse Azure) è fondamentale l'uso di una **ExpressRoute**, che garantisce una connessione privata e sicura (non utilizzando la rete pubblica di Internet), oppure di un **Gateway VPN**, che al contrario usa una connessione pubblica su Internet. [7] [14] [21]

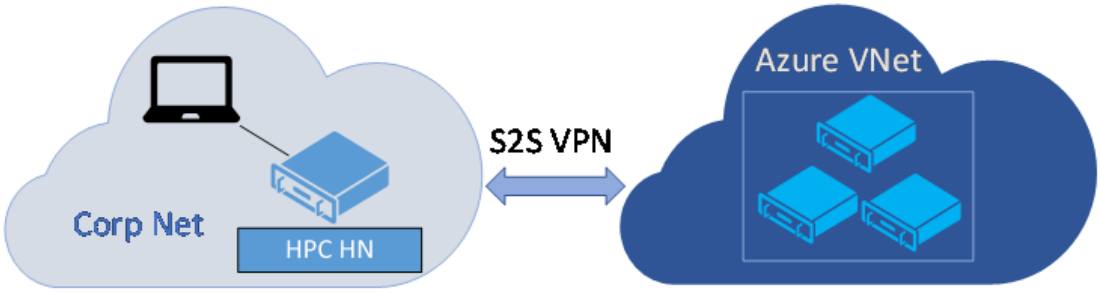

Fig. 19: Comunicazione all'interno di un cluster ibrido

Infine, HPC Pack mette a disposizione dell'utente un'opzione **totalmente sviluppata sul cloud** di Azure costituita da macchine virtuali. Prerequisito fondamentale è l'uso di almeno un certificato, distribuito su un Key Vault e impiegato dall'Head Node per la comunicazione nel *cluster*. La creazione di nuovi nodi di calcolo ne richiede numero

e dimensionamento come input. La soluzione presentata in questa tesi adotta la strategia di migrazione integrale su cloud, facendo leva sugli strumenti di Azure AD per la gestione del cluster (con i già citati vantaggi per la gestione delle identità, degli accessi e del *Single Sign-On*). [7] [14] [21]

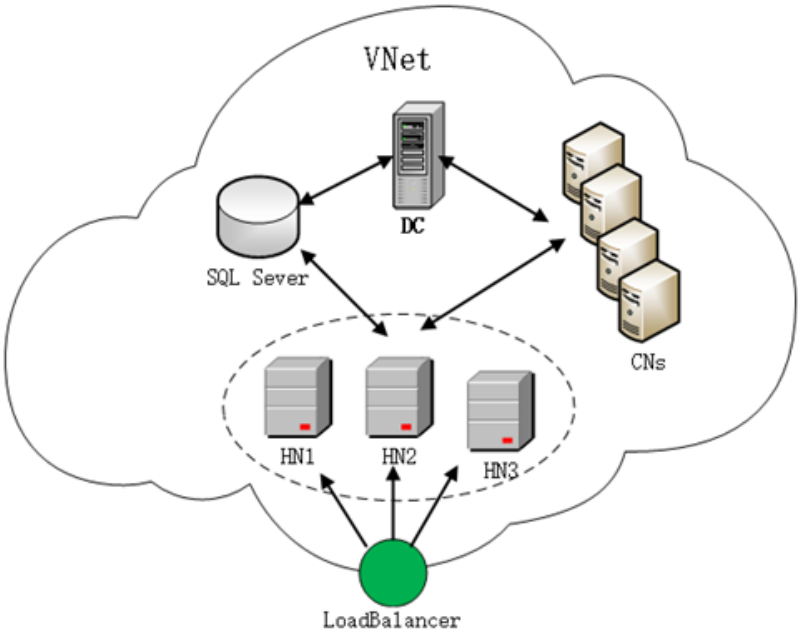

Fig. 20: Soluzione Cloud con Domain Controller e SQL Server dedicati

Se si opera in uno scenario distribuito su Azure, i nodi che risiedono sul cloud possono essere sottoposti a costante ottimizzazione (manuale o automatica). Si tratta della cosiddetta proprietà **AutoGrowShrink** del *cluster*. Essa è applicabile a ogni tipo di risorsa di calcolo presente su Azure (nodi PaaS o IaaS, Set di Scalabilità e *pool* Batch), e garantisce un controllo più o meno fine sulle unità disponibili. Ciò che AutoGrowShrink permette è la deallocazione di risorse inutilizzate, o l'allocazione di risorse quando necessario. Il suo punto di forza è la capacità automatica di controllo, estremamente configurabile: permette, infatti, di decidere quanto a lungo un nodo debba essere inattivo prima di essere stoppato, o quante risorse devono essere attivate in caso di maggiore stress computazionale. Un limite importante di questo scenario, tuttavia, è che non è possibile creare automaticamente nuove risorse di calcolo: esse,

infatti, dovranno già essere presenti all'interno di uno specifico gruppo di risorse. Pertanto, se richiesto, ulteriori nodi dovranno essere creati manualmente. [7]

La creazione di nuovi nodi di calcolo all'interno del proprio *cluster* HPC è delegata all'Head Node e richiede alcune operazioni preliminari, in base al tipo di risorse che si intende gestire. I nomi assegnati a ciascuno dei Compute Node creati vengono originati da un nome base, in cui è specificato un valore numerico identificativo incrementale. Ovviamente, in tutti gli scenari è richiesta una sottoscrizione Azure valida.

Se si desidera creare nuovi nodi **Azure PaaS**, bisogna preparare un certificato auto-firmato dall'Head Node, dopodiché vanno specificati un Azure Cloud Service e uno Storage Account. La procedura di creazione e avvio vera e propria è estremamente elementare (sia tramite interfaccia che con HPC Powershell), poiché necessita delle sole informazioni di dimensionamento e numero dei nodi di calcolo da creare.

Se, come nello scenario qui presentato, si intende attivare un cluster di **Azure IaaS Compute Node** (ossia macchine virtuali create su Azure), è necessario configurare le impostazioni di connessione: una rete virtuale nel caso di *cluster* distribuito su cloud, un *gateway* di rete nel caso di soluzione ibrida. Per l'aggiunta di nuovi nodi vanno specificate alcune informazioni aggiuntive: la creazione di risorse di calcolo richiede l'uso di una sottoscrizione (con autenticazione tramite credenziali oppure tramite Azure AD) e l'applicazione AAD collegata che si intende usare per gestire i nodi (con il certificato a essa relativo). Comunque, quest'ultima può anche essere creata in fase di configurazione di HPC. In seguito, i dati relativi alla creazione delle macchine virtuali consentono una configurazione corretta: gruppo di risorse in cui si opera, regione che lo ospita, nome della rete virtuale e della sottorete. Infine, indicando il Key Vault e il certificato di gestione di HPC (creati in fase di realizzazione del *cluster*), è possibile procedere alla creazione di nodi di calcolo. Tutte queste operazioni, comunque, sono attuabili sia tramite Powershell che mediante l'interfaccia proposta da HPC.

I Compute Node IaaS di Azure sono realizzati a partire da un oggetto di HPC noto come **Node Template**. In breve, esso specifica informazioni dettagliate sulla macchina virtuale da creare (*networking*, dominio, sistema operativo) e può indicare anche

operazioni aggiuntive. Una volta creato un nodo di calcolo, infatti, può essere richiesta la copia di alcuni file o l'esecuzione di uno *script*: ciò deve essere indicato sotto forma di comando. In fase di creazione e di avvio del nodo, infine, bisogna esplicitare il numero di macchine virtuali desiderate, il loro dimensionamento e il Node Template di riferimento.

Un'ultima tipologia di nodi implementabili tramite HPC consiste in un *pool* di nodi **Azure Batch**. Si tratta di una tecnologia ibrida tra IaaS e PaaS, studiata per risolvere in maniera efficiente processi paralleli e di HPC su larga scala. In particolare, Azure Batch è stato realizzato in modo da trasformare o analizzare grandi quantità di dati (per esempio applicazioni di *deep learning*, rendering d'immagini, simulazioni Monte Carlo) con livello di parallelizzazione estremamente elevato. Infatti, i vari nodi di calcolo non sono progettati per comunicare tra di loro, ma è possibile implementare questa caratteristica attraverso le API MPI (Message Passing Interface).

L'uso di nodi Azure Batch consiste in una serie di operazioni: innanzitutto, uno Storage Account deve ospitare i file di input e le applicazioni per l'elaborazione. Dopo aver creato un *pool* di nodi Azure Batch all'interno di un Azure Batch Account, il lavoro viene suddiviso in *job* (a loro volta composti da unità di lavoro elementari, note come *task*). Azure offre numerosi strumenti per il monitoraggio dei nodi e delle attività svolte durante l'esecuzione. Infine, i nodi Azure si coordinano per restituire l'output desiderato.

La creazione di unità di calcolo Azure Batch tramite HPC necessita di un Account Batch e di informazioni di Azure AD (ID e chiavi dell'applicazione, ID del *tenant*, ecc.). Come nello scenario di nodi IaaS, HPC Pack richiede la creazione di un Node Template (con impostazioni di configurazione differenti e specifiche dei nodi Azure Batch), dopodiché è possibile creare, avviare e arrestare un *pool* di nodi o delegarvi delle attività da eseguire. [7] [14] [21]

3.3 JOB E TASK

HPC Pack, comunque, non fornisce solo la gestione del *cluster*, ma anche l'organizzazione delle attività svolte dai nodi di calcolo. Si parla, in questo caso, di

job, ossia operazioni più o meno complesse assegnate all'**HPC Job Scheduler**. Un **job** è composto da uno o più **task**: questi definiscono il flusso di esecuzione in grado di fornire maggiore parallelismo, mediante la definizione di dipendenze tra *task*. Perché un *job* sia accodato nello *scheduler* di HPC, esso deve essere creato a partire da un **Job Template**. Una volta avviato, dunque, un *job* viene inserito in una coda: in base alle risorse di calcolo disponibili e alla priorità specifica lo *scheduler* si occuperà di gestire il flusso concorrente dei vari *task* e, più in generale, dei *job* presenti.

In HPC, dunque, un *job* è identificato da alcune proprietà fondamentali: tra gli altri la **priorità** (Lowest, BelowNormal, Normal, AboveNormal, Highest) consente di stabilire delle gerarchie nell'esecuzione, diverse impostazioni di **notifica** e di fallimento, il numero di **core**, **socket** e **nodi** (minimo e massimo) richiesti per l'esecuzione. È naturale, pertanto, che un *job* sia diviso tra più nodi di calcolo e, quindi, la sua esecuzione sia naturalmente parallelizzabile. Anche i *task* possono specificare numerose proprietà: in particolare, è possibile indicare anche per essi il numero di *core* e nodi richiesti, ma vanno specificati anche le **dipendenze** e i percorsi dei file di input e output. Ciò che indica nel dettaglio l'operazione da eseguire è rappresentato dalla proprietà **Command Line**, ossia il comando richiesto nel *task*.

Il ciclo di vita di un *job* è ben definito da HPC Pack da una serie di stati. Questi stessi sono validi anche per i *task*.

- Un *job* è in stato di **Submitted** se attende di essere validato per essere inserito in coda;
- Dopo una fase transitoria di **Validating** (consistente in una serie di controlli su permessi e impostazioni), il *job* è **Queued**, ossia schedulato e inserito in coda;
- La fase di esecuzione è rappresentata da **Running**;
- Se l'esecuzione è stata portata a termine con successo, lo stato finale è **Finished**; in caso contrario, si parla di **Failed**;
- Se per un *job*, in qualsiasi stato esso si trovi, è richiesta la cancellazione (dall'amministratore del *cluster*, dallo *scheduler* o dal *job owner*), esso andrà in **Canceled**. [7]

Tra i vari tipi di job paralleli che possono essere eseguiti da un *cluster* HPC, ve ne sono alcuni estremamente comuni. Tra questi, i **MPI Job** sono formati da *task*

intrinsecamente paralleli. Solitamente, si tratta di file eseguibili singoli che eseguono su più *core* in parallelo, garantendo comunicazione tra processi. I *task* MPI si distinguono per il fatto che la loro *Command Line* è preceduta da *mpiexec*.

Un **Parametric Sweep Job** è formato da un numero arbitrario di istanze della stessa applicazione (o comando singolo) che vengono eseguite in maniera concorrente. Solitamente non è prevista comunicazione tra i *task*, e i file di input e di output sono organizzati in modo da essere posti in una cartella comune (questo non costituisce comunque un vincolo).

Un **Task Flow Job**, al contrario, prevede una struttura dei *task* più complessa e basata su un ordine prestabilito di esecuzione: questo è rappresentato dalle dipendenze espresse, spesso perché un *task* ha un input che necessita (direttamente o indirettamente) dell'output di un altro *task*.

Un ultimo tipo di *job* riguarda l'approccio **Service Oriented Architecture** (SOA). Esso è realizzato per costruire sistemi distribuiti lascamente accoppiati. In questi modelli, funzionalità distinte sono organizzate in moduli software noti come servizi: essi sono creati in modo da essere quanto più esportabili e accessibili alle altre applicazioni. Il vantaggio principale di questo modello in HPC consiste nell'uso del calcolo distribuito senza la riscrittura di codice di basso livello. Lo scenario prevede un'applicazione client che fornisca mediante interfaccia l'accesso ai vari servizi offerti da HPC; lato server, il *job* è strutturato in modo da contenere un *task* chiamato Service (uno per ogni nodo di calcolo impegnato) che comunica con il Broker Node, inviando richieste e ricevendo risposte. [7]

Esistono diversi modi per comunicare con il gestore del *cluster* e dei *job*. Se si desidera un modello più automatizzato rispetto all'interfaccia, le API che HPC Pack mette a disposizione possono tornare molto utili: è possibile, infatti, **comunicare con l'HPC Scheduler** mediante linea di comando o, in alcuni scenari, tramite le API .NET. In particolare, esistono alcuni file eseguibili che possono essere invocati, mediante un'opportuna sintassi che ne definisce i parametri. Tra i più utili vi sono **cluscfg** (per la configurazione del Job Scheduler, che fornisce gestione di certificati, credenziali, parametri del *cluster* e sistema di notifica dello stato dei *job*), **clusrun** (per l'esecuzione in contemporanea del comando specificato su un insieme configurabili di

nodi del *cluster*), **hpcfile** (per la gestione dei file nei vari nodi del *cluster*), **hpctrace** (che si occupa del *logging* dei servizi HPC). Inoltre, una serie di comandi permette la manipolazione di *job*, *task*, nodi e applicazioni MPI: si tratta dei sottocomandi **job**, ***task***, **jobtemplate**, **node**, **mpiexec**.

Lo strumento di gestione del *cluster* nei suoi vari aspetti in maniera interattiva è rafforzato da una serie di *cmdlet* di HPC Powershell dalla sintassi estremamente semplificata. Esistono, dunque, comandi per creare, eliminare, avviare, arrestare le risorse di calcolo. È possibile anche ottenere una serie di informazioni circa tutto ciò che è correlato a HPC (*job*, nodi di calcolo di vario tipo, Node Template e Job Template tra gli altri). Per esempio, in uno scenario come quello che verrà presentato nel prossimo Capitolo, possono essere estremamente utili i comandi **Add-HpcNodeSet** e **Remove-HpcNodeSet**. Questi *cmdlet*, come il nome può suggerire, permettono di creare e rimuovere dal *cluster* delle istanze di macchina virtuale, delle quali sono specificate informazioni quali quantità, dimensionamento e Node Template. Inoltre, HPC Powershell garantisce il controllo dei *job* e delle richieste SOA gestire, oltre alla possibilità di importare o esportare Node Template e Job Template. Infine, consente (tra le altre proprietà) la configurazione di AutoGrowShrink con il comando **Set-HpcClusterProperty**. [7]

4. SOLUZIONE DI GESTIONE DI AZURE HPC

Gli strumenti ampiamente descritti nei capitoli precedenti permettono di introdurre e comprendere al meglio la realizzazione del progetto qui rappresentato.

Il proposito del lavoro svolto in questa tesi riguarda la fase di progettazione e sviluppo delle tecnologie HPC sviluppate sul cloud Azure. In particolare, l'obiettivo preposto è la migrazione delle risorse di calcolo computazionale da una soluzione *on-premise* a una prettamente virtualizzata, mediante gli strumenti di automazione delle risorse forniti da Microsoft.

I passi principali che hanno portato alla realizzazione di questo progetto riguardano:

- **Preparazione dell'ambiente di esecuzione** tramite script Powershell (o, equivalentemente, codice C# facendo uso delle API di Azure): in particolare la creazione di un gruppo di risorse, delle risorse e dei certificati preliminari per l'installazione di HPC e di uno Storage Account che fornisca i file applicativi e i database correttamente preconfigurati, pronti per il download e per l'impiego da parte delle macchine virtuali;
- **Realizzazione dell'infrastruttura** adibita all'uso di HPC Pack, comprendente macchine virtuali (ed estensioni) e risorse di rete attraverso l'uso di modelli di risorse di Azure Resource Management;
- **Creazione di una serie di script** legati alle macchine virtuali create, per consentire la configurazione ottimale legata all'ambiente di esecuzione: gestione dei database, dell'Active Directory, del *cluster*;
- Infine, un'ultima operazione consiste nel **comunicare direttamente con le API di HPC** attraverso la realizzazione di un programma sviluppato nel linguaggio C#. Esso permette all'utente di manipolare direttamente il proprio cluster, stabilendo il quantitativo di risorse computazionali create gestite con una serie di livelli di potenza di calcolo.

Il tutto sarà integrato all'interno dell'applicativo ERMAS offerto da Prometeia. Al suo interno l'utente avrà a disposizione un nuovo pannello che permetterà di monitorare lo stato del *cluster* e di cambiarne le proprietà in tempo reale. Le risorse ora possedute,

infatti, non sono più risorse fisiche in senso stretto ma virtualizzate e presenti sul cloud e, in quanto tali, totalmente manipolabili. Una volta distribuita l'infrastruttura, pertanto, sarà sufficiente utilizzare questo strumento per modellare le risorse di calcolo disponibili in base all'uso desiderato; ovviamente, la finalità principale di questa operazione riguarda una migliore gestione dei costi, ottimizzabili come descritto in precedenza.

Le scelte strutturali del progetto permettono di fornire all'utente un insieme di macchine virtuali su Azure, distribuite come normali risorse IaaS ma il cui controllo è fornito all'utente a livello PaaS. Per questo la scelta è ricaduta sull'uso di macchine virtuali: la possibilità di arrestarle e avviarle su necessità consente di sfruttare appieno il paradigma Pay-As-You-Go del cloud computing, garantendo un'ottimizzazione dei costi. Un discorso analogo può essere fatto per quanto riguarda l'istanza SQL Server: sebbene, come si vedrà nel Capitolo 5, il prezzo delle diverse soluzioni per ogni ora di utilizzo è pressoché simile per le varie soluzioni, una macchina virtuale risulta un'opzione più che valida a lungo termine. In fase di configurazione del modello, si è optato per l'uso di Powershell che consente nativamente l'automazione della preparazione delle macchine virtuali tramite la realizzazione di script personalizzati.

Di seguito sono mostrati gli elementi creati nel modello realizzato, dopodiché vengono presentati gli script e l'ARM *template* in ordine cronologico di utilizzo in fase di distribuzione delle risorse. Infine, la parte finale sarà dedicata alla presentazione del codice relativo al programma per la gestione in tempo reale dei nodi di calcolo.

4.1 MIGRAZIONE DI HPC SU AZURE

Per quanto riguarda l'infrastruttura realizzata, il modello si adegua perfettamente allo standard proposto da HPC Pack nel caso distribuito sul cloud: il cluster vero e proprio si compone di una macchina virtuale adibita al ruolo di **Head Node**, adoperato come Job Scheduler e Node Manager per l'orchestrazione di un numero arbitrario e flessibile di **Compute Node** (in particolare, Azure IaaS Virtual Machine). In questo caso, l'Head Node ospita anche i servizi centralizzati di **autenticazione** e di **business applicativi**. La centralità eccessiva dell'Head Node può condurre a spiacevoli peggioramenti nelle

prestazioni legate al fenomeno noto come *bottleneck*. La soluzione studiata per ERMAS risolve il problema impiegando due ulteriori macchine virtuali che si occuperanno di due aspetti trasversali ma comunque essenziali legati all'*High-Performance Computing*. La gestione dell'Active Directory viene demandata quindi a una Virtual Machine che funge da **Domain Controller**: la creazione di nuove utenze e l'autenticazione nel dominio di macchine virtuali aggiuntive è affidata a questa nuova risorsa. In merito ai database applicativi necessari ai nodi di calcolo, invece, la strategia stabilita è quella di dedicarvi una **SQL Virtual Machine** accessibile dalle utenze stabilite all'interno del dominio gestito nel *cluster*. In realtà, a questa istanza SQL Server è destinata anche la gestione dei database nativi di HPC. Infine, il modello propone la creazione di una **Presentation Virtual Machine** che ospita il front-end di ERMAS attraverso cui accedere ai database, creare *job* e visualizzarne i risultati. Come verrà mostrato in seguito, la configurazione così descritta viene garantita dall'esecuzione programmata di script preparati per il modello. Ovviamente, come descritto dall'ARM *template*, ognuna delle macchine virtuali dispone delle risorse correlate (dischi, interfacce di rete) e comunica con le altre attraverso una rete virtuale dedicata.

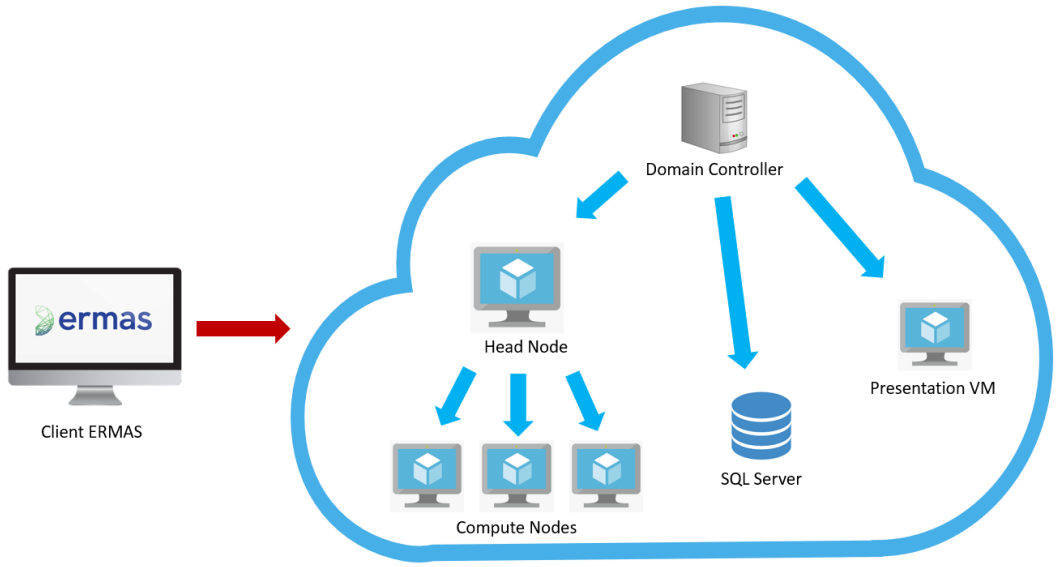

Fig. 21: Infrastrutura di calcolo distribuito di ERMAS

Il modello fornito è generalizzabile a qualsiasi scenario di calcolo distribuito che si intende creare, gestire e disfare in maniera del tutto automatizzata. La personalizzazione apportata per ERMAS non viene tanto dall'ARM *template* quanto dagli script Powershell preparati. La scelta dell'archiviazione locale dei file coinvolti, per motivi di comodità, vira sull'uso di un'unica cartella chiamata **PROM_HPC** (meno dispersiva per i casi di test e per l'upload dei file sui servizi di storage di Azure). L'organizzazione in *folder* permette una maggiore intuitività anche nell'uso futuro dei file archiviati. Nel dettaglio, le cartelle di cui il modello necessita sono:

- **Authentication Service**, che contiene i file richiesti dall'Head Node per l'installazione del servizio di autenticazione di ERMAS;
- **Business Services** ospita i file utili all'installazione del servizio di business di ERMAS, sempre destinato all'Head Node;
- **Database** fornisce i database applicativi utili al funzionamento della logica proposta da ERMAS, del servizio di autenticazione e di *logging*;
- **ErmasNet** contiene la logica del client di ERMAS, che verrà copiata dalla macchina virtuale di Presentation;
- **HPCFiles** è usata per contenere file accessori destinati all'Head Node. La scelta di una cartella separata è preferita per dividere le varie funzionalità offerte dalla stessa macchina virtuale: HPCFiles, infatti, conterrà file relativi a HPC Pack, come un Node Template per i nodi di calcolo da aggiungere al cluster e i certificati richiesti per la configurazione;
- **Script** si compone di alcuni file contenenti una sequenza di istruzioni Powershell in formato .ps1, realizzati appositamente per ciascuna delle macchine virtuali appartenenti all'infrastruttura;
- **SIMPro.net** racchiude il vero e proprio motore di calcolo e, in quanto tale, è destinato ai Compute Node che verranno creati in HPC su Azure.

Un'operazione iniziale resasi necessaria per l'uso di macchine virtuali su cloud consiste nell'aggiornamento di alcuni campi nei file di configurazione, all'interno delle cartelle Authentication Service, Business Service, ErmasNet e SIMPro.net: stabiliti i nomi e gli indirizzi IP privati dei vari attori all'interno dell'infrastruttura, infatti, bisogna configurare correttamente le stringhe relative alle informazioni di

Active Directory (nome del dominio, utenza amministratrice), gli indirizzi IP e le porte esposte dall'Head Node e le stringhe di connessione ai database.

Per ognuna di queste cartelle viene effettuato l'upload in uno Storage Account appositamente creato. Due strumenti ausiliari, utili soprattutto in fase di esecuzione degli *script*, sono il comando **AzCopy** e **PSExec**. Il primo, installabile con una sequenza di comandi oppure manualmente, definisce un file eseguibile con una sintassi ben definita attraverso il quale è possibile scaricare dei blob presenti sullo Storage Account specificato. Pertanto, ognuna delle macchine virtuali create lo installerà e userà per l'acquisizione delle cartelle richieste per il funzionamento. In particolare, i parametri di cui questo comando necessita sono /Source (sorgente dei dati), /Dest (destinazione della copia), /s (indica che una cartella deve essere copiata ricorsivamente). /SourceKey e /DestKey, invece, riguardano gli Storage Account usati e ne specificano le chiavi di accesso. PSExec, invece, consente l'esecuzione di *script* o, in generale, istruzioni acquisite dalla linea di comando usando un'utenza specificata interna alla macchina, previa definizione delle credenziali d'accesso. Questo strumento è utilizzato nella configurazione dell'istanza SQL Server.

Inoltre, la cartella PROM_HPC contiene due file essenziali per la distribuzione dell'intera struttura: si tratta dello script Powershell **StartingScript.ps1** e dell'ARM template **hpcAzure.json**. La presentazione dei dettagli implementativi partirà proprio da questi due file.

4.1.1 SCRIPT D'AVVIO

StartingScript.ps1 è lo script Powershell di avvio. Esso prepara l'ambiente per l'installazione di HPC e per la configurazione delle risorse. Dopo aver stabilito le credenziali Azure e i nomi di alcune risorse da distribuire sul cloud, lo script si occupa di installare AzCopy sulla macchina che esegue lo script, in quanto tornerà utile in seguito. Dopo aver effettuato la connessione a Azure con le credenziali specificate, viene creato un **gruppo di risorse** legato alla sottoscrizione collegata all'utente: esso, dunque, conterrà uno **Storage Account** e un **Key Vault**. Il primo ospiterà le cartelle sopra specificate, il cui upload è delegato al comando AzCopy. Segue la creazione di

due certificati: uno, utile all'installazione di HPC Pack sull'Head Node, viene inserito nel Key Vault, l'altro è usato per la creazione di un'applicazione di Azure Active Directory. Infine, eseguendo il comando **New-AZResourceGroupDeployment**, viene avviata la procedura di distribuzione delle risorse definite nell'ARM template hpcAzure.json.

```
install-module az -allowclobber -force
install-module azuread -allowclobber -force
import-module azuread
import-module az
#login e creazione Resource Group
$Credential = New-Object -TypeName 'System.Management.Automation.PSCredential' -ArgumentList
$User,$PWord;
Connect-AzAccount -Credential $Credential;
connect-azuread -Credential $Credential;
$rg = New-AzResourceGroup -Name $resGP -Location $location
#creazione Key Vault e certificato, poi caricamento del certificato sul Key Vault
$hpcKeyVault = New-AzKeyVault -Name $VaultName -ResourceGroupName $resGP -Location $location -
EnabledForDeployment -EnabledForTemplateDeployment
#creazione Storage Account e Container necessari
Enable-AzureRmAlias -Scope CurrentUser
New-AzStorageAccount -ResourceGroupName $resGP  -Name $storacc -Location $location -SkuName
Standard_RAGRS -Kind StorageV2
$key0 = (Get-AzStorageAccountKey -Name $storacc -ResourceGroupName $resGP )[0]
$Context = New-AzStorageContext -StorageAccountName $storacc -StorageAccountKey $Key0.value
New-AzStorageContainer -Permission Container -Name $container1 -Context $Context
New-AzStorageContainer -Permission Container -Name $container2 -Context $ContextSet-Location 'C:\Program
Files (x86)\Microsoft SDKs\Azure\AzCopy\'
.\AzCopy /Source:"$currentPath\script" /Dest:https://$($storAcc).blob.core.windows.net/script
/DestKey:$($key0.value) /s
.\AzCopy /Source:"$currentPath\authenticationservice"
/Dest:https://$($storAcc).blob.core.windows.net/cont/authenticationservice /DestKey:$($key0.value) /s
.\AzCopy /Source:"$currentPath\businessserviceshost"
/Dest:https://$($storAcc).blob.core.windows.net/cont/businessserviceshost /DestKey:$($key0.value) /s
.\AzCopy /Source:"$currentPath\ermasnet" /Dest:https://$($storAcc).blob.core.windows.net/cont/ermasnet
/DestKey:$($key0.value) /s
.\AzCopy /Source:"$currentPath\simpronet" /Dest:https://$($storAcc).blob.core.windows.net/cont/simpronet
/DestKey:$($key0.value) /s
.\AzCopy /Source:"$currentPath\dbs" /Dest:https://$($storAcc).blob.core.windows.net/cont/dbs
/DestKey:$($key0.value) /s
.\AzCopy /Source:"$currentPath\hpcfiles" /Dest:https://$($storAcc).blob.core.windows.net/cont/hpcfiles
/DestKey:$($key0.value) /s
##creazione certificati
```

```
$cert=New-SelfSignedCertificate -Subject "CN=HPC Pack 2016 Communication" -KeySpec KeyExchange -
TextExtension @("2.5.29.37={text}1.3.6.1.5.5.7.3.1,1.3.6.1.5.5.7.3.2") -CertStoreLocation cert:\CurrentUser\My -
KeyExportPolicy Exportable -HashAlgorithm SHA256 -NotAfter (Get-Date).AddYears(5) -NotBefore (Get-
Date).AddDays(-1)
Export-PfxCertificate -Cert "cert:\CurrentUser\My\$($cert.Thumbprint)" -FilePath C:\temp\cert2.pfx -Password
$secpass
$keyVaultCert = Import-AzKeyVaultCertificate -VaultName $VaultName -Name mycert -FilePath
"c:\temp\cert2.pfx" -Password $secpass
$cert = New-SelfSignedCertificate -CertStoreLocation "cert:\CurrentUser\My" `
  -Subject "CN=hpcapp" `
  -KeySpec KeyExchange
$keyValue = [System.Convert]::ToBase64String($cert.GetRawCertData())
export-pfxcertificate -FilePath $currentPath\hpcfiles\cert.pfx -Password $PWordcert -Cert $cert
#creazione AAD APP
$sp = New-AzADServicePrincipal -DisplayName hpcapp `
  -CertValue $keyValue `
  -EndDate $cert.NotAfter `
  -StartDate $cert.NotBefore
Sleep 50
New-AzRoleAssignment -RoleDefinitionName Contributor -ServicePrincipalName $sp.ApplicationId
$subscrId=(Get-AzContext).Subscription.id
$TenantId = (Get-AzSubscription -SubscriptionId $subscrId).TenantId
$ApplicationId = (Get-AzADApplication -DisplayNameStartWith hpcapp).ApplicationId
New-Azadappcredential -ApplicationId $ApplicationId.guid -CertValue $keyValue -enddate $cert.notafter
New-AzResourceGroupDeployment -ResourceGroupName $resGP -TemplateFile '$currentPath\hpcAzure.txt' -
startingStorageAccountName $storAcc -vaultName $VaultName -certificateUrl $keyVaultCert.SecretId -
certThumbprint $keyVaultCert.Thumbprint -computeNodeNumber 1 -applicationId $ApplicationId -tenantId
$tenantId
```

Fig. 22: Script d'avvio

Comunque, attraverso le API di Azure, la medesima procedura può essere tradotta in un programma scritto in C#. Una piccola nota va dedicata ai nomi delle varie risorse create: essi sono stati assegnati di default, ma è possibile apportare delle modifiche. Se si intende apportare dei cambiamenti in StartingScript.ps1, è necessario passare questi nuovi nomi come parametri del *cmdlet* New-AzResourceGroupDeployment per rifletterne le modifiche nel template hpcAzure.json. Inoltre, le nuove impostazioni vanno aggiornate manualmente negli altri script e, nel caso particolare dello scenario di ERMAS, anche nei file di configurazione dei servizi applicativi Authentication Service, Business Service, ErmasNet e SIMPro.net.

4.1.2. TEMPLATE AZURE RESOURCE MANAGER

Il fulcro dell'automatizzazione del modello implementato è la realizzazione di questo file. Esso contiene tutte le informazioni necessarie al *deployment* e, una volta messo in esecuzione da Azure, si occupa implicitamente dell'intero ciclo di vita delle risorse, a partire dalla loro creazione (con grado di parallelizzazione maggiore rispetto alle altre modalità di distribuzione) fino alla loro configurazione facendo uso di estensioni, DSC e script Powershell.

La realizzazione di questo progetto ha come punto di partenza il lavoro di **Sunbin Zhu**, ingegnere presso Microsoft, esplicato nel progetto GitHub https://github.com/Azure/hpcpack-template-2016. Esso offre diversi modelli di risorse in base alla configurazione desiderata: scenari ibridi o cloud, con domini nuovi o preconfigurati, e numerose opportunità di distribuzione dei database.

Come per ogni ARM *template*, la definizione di parametri e variabili precede quella delle risorse vere e proprie. I parametri sono raggruppati in base alla loro utilità: vi sono quelli globali, come ID dell'applicazione e del *tenant* dell'Azure AD (per la gestione dell'applicazione AAD), campi relativi allo Storage Account contenente i file da scaricare e nome del dominio che si intende creare; inoltre, bisogna specificare le informazioni circa il Key Vault e il certificato con cui si intende installare HPC Pack. I restanti parametri sono suddivisi a seconda delle risorse che li riguardano: si tratta perlopiù di nomi delle macchine virtuali, delle loro credenziali, dimensioni e informazioni sui dischi collegati. Si avranno, dunque, campi replicati per l'Head Node, per la Presentation Virtual Machine, per il Domain Controller e per l'istanza SQL Server. Inoltre, è fornito un prefisso dei nomi dei Compute Node, che in modo incrementale assegnerà un identificativo alle nuove istanze, e il numero di nodi di calcolo che si intende creare.

Anche le variabili del template sono organizzate allo stesso modo: a partire dai parametri, sono definite informazioni sullo Storage Account, sulla rete virtuale (gruppo di sicurezza, sottorete, informazioni sull'indirizzamento IP privato). Inoltre, per ciascuna macchina virtuale sono assegnati dei nomi per le risorse di interfaccia di rete e di indirizzi IP pubblici.

Le risorse vere e proprie costituiscono la parte principale del modello di risorse creato. Prima di elencare dettagliatamente i suoi componenti principali, è bene fare delle assunzioni. Tutte le macchine virtuali, che appartengano a HPC o meno, dispongono di un'interfaccia per comunicare all'interno della rete cui appartengono. A tal proposito, per motivi di configurazione, risulta utile assegnare indirizzi IP privati statici sia al Domain Controller, per definire in maniera univoca la macchina presso cui autenticarsi nel dominio, che all'Head Node, per evitare di dover modificare di volta in volta le stringhe nei file applicativi di configurazione dei servizi. Non tutte le istanze, tuttavia, dispongono di un indirizzo IP pubblico, in quanto non strettamente necessario tantomeno sicuro: è il caso dei Compute Node e della macchina che ospita SQL Server. Nella descrizione delle risorse, si è preferito inserirle (e quindi presentarle) in ordine cronologico di realizzazione. Così facendo, oltre a darne una lettura più comprensibile, si dà un'idea più chiara delle dipendenze tra risorse che Azure Resource Manager mette a disposizione.

- Il primo blocco di risorse che il *template* crea riguarda il *networking* in generale: alla realizzazione di una rete virtuale segue il gruppo di sicurezza a essa associato e gli indirizzi IP pubblici delle macchine virtuali che verranno create;

```
##Virtual Network, Network Security Group
      {
         "type": "Microsoft.Network/virtualNetworks",
         "apiVersion": "2017-10-01",
         "name": "[variables('virtualNetworkName')]",
         "location": "[resourceGroup().location]",
         "properties": {
            "addressSpace": {
               "addressPrefixes": [
                  "[variables('addressPrefix')]"
               ]
            },
            "subnets": [
               {
                  "name": "[variables('subnet1Name')]",
                  "properties": {
                     "addressPrefix": "[variables('subnet1Prefix')]"
                  }
               }
            ]
```

```
    }
},
{
    "type": "Microsoft.Network/networkSecurityGroups",
    "apiVersion": "2017-10-01",
    "name": "[variables('nsgName')]",
    "location": "[resourceGroup().location]",
    "properties": {
        "securityRules": [
            {
                "name": "allow-HTTPS",
                "properties": {
                    "description": "Allow Https",
                    "protocol": "Tcp",
                    "sourcePortRange": "*",
                    "destinationPortRange": "443",
                    "sourceAddressPrefix": "*",
                    "destinationAddressPrefix": "*",
                    "access": "Allow",
                    "priority": 1000,
                    "direction": "Inbound"
                }
            },
            {
                "name": "allow-RDP",
                "properties": {
                    "description": "Allow RDP",
                    "protocol": "Tcp",
                    "sourcePortRange": "*",
                    "destinationPortRange": "3389",
                    "sourceAddressPrefix": "*",
                    "destinationAddressPrefix": "*",
                    "access": "Allow",
                    "priority": 1010,
                    "direction": "Inbound"
                }
            },
            {
                "name": "allow-HPCSession",
                "properties": {
                    "description": "Allow HPC Session service",
                    "protocol": "Tcp",
                    "sourcePortRange": "*",
                    "destinationPortRange": "9090",
                    "sourceAddressPrefix": "*",
                    "destinationAddressPrefix": "*",
                    "access": "Allow",
                    "priority": 1020,
```

```
                "direction": "Inbound"
            }
        },
        {
            "name": "allow-HPCBroker",
            "properties": {
                "description": "Allow HPC Broker service",
                "protocol": "Tcp",
                "sourcePortRange": "*",
                "destinationPortRange": "9087",
                "sourceAddressPrefix": "*",
                "destinationAddressPrefix": "*",
                "access": "Allow",
                "priority": 1030,
                "direction": "Inbound"
            }
        },
        {
            "name": "allow-HPCBrokerWorker",
            "properties": {
                "description": "Allow HPC Broker worker",
                "protocol": "Tcp",
                "sourcePortRange": "*",
                "destinationPortRange": "9091",
                "sourceAddressPrefix": "*",
                "destinationAddressPrefix": "*",
                "access": "Allow",
                "priority": 1040,
                "direction": "Inbound"
            }
        },
        {
            "name": "allow-HPCDataService",
            "properties": {
                "description": "Allow HPC Data service",
                "protocol": "Tcp",
                "sourcePortRange": "*",
                "destinationPortRange": "9094 ",
                "sourceAddressPrefix": "*",
                "destinationAddressPrefix": "*",
                "access": "Allow",
                "priority": 1050,
                "direction": "Inbound"
            }
        }
    ]
  }
}
```

```
##esempio di Public IP
      {
         "type": "Microsoft.Network/publicIPAddresses",
         "apiVersion": "2018-08-01",
         "name": "[variables('dc_ipName')]",
         "location": "[resourceGroup().location]",
         "sku": {
            "name": "Basic"
         },
         "properties": {
            "publicIPAllocationMethod": "Dynamic"
         }
      }
```

Fig. 23: Definizione risorse di rete

- In seguito, avviene la creazione delle interfacce di rete, una per ognuna delle Virtual Machine. Vi è, tuttavia, una particolare relazione tra queste risorse: dal momento che, come già evidenziato, alcune risorse necessitano di un indirizzo IP privato statico, la creazione di una Network Interface per le istanze di Domain Controller e di Head Node precede tutte le altre, in modo da evitare conflitti legati a indirizzi già occupati;

```
##esempio di Network Interface
      {
         "type": "Microsoft.Network/networkInterfaces",
         "apiVersion": "2017-10-01",
         "name": "[variables('hn_nicName')]",
         "location": "[resourceGroup().location]",
         "dependsOn": [
            "[concat('Microsoft.Network/virtualNetworks/', variables('virtualNetworkName'))]",
            "[concat('Microsoft.Network/networkSecurityGroups/', variables('nsgName'))]",
            "[concat('Microsoft.Network/publicIPAddresses/', variables('hn_ipName'))]"
         ],
         "properties": {
            "ipConfigurations": [
               {
                  "name": "IPConfig",
                  "properties": {
                     "privateIPAllocationMethod": "Static",
                     "privateIPAddress": "10.0.0.5",
                     "publicIPAddress": "[variables('hn_ipId')]",
                     "subnet": {
```

```
                        "id": "[variables('subnetRef')]"
                    }
                }
            }
        ],
        "dnsSettings": {
            "dnsServers": [
                "10.0.0.4"
            ]
        },
        "networkSecurityGroup": "[variables('networkSecurityGroupId')]",
        "enableAcceleratedNetworking": "[equals(parameters('enableAcceleratedNetworking'), 'Yes')]"
    }
}
```

Fig. 24: Definizione interfaccia di rete

- Dopo queste operazioni preliminari, utili alla configurazione della comunicazione nel cluster, avviene la creazione delle risorse vere e proprie, ossia le macchine virtuali. In particolare, è presente una risorsa per ciascuna istanza a eccezione di SQL Server. La sua realizzazione, infatti, viene gestita attraverso una risorsa di tipo deployment innestata: ciò è attuato in modo da attivare l'estensione DSC che si occupa della configurazione dei database di HPC;

```
##esempio di Virtual Machine
    {
        "type": "Microsoft.Compute/virtualMachines",
        "apiVersion": "2017-12-01",
        "name": "[parameters('hn_VMName')]",
        "location": "[resourceGroup().location]",
        "dependsOn": [
            "[concat('Microsoft.Network/networkInterfaces/', variables('hn_nicName'))]",
            "[concat('Microsoft.Compute/availabilitySets/', variables('availabilitySetName'))]"
        ],
        "properties": {
            "availabilitySet": "[if(variables('createHNInAVSet'), variables('availabilitySet'), json('null'))]",
            "hardwareProfile": {
                "vmSize": "[parameters('hn_VMSize')]"
            },
            "osProfile": {
                "computerName": "[parameters('hn_VMName')]",
                "adminUsername": "[parameters('hn_adminUsername')]",
                "adminPassword": "[parameters('adminPassword')]",
```

```
                "windowsConfiguration": {
                    "enableAutomaticUpdates": false
                },
                "secrets": "[variables('certSecrets')]"
            },
            "storageProfile": {
                "imageReference": {
                    "publisher": "MicrosoftWindowsServerHPCPack",
                    "offer": "WindowsServerHPCPack",
                    "sku": "2016u3hn-ws2016",
                    "version": "latest"
                },
                "osDisk": {
                    "name": "[concat(parameters('hn_VMName'), '-osdisk')]",
                    "caching": "ReadOnly",
                    "createOption": "FromImage",
                    "managedDisk": {
                        "storageAccountType": "Standard_LRS"
                    }
                },
                "dataDisks": "[if(equals(parameters('hn_DataDiskCount'), 0), variables('emptyArray'),
variables('hn_DataDisks').hn_DataDisks)]"
            },
            "networkProfile": {
                "networkInterfaces": [
                    {
                        "id": "[resourceId('Microsoft.Network/networkInterfaces', variables('hn_nicName'))]"
                    }
                ]
            }
        }
    }

##creazione SQL VM e database HPC
    {
        "type": "Microsoft.Resources/deployments",
        "apiVersion": "2017-05-10",
        "name": "[concat('createDBServer', parameters('sql_VMName'))]",
        "dependsOn": [
            "[resourceId('Microsoft.Network/networkInterfaces/', variables('dc_nicName'))]",
            "[resourceId('Microsoft.Network/networkInterfaces/', variables('hn_nicName'))]",
            "[resourceId('Microsoft.Network/networkInterfaces/', variables('pres_nicName'))]"
        ],
        "properties": {
            "mode": "Incremental",
            "templateLink": {
                "uri": "[concat(variables('sharedResxBaseUrl'), '/windowsvm-dsc.json')]",
                "contentVersion": "1.0.0.0"
```

```
},
"parameters": {
    "subnetId": {
        "value": "[variables('subnetRef')]"
    },
    "vmName": {
        "value": "[parameters('sql_VMName')]"
    },
    "vmSize": {
        "value": "[parameters('sql_VMSize')]"
    },
    "storageAccountType": {
        "value": "Standard_LRS"
    },
    "imageReference": {
        "value": {
            "publisher": "MicrosoftSQLServer",
            "offer": "SQL2016SP2-WS2016",
            "sku": "Standard",
            "version": "latest"
        }
    },
    "adminUsername": {
        "value": "[parameters('sql_adminUsername')]"
    },
    "adminPassword": {
        "value": "[parameters('adminPassword')]"
    },
    "dataDiskSizeInGB": {
        "value": 200
    },
    "enableAcceleratedNetworking": {
        "value": "[equals(parameters('enableAcceleratedNetworking'), 'Yes')]"
    },
    "dnsServers": {
        "value": [
            "10.0.0.4",
            "8.8.8.8",
        ]
    },
    "dscExtensionName": {
        "value": "configSQLServer"
    },
    "dscSettings": {
        "value": {
            "configuration": {
                "url": "[concat(variables('sharedResxBaseUrl'),'/ConfigSQLServer.ps1.zip')]",
                "script": "ConfigSQLServer.ps1",
```

```
                    "function": "ConfigSQLServer"
                }
            }
        }
    }
}
```

Fig. 25: Definizione macchina virtuale e SQL Server

- Dopo aver realizzato le macchine virtuali è possibile lavorare su di esse, mediante una serie di estensioni appositamente preparate. La prima a essere attivata è **setupHpcHeadNode**, applicata al Domain Controller. Si tratta dell'operazione più lunga del *deployment*, grazie alla quale viene creata un'Active Directory che sfrutta le credenziali dell'utenza amministratrice e il nome di dominio specificato nell'input da cui attinge; con l'estensione **copyapp** uno script chiamato **copiaDC** si occupa, successivamente, della creazione di nuovi utenti di dominio relativi alle macchine virtuali che vi si uniranno in seguito, permettendone così l'autenticazione;

```
{
        "type": "Microsoft.Compute/virtualMachines/extensions",
        "apiVersion": "2017-12-01",
        "name": "[concat(parameters('dc_VMName'),'/setupHpcHeadNode')]",
        "location": "[resourceGroup().location]",
        "dependsOn": [
            "[concat('Microsoft.Compute/virtualMachines/', parameters('dc_VMName'))]"
        ],
        "properties": {
            "publisher": "Microsoft.Powershell",
            "type": "DSC",
            "typeHandlerVersion": "2.20",
            "autoUpgradeMinorVersion": true,
            "settings": {
                "configuration": {
                    "url": "[concat(variables('sharedResxBaseUrl'),'/CreateADPDC.ps1.zip')]",
                    "script": "CreateADPDC.ps1",
                    "function": "CreateADPDC"
                },
                "configurationArguments": {
                    "DomainName": "[parameters('domainName')]"
                }
```

```
            },
            "protectedSettings": {
                "configurationArguments": {
                    "AdminCreds": {
                        "UserName": "[parameters('dc_adminUsername')]",
                        "Password": "[parameters('adminPassword')]"
                    }
                }
            }
        }
    }
{
        "type": "Microsoft.Compute/virtualMachines/extensions",
        "apiVersion": "2018-06-01",
        "name": "[concat(parameters('dc_VMName'),'/', 'copyapp')]",
        "location": "[resourceGroup().location]",
        "dependsOn": [
            "[concat('Microsoft.Compute/virtualMachines/', parameters('dc_VMName'),
'/extensions/setupHpcHeadNode')]"
        ],
        "properties": {
            "publisher": "Microsoft.Compute",
            "type": "CustomScriptExtension",
            "typeHandlerVersion": "1.7",
            "autoUpgradeMinorVersion": true,
            "settings": {
                "fileUris": [
                    "[concat(variables('scriptStorageUrl'),'copiaDC.ps1')]"
                ],
                "commandToExecute": "powershell.exe -ExecutionPolicy Unrestricted -File copiaDC.ps1",
                "storageAccountName": "parameters('startingStorageAccountName')",
                "storageAccountKey": "listKeys(variables('startingStorageAccountId'),'2019-04-
01').keys[0].value"
            }
        }
    }
```

Fig. 26: Estensione Domain Controller

- Per il corretto accesso delle macchine virtuali all'Active Directory è richiesto che si configurino i server DNS della rete virtuale, come eseguito dal *deployment* chiamato **UpdateVnetDNS**;
- Il passo successivo riguarda la configurazione della macchina virtuale SQL Server: essa, infatti, non è destinata a ospitare i soli database HPC, ma anche quelli applicativi di cui ERMAS necessita. A tal fine, pertanto, è specificata

l'estensione chiamata **copydb**, attraverso la quale viene eseguito lo script chiamato **copiaDB** e, indirettamente, anche lo script **work**. L'esecuzione degli script avviene nelle modalità presentate nel caso di copyapp del Domain Controller. Entrambi verranno presentati più avanti;

- Perciò avviene l'installazione dell'Head Node di HPC, ossia l'operazione più importante di questo template. La configurazione consiste nell'esecuzione di un'estensione chiamata **joindomain** (che esegue un comando di tipo **JsonADDomainExtension**) a partire dalle credenziali fornite e dal nome del dominio: grazie a essa la macchina virtuale viene autenticata presso l'Active Directory di riferimento. L'installazione di HPC è prerogativa del *deployment* chiamato **installHpcHeadNode** mediante estensione incrementale DSC. L'input, in questo caso, riguarda una serie di informazioni circa i certificati precedentemente realizzati, la rete in cui il *cluster* dovrà risiedere e altri dati relativi alle risorse Azure, oltre a indicazioni rilevanti riguardanti la dimensione e il prefisso di nomi dei Compute Node. Se si intende usare un'istanza SQL separata per i database di HPC, infine, è necessario specificare in input il nome della macchina virtuale SQL nel campo *SQLServerInstance*. In ultimo, l'estensione **copyapp** specifica una sequenza di istruzioni modellate per l'installazione dei servizi applicativi sull'Head Node e degli ultimi accorgimenti sul servizio HPC;

```
      {
        "type": "Microsoft.Compute/virtualMachines/extensions",
        "apiVersion": "2015-06-15",
        "name": "[concat(parameters('hn_VMName'),'/joindomain')]",
        "location": "[parameters('location')]",
        "dependsOn": [
          "[concat('Microsoft.Compute/virtualMachines/', parameters('hn_VMName'),
'/extensions/installInfiniBandDriver')]",
          "[concat('Microsoft.Compute/virtualMachines/', parameters('hn_VMName'))]",

"[concat('Microsoft.Compute/virtualMachines/',parameters('dc_VMName'),'/extensions/copyapp')]"
        ],
        "properties": {
          "publisher": "Microsoft.Compute",
          "type": "JsonADDomainExtension",
          "typeHandlerVersion": "1.3",
```

```
            "autoUpgradeMinorVersion": true,
            "settings": {
                "Name": "[parameters('domainName')]",
                "User": "[concat(parameters('domainName'), '\\', parameters('hn_adminUsername'))]",
                "Restart": "true",
                "Options": "[variables('domainJoinOptions')]",
                "OUPath": ""
            },
            "protectedSettings": {
                "Password": "[parameters('adminPassword')]"
            }
        }
    }

    {
        "type": "Microsoft.Resources/deployments",
        "apiVersion": "2018-05-01",
        "name": "installHpcHeadNode",
        "dependsOn": [
            "[concat('Microsoft.Compute/virtualMachines/', parameters('hn_VMName'),
'/extensions/installInfiniBandDriver')]",
            "[concat('Microsoft.Compute/virtualMachines/', parameters('hn_VMName'))]",
            "[concat('Microsoft.Compute/virtualMachines/', parameters('hn_VMName'),
'/extensions/joindomain')]",
            "[concat('Microsoft.Compute/virtualMachines/', parameters('sql_VMName'),
'/extensions/copydb')]"
        ],
        "properties": {
            "mode": "Incremental",
            "template": {
                "$schema": "https://schema.management.azure.com/schemas/2015-01-
01/deploymentTemplate.json#",
                "contentVersion": "1.0.0.0",
                "resources": [
                    {
                        "type": "Microsoft.Compute/virtualMachines/extensions",
                        "name": "[concat(parameters('hn_VMName'),'/setupHpcHeadNode')]",
                        "apiVersion": "2017-12-01",
                        "location": "[resourceGroup().location]",
                        "properties": {
                            "publisher": "Microsoft.Powershell",
                            "type": "DSC",
                            "typeHandlerVersion": "2.20",
                            "autoUpgradeMinorVersion": true,
                            "settings": {
                                "configuration": {
                                    "url":
"[concat(variables('sharedResxBaseUrl'),'/InstallHpcSingleHeadNode.ps1.zip')]",
```

```
                        "script": "InstallHpcSingleHeadNode.ps1",
                        "function": "InstallHpcSingleHeadNode"
                    },
                    "configurationArguments": {
                        "SSLThumbprint": "[parameters('certThumbprint')]",
                        "SQLServerInstance": "[parameters('sql_VMName')]",
                        "CNSize": "[parameters('computeNodeVMSize')]",
                        "SubscriptionId": "[subscription().subscriptionId]",
                        "VNet": "[variables('virtualNetworkName')]",
                        "Subnet": "[variables('subnet1Name')]",
                        "Location": "[resourceGroup().location]",
                        "ResourceGroup": "[resourceGroup().name]",
                        "VaultResourceGroup": "[parameters('resourceGroupName')]",
                        "CertificateUrl": "[parameters('CertificateUrl')]",
                        "CNNamePrefix": "[parameters('computeNodeNamePrefix')]"
                    }
                },
                "protectedSettings": {
                    "configurationArguments": {
                        "SetupUserCredential": {
                            "UserName": "[concat(parameters('domainName'), '\\',
parameters('hn_adminUsername'))]",
                            "Password": "[parameters('adminPassword')]"
                        },
                        "AzureStorageConnString":
"[concat('DefaultEndpointsProtocol=https;AccountName=', parameters('startingStorageAccountName'),
';AccountKey=', listKeys(variables('startingStorageAccountId'), '2019-04-01').keys[0].value)]"
                    }
                }
            }
        }
    ]
  }
 }
}
```

Fig. 27: Installazione HPC sull'Head Node

- Nel momento in cui l'intero ambiente di esecuzione è stato configurato, il template passa alla preparazione della Virtual Machine Presentation che ospita il client, con l'estensione **copyapp** che esegue lo script **copiaErmasNet**;
- Contemporaneamente avviene la preparazione dei nodi di calcolo iniziali richiesti. Tra i parametri di input del *template* è specificato il numero di Compute Node da creare: questo passo, pertanto, non viene eseguito se il

numero indicato è 0. Ovviamente è possibile realizzare nuovi nodi anche in seguito, quindi tale campo può assumere qualsiasi valore. Per l'esecuzione di questo step vengono applicati *deployment* incrementali in numero pari a quello dei nodi richiesti (richiamando la funzione *copy*), in cui sono fornite le informazioni basilari di *networking* e di macchina virtuale; ma soprattutto i dati necessari per l'autenticazione nell'Active Directory e per l'installazione di HPC. Infatti, all'interno di questo *deployment*, viene applicata un'estensione di tipo **HpcPack** chiamata **HpcComputeNode**. Al termine della configurazione dell'Head Node, esso esegue lo script **copiaSIMPro.net** in cui scarica l'*engine* di calcolo progettato da Prometeia;

```
    {
        "type": "Microsoft.Resources/deployments",
        "apiVersion": "2017-05-10",
        "name": "[concat('create', parameters('computeNodeNamePrefix'), padLeft(string(copyIndex()),
3, '0'))]",
        "dependsOn": [
            "[concat('Microsoft.Compute/availabilitySets/', variables('availabilitySetName'))]",
            "Microsoft.Resources/deployments/updateVNetDNS",
            "installHpcHeadNode"
        ],
        "properties": {
            "mode": "Incremental",
            "templateLink": {
                "uri": "[concat(variables('sharedResxBaseUrl'), '/computenode.json')]",
                "contentVersion": "1.0.0.0"
            },
            "parameters": {
                "subnetId": {
                    "value": "[variables('subnetRef')]"
                },
                "vmName": {
                    "value": "[concat(parameters('computeNodeNamePrefix'), padLeft(string(copyIndex()), 3,
'0'))]"
                },
                "vmSize": {
                    "value": "[parameters('computeNodeVMSize')]"
                },
                "osDiskType": {
                    "value": "Standard_LRS"
                },
                "dataDiskSizeInGB": {
```

```
        "value": "[parameters('computeNodeDataDiskSize')]"
    },
    "dataDiskCount": {
        "value": "[parameters('computeNodeDataDiskCount')]"
    },
    "dataDiskType": {
        "value": "Standard_LRS"
    },
    "imageReference": {
        "value": {
            "publisher": "MicrosoftWindowsServer",
            "offer": "WindowsServer",
            "sku": "2016-Datacenter",
            "version": "latest"
        }
    },
    "imageOsPlatform": {
        "value": "windows"
    },
    "adminUsername": {
        "value": "[parameters('hn_adminUsername')]"
    },
    "adminPassword": {
        "value": "[parameters('adminPassword')]"
    },
    "availabilitySetName": {
        "value": "[if(variables('createCNInAVSet'), variables('availabilitySetName'), '')]"
    },
    "installRDMADriver": {
        "value": "[and(variables('createCNInAVSet'), variables('cnRDMACapable'))]"
    },
    "enableAcceleratedNetworking": {
        "value": "[equals(parameters('enableAcceleratedNetworking'), 'Yes')]"
    },
    "secrets": {
        "value": "[variables('certSecrets')]"
    },
    "certThumbprint": {
        "value": "[trim(parameters('certThumbprint'))]"
    },
    "headNodeList": {
        "value": "[parameters('hn_VMName')]"
    },
    "joinDomain": {
        "value": true
    },
    "domainName": {
        "value": "[parameters('domainName')]"
```

```
            },
            "dnsServers": {
                "value": [
                    "10.0.0.4",
                    "8.8.8.8"
                ]
            }
        }
    },
    "copy": {
        "name": "CN",
        "count": "[parameters('computeNodeNumber')]"
    }
}
```

Fig. 28: Configurazione Compute Node

- All'interno del *template* sono indicate anche due risorse aggiuntive: un *Availability Set*, che garantisce maggiore tolleranza ai guasti per le proprie macchine virtuali, e un'estensione che ne fa uso chiamata *InstallInfinibandDriver*. Entrambe, di default, sono disattivate in questo *template*, ma sono lasciate per dare facoltà di scelta all'utente.

Il diagramma di flusso seguente presenta visivamente le dipendenze nella creazione di risorse applicate nello sviluppo dell'infrastruttura presentata dal template *hpcAzure.json*.

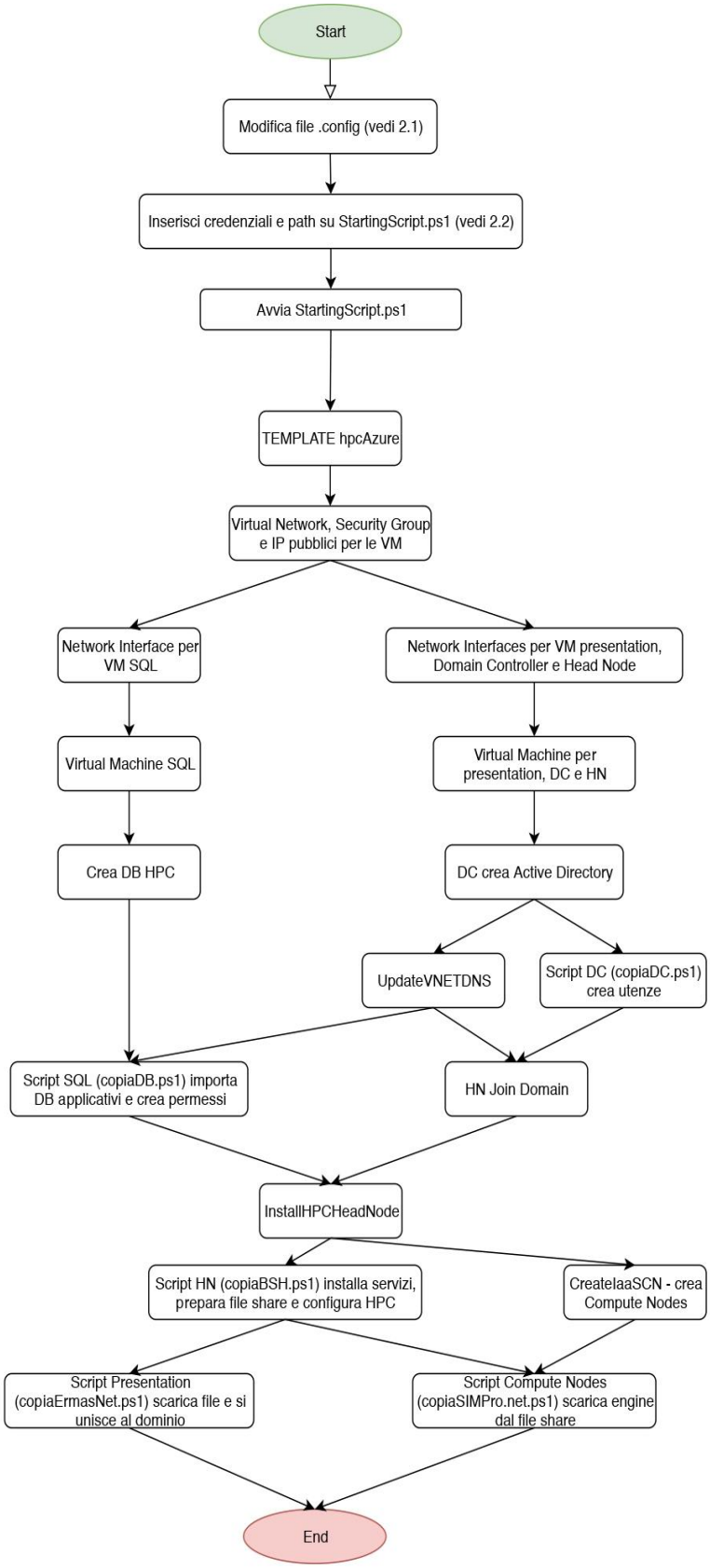

Fig. 29: Flusso di esecuzione di hpcAzure.json

4.1.3 SCRIPT DI CONFIGURAZIONE

Il *template hpcAzure.json* da solo fornisce un modello totalmente configurabile per qualsiasi contesto applicativo che richieda l'uso di calcolo parallelo. Esso, infatti, è stato realizzato secondo le linee guida fornite da Microsoft HPC Pack. Ciò che personalizza il progetto secondo le direttive di ERMAS è attuato mediante le estensioni indicate, che configurano l'ambiente attraverso l'uso di script Powershell. In linea generale, essi si occupano del download e della preparazione di una serie di file applicativi, impostando l'ambiente per la comunicazione con i servizi forniti dagli altri attori.

In ordine cronologico, **copiaDC** è lo script più semplice, viene eseguito dal Domain Controller e provvede esclusivamente alla creazione delle utenze e dei gruppi di dominio necessari.

```
Import-Module ActiveDirectory
New-ADUser hpcadmin -Server dcadmin -AccountPassword (ConvertTo-SecureString "PASSWORD" -AsPlainText
-Force) -ChangePasswordAtLogon 0 -PasswordNeverExpires 0 -Enabled 1
New-ADUser presadmin -Server dcadmin -AccountPassword (ConvertTo-SecureString "PASSWORD" -AsPlainText
-Force) -ChangePasswordAtLogon 0 -PasswordNeverExpires 0 -Enabled 1
New-ADUser sqladmin -Server dcadmin -AccountPassword (ConvertTo-SecureString "PASSWORD" -AsPlainText -
Force) -ChangePasswordAtLogon 0 -PasswordNeverExpires 0 -Enabled 1
Add-AdGroupMember -Identity "Domain Admins" -Members "hpcadmin"
Add-AdGroupMember -Identity "Domain Admins" -Members "sqladmin"
```

Fig. 30: Script copiaDC

copiadb è eseguito sulla macchina virtuale SQL e, al contrario, ha una struttura più articolata: lo strumento AzCopy viene installato e utilizzato per il download dei database applicativi dallo Storage Account creato in fase preliminare dallo script StartingScript; segue l'avvio di un secondo script, chiamato **work**, che usa le credenziali dell'amministratore dell'istanza SQL Server.

```
Set-Location 'C:\Program Files (x86)\Microsoft SDKs\Azure\AzCopy\'

.\AzCopy /Source:$storAccUrl /Dest:c:\prometeia\ermas /SourceKey:$storAccKey /s
```

```
#Esegue work.ps1 con utenza sql
$secpasswd = ConvertTo-SecureString 'PASSWORD' -AsPlainText -Force
set-location C:\prometeia\ermas\tools
.\psexec -accepteula -h -u sqladmin\sqladmin -p PASSWORD powershell.exe
"c:\prometeia\ermas\work\work.ps1"
```

Fig. 31: Script copiaDB

Questo nuovo script si rende necessario per usare i permessi richiesti per il *deployment* dei database mediante l'uso del file eseguibile **SqlPackage**: esso necessita in input, dell'operazione da svolgere (in questo caso Import), del file .bacpac da ottenere e di una stringa di connessione relativa all'istanza SQL Server cui il nuovo database è destinato. Per accedere ai database, le altre macchine del cluster necessitano di un'utenza con i corretti permessi: questo viene eseguito tramite *query* SQL che creano e configurano un'identità per i vari nodi della rete. In particolare, il comando Powershell che ne consente l'esecuzione è **Invoke-Sqlcmd**. Inoltre, la macchina virtuale appena creata viene aggiunta all'Active Directory gestita dal Domain Controller.

```
install-PackageProvider -Name nuget -Force
install-module -Force -Name sqlserver -AllowClobber
##Deploy DB
Set-Location "C:\Program Files (x86)\Microsoft SQL Server\140\DAC\bin"
.\sqlpackage /a:Import /sf:c:\prometeia\ermas\db.bacpac /tcs:"Server=sqladmin; Database=AUTH_DB; Integrated
Security=SSPI;"
##Join AD
$secpasswd = ConvertTo-SecureString 'PASSWORD' -AsPlainText -Force
$mycreds = New-Object System.Management.Automation.PSCredential ('hpc\sqladmin', $secpasswd)
Add-Computer -DomainName hpc.local -Credential $mycreds
#gestisce permessi
invoke-sqlcmd -Serverinstance sqladmin -query "create login [hpc\dcadmin] from windows; create user hpcadmin
from login  [hpc\dcadmin]; alter server role sysadmin add member [hpc\dcadmin]"
invoke-sqlcmd -Serverinstance sqladmin -query "create login [hpc\sqladmin] from windows; create user
hpcadmin from login  [hpc\sqladmin]; alter server role sysadmin add member [hpc\sqladmin]"
invoke-sqlcmd -Serverinstance sqladmin -query "grant create procedure to [hpc\sqladmin]"
invoke-sqlcmd -Serverinstance sqladmin -query "create login [hpc\hpcadmin] from windows; create user
hpcadmin from login  [hpc\hpcadmin]; alter server role sysadmin add member [hpc\hpcadmin]"
```

Fig. 32: Script work

In tempi più ridotti vengono applicate le istruzioni contenute nello script **copiaBSH** all'interno dell'Head Node. All'installazione di AzCopy segue il download dei servizi di autenticazione e business, oltre che dell'*engine* di calcolo destinato ai Compute Node. Questo comando attinge ai file necessari usando il solito Storage Account creato da StartingScript. Dopo che i servizi sono stati correttamente installati (tramite **InstallUtil**) lo script si occupa di creare una cartella condivisa con i nodi di calcolo facendo uso del *cmdlet* **New-SMBShare**: specificandovi nome della condivisione, percorso della cartella e permessi di lettura, i Compute Node potranno accedere e scaricare nei dischi a essi dedicati il motore di calcolo progettato per ERMAS.

```
.\AzCopy /Source:$storAccUrl1 /Dest:c:\ErmasNETServices\BusinessServices_Trunk /SourceKey:$storAccKey /s
.\AzCopy /Source:$storAccUrl2 /Dest:c:\prometeia\ermas\authenticationservice /SourceKey:$storAccKey /s
.\AzCopy /Source:$storAccUrl3 /Dest:c:\prometeia\ermas\simpronet /SourceKey:$storAccKey /s
.\AzCopy /Source:$storaccUrl4 /Dest:c:\hpcfiles /SourceKey:$storAccKey /s
##Installa servizi
Set-Location c:\ErmasNETServices\BusinessServices_Trunk
.\Install.bat
c:\Windows\Microsoft.NET\Framework64\v4.0.30319\InstallUtil.exe
c:\prometeia\ermas\authenticationservice\Prometeia.ALMPro.AuthenticationServiceHost.exe
##crea file share
New-SMBShare –Name "simpronet" –Path "c:\prometeia\ermas\simpronet" -ReadAccess Everyone
##importa certificato  e configura hpc per l'  aggiunta di nuovi nodi di calcolo
Install-PackageProvider NuGet -Force;
Set-PSRepository PSGallery -InstallationPolicy Trusted
Install-Module az -Repository PSGallery
import-module az
$PWord = ConvertTo-SecureString -String 'PASSWORD' -AsPlainText -Force;
Import-PfxCertificate -FilePath C:\hpcfiles\cert.pfx -Password $PWord -CertStoreLocation Cert:\LocalMachine\my
$Thumbprint = (Get-ChildItem cert:\LocalMachine\My\ | Where-Object {$_.Subject -eq "CN=hpcapp"
}).Thumbprint

Connect-AzAccount -ServicePrincipal `
  -CertificateThumbprint $Thumbprint `
  -ApplicationId $ApplicationId.Tostring() `
  -TenantId $TenantId
Add-PSSnapin Microsoft.Hpc
Set-HpcClusterRegistry -PropertyName SubscriptionId -PropertyValue $subscrID
Set-HpcClusterRegistry -PropertyName TenantId -PropertyValue $tenantID
Set-HpcClusterRegistry -PropertyName ApplicationId -PropertyValue $applicationID.tostring()
Set-HpcClusterRegistry -PropertyName Thumbprint -PropertyValue $Thumbprint
# Set Virtual network information
Set-HpcClusterRegistry -PropertyName VNet -PropertyValue vnet
Set-HpcClusterRegistry -PropertyName Subnet -PropertyValue Subnet-1
Set-HpcClusterRegistry -PropertyName Location -PropertyValue westeurope
```

```
Set-HpcClusterRegistry -PropertyName ResourceGroup -PropertyValue mich
# Set Azure Key vault certificate
Set-HpcKeyVaultCertificate -ResourceGroup mich -CertificateUrl $secreturl -CertificateThumbprint $secretthumb
##importa node template
Set-Location 'C:\Program Files\Microsoft HPC Pack 2016\Bin'
Add-PsSnapin Microsoft.HPC
import-hpcnodetemplate c:\hpcfiles\nodetmp.xml
```

Fig. 33: Script copiaBSH

L'ultima operazione eseguita è richiesta dal servizio di autenticazione di ERMAS e consiste nella creazione di un'identità (quella della macchina virtuale che ospita il client) all'interno del database di autenticazione.

La fase conclusiva della configurazione riguarda due script che, almeno a livello concettuale, possono essere applicati in parallelo: **copiaErmasNet** sulla Presentation Virtual Machine, **copiaSIMPro.net** sui nodi di calcolo realizzati nel *template* (se ve ne sono). Il primo script, oltre alle solite operazioni di installazione di AzCopy e download del client, si occupa dell'inserimento della macchina al dominio. Il secondo non necessita di questa operazione, in quanto viene aggiunto al dominio già in fase di creazione, e lo script si limita alla copia dell'*engine* di calcolo dalla cartella condivisa dall'Head Node.

```
##comandi per Presentation
##Copia file
Set-Location 'C:\Program Files (x86)\Microsoft SDKs\Azure\AzCopy\'
.\AzCopy /Source:$storAccUrl /Dest:c:\prometeia\ermas\ermasnet /SourceKey:$storAccKey /s
##Join AD
$secpasswd = ConvertTo-SecureString 'PASSWORD' -AsPlainText -Force
$mycreds = New-Object System.Management.Automation.PSCredential ('hpc\presadmin', $secpasswd)
Add-Computer -DomainName hpc.local -Credential $mycreds
```

Fig. 34: Script copiaErmasNet

```
##comando per compute nodes
copy \\hpcadmin\simpronet c:\prometeia\ermas\ -Recurse
```

Fig. 35: Script copiaSIMPro.net

4.2 PROGRAMMA DI GESTIONE DI HPC

Dopo che l'intera infrastruttura è stata correttamente implementata sul cloud e configurata, è pronta all'uso: in particolare, se si desidera modificare il *cluster* aggiungendo o rimuovendo un certo numero di Compute Node, esistono diverse vie più o meno complicate: per esempio sarebbe possibile usare Remote Desktop Protocol per accedere all'Head Node (tramite il suo indirizzo IP pubblico) e usare l'interfaccia di HPC Pack o i *cmdlet* che forniscono comunicazione diretta con il *cluster*. In alternativa, si può usare Azure (Powershell, Azure CLI, ecc.) per distribuire nuove risorse dello stesso tipo dei nodi costruiti in fase di avvio. Comunque, strategie di questo tipo richiedono un intervento manuale estremamente macchinoso: per evitare scenari del genere, l'idea da implementare nel client di ERMAS consente di comunicare direttamente con le API fornite da HPC usando il .NET Framework. Il codice di seguito presentato intende descrivere la soluzione applicata: il suo obiettivo principale è consentire, tramite interfaccia estremamente intuitiva realizzata con WPF (Windows Presentation Foundation), la totale manipolazione del *cluster*. La comunicazione con il Node Manager di HPC permette di organizzare le proprie risorse di calcolo in base all'uso richiesto, incrementandone la potenza in scenari di forte stress computazionale e riducendola quando non richiesta, tramite l'uso di un pannello controllato direttamente dall'utente. Il vantaggio principale di questo strumento riguarda la totale automazione delle operazioni di basso livello, consentendo di fare apparire all'utente come uno scenario PaaS quella che, a tutti gli effetti, è una soluzione IaaS: il cliente, infatti, non ha alcuna visibilità a livello di configurazione delle risorse gestite e l'unica attività di cui deve occuparsi è quella di manipolare il *cluster* in base all'uso.

Le funzionalità offerte si fondano sulla creazione, l'avvio, l'arresto e la cancellazione di nodi di calcolo nel *cluster*: in questo modo sono realizzate delle funzioni più complesse che offrono tre livelli di potenza, ovviamente dopo la corretta connessione al Node Manager (LOW, MEDIUM, HIGH). Ciò è attuabile comunicando con un'istanza di una classe appositamente creata, chiamata **AzureConfigurationManager**, che dispone di alcuni metodi:

- **SetState** stabilisce la configurazione in cui il *cluster* deve essere portato;

- **SetConf** è usato per definire i vari livelli di potenza di cui l'utente può aver bisogno;
- **SetDel** è necessario per decidere se i nodi di calcolo non più necessari devono essere solo arrestati o anche eliminati dal *cluster*.

Inoltre, sono forniti servizi aggiuntivi: un box dell'interfaccia è dedicato all'output delle varie funzioni, permettendo di tracciare uno storico delle operazioni effettuate nell'intero ciclo di vita del cluster tramite *polling*: si tratta del metodo **UpdateLists**.

Prima di mostrare nel dettaglio come queste funzioni sono applicate, è necessaria una rapida panoramica delle API utilizzate: in particolare, quelle rilevanti riguardano i *namespace* **Microsoft.Hpc.Scheduler** e Microsoft.ComputeCluster.Management.

Per il primo l'utilizzo è limitato all'interfaccia **IScheduler**: essa offre numerose funzioni che consentono la manipolazione di *job* e *task*, ma è anche possibile ottenere in modo estremamente accurato lo stato di attività dei nodi e dei *core* che li compongono. La connessione, dunque, avviene tramite il metodo **ConnectAsync** che, a partire dalla creazione di un oggetto **SchedulerConnectionContext** e da un **CancellationToken**, stabiliscono la connessione allo Scheduler. Questa connessione ritornerà utile all'interno dei metodi usati per impostare i vari livelli di potenza: può succedere che si desideri diminuire i nodi usati, ma che alcuni di essi stiano eseguendo dei *task* e che quindi non si possano arrestare immediatamente. Perciò, l'uso del metodo **FindIdle** consente di ottenere una lista di nodi (**ISchedulerNode**) composti da **ISchedulerCore**, che ne rappresentano i *core*: in particolare, viene usata la loro proprietà chiamata State per stabilire se un nodo è inattivo e, quindi, può essere arrestato immediatamente.

Inoltre, il progetto da implementare in ERMAS necessita che non vengano demandate troppe operazioni da effettuare, in quanto il *cluster* sarebbe impegnato in configurazioni per tempi troppo prolungati: la gestione tramite eventi permette di tracciare e lasciare in attesa solo l'ultima operazione richiesta, per poi eseguirla quando la precedente è stata conclusa: lo *scheduling* di queste operazioni è delegato al metodo **CheckNext**.

Ma ciò che rende possibile aggiungere e rimuovere nodi dal *cluster*, oltre che manipolarne l'attività, è fornito dal secondo *namespace*. La classe usata è **ClusterManager**, la cui istanza è fornita tramite il metodo statico **ConnectAsync**. I metodi offerti dall'oggetto restituito dalla connessione operano soprattutto usando una serie di istanze appartenenti allo stesso spazio di nomi, quali **TemplateDescription**, che identifica i Node Template, e **NodeDescription**, che rappresenta i nodi su cui si intende operare. L'uso che viene fatto degli oggetti NodeDescription riguarda perlopiù le loro proprietà Name, NodeState e NodeHealth. La classe ClusterManager mette a disposizione metodi per compiere numerose operazioni, ma quelle usate particolarmente in questo caso impiegano due sue proprietà specifiche, ossia **TemplateManager** e **NodeManager**:

- **GetTemplateView** restituisce una collezione di Node Template;
- **GetNodeView** offre una lista di nodi di calcolo, con possibilità di operazioni di filtro;
- **AddAzureIaaSNodes** permette di creare nuovi nodi e portarli nello stato Not-Deployed: avviarli è un compito che viene eseguito successivamente, se necessario. Questo metodo richiede delle impostazioni riguardanti la quantità di risorse da creare, il loro dimensionamento e il Node Template da usare al momento dell'avvio: la comunicazione con Azure avviene in maniera implicita e non richiede l'uso di credenziali, poiché informazioni sulla sottoscrizione sono state inserite in fase di *deployment* delle risorse, richiamando lo *script* copiaBSH;
- **StartAzureIaaSNodes** avvia i nodi di calcolo specificati (aggregati in una lista) e li porta in stato Online;
- **StopAzureIaaSNodes** arresta i nodi e li porta in stato Not-Deployed: il vantaggio principale riguarda il fatto che, quando arrestati, i nodi non gravano nei costi relativi all'uso di macchine virtuali su Azure;
- **DeleteAzureIaaSNodes** elimina dal *cluster* l'insieme di nodi indicati.

Una parentesi va aperta riguardo questi due ultimi comandi: il risultato di entrambi è scontato, in quanto non vi sono addebiti legati alle risorse che sono manipolate dai due metodi. Ma la differenza principale riguarda le tempistiche di *deployment* in chiave

futura. Se si intende arrestare un numero di nodi che, nel breve periodo, sarà nuovamente richiesto, conviene usare StopAzureIaaSNodes, in quanto eliminarli dal *cluster* porta la necessità di creare *ex novo* altri nodi di calcolo. Questo si traduce principalmente in un rallentamento notevole del processo: riavviare un nodo comporta circa due minuti, mentre crearli e avviarli può impiegare anche un quarto d'ora, e i tempi possono crescere se le operazioni indicate dal Node Template sono molto lunghe (un nodo già configurato non necessita di effettuarle nuovamente). La scelta dipende dalle intenzioni di utilizzo che si hanno circa la gestione del *cluster* nel breve periodo.

La classe **AzureConfigurationManager** si compone di un insieme di campi e metodi costruiti a partire da quelli appena presentati. Le sue proprietà comprendono:

- **State**, che può assumere quattro diversi valori: LOW, MEDIUM, HIGH indicano i livelli di potenza configurabili, mentre ERR viene usato in fase di avvio per rilevare se il *cluster* non è in nessuno di questi stati e viene gestito impostando uno degli altri stati consistenti;
- **Current** rappresenta una stringa che specifica la prossima operazione da eseguire, se il *cluster* è attualmente impegnato in altre operazioni;
- **Ifdel** è un valore booleano che stabilisce se le risorse debbano essere arrestate o eliminate in caso di ridimensionamento del *cluster*;
- **Low, Med, High** rappresentano i tre livelli di potenza possibili in termini di numero di nodi da avere nelle tre configurazioni;
- **ClusterName, Size, TemplateName** sono stringhe usate per la connessione e la configurazione dei vari nodi;
- **Clusmgr** e **Template** definiscono gli oggetti (precedentemente illustrati) le cui primitive sono usate per comunicare con HPC Pack;
- **Nodes** è un'istanza **NodeCollection** che fornisce una lista dei nodi attualmente nel *cluster*; l'aggiornamento di questa lista è delegata al metodo **Update**.

I metodi offerti dalla classe AzureConfigurationManager permettono le operazioni precedentemente illustrate: innanzitutto, avviene la connessione tramite **Connect** che, si connette in maniera asincrona al NodeManager.

Il secondo metodo, **UpdateLists**, gestisce un *thread* separato che fornisce il servizio di *polling*: esso rileva, pertanto, variazioni nello stato di nodi notificando se, per esempio, l'avvio di un nodo è stato completato o se ne è stato richiesto l'arresto. Il metodo accessorio **FindIdle** è usato per comunicare con lo Scheduler: la sua funzione è quella di rilevare nodi inattivi per eseguirne l'arresto (o l'eliminazione). Il vantaggio apportato da questo metodo è che, se non è possibile arrestare il numero di nodi desiderato, AzureConfigurationManager provvede a operare su quelli liberi per poi, iterativamente, rilevarne di nuovi.

SetState rappresenta il cuore di questa fase del progetto: invocando questo metodo, infatti, è possibile portare il *cluster* nello stato desiderato. Se bisogna avviare dei nodi, esso si occupa automaticamente di controllare se è necessario prima crearne altri; se, invece, è richiesto che alcuni nodi vengano arrestati (o eliminati), usa il metodo FindIdle e opera su quelli disponibili. Inoltre, questo metodo rimane in attesa finché le operazioni richieste siano terminate prima di terminare l'esecuzione. **SetDel** si occupa, come accennato, di stabilire se arrestare o eliminare i nodi nelle richieste future. SetConf, invece, imposta il numero di nodi di Low, Med e High ai valori specificati.

Infine, il metodo **CheckNext**, tramite la gestione di eventi, viene usato per stabilire la configurazione corrente e schedulare le operazioni da eseguire al termine di quelle precedentemente avviato. Invoca quindi il metodo **Configure** che fornisce il Node Template e aggiorna la lista di nodi. Dopo aver controllato, tramite un servizio di *polling*, che non vi siano nodi in fase di configurazione, stabilisce lo stato corrente (sulla base del numero di nodi dei tre livelli): se rileva un'inconsistenza, è segnalato lo stato ERR e il cluster è portato al livello LOW. L'avvio della successiva operazione schedulata avviene usando il valore Current per invocare il metodo identificato dal costrutto *switch* e, alla sua conclusione, invoca ricorsivamente sé stesso per schedulare le prossime operazioni. Per motivi di test è stato aggiunto un ulteriore metodo, chiamato **GetInput**: esso genera un semplice *thread* costantemente in attesa di un input: se esso corrisponde a una delle operazioni indicate nello *switch* di CheckNext, la proprietà Current assume il suo valore. Per chiarezza, è ora mostrato il codice completo della classe AzureConfigurationManager.

```
using System;
using System.Threading;
using Microsoft.ComputeCluster.Management;
using System.Collections.Generic;
using Microsoft.Hpc.Scheduler;
using System.Linq;
using System.Threading.Tasks;

namespace HpcMgr
{
    class AzureConfigurationManager
    {
        enum Level { ERR, LOW, MED, HIGH };
        string[] accept = new string[] { "1", "2", "3", "4", "5", "6", "7", "8", "9", "0" };
        ManualResetEvent free = new ManualResetEvent(false);
        ManualResetEvent ev = new ManualResetEvent(false);
        ManualResetEvent notbusy = new ManualResetEvent(false);
        Bool starting;
        Level State { get; set; }
        string Current { get; set; }
        bool Ifdel { get; set; }
        int Low { get; set; }
        int Med { get; set; ;
        int High { get; set; }
        string ClusterName { get; set; }
        string Size { get; set; }
        string TemplateName { get; set; }
        ClusterManager Clusmgr { get; set; }
        TemplateDescription Template { get; set; }
        NodeCollection Nodes { get; set; }
//Connect to the cluster, wait for previous operations and set the current state
        public async Task Connect(string clusterName)
        {
            ClusterName=clustername;
            CancellationToken token = new CancellationTokenSource().Token;
            Clusmgr = ClusterManager.ConnectAsync(ClusterName, token).Result;
            Console.WriteLine("Connected to cluster: " + Clusmgr.ClusterName);
        }
        Internal async Task Configure()
        {
            TemplateCollection templates = Clusmgr.Templates.GetTemplateView(null);
            templates.Update();
            IEnumerable<TemplateDescription> temps = (IEnumerable<TemplateDescription>)templates;
            Template = temps.ToList<TemplateDescription>().Find(desc => desc.Name == TemplateName);
            Console.WriteLine("Found template: " + TemplateName);
            Nodes = Clusmgr.Nodes.GetNodeView(null, false, true);
            Nodes.Update();
            free.Set();
```

```
            notbusy.Set();
            await Task.Run(() =>
            {
                Console.WriteLine("Waiting for previous operations ");
                while (Nodes.Count > 0 && !Nodes.All(node => node.NodeState ==
Microsoft.ComputeCluster.Management.ClusterModel.ClusterNodeState.Online ||
                  node.NodeState == Microsoft.ComputeCluster.Management.ClusterModel.ClusterNodeState.Offline ||
                  node.NodeState ==
Microsoft.ComputeCluster.Management.ClusterModel.ClusterNodeState.NotDeployed))
                {
                    Thread.Sleep(5000);
                    Nodes.Update();
                }
            });
            //Setta stato
            if (Nodes.Where(node => node.NodeState ==
Microsoft.ComputeCluster.Management.ClusterModel.ClusterNodeState.Offline ||
                    node.NodeState ==
Microsoft.ComputeCluster.Management.ClusterModel.ClusterNodeState.Online).Count() == Low)
                State = Level.LOW;
            else if (Nodes.Where(node => node.NodeState ==
Microsoft.ComputeCluster.Management.ClusterModel.ClusterNodeState.Offline ||
                    node.NodeState ==
Microsoft.ComputeCluster.Management.ClusterModel.ClusterNodeState.Online).Count() == Med)
                State = Level.MED;
            else if (Nodes.Where(node => node.NodeState ==
Microsoft.ComputeCluster.Management.ClusterModel.ClusterNodeState.Offline ||
                    node.NodeState ==
Microsoft.ComputeCluster.Management.ClusterModel.ClusterNodeState.Online).Count() == High)
                State = Level.HIGH;
            else
            {
                Console.WriteLine("Inconsistent; bringing to Low");
                await SetState(Level.LOW, Low);
                State = Level.LOW;
            }
            Console.WriteLine("State set to " + State);
            Console.WriteLine("The manager is READY");
        }
        //Detect nodes operations
        public async Task UpdateLists()
        {
            await Task.Run(() =>
            {
                free.WaitOne();
                while (true)
                {
                    List<NodeDescription> old = Nodes.ToList().GetRange(0, Nodes.Count);
```

```
                var tr = old.FindAll(n => n.NodeHealth ==
Microsoft.ComputeCluster.Management.ClusterModel.NodeHealthState.Transitional);
                var ok = old.Except(tr);
                Thread.Sleep(10000);
                Nodes.Update();
                foreach (var node in old)
                {
                    if (!Nodes.Any(n => n.Name == node.Name))
                        Console.WriteLine(node.Name + " deleted");
                }
                foreach (var node in Nodes)
                {
                    if (tr.Any(n => n.Name == node.Name))
                    {
                        if (node.NodeHealth !=
Microsoft.ComputeCluster.Management.ClusterModel.NodeHealthState.Transitional)
                        {
                            if (node.NodeState ==
Microsoft.ComputeCluster.Management.ClusterModel.ClusterNodeState.NotDeployed)
                                Console.WriteLine(node.Name + " set to " + node.NodeState);
                            else
                                Console.WriteLine(node.Name + " set to " + node.NodeState);
                        }
                    }
                    else if (ok.Any(n => n.Name == node.Name))
                    {
                        if (node.NodeHealth ==
Microsoft.ComputeCluster.Management.ClusterModel.NodeHealthState.Transitional)
                        {
                            if (node.NodeState ==
Microsoft.ComputeCluster.Management.ClusterModel.ClusterNodeState.Provisioning)
                                Console.WriteLine(node.Name + " set to " + node.NodeState);
                            else if (node.NodeState ==
Microsoft.ComputeCluster.Management.ClusterModel.ClusterNodeState.Stopping)
                                Console.WriteLine(node.Name + " set to " + node.NodeState);
                            else if (node.NodeState ==
Microsoft.ComputeCluster.Management.ClusterModel.ClusterNodeState.Removing)
                                Console.WriteLine(node.Name + " set to " + node.NodeState);
                        }
                    }
                    else if (!old.Any(n => n.Name == node.Name))
                        Console.WriteLine(node.Name + " created");
                }
            }
        });
    }

                    //find idle nodes
```

```
        internal List<NodeDescription> FindIdle(List<NodeDescription> active)
        {
            List<NodeDescription> tostop = new List<NodeDescription>();
           //find idle nodes
            foreach (var node in active)
            {
                if (node.HasOngoingOperations())
                        tostop.Add(node);
            }
            return tostop;
        }
        //Set the state of the cluster
        private async Task SetState(Level newlv, int newnd)
        {
            await Task.Run(() =>
            {
                if (State != newlv)
                    Console.WriteLine(State + "->" + newlv);
                else
                    Console.WriteLine(State + " has now " + newnd);
                if (Nodes.Where(node => node.NodeState ==
Microsoft.ComputeCluster.Management.ClusterModel.ClusterNodeState.Offline ||
                    node.NodeState ==
Microsoft.ComputeCluster.Management.ClusterModel.ClusterNodeState.Online).ToList().Count() < newnd)
                {
                    if (Nodes.Where(node => node.NodeState ==
Microsoft.ComputeCluster.Management.ClusterModel.ClusterNodeState.NotDeployed).ToList().Count() < newnd)
                    {
                        Clusmgr.Nodes.AddAzureIaaSNodes(newnd - Nodes.Where(node => node.NodeHealth !=
Microsoft.ComputeCluster.Management.ClusterModel.NodeHealthState.Transitional).ToList().Count(), Size, false,
Template);
                        while (Nodes.Where(node => node.NodeState ==
Microsoft.ComputeCluster.Management.ClusterModel.ClusterNodeState.NotDeployed).ToList().Count() < newnd -
Nodes.Where(node => node.NodeState ==
Microsoft.ComputeCluster.Management.ClusterModel.ClusterNodeState.Offline ||
                    node.NodeState ==
Microsoft.ComputeCluster.Management.ClusterModel.ClusterNodeState.Online).ToList().Count())
                        {
                            Thread.Sleep(5000);
                            Nodes.Update();
                        }
                    }
                    Clusmgr.Nodes.StartAzureIaaSNodes(Nodes.Where(node => node.NodeState ==
Microsoft.ComputeCluster.Management.ClusterModel.ClusterNodeState.NotDeployed).ToList().GetRange(0,
newnd -
                        Nodes.Where(node => node.NodeState ==
Microsoft.ComputeCluster.Management.ClusterModel.ClusterNodeState.Offline ||
```

```
                node.NodeState ==
Microsoft.ComputeCluster.Management.ClusterModel.ClusterNodeState.Online).Count()));
                 while (Nodes.Where(node => node.NodeState ==
Microsoft.ComputeCluster.Management.ClusterModel.ClusterNodeState.Offline ||
                node.NodeState ==
Microsoft.ComputeCluster.Management.ClusterModel.ClusterNodeState.Online).ToList().Count() < newnd)
                {
                    Thread.Sleep(5000);
                    Nodes.Update();
                }
            }
            else if (Nodes.Where(node => node.NodeState ==
Microsoft.ComputeCluster.Management.ClusterModel.ClusterNodeState.Offline ||
                node.NodeState ==
Microsoft.ComputeCluster.Management.ClusterModel.ClusterNodeState.Online).ToList().Count() > newnd)
            {
                 while (Nodes.Where(node => node.NodeState ==
Microsoft.ComputeCluster.Management.ClusterModel.ClusterNodeState.Offline ||
                node.NodeState ==
Microsoft.ComputeCluster.Management.ClusterModel.ClusterNodeState.Online).ToList().Count() > newnd)
                {
                    if (!Ifdel)
                    {
                        var tostop = FindIdle(Nodes.Where(node => node.NodeState ==
Microsoft.ComputeCluster.Management.ClusterModel.ClusterNodeState.Offline ||
                node.NodeState ==
Microsoft.ComputeCluster.Management.ClusterModel.ClusterNodeState.Online).ToList());
                        Clusmgr.Nodes.StopAzureIaaSNodes(tostop.GetRange(0, Math.Min(tostop.Count,
(Nodes.Where(node => node.NodeState ==
Microsoft.ComputeCluster.Management.ClusterModel.ClusterNodeState.Offline ||
                node.NodeState ==
Microsoft.ComputeCluster.Management.ClusterModel.ClusterNodeState.Online).ToList().Count() - newnd))));
                        Thread.Sleep(5000);
                        Nodes.Update();
                    }
                    else
                    {
                        var todel = FindIdle(Nodes.Where(node => node.NodeState ==
Microsoft.ComputeCluster.Management.ClusterModel.ClusterNodeState.Offline ||
                node.NodeState ==
Microsoft.ComputeCluster.Management.ClusterModel.ClusterNodeState.Online).ToList());
                        Clusmgr.Nodes.DeleteAzureIaaSNodes(todel.GetRange(0, Math.Min(todel.Count,
Nodes.Where(node => node.NodeState ==
Microsoft.ComputeCluster.Management.ClusterModel.ClusterNodeState.Offline ||
                node.NodeState ==
Microsoft.ComputeCluster.Management.ClusterModel.ClusterNodeState.Online).ToList().Count() - newnd))
                            .Union(Nodes.Where(node => node.NodeState ==
Microsoft.ComputeCluster.Management.ClusterModel.ClusterNodeState.NotDeployed)));
```

```
                    Thread.Sleep(5000);
                    Nodes.Update();
                }
            }
            while (Nodes.Any(n => n.NodeHealth ==
Microsoft.ComputeCluster.Management.ClusterModel.NodeHealthState.Transitional))
            {
                Thread.Sleep(5000);
                Nodes.Update();
            }
        }
        State = newlv;
    });
}
//Set the boolean deletion value
private void SetDel()
{

    Ifdel = !Ifdel;
}
 //Change the value of one of the levels
private async Task SetConf(Level lv, int newvalue)
{
    Console.WriteLine(lv + " has now " + newvalue);
    switch (lv)
    {
        case Level.LOW:
            Low = newvalue;
            if (State == lv)
                await SetState(lv, newvalue);
            break;
        case Level.MED:
            Med = newvalue;
            if (State == lv)
                await SetState(lv, newvalue);
            break;
        case Level.HIGH:
            High = newvalue;
            if (State == lv)
                await SetState(lv, newvalue);
            break;
    }
}
 //Schedule next operations (or wait new ones)
public async Task CheckNext()
{
   If (starting)
        await Configure();
```

```
while (true)
{
    notbusy.WaitOne();
    ev.WaitOne();
    ev.Reset();
    string next = Current;
    Current = null;
    await Task.Run(async () =>
    {
        switch (next)
        {
            case "1":
                await SetState(Level.LOW, Low);
                break;
            case "2":
                await SetState(Level.MED, Med);
                break;
            case "3":
                await SetState(Level.HIGH, High);
                break;
            case "4":
                if (Low > 0)
                {
                    Low -= 1;
                    await SetConf(Level.LOW, Low);
                }
                break;
            case "5":
                if (Med > Low + 1)
                {
                    Med -= 1;
                    await SetConf(Level.MED, Med);
                }
                break;
            case "6":
                if (High > Med + 1)
                {
                    High -= 1;
                    await SetConf(Level.HIGH, High);
                }
                break;
            case "7":
                if (Low < Med - 1)
                {
                    Low += 1;
                    await SetConf(Level.LOW, Low);
                }
                break;
```

```
                    case "8":
                        if (Med < High - 1)
                        {
                            Med += 1;
                            await SetConf(Level.MED, Med);
                        }
                        break;
                    case "9":
                        High += 1;
                        await SetConf(Level.HIGH, High);
                        break;
                    case "0":
                        SetDel();
                        break;
                    default:
                        break;
                }
            }
        );
            notbusy.Set();
            await CheckNext();
        }
    }
    //Listen for the next input
    public async Task GetInput()
    {
        await Task.Run(() =>
        {
            string input = null;
            while (true)
            {
                input = Console.ReadLine();
                if (accept.Contains(input))
                {
                    Current = input;
                    this.ev.Set();
                }
            }
        });
    }
  }
}
```

Fig. 36: Classe AzureConfigurationManager

5. RISULTATI

Il vantaggio apportato dalla soluzione proposta rispetto a scenari *on-premise* è evidente. L'uso di macchine virtuali elimina ogni necessità di gestione di un insieme di server fisici, inclusi aggiornamenti, manutenzione e amministrazione del sistema in caso di guasti.

Un confronto più approfondito può essere fatto anche rispetto a un modello molto più statico, in cui l'intera infrastruttura viene mantenuta attiva per un tempo indefinito. In tal caso, infatti, i costi delle risorse gestite continuano a gravare sul cliente, anche quando non ne sta facendo alcun uso effettivo. Azure, in tal senso, offre due modelli particolari di pagamento: un contratto in base al consumo, in cui il cliente paga solo per le risorse che utilizza e in cui i costi sono disattivati quando le stesse sono deallocate; una modalità di istanze riservate in cui, a prescindere dall'uso effettivo, le risorse sono mantenute e i relativi costi addebitati per un periodo prestabilito (di solito, uno o tre anni). Ovviamente, per quest'ultima soluzione Azure offre una sensibile riduzione dei costi rispetto al pagamento in base al consumo, consentendo un risparmio fino al 72% per alcune risorse.

In questo modo, Microsoft va incontro alle necessità dei clienti che hanno bisogno di **istanze dedicate** per lunghi periodi di tempo, come nel caso di applicazioni *stand-alone*, database e siti web che necessitano di essere mantenuti indefinitamente.

Comunque, questa offerta non si adatta alle richieste di ERMAS e, in particolare, ai requisiti di calcolo distribuito di cui necessita. È preferibile, al contrario, una soluzione **Pay-As-You-Go** (PAYG): in base alle ore di utilizzo che l'infrastruttura richiederebbe, infatti, non conviene prenotare le istanze di macchina virtuale per la durata di un intero anno o, addirittura, tre. I tempi stimati di impiego sono ben inferiori: le simulazioni e i calcoli dei fattori di rischio offerti da Prometeia comportano operazioni commissionate al sistema solitamente in periodi predefiniti e con cadenza regolare, per esempio ogni fine del mese. Supponendo, quindi, di aver bisogno dell'infrastruttura nella sua completezza per un'intera settimana ogni mese, è evidente come ERMAS benefici il modello PAYG abbattendo i costi di utilizzo.

Come già descritto, Azure mette a disposizione numerose istanze di macchina virtuale a seconda dell'utilizzo per cui esse sono state pianificate: il carico di lavoro che si

intende delegare loro porta a requisiti ben definiti in termini di potenza di calcolo e di memoria fornita. L'effetto principale di questa offerta si traduce in una vasta gamma di dimensionamenti e prezzi messi a disposizione.

A seconda degli scopi per cui sono progettate, le macchine virtuali di Azure si dividono in sei macrocategorie:

- **General purpose**, per carichi di lavoro equilibrati o per creare ambienti di test e sviluppo;
- **Compute optimized**, ideale per scenari che richiedono capacità computazionale costantemente ottimizzata;
- **Memory optimized**, per applicazioni che effettuano operazioni intensive in memoria, che necessitano di *throughput* elevato. Un caso particolare può essere la presenza di database con elevato carico di richieste da gestire;
- **Storage optimized**, se si necessita di un rapporto molto elevato di operazioni input/output al secondo;
- **GPU**, che fa uso di NVIDIA GPU;
- **High-Performance Compute**, ideale in contesti a elevato utilizzo della rete, sforzo computazionale per soluzioni di High-Performance Computing.

In tabella sono mostrate le varie classi di macchine virtuali che Azure mette a disposizione. Alcune considerazioni riguardano la disponibilità dei vari tipi di Virtual Machine nelle diverse regioni, non sempre garantita. Il tipo scelto, inoltre, influisce sul numero massimo di dischi dati che è possibile connettere all'istanza di VM creata. [7]

VM Series	Tipo	Utilizzi	Prezzi minimi(€/mese)
A-series	General purpose	Sviluppo/test	9.85
B-series	General purpose	Carichi di lavoro contenuti	2.31
D-series	General purpose	RDBMS, analisi, caching	35.09
DC-series	General purpose	Carichi di lavoro con requisiti di sicurezza	41.25
E-series	Memory optimized	RDBMS, caching, analisi in-memory	77.57
F-series	Compute optimized	Web server, batch processing, analisi	30.17
G-series	Storage optimized	Web server, batch processing, analisi	270.26
H-series	Compute optimized	Simulazioni su larga scala (HPC)	490.03
Ls-series	Storage optimized	Data Warehouse, DB transazionali, NoSQL	230.24
M-series	Compute optimized	Workload in-memory	945.89
Mv2-series	Storage optimized	SQL, calcolo parallelo	13 734.24
N-series	GPU optimized	AI, deep learning, editing, simulazioni	554.05

Fig. 37: Serie di Azure VM

Le stime di prezzo sono fatte sulla base delle dimensioni delle macchine virtuali e di storage di Azure meglio adattabili al modello, ma il concetto è facilmente estendibile ad altri possibili scenari. Innanzitutto, le istanze di Virtual Machine richiedono un

costo per la licenza di Windows Server, a eccezione dell'istanza SQL Server. Quest'ultima richiede un dimensionamento adatto a un contesto di applicazioni con uso intensivo di memoria: per esempio, è possibile usare un'istanza E8a v4, con 8 vCore, 64 GiB di RAM e 200 GiB di archiviazione temporanea.

Per quanto riguarda la Presentation Virtual Machine, il Domain Controller e l'Head Node, invece, si può optare per istanze D8as v4, che dispongono di 8 vCore, 32 GiB di RAM e 64 GiB di archiviazione: questa categoria di macchine, infatti, è ideata per supportare numerosi scenari e si adatta molto bene alla maggior parte dei contesti applicativi. Infine, i nodi di calcolo del cluster HPC usano la serie H, appositamente designata per la modellazione dei rischi finanziari, per simulazioni e analisi molecolari e genomiche.

In particolare, si può supporre di usare istanze H8 dotate di 8 vCore e 56 GiB di RAM. Per la gestione dei Compute Node, solitamente i requisiti di un cluster standard si attestano ai 64 vCore, con possibilità di picchi anche fino a 128 e, in momenti di minor carico, anche intorno ai 16: le analisi si basano sull'uso di istanze dotate di 8 vCore.

Ora verrà presentato un confronto dei costi con riservazione nei casi di soluzione PAYG, prenotazione per un anno e per tre anni, tenendo conto dei tre scenari possibili. In questo modo, è possibile tenere traccia dei costi fissi (legati alle istanze statiche) e di quelli variabili legati ai nodi di calcolo. Le stime di seguito sono annuali e si basano sull'utilizzo delle macchine virtuali per una durata totale di una settimana al mese.

Innanzitutto, viene mostrata la tabella che esprime in dettaglio un confronto di prezzi per le istanze base del *cluster*, che costituiscono il cuore dell'infrastruttura.

VM	Dimensioni	PAYG (€/anno)	1 anno (€/anno)	3 anni (€/anno)	Risparmio 1 – 3 anni
Domain Controller	D8as v4	1 407.77	4 979.18	4 274.00	71.7% - 67.0%
Presentation	D8as v4	1 407.77	4 979.18	4 274.00	71.7% - 67.0%
Head Node	D8as v4	1 407.77	4 979.18	4 274.00	71.7% - 67.0%
SQL Server	E8a v4	1 033.80	2 642.00	1 773.90	60.8% - 41.7%
TOTALE	**//**	**5 257.11**	**17 579.54**	**14 595.9**	**70.1% - 64.0%**

Fig. 38: Costi e risparmi di PAYG per VM fisse

La scelta di una macchina virtuale SQL Server così dimensionata, inoltre, garantisce una riduzione dei costi rispetto alle soluzioni di istanza gestita e di pool elastico. Infatti, queste soluzioni necessitano che le risorse siano mantenute all'interno di Azure anche per lunghi periodi in cui esse non risultano necessarie; il costo orario, pertanto, è notevolmente inferiore: si passa da € 1.35/ora per il pool elastico e da € 1.81/ora per l'istanza gestita a € 1.70/ora. Sebbene la differenza di prezzo non sembri avvantaggiarla, la soluzione di istanziare una macchina virtuale SQL Server è ideale perché, quando non più necessaria, è sufficiente deallocare l'istanza per interrompere qualsiasi addebito.

Infine, la tabella seguente mostra i vari scenari di potenza di calcolo con 2, 8 e 16 Compute Node dedicati. Qui è illustrata dunque la parte variabile del *cluster*, con i vari livelli configurati. Si noti come l'aggiunta di un nuovo nodo di calcolo incrementi il costo di € 1.44/ora. [7]

Potenza di calcolo	Dimensioni	PAYG (€/anno)	1 anno (€/anno)	3 anni (€/anno)	Risparmio 1 – 3 anni
Low (~2)	H8	5 800.83	15 284.45	12 532.06	62.04% - 53.71%
Medium (~8)	H8	23 203.35	61 137.80	50 128.24	62.04% - 53.71%
High (~16)	H8	46 406.64	122 275.60	100 256.48	62.04% - 53.71%

Fig. 39: Costi e risparmi di PAYG per Compute Node di HPC

CONCLUSIONE

In base a quanto sviluppato nel corso di questo progetto, risulta evidente l'importanza dell'utilizzo delle infrastrutture di *cloud computing* proposte da Microsoft, soprattutto tenendo conto che tecnologie di questo tipo si adattano molto bene a scenari di calcolo distribuito. HPC Pack supporta nativamente l'uso di risorse virtuali tramite Microsoft Azure e, grazie a questa soluzione, è possibile implementare l'intero sistema desiderato all'interno della piattaforma cloud. In particolare, la migrazione del modello proposto da ERMAS è semplificata mediante l'uso delle macchine virtuali e delle tecniche di automazione messe a disposizione da Azure. La virtualizzazione garantisce il vantaggio essenziale di non doversi più occupare di server fisici e, sulla base di ciò, tutto ciò che riguarda la gestione del livello strutturale di *networking*, installazione e configurazione viene delegato a terze parti. Inoltre, tutto ciò viene attuato senza tener conto della compatibilità con le soluzioni precedenti e, soprattutto, senza rivedere alcun modulo software precedentemente implementato.

La seconda parte del progetto costituisce un ulteriore passo avanti nel passaggio al cloud: i costi dell'utente sono gestiti in maniera molto più efficiente, in quanto è possibile modellare l'infrastruttura in tempo reale, implementandola solo quando necessario e deallocandola interamente dove non dovesse più essere necessaria. Il vantaggio è duplice: non sono la configurazione non è più un compito diretto del cliente, ma anche i costi che vi vengono addebitati sono molto più flessibili e adattati alle sue esigenze.

Il prossimo passo del progetto consiste nell'implementazione del servizio all'interno delle già consolidate tecnologie di ERMAS, in modo da inserire un'interfaccia utente con la quale effettuare le già citate tecniche di gestione del cluster HPC. Il notevole risparmio economico incentiverà clienti di diversa natura ad adottare un approccio *cloud-oriented* garantendo allo stesso tempo un sistema molto più stabile su cui il software progettato può operare.

BIBLIOGRAFIA

1. https://www.prometeia.it/home
2. Reese G., *Cloud Application Architectures*, O'Reilly, Sebastopol, USA, 2009, pp. 2-18; 23
3. Velte A., Velte T., Elsenpeter R., *Cloud Computing A Practical Approach*, McGrawHill, USA, 2009, pp. 3-18
4. Baun C., Kunze M., Nimis J., Tai S., *Cloud Computing Web-Based Dynamic IT Services*, Springer, Berlin, Germany, 2011, pp. 3-4; 36-37
5. Ferrari A., Zanleone E., *Cloud Computing Aspettative, problemi, progetti e risultati di aziende passate al modello "as a service"*, FrancoAngeli, Milano, Italy, 2011, pp.15-20
6. Jamsa K., *Cloud Computing SaaS, PaaS, IaaS, Virtualization, Business Models, Mobile, Security, and More*, Jones & Bartlett, Burlington, USA, 2013, pp. 7-8; 19-56
7. https://docs.microsoft.com/it-it/
8. Copeland M., Soh J., Puca A., Manning M., Gollob D., *Microsoft Azure Planning, Deploying, and Managing Your Data Center in the Cloud*, Apress, New York, USA, 2015, pp. 10-21
9. Collier M., Shahan R., *Fundamentals of Azure Second Edition*, Microsoft Press, Redmond, USA, 2016, pp. 5-8; 70-78; 101-107; 148-149; 157-180
10. Tulloch M., *Introducing Windows Azure For IT Professionals*, Microsoft Press, Redmond, USA, 2013, pp. 31-32; 115-121
11. Foulds I., *Scopri Azure a piccole dosi in un mese*, Manning, Shelter Island, USA, 2018, pp. 65-66; 83-89; 185

12. Soueidi C., *Microsoft Azure Storage Essentials*, Packt, Birmingham, UK, 2015, pp. 1-7
13. Zaal S., *Architecting Microsoft Azure Solutions – Exam Guide 70-535*, Packt, Birmingham, UK, 2018, pp. 29; 159-163; 193-194
14. Waly M., *Learning Microsoft Azure Storage*, Pack, Birmingham, UK, 2017, pp. 13-20
15. Crump M., Luijbregts, *The Developer's Guide to Azure*, ebook, 2019, pp. 42
16. Savill J., *Microsoft Azure Infrastructure Services for Architects Designing Cloud Solutions*, Indianapolis, USA, 2020, pp. 171-176
17. Karthikeyan S., *Practical Microsoft Azure IaaS*, Apress, Bangalore, India, 2018, pp. 2-16; 23-26
18. Zaal S., Malik A., Rossel S., Marston J., Wali M., Demiliani S., *Migrating Applications to the Cloud with Azure*, Packt, Birmingham, UK, 2019, p. 176
19. Modi R., *Azure Resource Manager Templates Quick Start Guide*, Packt, Birmingham, UK, 2019, pp. 113-118
20. Mitchell T., *Azure Powershell Quick Start Guide*, Packt, Birmingham, UK, 2018, pp. 5-18
21. Chapman J., Dhally A., *Automating Microsoft Azure with Powershell*, Packt, Birmingham, UK, 2015, pp. 1-14