1. ***Introduzione: Obiettivi, Overview dell'applicazione e Componenti***

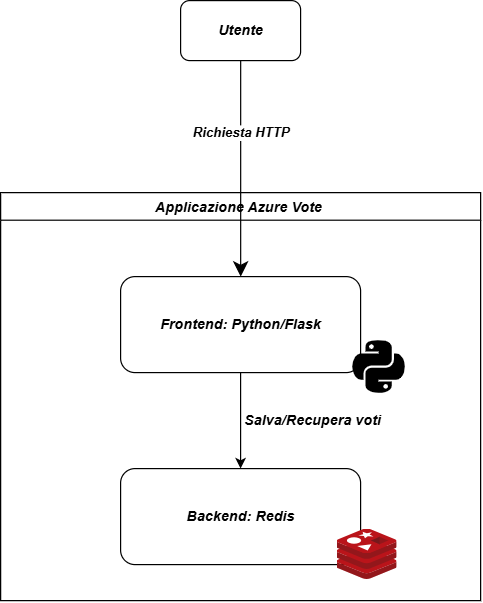
L'applicazione in questione è una semplice web app di voto, che permette agli utenti di votare per due opzioni: "Cats" o "Dogs". I voti vengono raccolti in tempo reale e visualizzati sullo schermo.

Il codice sorgente dell'applicazione è disponibile pubblicamente nel repository GitHub di Azure: <https://github.com/Azure-Samples/azure-voting-app-redis>.

L'architettura dell'applicazione è basata su microservizi ed è composta da due componenti principali:

Frontend: Un'applicazione web basata su Python (Flask) e un server web NGINX. È l'interfaccia utente che gestisce la logica di voto e l'interazione con l'utente.

Backend: Un database NoSQL in memoria, Redis. È responsabile dell'archiviazione e del recupero dei voti.



### **Panoramica dei File nel Repository**

All'interno del repository azure-voting-app-redis, qui presenti elencati i seguenti file per la comprensione e il funzionamento dell'applicazione.

* **main.py**: Il cuore dell'applicazione. Questo file contiene il codice sorgente dell'app web in Python, sviluppata con il framework Flask.
* **Dockerfile**: Le istruzioni per costruire l'immagine Docker del frontend. Definisce l'ambiente di runtime dell'applicazione.
* **requirements.txt**: Elenca tutte le dipendenze Python necessarie al funzionamento dell'app.
* **docker-compose.yaml**: Utilizzato per l'ambiente di sviluppo locale. Questo file definisce e avvia i container dell'applicazione in locale, senza Kubernetes.
* **azure-vote-all-in-one-redis.yaml**: Il file di configurazione per Kubernetes (manifest) che ho utilizzato per il deployment finale. Definisce i servizi, i deployment e i volumi.

**Nota:** Per una spiegazione più dettagliata e un'analisi approfondita di ogni singolo file, si è creato un archivio di progetto separato dedicato all'apprendimento personale.

1. ***Architettura Selezionata:***

Dopo un'attenta analisi delle opzioni disponibili, ho optato per una Architettura AKS Standard .

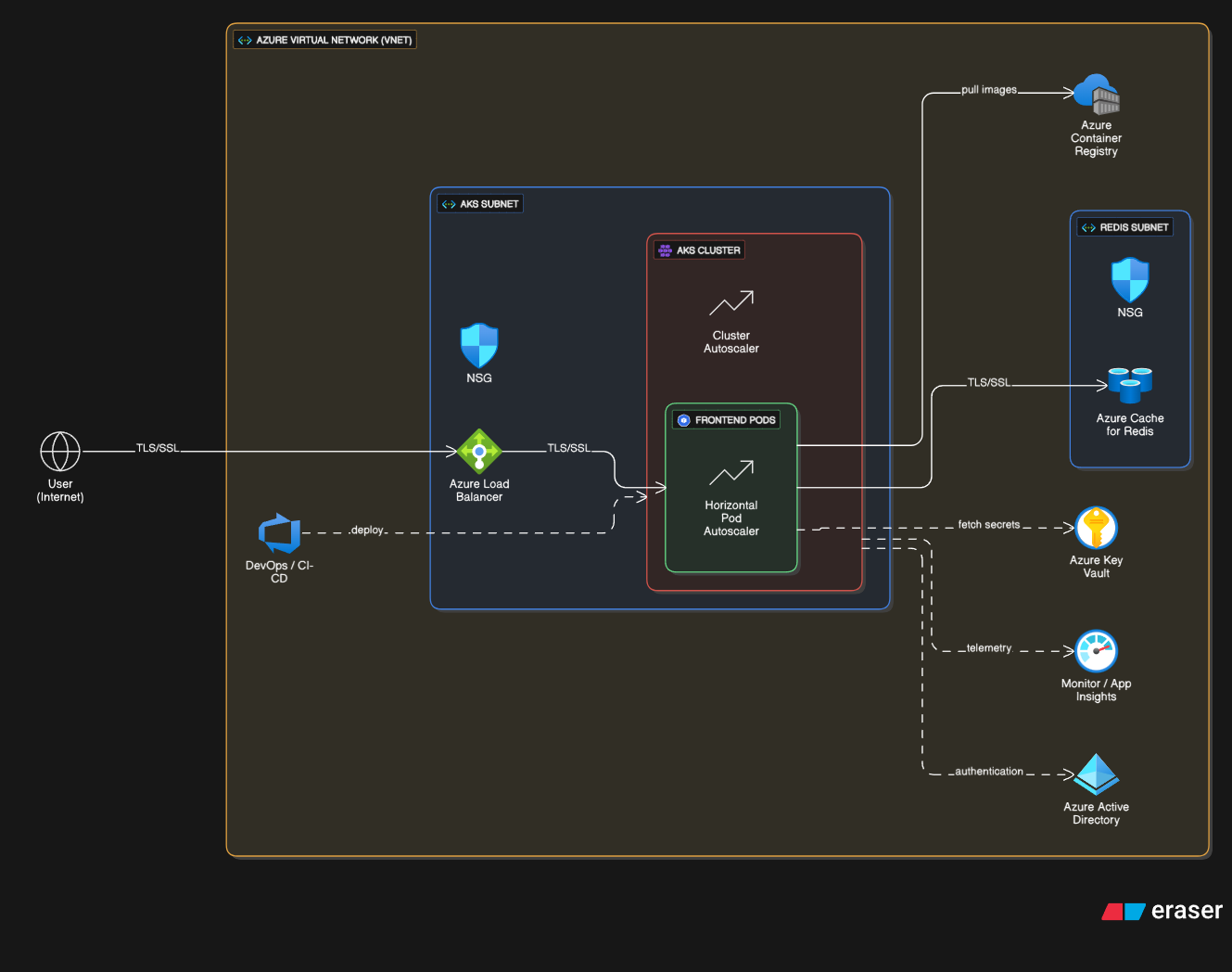
La scelta di questa architettura è guidata dalla necessità di bilanciare scalabilità, affidabilità e gestione operativa, garantendo al contempo la portabilità dell'applicazione.

* ***Perché Azure Kubernetes Service (AKS)?***

AKS è stato selezionato come piattaforma di orchestrazione principale. A differenza di soluzioni basate su macchine virtuali, AKS fornisce un'infrastruttura standardizzata e portabile per i container. Gestisce automaticamente la scalabilità, l'alta disponibilità e gli aggiornamenti dei nodi worker, riducendo significativamente l'onere operativo e permettendo al team di concentrarsi sullo sviluppo dell'applicazione piuttosto che sulla gestione dell'infrastruttura.

* ***Perché Azure Container Registry (ACR)?***

L'utilizzo di un registro di container privato su Azure garantisce che le immagini dell'applicazione siano sempre disponibili per il cluster AKS con basse latenze. Inoltre, ACR si integra perfettamente con i sistemi di sicurezza e di identità di Azure, offrendo un'ulteriore garanzia di sicurezza e controllo sull'accesso alle immagini.



***Servizi Azure Utilizzati:***

* ***Azure Kubernetes Service (AKS***): Il fulcro dell'infrastruttura. AKS gestisce i nodi Kubernetes e l'orchestratore, assicurando la scalabilità e la resilienza dei tuoi container. (<https://learn.microsoft.com/en-us/azure/aks/intro-aks-automatic>).
* ***Azure Container Registry (ACR):*** Un registro di immagini Docker privato e sicuro, utilizzato per memorizzare le immagini container dell'applicazione. AKS si integra nativamente con ACR per il pull delle immagini. [(https://learn.microsoft.com/en-us/azure/container-registry/container-registry-intro](https://learn.microsoft.com/en-us/azure/container-registry/container-registry-intro)).
* ***Azure Cache for Redis:*** Un'implementazione fully managed del database Redis. Gestirà l'alta disponibilità, la scalabilità e gli aggiornamenti del database in memoria, sostituendo l'istanza containerizzata.(<https://learn.microsoft.com/en-us/azure/azure-cache-for-redis/cache-overview>).
* ***Azure Load Balancer:*** Incluso in AKS, esporrà il servizio frontend su Internet, distribuendo il traffico in entrata tra i vari pod del frontend per garantire l'alta disponibilità.(<https://learn.microsoft.com/it-it/azure/load-balancer/load-balancer-overview>).
* ***Azure Key Vault***: Un servizio per la gestione centralizzata di chiavi, segreti e certificati. In un'architettura di produzione, verrebbe utilizzato per memorizzare in modo sicuro la stringa di connessione a Redis e altre credenziali, evitando di inserirle direttamente nei file di configurazione.[(https://learn.microsoft.com/en-us/azure/key-vault/general/basic-concepts](https://learn.microsoft.com/en-us/azure/key-vault/general/basic-concepts)).
* ***Azure Monitor / Application Insights:*** Un servizio di telemetria completo per monitorare le performance dell'applicazione. Permette di raccogliere log, tracciare le metriche di utilizzo e diagnosticare i problemi in produzione.(<https://learn.microsoft.com/it-it/azure/azure-monitor/app/app-insights-overview>).

***4. Processo di Deployment: Guida Passo-Passo***

Questa sezione illustra la procedura completa per distribuire l'applicazione ***azure-vote-app*** su un ***cluster Azure Kubernetes Service (AKS)***, a partire dalla creazione dell'infrastruttura fino alla verifica finale.

***Prerequisiti***

* ***Azure CLI***: Per la gestione delle risorse Azure. [Documentazione](https://learn.microsoft.com/it-it/cli/azure/get-started-with-azure-cli?view=azure-cli-latest)
* ***kubectl***: Per interagire con il cluster Kubernetes. [Documentazione](https://kubernetes.io/docs/tasks/configure-pod-container/)
* ***Podman (o Docker):*** Per la containerizzazione dell'applicazione.

[Documentazione](https://docs.podman.io/en/stable/markdown/podman.1.html)

* Credenziali di login valide per l’account Azure.

***Fase 1: Preparazione dell'Infrastruttura Azure***

In questa fase, verranno create tutte le risorse Azure necessarie per ospitare e gestire l'applicazione.

*Accedere ad Azure e creare il gruppo di risorse*.

| az login |
| --- |

| az group create --name azure-vote-rg --location westeurope |
| --- |

*Creare il Registro di Container (ACR), il Cluster Kubernetes (AKS) e l'istanza di Redis Cache:*

| Bash  az acr create --resource-**group** **azure-vote-rg** --name appvoteregistry --sku Basic az redis create --resource-**group** **azure-vote-rg** --name azure-vote-redis-its --**location** **westeurope** --sku Basic --vm-size C1 az aks create --resource-**group** **azure-vote-rg** --name azure-vote-cluster --**node-count** 2 --enable-managed-identity --attach-acr appvoteregistry |
| --- |

Configurare kubectl per connettersi al nuovo cluster:

| Bash  az aks get-credentials --resource-group azure-vote-rg --name azure-vote-cluster |
| --- |

***Fase 2: Containerizzazione e Pubblicazione dell'Applicazione***

Qui l'applicazione viene trasformata in un'immagine container e caricata sul proprio registro privato.

Eseguire il login a Podman (o Docker) per connettersi al AC:

| Bash  podman login appvoteregistry.azurecr.io |
| --- |

Costruire l’immagine container :

| **Bash**  podman **build** -t appvoteregistry.azurecr.io/azure-vote-app-redis:v1 . |
| --- |

Caricare l'immagine sul proprio ACR:

| **Bash**  podman **push** appvoteregistry.azurecr.io/azure-vote-app-redis:v1 |
| --- |

***Fase 3: Deployment dell'Applicazione su AKS***

Questo è il passaggio che distribuisce i servizi e i container sul cluster **Kubernetes.** Si basa sul file **‘azure-vote-all-in-one-redis.yaml’** , che definisce l'architettura finale con **Redis** all'interno del cluster.

Eseguirei il deployment utilizzando il file YAML:

| Bash  kubectl apply -f azure-vote-**all**-**in**-one-redis.yaml |
| --- |

***Fase 4: Troubleshooting e Diagnosi***

Questi comandi sono stati cruciali per diagnosticare e risolvere i problemi di connettività e configurazione.

Verifica dello stato dei nodi del cluster:

| **Bash**  kubectl get nodes |
| --- |

Verifica dello stato dei pod per diagnosticare errori:

| **Bash**  kubectl get pods |
| --- |

Visualizzare i log di un pod specifico:

| Bash  kubectl logs [NOME\_POD] Accedi **al** pod per **il** **debug** dei problemi **di** rete. |
| --- |

| Bash  kubectl exec -it $(kubectl get pods --selector=app=azure-vote-front --output=jsonpath='{.items[0].metadata.name}') -- /bin/bash |
| --- |

***Fase 5 : Implementazione di Azure Key Vault***

L'obiettivo è rimuovere le credenziali dai file di configurazione e gestirle in modo sicuro.

Creazione del Key Vault e Configurazione dell'Accesso:

| Bash  az keyvault create --resource-**group** **azure-vote-rg** --name your-keyvault-name --**location** **westeurope** |
| --- |

Ottenere l'ID dell'identità gestita del tuo cluster AKS. Questo ID è necessario per assegnare i permessi.

| Bash  $aks\_identity = az aks show --resource-group azure-vote-rg --name azure-vote-cluster --query "identity.principalId" --output tsv |
| --- |

Assegnare il ruolo Key Vault Secrets User al cluster AKS. Questo permette al cluster di leggere i segreti :

| Bash  az **role** **assignment** create --**role** **"Key** Vault Secrets User" --assignee $aks\_identity --scope "/subscriptions/$(az account show --query id -o tsv)/resourceGroups/azure-vote-rg/providers/Microsoft.KeyVault/vaults/your-keyvault-name" |
| --- |

Ottenere i l'ID del proprio utente , per ottenere i permessi di scrittura : .

| Bash  $my\_user\_identity = az ad signed-**in**-user show --query "id" --output tsv |
| --- |

Assegnare il ruolo Key Vault Secret Officer al proprio utente. Questo darà i permessi necessari per scrivere e gestire i segreti :

| Bash  az **role** **assignment** create --**role** **"Key** Vault Secrets Officer" --assignee $my\_user\_identity --scope "/subscriptions/$(az account show --query id -o tsv)/resourceGroups/azure-vote-rg/providers/Microsoft.KeyVault/vaults/your-keyvault-name" |
| --- |

Salvare la stringa di connessione a Redis come segreto. Sostituire il proprio nome e il valore del segreto con il nome del servizio interno di Kubernetes.

| ***Bash  az keyvault secret set --vault-name your-keyvault-name --name RedisConnectionString --value "\*\*\*\*\*"*** |
| --- |

### ***Fase 6: Abilitazione del Monitoraggio con Azure Monitor***

Eseguendo il seguente comando, Abiliterà l'add-on di monitoraggio sul proprio cluster AKS. Questo add-on si occuperà di installare gli agenti necessari su ogni nodo, che inizieranno a inviare dati a un'area di lavoro Log Analytics.

Questo passaggio è fondamentale per:

* **Monitoraggio in tempo reale**: Raccogliere metriche e dati sulle prestazioni del tuo cluster e dei tuoi container.
* **Log centralizzati**: Visualizzare i log dell'applicazione e del sistema in un unico posto per una facile diagnostica.

| **bash**  az aks enable-**addons** --resource-group azure-vote-rg --name azure-vote-cluster --**addons** monitoring |
| --- |

***Fase 7: Adattamento e controllo Finale***

Questa è stata la fase in cui ho adattato l'infrastruttura per renderla compatibile con l'applicazione, portando al deployment di successo.

Re-deployment dell'applicazione con il file YAML.:

| Bash  kubectl apply -f azure-vote-**all**-**in**-one-redis.yaml |
| --- |

Verifica finale dei servizi e dei pod per confermare il funzionamento.

| Bash  kubectl get service kubectl get pods |
| --- |

***Accesso all'Applicazione***

Una volta completato il deployment, l'applicazione è accessibile pubblicamente tramite l'indirizzo IP assegnato. Questo indirizzo espone il servizio frontend e permette di interagire con la web app di voto.

L'applicazione è ora disponibile e può essere testata al seguente indirizzo:

<http://108.142.112.172/>