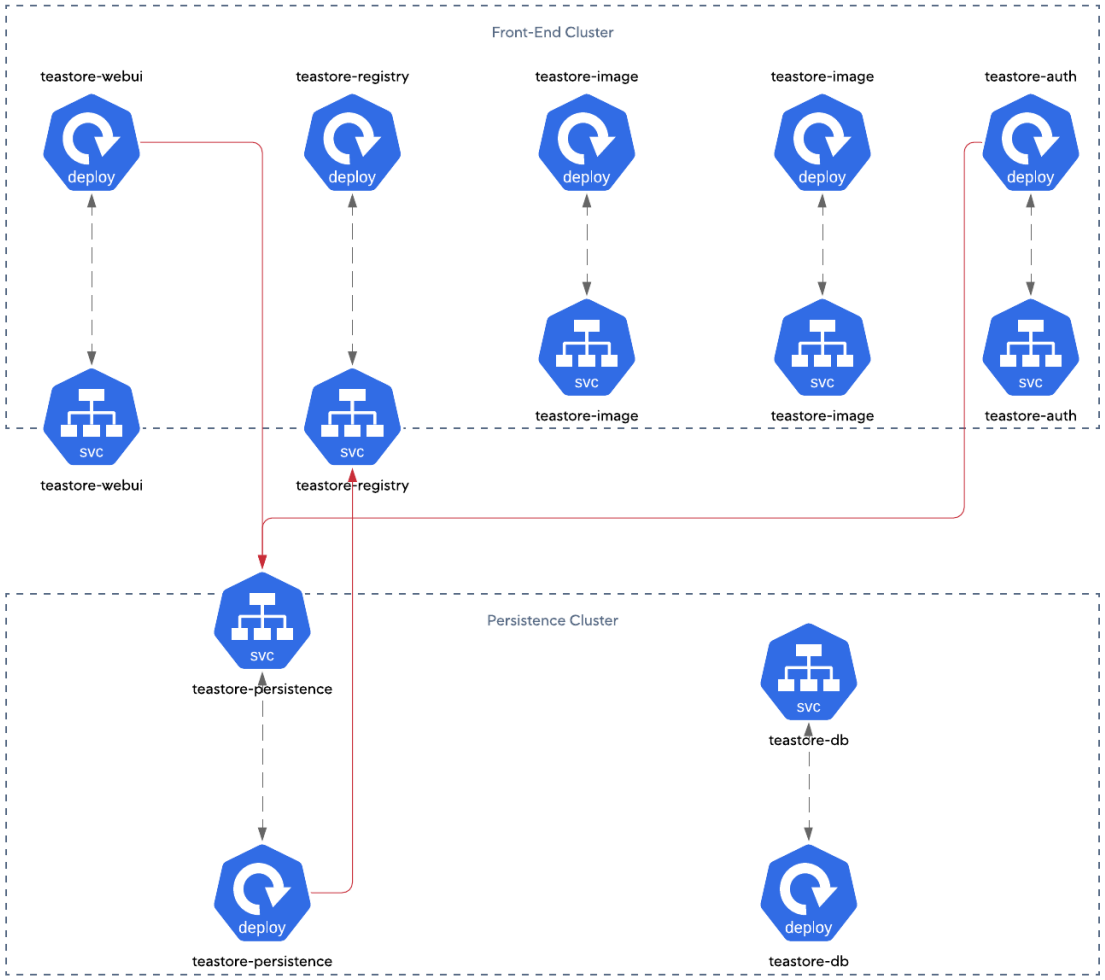
Deploying the TeaStore Application Across Multiple Kubernetes Clusters

Introduction

This doc outlines the process for deploying the TeaStore application across multiple Kubernetes clusters. We will be separating the persistence and database layers into a separate cluster.

End Architecture

The end architecture diagram is shown below. The redlines indicate cross-cluster network connections between services.



Approach

The original authors of the TeaStore application documented multiple deployment strategies centered around two different technologies: containers and Java application servers. As part of the container approach, the following options were documented:

1. Single container
2. Multiple single-service containers
3. Docker-compose
4. Single Kubernetes cluster

To achieve the desired architecture, we used the deployment strategy for a single Kubernetes cluster. In the Kubernetes deployment strategy, an environment variable for the host name of each service is set to the simple name of the service (i.e., teastore-webui and teastore-registry), and the services use this environment variable when registering with the registry service. Since using this simple host name is not reachable from services residing in a separate cluster, we modified the deployment strategy to use patterns derived from the multiple single-service container strategy.

All services need to communicate with the registry service, so we exposed the Kubernetes service for the registry as a load balancer. In addition, because the webui and auth services communicate with the persistence service in certain transactions, we exposed the persistence service as a load balancer. These are denoted in the diagram above as services sitting on the border of the dotted box representing the cluster.

The final deployment artifacts consist of 4 YAML files: one for the registry service, one for the persistence and db services, one for the remaining services, and one for load generation on the application.

Deployment Directions

Assumptions

- These steps used Kubernetes clusters created in public cloud infrastructure.
- The steps do not include directions for cluster creation.
- The steps do not include instrumentation using AppDynamics.
- The implementer will need to switch Kubernetes contexts as needed throughout the steps
- Additional steps may be required for configuring access into the IKS environment (proxies, etc.).

Steps

Cluster	Step	Terminal Command
Front-End Cluster	1. Deploy registry to front-end cluster.	kubectl apply -f ./teastore-registry.yaml
Front-End Cluster	2. Save registry load balancer address as environment variable.	export REGISTRY_LOADBALANCER_HOST=$(kubectl get svc teastore-registry -o json | jq -r '.status.loadBalancer.ingress[0] | .[]')
Persistence Cluster	3. Set registry URL in persistence deployment and deploy persistence to persistence cluster.	cat teastore-persistence.yaml | sed -e "s/\$REGISTRY_LOADBALANCER_HOST/$REGISTRY_LOADBALANCER_HOST/g" | kubectl apply -f -
Persistence Cluster	4. Save persistence load balancer address as environment variable.	export PERSISTENCE_LOADBALANCER_HOST=$(kubectl get svc teastore-persistence -o json | jq -r '.status.loadBalancer.ingress[0] | .[]')
Persistence Cluster	5. Update host name environment variable of persistence deployment.	kubectl set env deployment/teastore-persistence HOST_NAME=$PERSISTENCE_LOADBALANCER_HOST
Front-End Cluster	6. Deploy remaining front-end services to front-end cluster.	kubectl apply -f ./teastore-frontend.yaml
Front-End Cluster	7. Deploy load generation deployments.	Kubectl apply -f ./teastore-loadgen.yaml