Seamlessly deploying & managing Kubernetes across multi-cloud with VMware Technologies
Table of Contents:

Seamlessly deploying & managing Kubernetes across multi-cloud with VMware Technologies..................1
Multi-Cloud Challenges and VMware Cloud Capabilities: ........................................................................3
What Defines the Ideal Cloud Environment? .........................................................................................3
Kubernetes provides an ideal platform for Multi and Hybrid Cloud: ..................................................4
  Tanzu Kubernetes Grid: .....................................................................................................................5
  Simplified Installation .......................................................................................................................5
  Automated multi-cluster operations .................................................................................................6
  Integrated Platform Services .............................................................................................................6
  Open-Source Alignment ....................................................................................................................6
Where does Tanzu Kubernetes Grid run? ...............................................................................................6
  Private cloud .....................................................................................................................................6
  Public cloud ......................................................................................................................................6
  Edge: ..............................................................................................................................................7
Tanzu Service Mesh: ..............................................................................................................................7
  What Makes Tanzu Service Mesh Different? .......................................................................................8
Tanzu Mission Control: ..........................................................................................................................9
Provider and Customer Challenges ........................................................................................................10
  The Problem: .................................................................................................................................10
Solution Components: ........................................................................................................................10
  The Solution: ..................................................................................................................................11
Solution Architecture: ........................................................................................................................11
Solution Configuration: ........................................................................................................................12
Conclusion: ..........................................................................................................................................19
Appendix A: Fitness_Cluster.yaml .....................................................................................................20
Appendix B: Catalog YAML file: ........................................................................................................24
Appendix C: Load Generator ..............................................................................................................26
Multi-Cloud Challenges and VMware Cloud Capabilities

The impact of cloud continues to be undeniable to both business and IT. Cloud has redefined the relationship between business and IT, reshaping business models, accelerating delivery of new business services, created new models for customer engagement and improved the efficiency and effectiveness of employees.

**Figure 1: Challenges with Multi-Cloud**

But the cloud market is at an inflection point. Organizations have hundreds or thousands of applications - both existing monoliths and new cloud microservices. They are all different. But they are also critical to their business. Longstanding app architectures are giving way to new cloud native models. The worlds of datacenter, cloud and edge are converging. And the diversity of multi-cloud, once viewed as chaotic and complex, is emerging as the most powerful source of innovation.

**What Defines the Ideal Cloud Environment?**

- Freedom to build and run applications for ANY environment
- With development and operations teams collaborating freely
- Ability to manage diverse environments CONSISTENTLY
- With applications and data that are secure and protected EVERYWHERE
- And the freedom to change my mind in the future without PENALTY
Figure 2: The VMware Multi-Cloud Strategy

Only VMware can drive the next generation of cloud, supporting ambitious multi-cloud strategies, for all application initiatives to deliver unprecedented business value. VMware App Modernization delivers the technology to build, run, and manage all these applications across any cloud, and the team to guide any organization's application modernization effort.

Figure 3: Portfolio of a multi-cloud journey

VMware offers the complete portfolio for the multi-cloud journey for any enterprise on any cloud. It provides a platform where both legacy and modern apps can co-exist and ubiquitously run across different cloud without re-platforming.

Kubernetes provides an ideal platform for Multi and Hybrid Cloud

Kubernetes provides the capability for container orchestration, while also facilitating an easy way to encapsulate applications. The Kubernetes management system provides a standardized mechanism for application delivery that is decoupled from the underlying infrastructure and can run in any cloud. All public & private cloud providers have adopted cloud-native technology and make it possible for running
applications in a standardized manner across a multi-cloud infrastructure. Modern developers can now leverage Kubernetes APIs in multi-cloud environments anywhere in the world to deploy their applications. Kubernetes has energized the software industry’s need for productivity, efficiency, by leveraging cloud-native technology available anywhere across public and private clouds.

Tanzu Kubernetes Grid (TKG)

Things get complex while running tens of thousands of containers across your enterprise at scale in production. Automation is mandatory for the deployment and management of all those containers on clusters of virtual or physical machines. Kubernetes, the industry-standard for container management, can streamline container orchestration to avoid the complexities of interdependent system architectures. However, there’s still considerable heavy lifting that an operations team must do to stand-up and manage a Kubernetes runtime consistently, while running in multiple data centers and clouds. They must also have the in-house expertise to design, deploy and integrate all the necessary components.

Figure 4: Tanzu Kubernetes Grid logical schematic

Tanzu Kubernetes Grid is engineered to simplify installation and Day 2 operations of Kubernetes across enterprises. It is tightly integrated with vSphere and can be extended to run with consistency across public cloud and edge environments. Tanzu Kubernetes Grid delivers multiple benefits to unlock the full potential of upstream Kubernetes and its burgeoning ecosystem of open-source cloud native technology through:

Simplified Installation

Tanzu Kubernetes Grid is engineered to include the tools and open-source technologies needed to deploy and consistently operate a scalable Kubernetes environment across VMware private cloud, public cloud, edge, or encompassing multiple clouds.
Automated multi-cluster operations
With declarative, multi-cluster lifecycle management, a CLI tool, and streamlined upgrades and patching, Tanzu Kubernetes Grid helps enterprises easily manage large-scale, multi-cluster Kubernetes deployments and automate manual tasks to reduce business risk and focus on more strategic work.

Integrated Platform Services
Tanzu Kubernetes Grid streamlines the deployment of local and in-cluster services to simplify the configuration of container image registry policies, monitoring, logging, ingress, networking & storage, and enables the Kubernetes environment for production workloads.

Open-Source Alignment
Containerized applications can be run on an upstream-aligned Kubernetes distribution and key open-source technologies like Cluster API, Fluentbit, and Contour, enabling portability and the support and innovation of the global Kubernetes community.

Where does Tanzu Kubernetes Grid run?

Private cloud
With Tanzu Kubernetes Grid Service integrated with vSphere, existing data center tooling and workflows can be leveraged to give developers on-demand access to conformant Kubernetes clusters in the private cloud and managing cluster lifecycle through automated, API-driven workflows.
Public cloud
Tanzu Mission Control can be used to enable development teams to quickly spin up managed Kubernetes clusters in their public cloud accounts, while operations maintain access to the control plane for security and customization.

Edge
Tanzu Kubernetes Grid’s open architecture enables lightweight deployments and streamlined multicluster operations in highly distributed edge environments, like retail remote site locations.

Tanzu Service Mesh
_Tanzu Service Mesh_ provides consistent connectivity and _security for microservices_ across all Kubernetes clusters and clouds in the most demanding multi-cluster and multi-cloud environments. Tanzu Service Mesh can be installed in _Tanzu Kubernetes Grid_ (TKG) clusters and third-party Kubernetes-conformant clusters. It can be used with clusters managed by Tanzu Mission Control (i.e., Tanzu-managed clusters) or clusters managed by other Kubernetes platforms and managed services.
What Makes Tanzu Service Mesh Different?

Beyond its multi-cloud focus, one of the other differentiating characteristics of Tanzu Service Mesh is its ability to support cross-cluster and cross-cloud use cases via Global Namespaces (GNS). A GNS abstracts an application from the underlying Kubernetes cluster namespaces and networking, allowing you to transcend infrastructure limitations and boundaries, and securely stretch applications across clusters and clouds. Global Namespaces allow you to have consistent traffic routing, application resiliency, and security policies for your applications across cloud siloes, regardless of where the applications are running.

By enabling and delivering true multi-cloud capabilities, GNS can offer improved agility, business continuity, visibility, and better security for your modern applications.

In addition to providing an abstraction for applications, GNS also provides strong isolation that can be used for multi-tenancy model for application teams and business units. Each of these groups can have as many GNSs as they need for their application. More about GNS can be found at “Using Global Namespaces to secure multi-cloud applications”.
Tanzu Service Mesh can also automate and simplify the installation and lifecycle management of the service mesh bits running inside your Kubernetes clusters, while maintaining intended configuration values. One can also “move” application services without having to change anything in the application itself, which brings the idea of multi-cloud or hybrid-cloud workloads to life. This cross-domain/cross-cloud communication requires additional security considerations, so GNS encrypts the traffic, end to end, between the services across clusters and clouds.

**Tanzu Mission Control:**
Tanzu Mission Control, now available through VMware Cloud Partner Navigator, is a centralized management platform for consistently operating and securing your Kubernetes infrastructure and modern applications across multiple teams and clouds. Tanzu Mission Control provides operators with a single control point to give developers the independence they need to drive business forward, while enabling consistent management and operations across environments for increased security and governance.

**VMware Tanzu Mission Control**

A centralized management platform for operating and securing your Kubernetes infrastructure and modern applications across teams and clouds.

- **Productivity**
  - Enables developers with self-service access to Kubernetes for running and deploying applications

- **Consistency**
  - Centrally operates and manages all your Kubernetes clusters and applications at scale

- **Security**
  - Manage security and compliance confidently and efficiently across your entire Kubernetes footprint
The infrastructure and platform teams use Tanzu Mission Control to enable developers with self-service access to Kubernetes. It also allows them to centrally operate and manage the Kubernetes clusters and modern apps running on them with efficiency, consistency, and security. Application teams use Tanzu Mission Control to better manage and maintain applications by easily deploying services and workloads across clusters, better understanding the health of their applications and quickly troubleshooting issues.

**Provider and Customer Challenges**
There are a variety of Kubernetes distributions out there. Managing access, policy, security and cost across isolated distributions can be a challenge. Tanzu Mission Control provides a centralized management platform, giving managed service providers and their users the independence, they need to drive business forward, while enabling consistent management and operations across environments at scale.

**Consistent operation across clusters and clouds**

Centralized control and governance

- Multi-Cluster Management
- Manage Kubernetes on vSphere 7
- Kubernetes Deployment at the Edge

*Figure 12: Tanzu Mission provides consistent operations across clouds*

**The Problem**
It is hard to consistently connect, control, monitor, and remediate cloud native apps. Moderns App are running in multiple platforms and clouds. There are multiple endpoints to monitor, scale, and make them resilient. Operational and remediation policies differ across clouds. Security, auditing and compliance are disoriented.

**Solution Components**
TKG allows use of existing data center tools and workflows to give developers secure, self-serve access to conformant Kubernetes clusters in their VMware private cloud and extend the same consistent Kubernetes runtime across their public cloud and edge environments. TKG can enable consistent Kubernetes everywhere with automated multi-cluster operations, validated integrated services and enterprise-wide management.

VMware Tanzu Mission Control gives teams self service capabilities to spin up their own Kubernetes clusters, while keeping track of all their services using workspaces. Workspaces work across clusters provides teams the flexibility they need to run their services, while conforming to organizational policies.
Workspaces also allow operations teams to assign policy in a hierarchical way at the global, cluster, and workspace level.

Tanzu Service mesh provides the ability to run applications across multi-cloud environments. It ensures application high availability and resiliency to deliver on application SLAs and ensure a positive experience for application users, while protecting sensitive data and ensuring compliance. It enables operational Control to deliver consistent and intelligent operations across cloud environments.

This solution seeks to combine the capabilities of TKG, Tanzu Mission Control and Tanzu Service Mesh to host an end to end secure and optimized multi-cloud application. Kubernetes is deployed distinct multi-cloud locations that include VMC on AWS and VMC on Dell EMC.

The Solution
This solution showcases a multi-cloud deployment of a distributed application leveraging Tanzu Kubernetes Grid. The multi-cloud TKG solution is deployed in a distributed fashion across two different cloud environments that includes a VMC on AWS SDDC in Oregon and VMC on Dell EMC SDDC in Santa Clara. Tanzu Mission Control and Tanzu service mesh described below are used to operationalize, secure and manage the environment.

Solution Architecture

The logical schematic of the solution is shown. TKG is deployed independently in two distinct multicloud locations that include a VMC on AWS SDDC and a VMC on Dell EMC Edge location. Tanzu Mission Control is used to manage these TKG clusters in a centralized manner as shown. Tanzu Service mesh is used to create a global namespace and provides for monitoring, automation, policy management and secure communications across the multi-cloud infrastructure. An example e-commerce application was deployed across the multicloud environment to showcase the capabilities of the solution.
Solution Configuration

VMware Tanzu Mission Control (TMC) is used to centrally manage Kubernetes clusters. The two Kubernetes clusters, one running in VMC on AWS and the other in VMC in Dell EMC are shown in the TMC console.
VMware Tanzu Mission Control provides insight into all aspects of the Kubernetes clusters it manages. It provides a graphical view of all the health metrics, the nodes, namespaces and workloads.
The VMC on Dell EMC Kubernetes cluster is shown below. The master node is identified as the control plane and the four worker nodes are shown below that.

![VMC on Dell EMC TKG Cluster nodes as seen in TMC](image1)

Figure 16: VMC on Dell EMC TKG Cluster nodes as seen in TMC

Tanzu Service Mesh (TSM) console is shown with all its components. The sample global namespace used by the multi-cloud web application is shown.

![Global Namespace for solution in Tanzu Service Mesh](image2)

Figure 17: Global Namespace for solution in Tanzu Service Mesh
The two clusters are combined into a global namespace as shown below with Tanzu Service Mesh. TSM secures and manages the communication across the clusters and the namespace.

![Tanzu Service Mesh home page with clusters combined into a global namespace.](image)

*Figure 18: VMC on AWS TKG Cluster overview in TMC*

Details about the sample service “sample.acme.com” and its details in TSM are shown. Details about the application security and certificates are shown.

![Tanzu Mission Control with sample service details.](image)

*Figure 19: Details of the Global Namespace and associated application service*
A snippet from the yaml file from acme fitness service is shown with details of the shopping application. Full version of the YAML file can be seen in Appendix A.

![Snippet of the ACME Fitness Application YAML file](image)

**Figure 20:** Snippet of the ACME Fitness Application YAML file

Shown is a listing of some of the commands that were run for the creation of a mongo dB database and other components of the acme fitness application.

![Commands showing creation of ACME fitness App service](image)

**Figure 21:** Commands showing creation of ACME fitness App service
Components of the acme application in the VMC on Dell EMC Kubernetes is shown in a flow chart format. TSM shows compelling views of the applications it manages and their relationship.

Figure 22: Components of the ACME app and their relationships as seen in TSM

The web interface of the multi-tiered application is shown here. The application components are deployed across two different Kubernetes clusters across a multi-cloud environment.

Figure 23: Web Interface of the ACME multi-tiered application
The catalog service is hosted in the VMC on AWS Kubernetes cluster while the other aspects of the solution and the mongo DB are hosted in the Kubernetes cluster running on the VMC on Dell EMC SDDC. The Tanzu Service Mesh makes it seamless for the application components to communicate with each other securely across clouds while showing a unified front to the users.

![Tanzu Service Mesh](image)

*Figure 24: Global Namespace showing the application components dispersed across multi-cloud*

The Kubernetes dashboard for each individual cluster helps monitor and maintain the different pods that make up the distributed web application.
Conclusion:

In summary we have shown that this solution can leverage TKG clusters deployed across a multi-cloud environment. Tanzu mission control provides visibility and operational capabilities to manage these Kubernetes clusters with a single pane of glass. Tanzu service mesh provides the ability to combine multi-cloud applications with global namespaces and secures the application effectively across cloud boundaries. All the components showcased in this multi-cloud solution are part of the **VMware Tanzu Advanced Edition**. VMware Tanzu can be effectively leveraged by enterprises to deploy and manage Kubernetes application across a multi-cloud Kubernetes based environment.
Appendix A: Fitness_Cluster.yaml

```yaml
---
apiVersion: v1
kind: Service
metadata:
  name: cart-redis
labels:
  app: cart-redis
  service: cart-redis
spec:
  ports:
    - port: 6379
      name: tcp-redis-cart
      selector:
        app: cart-redis
        service: cart-redis

---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: cart-redis
labels:
  app: cart-redis
  service: cart-redis
spec:
  replicas: 1
  template:
    metadata:
      labels:
        app: cart-redis
        service: cart-redis
    spec:
      containers:
        - name: cart-redis
          image: redis:5.0.3-alpine
          command: ["redis-server"]
          imagePullPolicy: Always
          resources:
            requests:
              cpu: "100m"
              memory: "100Mi"
          ports:
            - name: tcp-redis
              containerPort: 6379
              protocol: "TCP"
          env:
            - name: REDIS_HOST
              value: "cart-redis"
            - name: REDIS_PASS
              valueFrom:
                secretKeyRef:
                  name: redis-pass
                  key: password
            - name: REDIS_PORT
              value: "6379"
            - name: CART_PORT
              value: "5000"
          volumes:
            - name: cart-data
              emptyDir: {}
            - image: gcr.io/vmwarecloudadvocacy/acmeshop-cart:1.0.0
              name: cart
              env:
                - name: REDIS_HOST
                  value: "cart-redis"
                - name: REDIS_PASS
                  valueFrom:
                    secretKeyRef:
                      name: redis-pass
                      key: password
                - name: REDIS_PORT
                  value: "6379"
                - name: CART_PORT
                  value: "5000"
              ports:
                - containerPort: 5000
                  name: http-cart
              volumeMounts:
                - name: cart-data
                  mountPath: /data
                - image: gcr.io/vmwarecloudadvocacy/acmeshop-cart:1.0.0
                  name: cart
                  env:
                    - name: REDIS_HOST
                      value: "cart-redis"
                    - name: REDIS_PASS
                      valueFrom:
                        secretKeyRef:
                          name: redis-pass
                          key: password
                    - name: REDIS_PORT
                      value: "6379"
                    - name: CART_PORT
                      value: "5000"
              ports:
                - containerPort: 5000
                  name: http-cart
              volumeMounts:
                - name: cart-data
                  mountPath: /data
                - image: gcr.io/vmwarecloudadvocacy/acmeshop-cart:1.0.0
                  name: cart
                  env:
                    - name: REDIS_HOST
                      value: "cart-redis"
                    - name: REDIS_PASS
                      valueFrom:
                        secretKeyRef:
                          name: redis-pass
                          key: password
                    - name: REDIS_PORT
                      value: "6379"
                    - name: CART_PORT
                      value: "5000"
              ports:
                - containerPort: 5000
                  name: http-cart
              volumeMounts:
                - name: cart-data
                  mountPath: /data
                - image: gcr.io/vmwarecloudadvocacy/acmeshop-cart:1.0.0
                  name: cart
                  env:
                    - name: REDIS_HOST
                      value: "cart-redis"
                    - name: REDIS_PASS
                      valueFrom:
                        secretKeyRef:
                          name: redis-pass
                          key: password
                    - name: REDIS_PORT
                      value: "6379"
                    - name: CART_PORT
                      value: "5000"
              ports:
                - containerPort: 5000
                  name: http-cart
              volumeMounts:
                - name: cart-data
                  mountPath: /data
                - image: gcr.io/vmwarecloudadvocacy/acmeshop-cart:1.0.0
                  name: cart
                  env:
                    - name: REDIS_HOST
                      value: "cart-redis"
                    - name: REDIS_PASS
                      valueFrom:
                        secretKeyRef:
                          name: redis-pass
                          key: password
                    - name: REDIS_PORT
                      value: "6379"
                    - name: CART_PORT
                      value: "5000"
              ports:
                - containerPort: 5000
                  name: http-cart
              volumeMounts:
                - name: cart-data
                  mountPath: /data
                - image: gcr.io/vmwarecloudadvocacy/acmeshop-cart:1.0.0
                  name: cart
                  env:
                    - name: REDIS_HOST
                      value: "cart-redis"
                    - name: REDIS_PASS
                      valueFrom:
                        secretKeyRef:
                          name: redis-pass
                          key: password
                    - name: REDIS_PORT
                      value: "6379"
                    - name: CART_PORT
                      value: "5000"
              ports:
                - containerPort: 5000
                  name: http-cart
              volumeMounts:
                - name: cart-data
                  mountPath: /data
                - image: gcr.io/vmwarecloudadvocacy/acmeshop-cart:1.0.0
                  name: cart
                  env:
                    - name: REDIS_HOST
                      value: "cart-redis"
                    - name: REDIS_PASS
                      valueFrom:
                        secretKeyRef:
                          name: redis-pass
                          key: password
                    - name: REDIS_PORT
                      value: "6379"
                    - name: CART_PORT
                      value: "5000"
              ports:
                - containerPort: 5000
                  name: http-cart
              volumeMounts:
                - name: cart-data
                  mountPath: /data
                - image: gcr.io/vmwarecloudadvocacy/acmeshop-cart:1.0.0
                  name: cart
                  env:
                    - name: REDIS_HOST
                      value: "cart-redis"
                    - name: REDIS_PASS
                      valueFrom:
                        secretKeyRef:
                          name: redis-pass
                          key: password
                    - name: REDIS_PORT
                      value: "6379"
                    - name: CART_PORT
                      value: "5000"
              ports:
                - containerPort: 5000
                  name: http-cart
              volumeMounts:
                - name: cart-data
                  mountPath: /data
                - image: gcr.io/vmwarecloudadvocacy/acmeshop-cart:1.0.0
                  name: cart
                  env:
                    - name: REDIS_HOST
                      value: "cart-redis"
                    - name: REDIS_PASS
                      valueFrom:
                        secretKeyRef:
                          name: redis-pass
                          key: password
                    - name: REDIS_PORT
                      value: "6379"
                    - name: CART_PORT
                      value: "5000"
              ports:
                - containerPort: 5000
                  name: http-cart
              volumeMounts:
                - name: cart-data
                  mountPath: /data
                - image: gcr.io/vmwarecloudadvocacy/acmeshop-cart:1.0.0
                  name: cart
                  env:
                    - name: REDIS_HOST
                      value: "cart-redis"
                    - name: REDIS_PASS
                      valueFrom:
                        secretKeyRef:
                          name: redis-pass
                          key: password
                    - name: REDIS_PORT
                      value: "6379"
                    - name: CART_PORT
                      value: "5000"
              ports:
                - containerPort: 5000
                  name: http-cart
              volumeMounts:
                - name: cart-data
                  mountPath: /data
                - image: gcr.io/vmwarecloudadvocacy/acmeshop-cart:1.0.0
                  name: cart
                  env:
                    - name: REDIS_HOST
                      value: "cart-redis"
                    - name: REDIS_PASS
                      valueFrom:
                        secretKeyRef:
                          name: redis-pass
                          key: password
                    - name: REDIS_PORT
                      value: "6379"
                    - name: CART_PORT
                      value: "5000"
              ports:
                - containerPort: 5000
                  name: http-cart
              volumeMounts:
                - name: cart-data
                  mountPath: /data
                - image: gcr.io/vmwarecloudadvocacy/acmeshop-cart:1.0.0
                  name: cart
                  env:
                    - name: REDIS_HOST
                      value: "cart-redis"
                    - name: REDIS_PASS
                      valueFrom:
                        secretKeyRef:
                          name: redis-pass
                          key: password
                    - name: REDIS_PORT
                      value: "6379"
                    - name: CART_PORT
                      value: "5000"
              ports:
                - containerPort: 5000
                  name: http-cart
              volumeMounts:
                - name: cart-data
                  mountPath: /data
                - image: gcr.io/vmwarecloudadvocacy/acmeshop-cart:1.0.0
                  name: cart
                  env:
                    - name: REDIS_HOST
                      value: "cart-redis"
                    - name: REDIS_PASS
                      valueFrom:
                        secretKeyRef:
                          name: redis-pass
                          key: password
                    - name: REDIS_PORT
                      value: "6379"
                    - name: CART_PORT
                      value: "5000"
              ports:
                - containerPort: 5000
                  name: http-cart
              volumeMounts:
                - name: cart-data
                  mountPath: /data
                - image: gcr.io/vmwarecloudadvocacy/acmeshop-cart:1.0.0
                  name: cart
                  env:
                    - name: REDIS_HOST
                      value: "cart-redis"
                    - name: REDIS_PASS
                      valueFrom:
                        secretKeyRef:
                          name: redis-pass
                          key: password
                    - name: REDIS_PORT
                      value: "6379"
                    - name: CART_PORT
                      value: "5000"
              ports:
                - containerPort: 5000
                  name: http-cart
              volumeMounts:
                - name: cart-data
                  mountPath: /data
                - image: gcr.io/vmwarecloudadvocacy/acmeshop-cart:1.0.0
                  name: cart
                  env:
                    - name: REDIS_HOST
                      value: "cart-redis"
                    - name: REDIS_PASS
                      valueFrom:
                        secretKeyRef:
                          name: redis-pass
                          key: password
                    - name: REDIS_PORT
                      value: "6379"
                    - name: CART_PORT
                      value: "5000"
              ports:
                - containerPort: 5000
                  name: http-cart
              volumeMounts:
                - name: cart-data
                  mountPath: /data
                - image: gcr.io/vmwarecloudadvocacy/acmeshop-cart:1.0.0
                  name: cart
                  env:
                    - name: REDIS_HOST
                      value: "cart-redis"
                    - name: REDIS_PASS
                      valueFrom:
                        secretKeyRef:
                          name: redis-pass
                          key: password
                    - name: REDIS_PORT
                      value: "6379"
                    - name: CART_PORT
                      value: "5000"
```
service: shopping
spec:
ports:
- name: http-shopping
  protocol: TCP
  port: 3000
selector:
  app: shopping
service: shopping

apiVersion: apps/v1 # for versions before 1.8.0 use apps/v1beta1
kind: Deployment
metadata:
  name: shopping
labels:
  app: shopping
service: shopping
spec:
  selector:
    matchLabels:
      app: shopping
      service: shopping
  strategy:
    type: Recreate
  replicas: 1
  template:
    metadata:
      labels:
        app: shopping
        service: shopping
    spec:
      containers:
        - image: gcr.io/vmwarecloudadvocacy/acmeshop-front-end:rel1
          name: shopping
          env:
            - name: FRONTEND_PORT
              value: '3000'
            - name: USERS_HOST
              value: 'users'
            - name: CATALOG_HOST
              value: 'catalog.acme.com'
            - name: ORDER_HOST
              value: 'order'
            - name: CART_HOST
              value: 'cart'
            - name: USERS_PORT
              value: '8081'
            - name: CATALOG_PORT
              value: '8082'
            - name: CART_PORT
              value: '5000'
            - name: ORDER_PORT
              value: '6000'
          ports:
            - containerPort: 3000
              name: http-shopping
            ---
        apiVersion: v1
        kind: Service
        metadata:
          name: shopping
        spec:
          ports:
            - name: http-shopping
              protocol: TCP
              port: 3000
          selector:
            app: shopping
            service: shopping

apiVersion: apps/v1 # for versions before 1.8.0 use apps/v1beta1
kind: Deployment
metadata:
  name: order
labels:
  app: order
service: order
spec:
  selector:
    matchLabels:
      app: order
      service: order
  strategy:
    type: Recreate
  replicas: 1
  template:
    metadata:
      labels:
        app: order
        service: order
    spec:
      containers:
        - image: mongo:4
          name: order-mongo
          resources: {}
          ports:
            - containerPort: 27017
              name: mongo
          protocol: "TCP"
          env:
            - name: MONGO_INITDB_ROOT_USERNAME
              value: 'mongoadmin'
            - name: MONGO_INITDB_ROOT_PASSWORD
              valueFrom:
                secretKeyRef:
                  name: order-mongo-pass
                  key: password

          volumeMounts:
            - mountPath: /data/db
              name: mongodata
          volumes:
            - name: mongodata
              emptyDir: {}
              # persistentVolumeClaim:
              #   claimName: mongodata

---

apiVersion: v1
kind: Service
metadata:
  name: order
labels:
  app: order
service: order
spec:
  ports:
    - name: http-order
      protocol: TCP
      port: 6000
      selector:
        app: order
        service: order
  ---

apiVersion: apps/v1 # for versions before 1.8.0 use apps/v1beta1
kind: Deployment
metadata:
  name: order-mongo
labels:
  app: order-mongo
service: order-mongo
spec:
  selector:
    matchLabels:
      app: order-mongo
      service: order-mongo
  strategy:
    type: Recreate
  replicas: 1
  template:
    metadata:
apiVersion: v1
kind: Deployment
metadata:
  name: order
  labels:
    app: order
    service: order
spec:
  selector:
    matchLabels:
      app: order
      service: order
  strategy:
    type: Recreate
  replicas: 1
  template:
    metadata:
      labels:
    spec:
      containers:
        - image: gcr.io/vmwarecloudadvocacy/acmeshop
          name: order
          ports:
            - containerPort: 6000
              name: http
              protocol: TCP
          env:
            - name: ORDER_DB_HOST
              value: 'order
              mongo'
            - name: ORDER_DB_PASSWORD
              valueFrom:
                secretKeyRef:
                  name: order
                  key: password
            - name: ORDER_DB_PORT
              value: '9000'
            - name: ORDER_DB_USERNAME
              value: 'mongoadmin'
            - name: ORDER_PORT
              value: '6000'
            - name: PAYMENT_PORT
              value: '9000'
            - name: PAYMENT_HOST
              value: 'payment'
            - name: PAYMENT_HOST
              value: 'payment'
            - name: PAYMENT_PORT
              value: '9000'
            - name: PAYMENT_HOST
              value: 'payment'
            - name: PAYMENT_PORT
              value: '9000'
          volumeMounts:
            - mountPath: /data
              name: order-data
          resources:
            requests:
              memory: "64Mi"
              cpu: "100m"
            limits:
              memory: "256Mi"
              cpu: "500m"
          ---
          apiVersion: v1
          kind: ConfigMap
          metadata:
            name: users-initdb-config
          data:
            seed.js:
              |
              db users.insertMany([{
                "firstname": "Walter", "lastname": "White", "email": "walter@acmefitness.com", "username": "walter", "password": "6837ea9b06409112a824d113927ad74fabc5c76e", "salt": ""}
              ,{
                "firstname": "Dwight", "lastname": "Schrute", "email": "dwright@acmefitness.com", "username": "dwright", "password": "6837ea9b06409112a824d113927ad74fabc5c76e", "salt": ""}
              ,{
                "firstname": "Eric", "lastname": "Cartman", "email": "eric@acmefitness.com", "username": "eric", "password": "6837ea9b06409112a824d113927ad74fabc5c76e", "salt": ""}
              ,{
                "firstname": "Han", "lastname": "Solo", "email": "han@acmefitness.com", "username": "han", "password": "6837ea9b06409112a824d113927ad74fabc5c76e", "salt": ""}
              ,{
                "firstname": "Phoebe", "lastname": "Buffay", "email": "phoebe@acmefitness.com", "username": "phoebe", "password": "6837ea9b06409112a824d113927ad74fabc5c76e", "salt": ""}
              ,{
                "firstname": "Elaine", "lastname": "Benes", "email": "elaine@acmefitness.com", "username": "elaine", "password": "6837ea9b06409112a824d113927ad74fabc5c76e", "salt": ""}
              ]);
          ---
          apiVersion: v1
          kind: Service
          metadata:
            name: users-mongo
            labels:
              app: users-mongo
              service: users-mongo
          spec:
            ports:
              - port: 27017
                name: mongo
                protocol: TCP
            selector:
              app: users-mongo
              service: users-mongo
          ---
          apiVersion: apps/v1
          kind: Deployment
          metadata:
            name: users-mongo
            labels:
              app: users-mongo
              service: users-db
          spec:
            selector:
              matchLabels:
                app: users-mongo
                service: users-db
            replicas: 1
            template:
              metadata:
                labels:

- name: users-mongo
  image: mongo:4
  resources: {}
  ports:
  - name: mongo-users
    containerPort: 27017
    protocol: "TCP"
  env:
  - name: MONGO_INITDB_ROOT_USERNAME
    value: 'mongoadmin'
  - name: MONGO_INITDB_DATABASE
    value: 'acmefit'
  - name: MONGO_INITDB_ROOT_PASSWORD
    valueFrom:
      secretKeyRef:
        name: users-mongo-pass
        key: password
  volumeMounts:
  - mountPath: /data/db
    name: mongodata
  - mountPath: /docker-entrypoint-initdb.d
    name: mongo-initdb
  volumes:
  - name: mongodata
  - name: mongo-initdb
    configMap:
      name: users-initdb-config
      # persistentVolumeClaim:
      # claimName: mongodata

---

apiVersion: v1
kind: Service
metadata:
  name: users
  labels:
    app: users
    service: users
spec:
  ports:
  - name: http-users
    protocol: TCP
    port: 8081
  selector:
    app: users
    service: users

---

apiVersion: apps/v1
kind: Deployment
metadata:
  name: users
  labels:
    app: users
    service: users
spec:
  selector:
    matchLabels:
      app: users
  strategy:
    type: Recreate
    replicas: 1
  template:
    metadata:
      labels:
        app: users
        service: users
    spec:
      volumes:
      - name: users-data
        emptyDir: {}
      containers:
      - image: gcr.io/vmwarecloudadvocacy/acmeproxy:1.0.0
        name: users
        env:
        - name: USERS_DB_HOST
          value: 'users-mongo'
        - name: USERS_DB_PASSWORD
          valueFrom:
            secretKeyRef:
              name: users-mongo-pass
              key: password
        - name: USERS_DB_PORT
          value: '27017'
        - name: USERS_DB_USERNAME
          value: 'mongoadmin'
        - name: USERS_PORT
          value: '8081'
        ports:
        - containerPort: 8081
          name: http-users
          volumeMounts:
            - mountPath: "/data"
              name: users-data
              resources:
                requests:
                  memory: "64Mi"
                  cpu: "100m"
                limits:
                  memory: "256Mi"
                  cpu: "500m"
---
Appendix B: Catalog YAML file:

```yaml
apiVersion: v1
template:
  metadata:
    labels:
      name: catalog
  spec:
    replicas: 1
    selector:
      matchLabels:
        name: catalog
    containers:
      - name: catalog
        image: acmefit
        ports:
        - containerPort: 8080
        - containerPort: 1025
        - containerPort: 80
        imagePullPolicy: Always
        livenessProbe:
          httpGet:
            path: /health
            port: 8080
        readinessProbe:
          httpGet:
            path: /ready
            port: 8080
        resources:
          limits:
            cpu: 1
            memory: 1Gi
          requests:
            cpu: 0.5
            memory: 512Mi
    volumes:
      - name: myvolume
        hostPath: /data
```
service: catalog
template:

spec:
  selector:
    matchLabels:
      app: catalog
      service: catalog
      version: v2

strategy:
  type: Recreate
  replicas: 1

metadata:
  labels:
    app: catalog
    service: catalog
    version: v2

spec:
  volumes:
    - name: catalog-data
      emptyDir: {}
  containers:
    - image: gcr.io/vmwarecloudadvocacy/acmeshop-catalog:1.0.0
      name: catalog
      env:
        - name: CATALOG_DB_HOST
          value: 'catalog-mongo-v2'
        - name: CATALOG_DB_PASSWORD
          valueFrom:
            secretKeyRef:
              name: catalog-mongo-pass
              key: password
        - name: CATALOG_DB_PORT
          value: '27017'
        - name: CATALOG_DB_USERNAME
          value: 'mongoadmin'
        - name: CATALOG_PORT
          value: '8082'
      ports:
        - containerPort: 8082
          name: http-catalog
      volumeMounts:
        - mountPath: "/data"
          name: "catalog-data"
      resources:
        requests:
          memory: "64Mi"
          cpu: "100m"
      limits:
        memory: "256Mi"
        cpu: "500m"
Appendix C: Load Generator

---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: acme-locust
  namespace: loadgen
spec:
  selector:
    matchLabels:
      app: acme-locust
  replicas: 4
  template:
    metadata:
      labels:
        app: acme-locust
    spec:
      containers:
      - name: main
        image: harbor.tanzuworld.com/apps/locust:1.4.1
        imagePullPolicy: IfNotPresent
        ports:
        - containerPort: 8089
        env:
        - name: LOCUSTFILE
          value: /mnt/locust/locustfile.py
        - name: LOCUSTFILE
          value: /mnt/locust/locustfile.py
        command:
        - locust
        args:
          -f
          - /mnt/locust/locustfile.py
          - --headless
          - --host=http://istio-ingressgateway.istio-system
        resources:
          requests:
            cpu: 300m
            memory: 256Mi
          limits:
            cpu: 1
            memory: 512Mi
        volumeMounts:
        - name: acme-locustfile
          mountPath: /mnt/locust
---
apiVersion: v1
kind: ConfigMap
metadata:
  name: acme-locustfile
  namespace: loadgen
data:
  locustfile.py:
  # This program will generate traffic for ACME Fitness Shop App. It simulates both Authenticated and Guest user scenarios. You can run this program either from Command line or from the web based UI. Refer to the "locust" documentation for further information.
from locust import HttpUser, TaskSet, task, SequentialTaskSet, Locust, LoadTestShape, between
import random
import math
# List of users (pre-loaded into ACME Fitness shop)
users = ["eric", "phoebe", "dwiht", "han"]
# List of products within the catalog
products = []
import logging
# GuestUserBrowsing simulates traffic for a Guest User (Not logged in)
class GuestUserBrowsing(SequentialTaskSet):
    def on_start(self):
        self.getProducts()
    def listCatalogItems(self):
        items = self.client.get("/products").json()[
            "data"
        ]
        for item in items:
            products.append(item["id"])
def listCatalogItems(self):
    items = self.client.get("/products").json()["data"]
    for item in items:
        products.append(item["id"])
    return products

@task(2)
def index(self):
    self.client.get("/")

class UserBehavior(SequentialTaskSet):
    tasks = [AuthUserBrowsing, GuestUserBrowsing]

class WebSiteUser(HttpUser):
    tasks = [UserBehavior]
    userid = ""
    #min_wait = 2000
    #max_wait = 10000
    wait_time = between(0.5, 3)

class StagesShape(LoadTestShape):
    
    A simply load test shape class that has different user and spawn_rate at
different stages.
    Keyword arguments:
    stages -- A list of dicts, each representing a stage with the following
    keys:
        duration -- When this many seconds pass the test is advanced to the
        next stage
        users -- Total user count
        spawn_rate -- Number of users to start/stop per second
        stop -- A boolean that can stop that test at a specific stage
        stop_at_end -- Can be set to stop once all stages have run.

    total_runtime = 1200
    stages = [
        {"duration": 300, "users": 100, "spawn_rate": 5},
        {"duration": 450, "users": 150, "spawn_rate": 5},
        {"duration": 600, "users": 600, "spawn_rate": 50},
        {"duration": 750, "users": 400, "spawn_rate": 5},
        {"duration": 900, "users": 300, "spawn_rate": 1},
        {"duration": 1050, "users": 100, "spawn_rate": 5},
        {"duration": 1200, "users": 50, "spawn_rate": 1},
    ]

def tick(self):
    run_time = self.get_run_time() % self.total_runtime

    for stage in self.stages:
        if run_time < stage["duration"]:
            tick_data = (stage["users"], stage["spawn_rate"])  
            return tick_data

    return None