Apigee Micro Gateways on Kubernetes Pilot Demo

Updated Date: 2/15/2018

Our Understanding

Equifax is adopting microservices for their enterprise. As they embark on this road, they would like to prove out their architecture for microservices. To this end Equifax has proposed a target state architecture and would like to validate the following

- Architecture is cloud agnostic
- Architecture is portable
- Architecture supports future goal of adopting istio
- Architecture is scalable

Equifax would like to implement a pilot to help them validate the above

Pilot Details

Scope :

- Create 3 node Kubernetes cluster on AWS. Cluster should have 1 master and 2 worker nodes
- Create 3 node Kubernetes cluster on Azure. Cluster should have 1 master and 2 worker nodes
- Create and Configure ingress controllers on both AWS and Azure
- Create a total of 4 different work spaces
- Build and configure Apigee micro gateway (AMG) container capable of being initialized with environment specific information
- Create and deploy service composed of AMG and helloworld microservices as shown in the architecture to the right in parallel on AWS and Azure
- Demonstrate that the architecture is scalable, portable and cloud agnostic

Deliverables:

1. Kubernetes (k8s) Cluster on AWS with 1 master and 2 worker nodes with ingress controllers
2. Kubernetes Cluster on Azure with 1 master and 2 worker nodes with ingress controllers
3. Jenkins Server with Two (2) Jenkins Jobs to deploy Service to AWS and Azure k8s
4. Git repo to build a AMG image capable of taking environment information as a runtime variable
5. Demo showing the following
   1. Solution is cloud agnostic
   2. Solution is scalable
6. Documentation of Steps to repeat items #1, #2 and #3

Pilot Solution

2 Options were considered as a part of this solution. Edge Micro gateway as a sidecar vs Kubernetes Service

- Option A: Micro Gateway as a Sidecar
  - Pros
    - Autonomy for developers to customize micro gateways
  - Cons
    - Micro gateway not a natural fit
    - Heavy container for a sidecar
    - Increases the complexity of Upgrade process
    - Does not use the networking features of K8s for routing
    - Tight coupling with individual micro services
    - Increases the compute requirements for K8S clusters

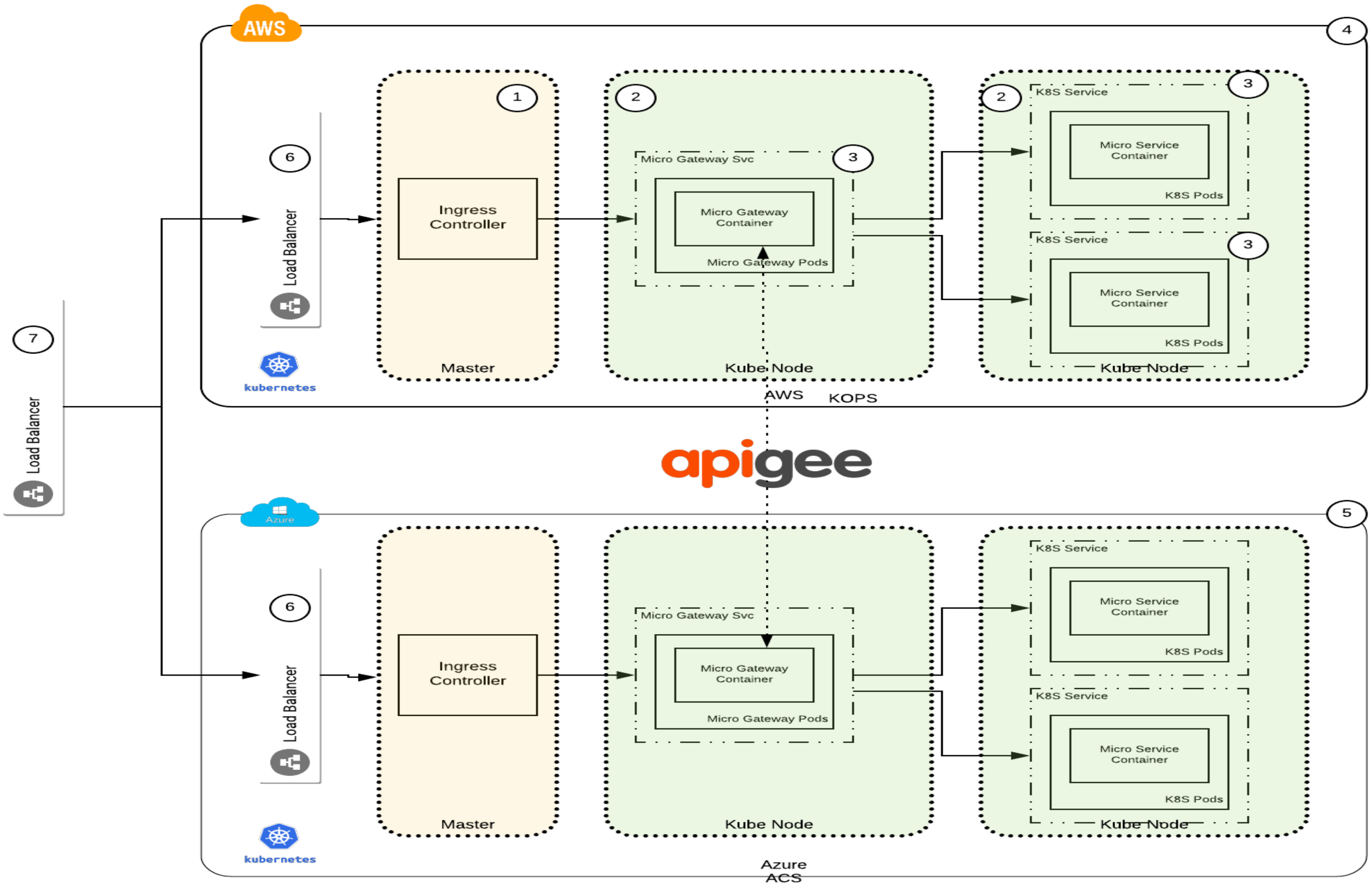
- Option B: Micro Gateway as a Kubernetes Service
  - Pros
    - Micro gateway a natural fit
    - Simpler Upgrade with rolling deployments
    - Maximizes the full use of K8S internal networking and service identification
    - Micro Services do not need to be micro gateway aware
    - Optimized the compute requirements for use of Micro Gateway
  - Cons
    - Limits autonomy of developers to customize edge micro gateway

Option B was selected and implemented

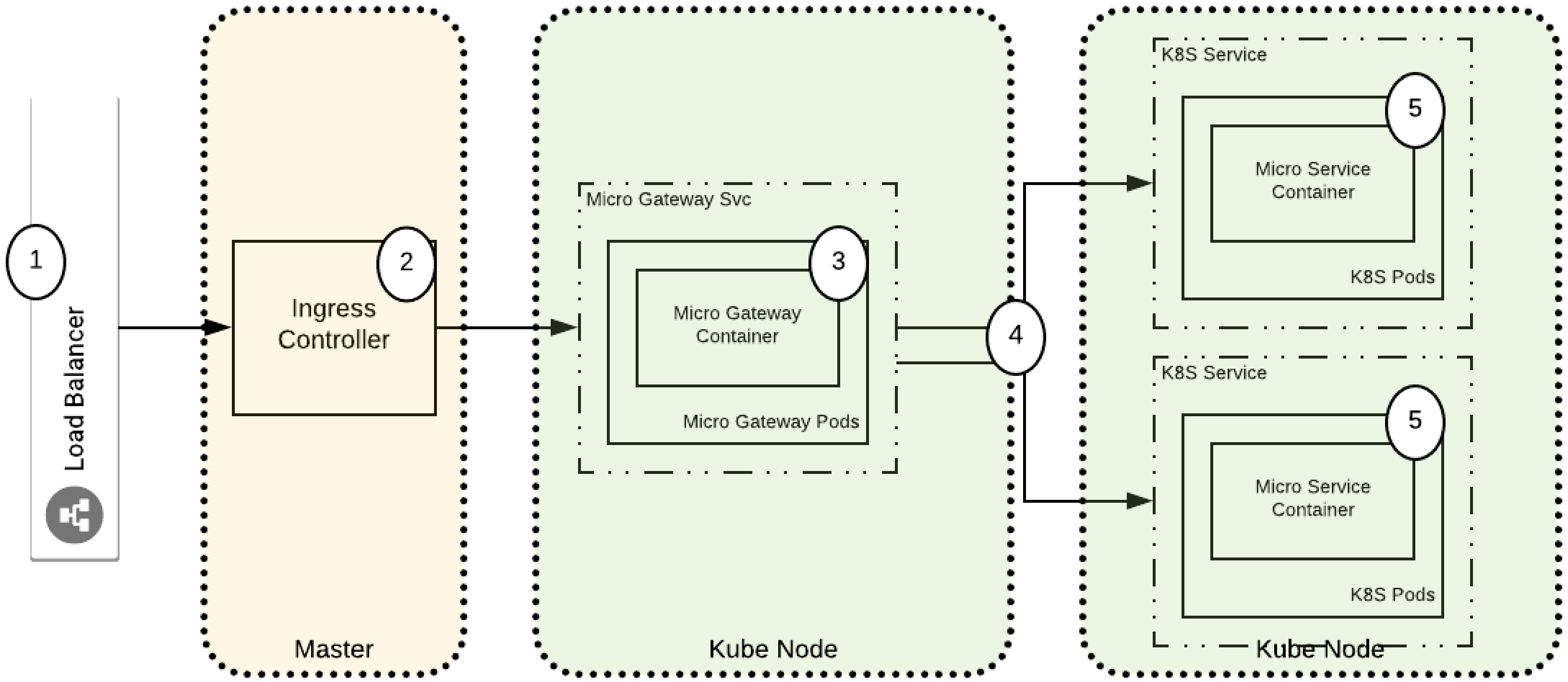
Platform Solution

- Cloud agnostic, runs on AWS and Azure
- Supports future goal of adopting istio
- Scalable to other Clouds like GCP

1. K8S Ingress
2. Kube Node
3. Edge Micro Gateway Container and Backend Services
4. K8S on AWS
5. K8S on AZURE



Option B- Scalability



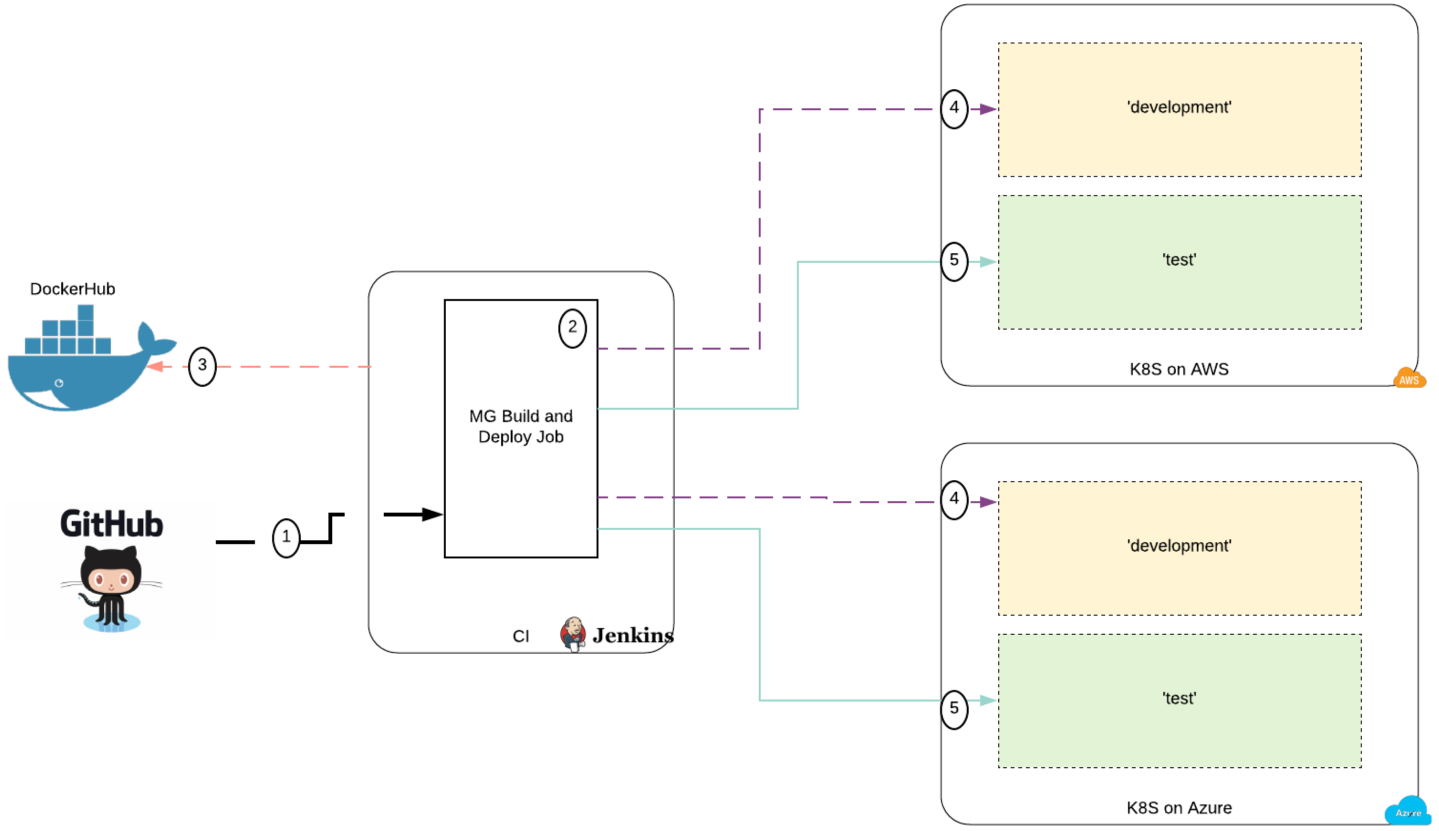
1. This can be ELB or F5 and can be scaled
2. The request is received by a Ingress Controller. This is a ELB for Azure Load balancer that are capable of auto scaling
3. The Micro Gateway Service is aware of numerous MG Pods that are running MG Containers . Deployments can be managed using K8S manifests to have multiple pods load balanced across a single K8S service providing scalability
4. K8S manages the DNS for the PODS and provides auto Service discovery and in-cluster routing
5. The micro service receives the request and provides an appropriate response back to the MG Container . Each of these micro services are access using a K8S Service vs direct POD Access. This allows of auto scaling PODS as needed

* Architecture is scalable

All points along this flow are individually Scalable

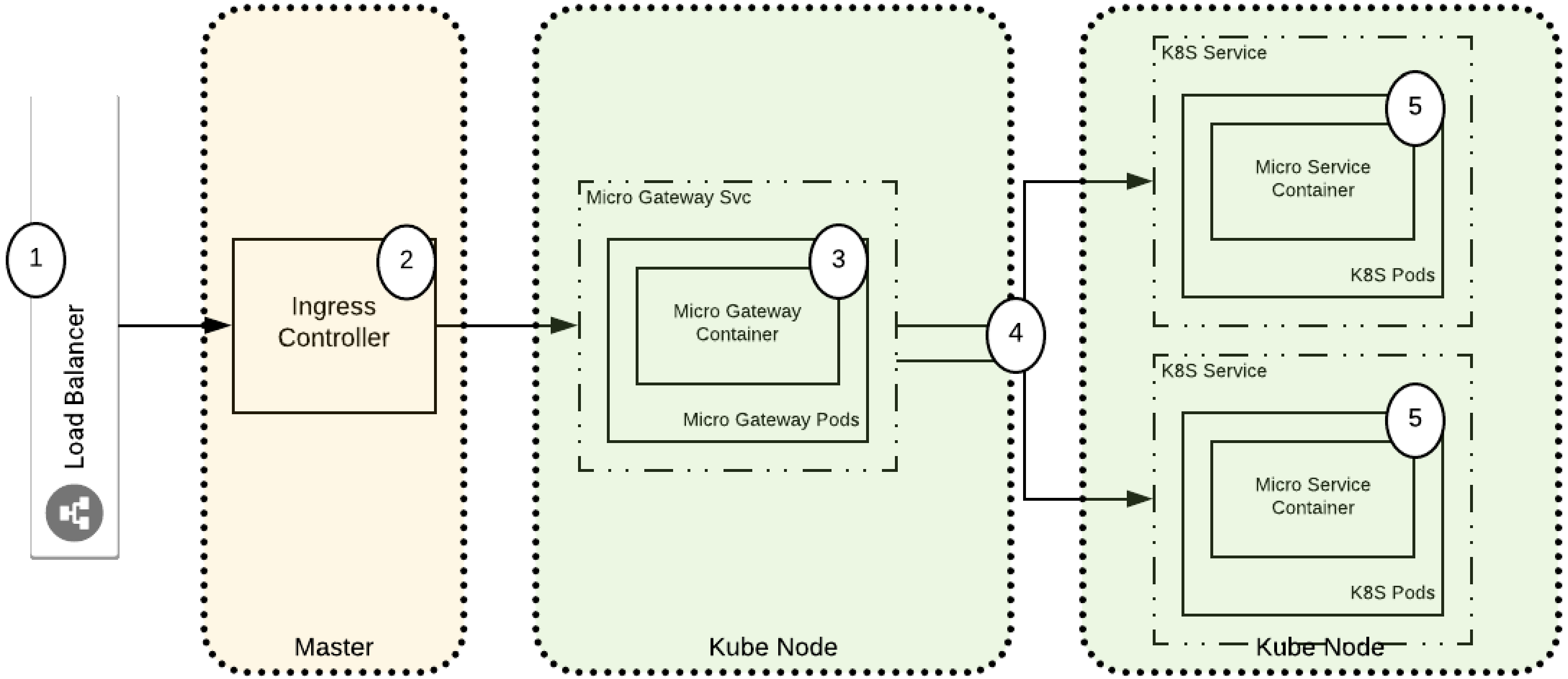
Build and Release Model

1. Developers make changes to Code in a common SCM system irrespective of final deployment location
2. A Job Triggered via a SCM commit or Manual Trigger that initiates a build and deploy
3. Application is complied, tested and a container image is built for deployment
4. Application specific pods and service is deployed to 'development' name space simultaneously on AWS and Azure Clouds Assumption : Configuration and secrets needed for the application are already present by this in in the namespace
5. Application specific pods and service is deployed to 'test' name space simultaneously on AWS and Azure Clouds Note : This usually happens after the application is tested and validated in 'dev'



* Architecture is portable

Option B- Ingress

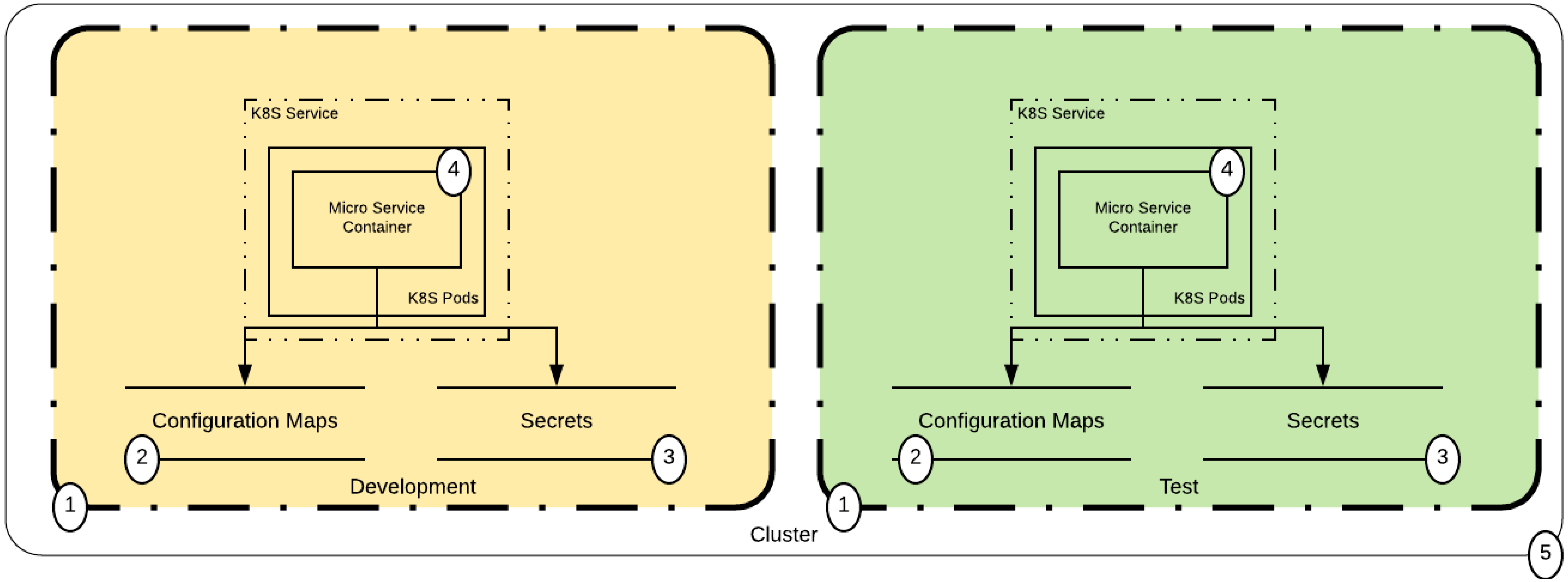


* Architecture is scalable

All points along this flow are individually Scalable

1. All requests to a services running inside of K8S are sent via a central load balancer. This can be ELB or F5
2. 2. The request is received by a Ingress Controller, and routed to the Micro Gateway Svc
3. The Micro Gateway Service is aware of numerous MG Pods that are running MG Containers . The Request is sent to one of the MG containers inside a MG POD
4. The MG Container receives the request, based on the target service associated to the request the MG proxies the request to the respective target backend
   Note: K8S DNS has been used to route to the target backend micro service running inside the cluster . For example the contact service is deployed to run with the name contact on port 8080. This is addressed inside the cluster as http://contact:8080
5. The micro service receives the request and provides an appropriate response back to the MG Container which in-turn sends the response back via the ingress

Managing Configuration



1. Kubernetes supports the logical separation of its cluster using namespaces. Namespaces are akin to SDLC environments which are capable of running the same application with differentiated sets of configuration
2. Config Maps: Allow of developers and administrators to define namespace specific properties that are available to applications running the name space
3. Secrets: Allows for sensitive information such as tokens, credentials etc to be stored in namespaces in a obfuscated manner and be supplied to the containers running inside the namespace
4. Represents the edge micro gateway . In the Development Name space it is configured to connect to the 'dev-qa' environment on Apigee. The configuration and credential information come from Configuration Map and Secrets and provided to the container using deployment.yml file
5. A Kubernetes cluster on AWS and/or Azure that is hosting the development and Test name spaces

Demo

- Deploy a Micro Gateway on AWS and AZURE using common manifest
- Deploy Micro Service on AWS and AZURE using common manifest
- Observe Scaling and Load balancing on both clusters
- Create 1 proxy for both Micro gateways with a cloud and cluster agnostic target
- Observe Cloud agnostic Routing

Thank You

Digital Transformation Experts

Ajay Naidu	Sunil Kumar	Raghu Gonihalli
ajayn@sidgs.com	*sunilb@sidgs.com*	raghug@sidgs.com
215-805-4478	*203-809-2378*	949-680-9734

SIDGS Intro

Technology Services and Solutions

Selenium
PostgreSQL
spring by Pivotal
CHEF
iOS
ANGULAR
puppet
Drupal
chai
mocha
ANSIBLE
TERRAFORM
React
SALTSTACK
Cucumber
Java
php $_SESSION
apigee
Microsoft .NET
JASMINE
cassandra
node JS
ZooKeeper
Microsoft SQL Server
amazon web services
docker
kubernetes
hadoop
SharePoint

Company Profile

Full Stack Digital Experts

Our Expertise	Global Delivery
Agile Development	*Philadelphia*
Digital Transformation	*Chicago*
API Development	*Virginia*
BPM	*Florida*
DevOps/Cloud	*Hyderabad*
UX/UI	*Vizag*
Big Data/BI	*Bangalore*
Mobile Development	