ISTIO DEPLOYMENT ON EKS

Version	Description	Revision date	Amendment/ Modification/ Deletion	Reviewer Name	Approver Name
1.0	Initial Version	26th, October 2023	Amendment	Ashish Kapoor Kiran Akshaya	Himanshu

Introduction:

The purpose of this runbook is to provide a detailed guide for deploying Istio on Amazon EKS, a managed Kubernetes service provided by AWS.

Prerequisites:

Before you begin the Istio deployment process, make sure you have the following prerequisites:

- AWS CLI and kubectl installed
- Helm and Istio CLI (Istioctl) installed.
- Amazon EKS cluster

Steps-by-step deployment

DOWNLOAD ISTIO

- Using the curl -L command, download all of the essential istio binary files
```
curl -L https://istio.io/downloadIstio | sh -
```

- Cd istio-1.11.1 in the terminal (will change depending on the version) Run the command export to set the path for the istio binary.
```
 export PATH=$PWD/bin:$PATH
```

INSTALL ISTIO

- Using istioctl to install Istio on a Kubernetes cluster here. Backend istioctl is a go template based on the helm.
```
istioctl install --set profile=demo -y
```

- Add a namespace label to instruct Istio to automatically inject Envoy sidecar proxies when you deploy your application later:
```
 kubectl label namespace default istio-injection=enabled
```

DEPLOY SAMPLE BOOK APPLICATION

- Deploy the book sample application:

```
kubectl apply -f samples/bookinfo/platform/kube/bookinfo.yaml
```

OPEN THE APPLICATION TO OUTSIDE TRAFFIC

INGRESS

The Istio will expose to a load balancer, which is a standard load balancer in the case of AWS it's a Classic Load balancer, and instead of building an ALB, we are modifying it to expose it to clusterIP so we can use ALB here.

Before building an AWS load balancer, the next step is to create an AWS ingress controller

Refer - AWS ingress controller

Next is the yaml file for creating aws alb which transfers all the incoming traffic to the Istio ingress gateway

Please refer and make the appropriate changes depending on your env

Ingress.yaml - Ingress resource is used to configure an AWS Application Load Balancer (ALB) to route traffic to the "istio-ingress gateway" service in the "istio-system" namespace, perform health checks, and handle SSL termination and HTTP to HTTPS redirection. The Ingress is associated with a specific hostname and listens on both HTTP and HTTPS ports (80 and 443, respectively).

```
        Kubectl apply -f Ingress.yaml
```

The new alb is formed and in route 53 update the domain to use the newly created alb.

GATEWAY

Gateway config - Istio Gateway configuration defines a route for incoming HTTPS traffic on port 443 to the Istio Ingress Gateway. The traffic is accepted for any host ("*"), and it terminates TLS encryption using a certificate and private key stored in a Kubernetes Secret named "bookinfo-tls." The Gateway is named "book-gateway" and is located in the "istio-system" namespace, which is a common namespace for Istio control plane components.

```
Kubectl apply -f gateway.yaml
```

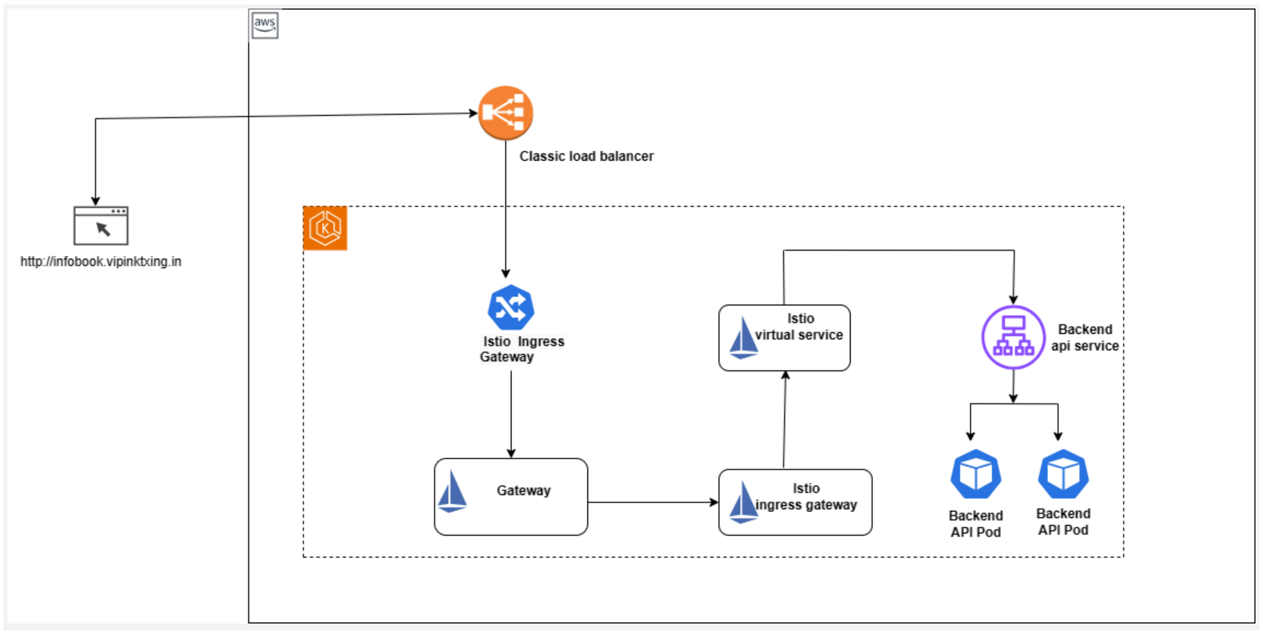
The request will be received by the AWS load balancer, which will then transmit it to istio-ingress-gateway .

VIRTUAL SERVICE

Virtual service config - VirtualService defines routing rules for various URIs, directing traffic to the "product page" service within the Kubernetes cluster. It's important to ensure that the "product page" service and the gateway "book gateway" are correctly set up and that Istio is properly configured to make this routing effective. Additionally, make sure this VirtualService is applied within the "istio-system" namespace where Istio is managing the service.

```
Kubectl apply -f virutalservice.yaml
```

TRAFFIC FLOW WITHOUT MTLS

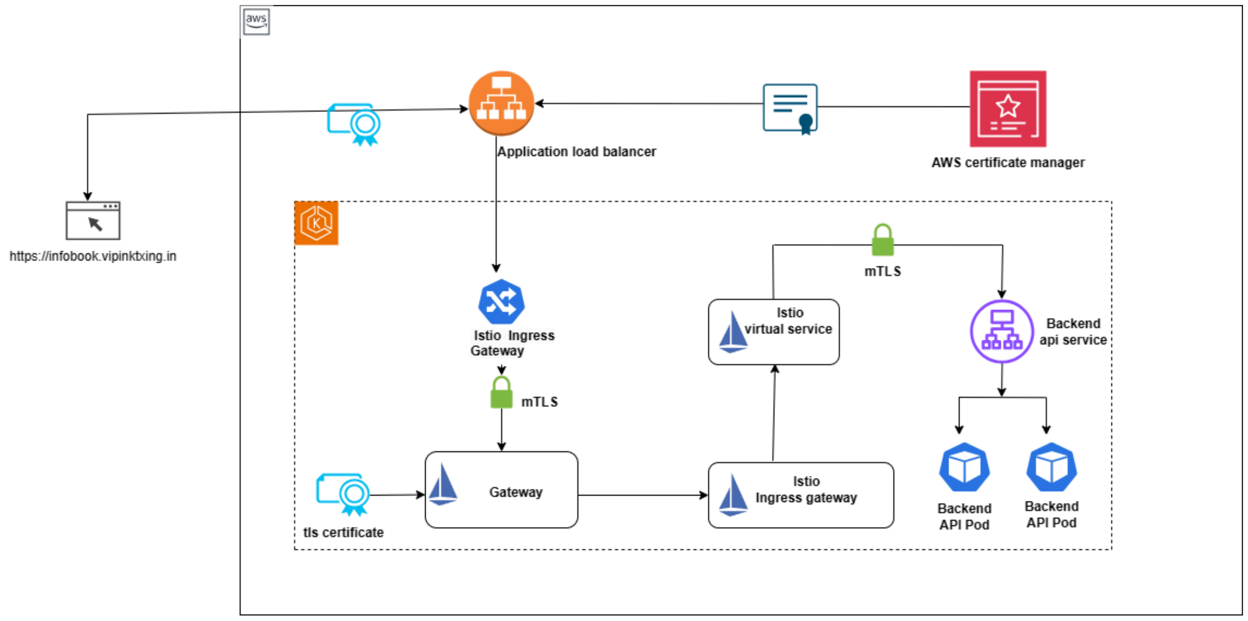


Mutual Transport Layer Security (mTLS) - Mutual Transport Layer Security (mTLS) is an essential security feature provided by Istio for securing communication between services in a microservices architecture

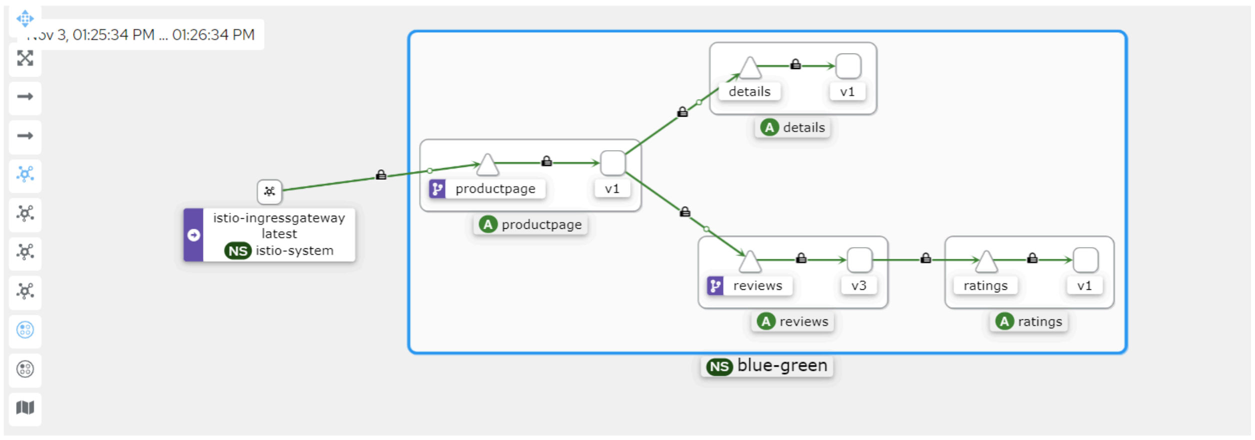
mTLS - The purpose of this PeerAuthentication resource is to enforce strict mTLS within the Istio service mesh, meaning that all services must establish secure, mutually authenticated connections. This configuration enhances security by ensuring that unauthorized or unauthenticated communication cannot occur within the service mesh.

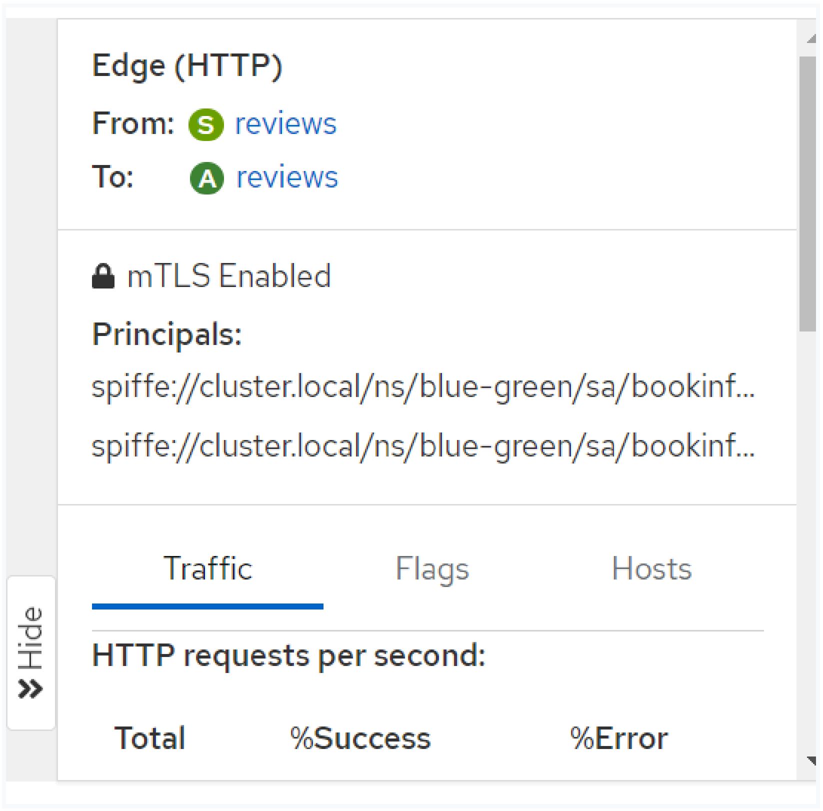
```
Kubectl apply -f PeerAuthentication.yaml
```

TRAFFIC FLOW WITH MTLS



RESULT





Edge (HTTP)

From: S reviews

To: A reviews

mTLS Enabled

Principals:

spiffe://cluster.local/ns/blue-green/sa/bookinf...

spiffe://cluster.local/ns/blue-green/sa/bookinf...

Traffic Flags Hosts

HTTP requests per second:

Total	%Success	%Error

Hide

Reference link - [Istio / Getting Started]()

CANARY DEPLOYMENT ON EKS USING ISTIO

Objective:

The goal of this document is to provide a concise solution approach for setting up a canary deployment using Istio on an Amazon EKS (Elastic Kubernetes Service) cluster. Canary deployment allows for a controlled release of new features to a subset of users, minimizing risks associated with new releases.

Problem Statement:

Deploying applications into production systems with minimal outages is always a challenge. The shift of moving monolithic applications to microservices has solved the problem of deployments to some extent where individual microservices can be deployed independently of each other. However large applications still need a mechanism where an application/microservice can be tested on production before we roll out the application for actual users.

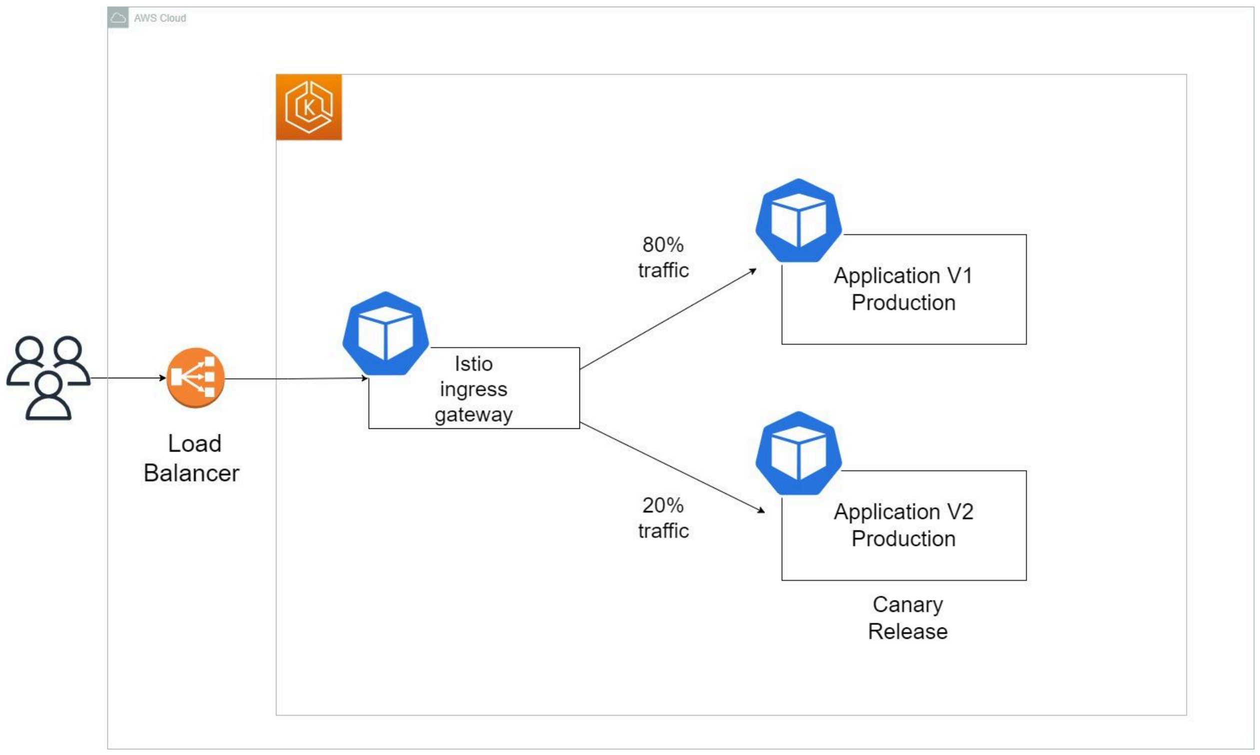
- Deploying microservices with an Istio service mesh can address this by enabling a clear separation between different application versions and traffic management.
- Istio mesh allows fine-grained traffic control to decouple traffic distribution and management from replica scaling. You can define the traffic percentages and Istio will manage the rest.

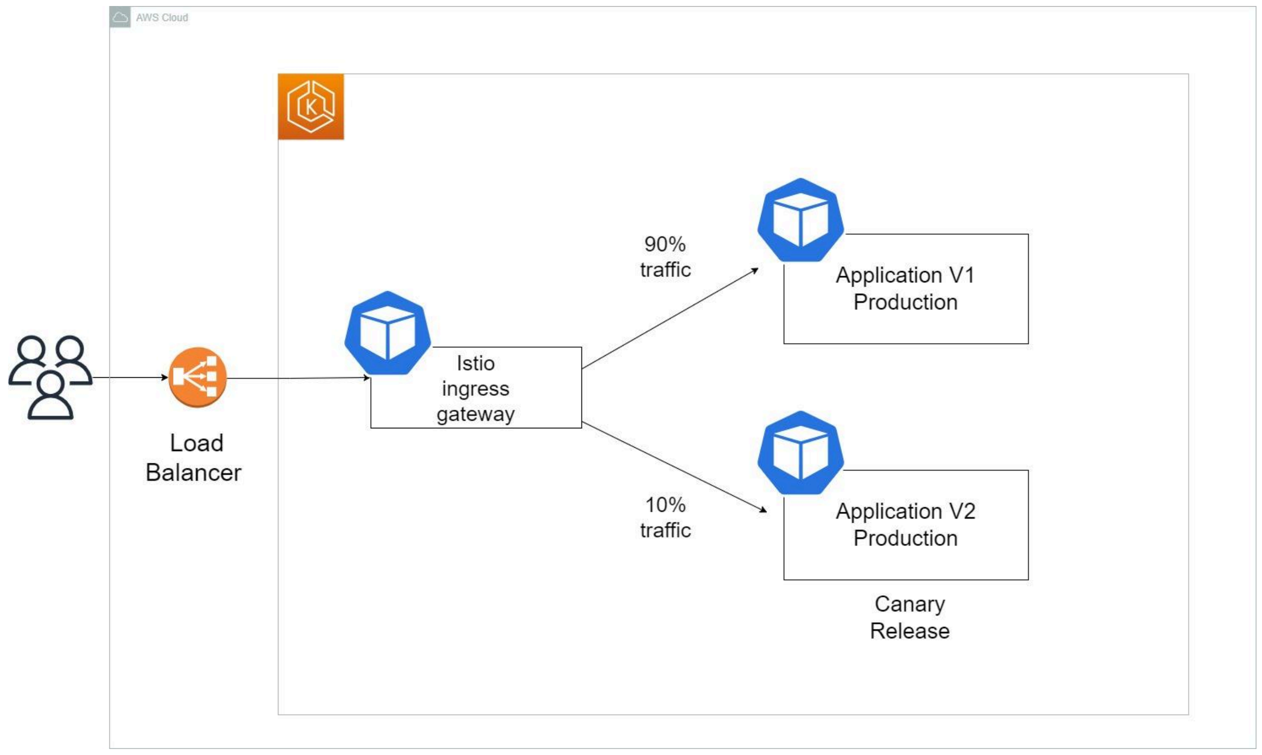
Canary deployment using Istio:

- A canary deployment is an upgraded version of an existing deployment, with all the required application code and dependencies. It is used to test out new features and upgrades to see how they handle the production environment.
- When you add the canary deployment to a Kubernetes cluster, it is managed by a service through ***selectors*** and ***labels***. The service routes traffic to the pods that have the specified label. This allows you to add or remove deployments easily.

- The amount of traffic that the canary gets corresponds to the number of pods it spins up. In most cases, you start by routing a smaller percentage of traffic to the canary and increase the number over time. With both deployments, you monitor the canary behavior to see whether any issues arise.
- Using Istio we can create canary deployment i.e., you can route the traffic on a specific service(targeting a specific application).

Canary deployment workflow :





Application used for canary deployment :

- Here we are using the **bookinfo** sample application for testing the canary deployment.
- As we already deployed the bookinfo application to test the Istio, we shall use the same for canary deployment

Destination rule :

Destination Rule allows you to define subsets of an application based on a set of labels. For example, we have deployed two subsets **v1** and **v2** of an application, and they are identified by the label version.

- Apply this destination-rule-all.yaml file to set the destination rule for the virtual service file.

Istio virtual service :

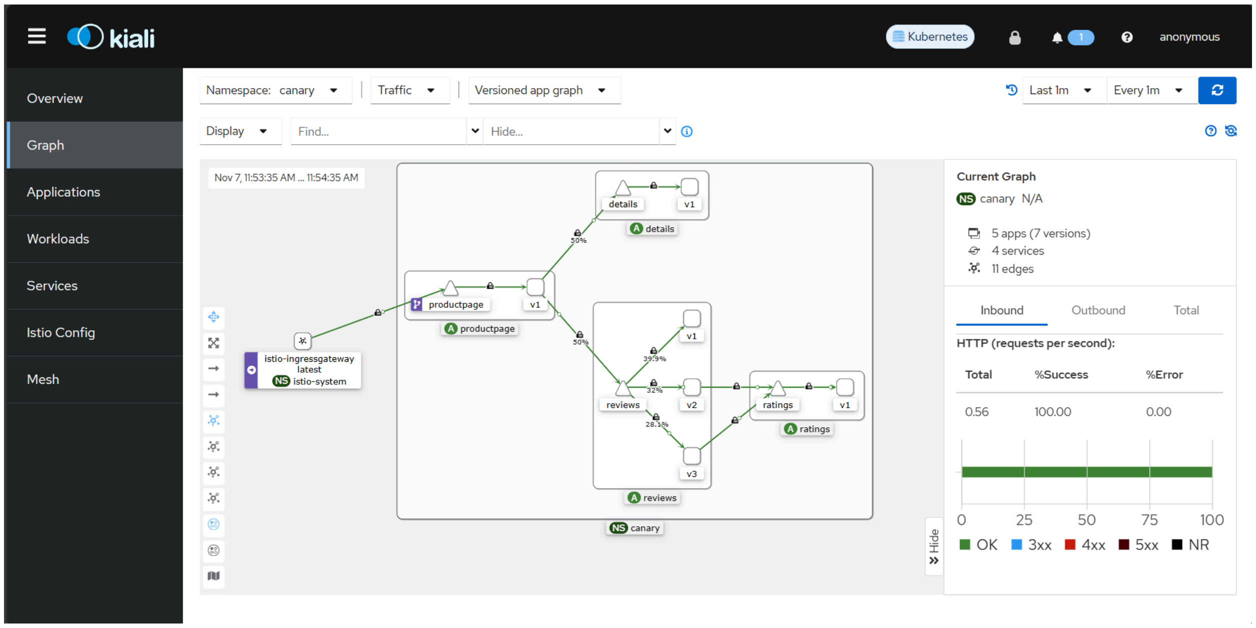
- Istio virtual service is one level higher than Kubernetes service. It can be used to apply traffic routing, fault injection, retries, and many other configurations to services.
- The virtual service will route the traffic on a specific version of applications when the user hits on mentioned hosts in the manifest file of the virtual service.
- This virtual service is connected to gateway CRDs. When a user hits the gateway URL then virtual service automatically plays its role.
- Once the destination rule is set, then apply the **virtual-service-90-10.yaml**
- Which sets the traffic to 90% for v1 and 10% for v2.
- Below is the virtual-service-90-10.yaml file, which is used to assign weight for the subsets.

```
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
 name: reviews
spec:
 hosts:
  - reviews
 http:
 - route:
  - destination:
     host: reviews
     subset: v1
    weight: 90
  - destination:
     host: reviews
     subset: v2
    weight: 10
```

- If you want to route the traffic only for the version you require, then change the percentage as you need in the same virtual service file and apply it again.
- After applying it, you will be able to see the traffic percentage splitting for the different versions in the Kiali dashboard.

Results :

- Before canary deployment



Namespace: canary | Traffic | Versioned app graph

Last 1m | Every 1m

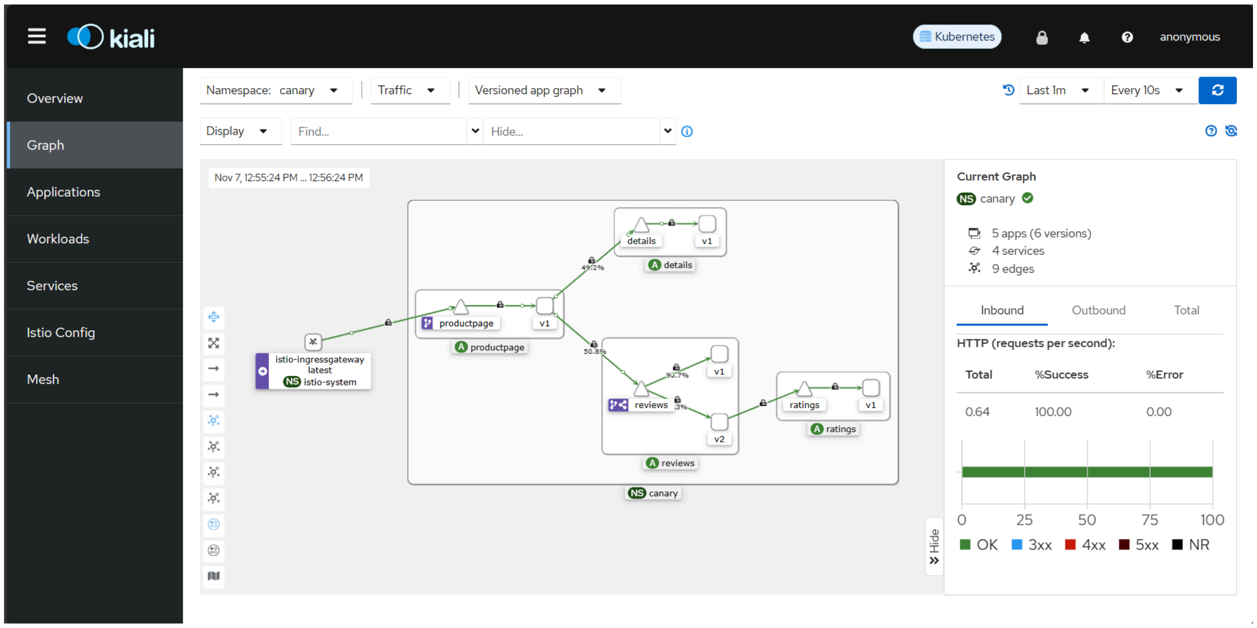
Nov 7, 11:53:35 AM ... 11:54:35 AM

Current Graph
NS canary N/A
5 apps (7 versions)
4 services
11 edges

HTTP (requests per second):

Total	%Success	%Error
0.56	100.00	0.00

- After canary deployment



Namespace: canary | Traffic | Versioned app graph | Last 1m | Every 10s

Current Graph
NS canary

5 apps (6 versions)
4 services
9 edges

HTTP (requests per second):

Total	%Success	%Error
0.64	100.00	0.00

BLUE-GREEN DEPLOYMENT ON EKS USING ISTIO

Objective:

- As a Cloud Engineer, I want to perform a POC for Setting up a blue-green deployment on EKS with istio. The goal of this document is to provide a concise solution approach for setting up a blue-green deployment on Amazon Elastic Kubernetes Service (EKS) with Istio includes establishing a controlled and automated process for transitioning between two distinct versions of an application while ensuring minimal downtime and the ability to quickly roll back if issues arise.

Acceptance criteria:

Application Containerization:

- The application is containerized and can be deployed within the EKS cluster.

Application Versions:

- Two distinct versions of the application (blue and green) are available, each with their respective container images or configurations.

Istio Virtual Services:

- Virtual services are defined to control traffic routing between the blue and green versions.
- Initially, the VirtualService directs all traffic to the blue version.

Traffic Shift:

- The deployment allows for gradual traffic shifting from the blue to the green version.
- Traffic shifting is adjustable and can be monitored in real-time.
- Traffic to the green version can be increased incrementally as needed.

Monitoring and Testing:

- Monitoring and observability tools, such as Kiali need set up to track the performance of both versions.
- Automated tests are conducted on the green version to ensure its functionality and reliability.

Full Transition:

- It is possible to complete the full transition from blue to green by adjusting the VirtualService routing rules.

- The transition occurs smoothly, with no adverse impact on users.

Performance Validation:

- The green version is validated for its performance, scalability, and functionality in a production-like environment.

Problem Statement:

We need to implement a robust and efficient deployment strategy for our microservices-based application hosted on Kubernetes with Istio as the service mesh, without using the traditional blue-green deployment technique. The challenge is to ensure seamless updates and rollbacks while minimizing service downtime and potential disruptions to end-users. Our goal is to find an alternative deployment approach that balances safety, resource efficiency, and ease of management within the Istio ecosystem, thereby addressing the limitations of the blue-green deployment method.

Blue-green deployment using Istio :

Blue-green deployment using Istio is a strategy for safely updating and releasing new versions of microservices within a Kubernetes environment while leveraging the capabilities provided by Istio as a service mesh. This approach aims to minimize service downtime, reduce deployment risks, and allow for efficient testing and rollback options. Here's an overview of the blue-green deployment process using Istio:

Environment Setup:

- Ensure that Istio is installed and configured within your Kubernetes cluster. Istio will be responsible for service discovery, load balancing, traffic routing, and canary release management.

Blue and Green Environments:

- Maintain two separate environments: the "Blue" environment representing the current live version of your microservices, and the "Green" environment for the new version you want to deploy.

Traffic Routing:

- Initially, all incoming traffic is directed to the Blue environment, where the stable version of your microservices is running.
- Istio's traffic routing capabilities, like VirtualServices and DestinationRules, are used to control how traffic flows between Blue and Green environments.

Deploy the Green Environment:

- Deploy the new version of your microservices into the Green environment. This deployment can be done using Kubernetes manifests or other deployment tools.

Canary Testing:

- Gradually shift a portion of the incoming traffic to the Green environment to perform canary testing. Istio's traffic shifting allows you to control the percentage of traffic sent to the new environment.

Monitoring and Validation:

- Continuously monitor the performance and stability of the Green environment during canary testing. Istio's observability tools, such as Kiali, Prometheus, and Grafana, can help in tracking key metrics.

Rollback Option:

- If issues are detected during canary testing or if performance metrics degrade beyond acceptable thresholds, you can quickly roll back by directing all traffic back to the Blue environment.

Gradual Promotion:

- If canary testing is successful and the new version proves to be stable, gradually increase the traffic share to the Green environment until it receives 100% of incoming traffic.

Clean-Up:

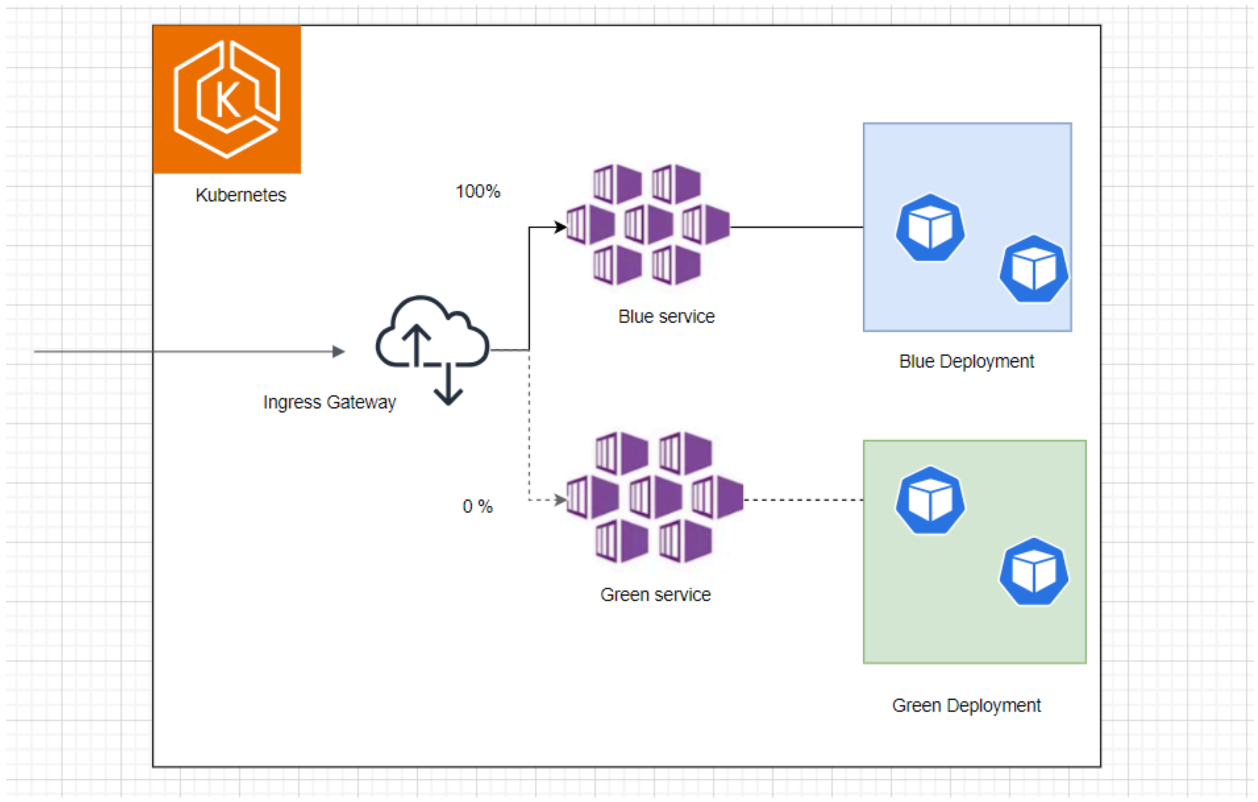
- Once the Green environment becomes the new stable version, the Blue environment can be safely decommissioned, and resources can be reclaimed.

Key benefits of using Istio for blue-green deployments include:

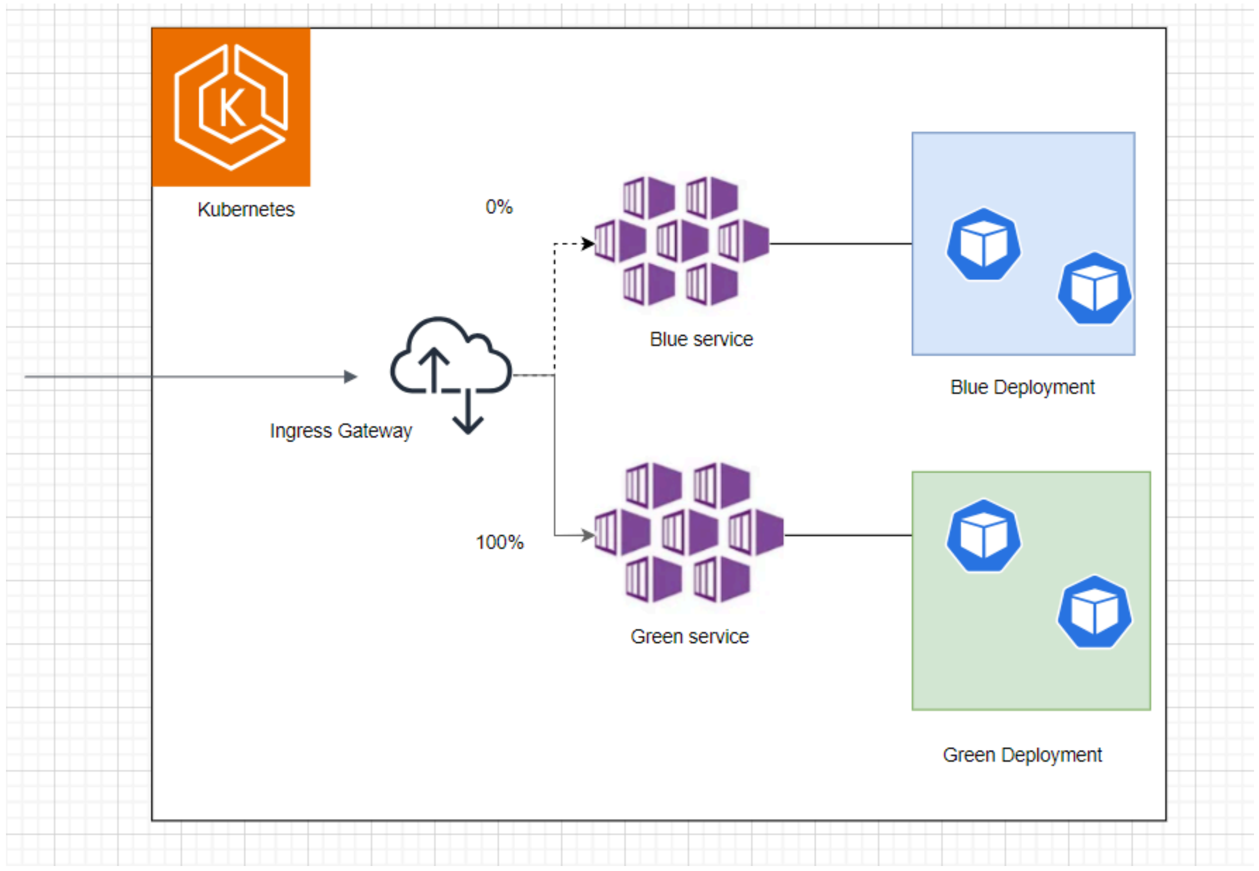
- Fine-grained traffic control: Istio allows you to manage traffic routing based on rules and metrics, ensuring precise control over the release process.
- Real-time monitoring and observability: Istio provides extensive metrics and monitoring tools to keep track of the deployment's health and performance.
- Automated rollback: Istio simplifies the rollback process, making it easier to revert to the previous version in case of issues.
- Improved fault tolerance: Istio's fault injection and circuit-breaking features can help uncover potential issues in the new version while isolating them from the rest of the system.

Overall, blue-green deployment with Istio offers a reliable and flexible approach for updating microservices, enhancing the resilience and agility of your application while minimizing the risk of service disruptions.

Blue-green Deployment workflow:



Blue Deployment is serving the production traffic.



Switch to Green deployment to serve the production traffic.

The application used for Blue-Green deployment :

- Here we are using the **bookinfo** sample application for testing the blue-green deployment.
- As we already deployed the book info application to test the Istio, we shall use the same for the blue-green deployment

Istio virtual service :

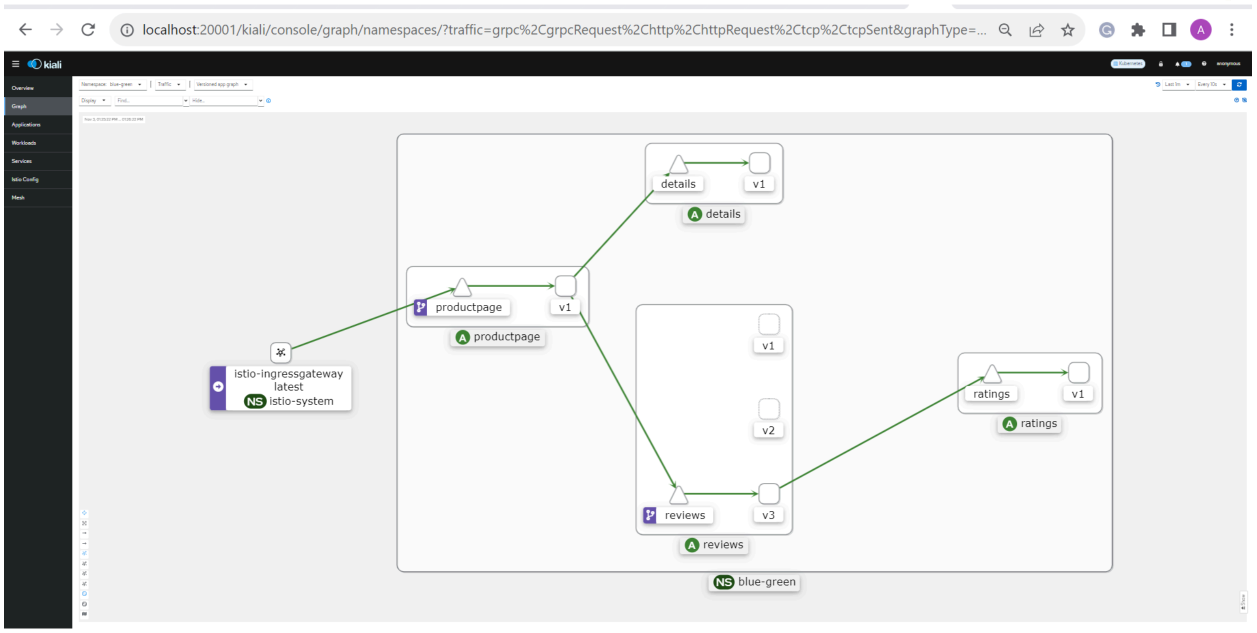
- Istio VirtualService is a fundamental resource that plays a central role in managing traffic routing, shaping, and control within the Istio service mesh. It allows you to define and configure how incoming requests are distributed to different services based on a set of routing rules, headers, paths, or other criteria.

- The virtual service will route the traffic on a specific version of applications when the user hits on mentioned hosts in the manifest file of the virtual service.

- This virtual service is connected to gateway CRDs. When a user hits the gateway URL then virtual service automatically plays its role.
- Change to the path **istio-1.19.3/samples/bookinfo/networking**
- In that path, there will be VirtualService files for blue-green deployment.
- The below is **virtual-service-reviews-v3.yaml** file, which is used to fully shift the traffic for the subsets. When you apply this file, the traffic will gradually be shifted to v3.

```
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: reviews
spec:
  hosts:
    - reviews
  http:
  - route:
    - destination:
        host: reviews
        subset: v3
```

You can able to see the gradual shifting when we use the above virtual service using the Kiali dashboard like below.



We can also do the same full traffic shifting for v1 and v2 by editing the subset in the yaml file.

Result:

As a result, This controlled traffic management approach enables seamless transitions between different versions of the service while maintaining fine-grained control over which components are visible to users, ensuring a smooth user experience.

Reference:

- Book-info Application
- Configuration for application
- Traffic-shifting