Microservices and Kubernetes Deployment Strategies

One of the biggest challenges in developing cloud native applications today is speeding up the number of your deployments. With a microservices approach, developers are already working with and designing completely modular applications that allow multiple teams to write and deploy changes simultaneously to an application.

Shorter and more frequent deployments offer the following benefits:

- Reduced time-to-market.
- Customers can take advantage of features faster.
- Customer feedback flows back into the product team faster, which means the team can iterate on features and fix problems faster.
- Higher developer morale with more features in production.

But with more frequent releases, the chances of negatively affecting application reliability or customer experience can also increase. This is why it’s essential for operations and DevOps teams to develop processes and manage deployment strategies that minimize risk to the product and customers. Learn more about CICD pipeline automation.

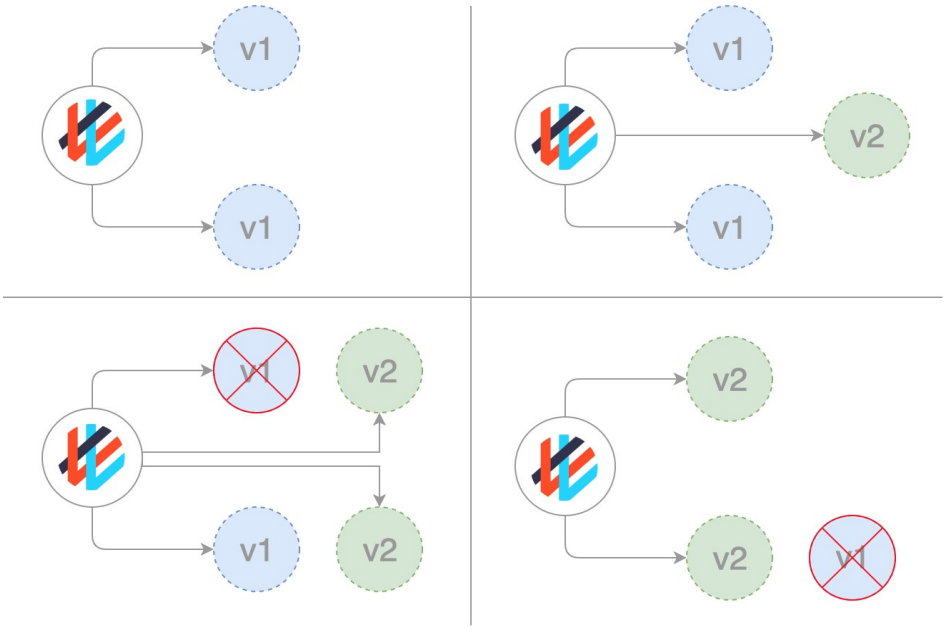
In this post, we’ll discuss Kubernetes deployment strategies, including rolling deployments and more advanced methods like canary and its variants.

Deployment Strategies

There are several different types of deployment strategies you can take advantage of depending on your goal. For example, you may need to roll out changes to a specific environment for more testing, or a subset of users/customers or you may want to do some user testing before making a feature 'Generally Available'.

Rolling Deployment

The rolling deployment is the standard default deployment to Kubernetes. It works by slowly, one by one, replacing pods of the previous version of your application with pods of the new version without any cluster downtime.



A rolling update waits for new pods to become ready via your readiness probe before it starts scaling down the old ones. If there is a problem, the rolling update or deployment can be aborted without bringing the whole cluster down. In the YAML definition file for this type of deployment, a new image replaces the old image.

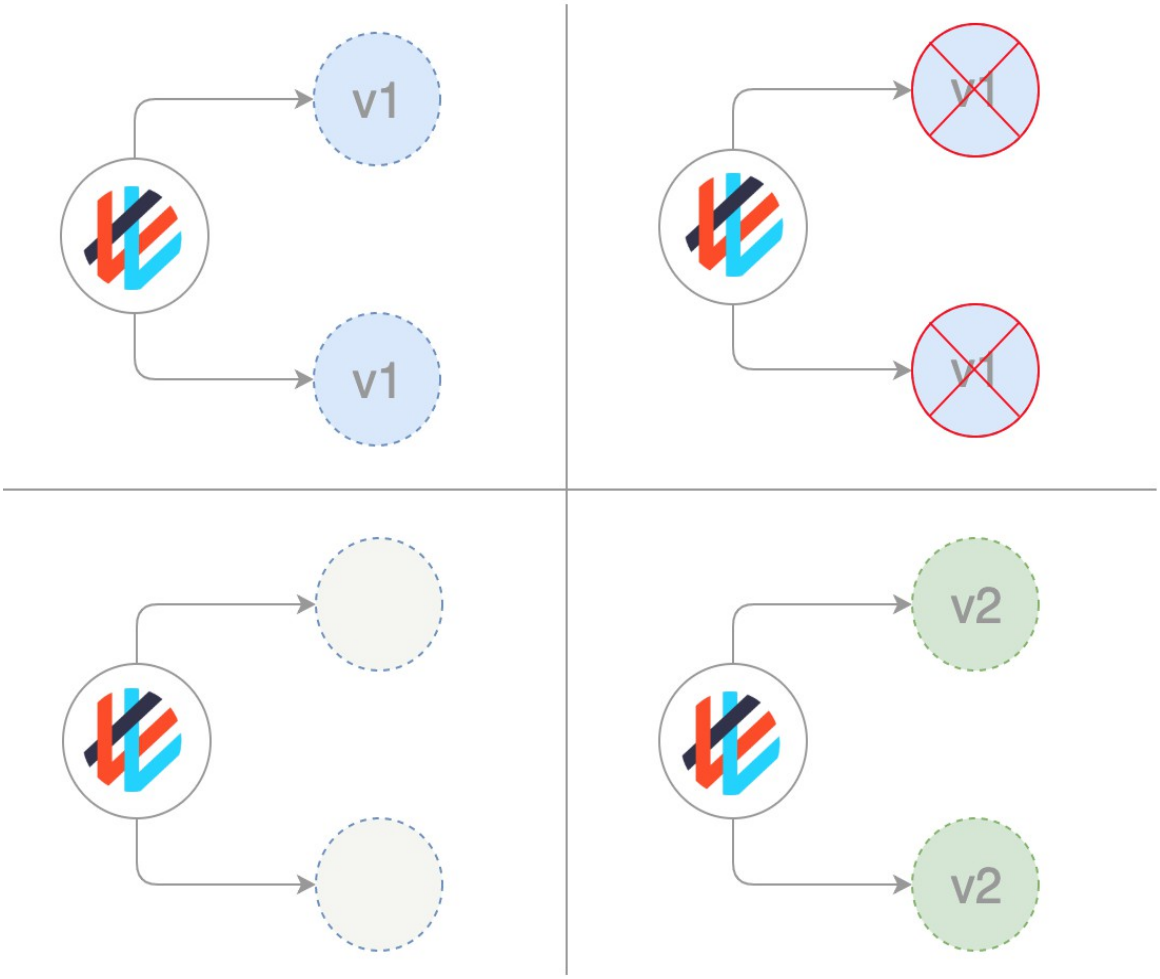
```
apiVersion: apps/v1beta1
kind: Deployment
metadata:
  name: awesomeapp
spec:
  replicas: 3
  template:
        metadata:
           labels:
             app: awesomeapp
        spec:
          containers:
            - name: awesomeapp
              image: imagerepo-user/awesomeapp:new
              ports:
                - containerPort: 8080
```

Rolling updates can be further refined by adjusting the parameters in the manifest file:

```
spec:
  replicas: 3
  strategy:
        type: RollingUpdate
        rollingUpdate:
           maxSurge: 25%
           maxUnavailable: 25%
  template:
  ...
```

Recreate

In this type of very simple deployment, all of the old pods are killed all at once and get replaced all at once with the new ones.



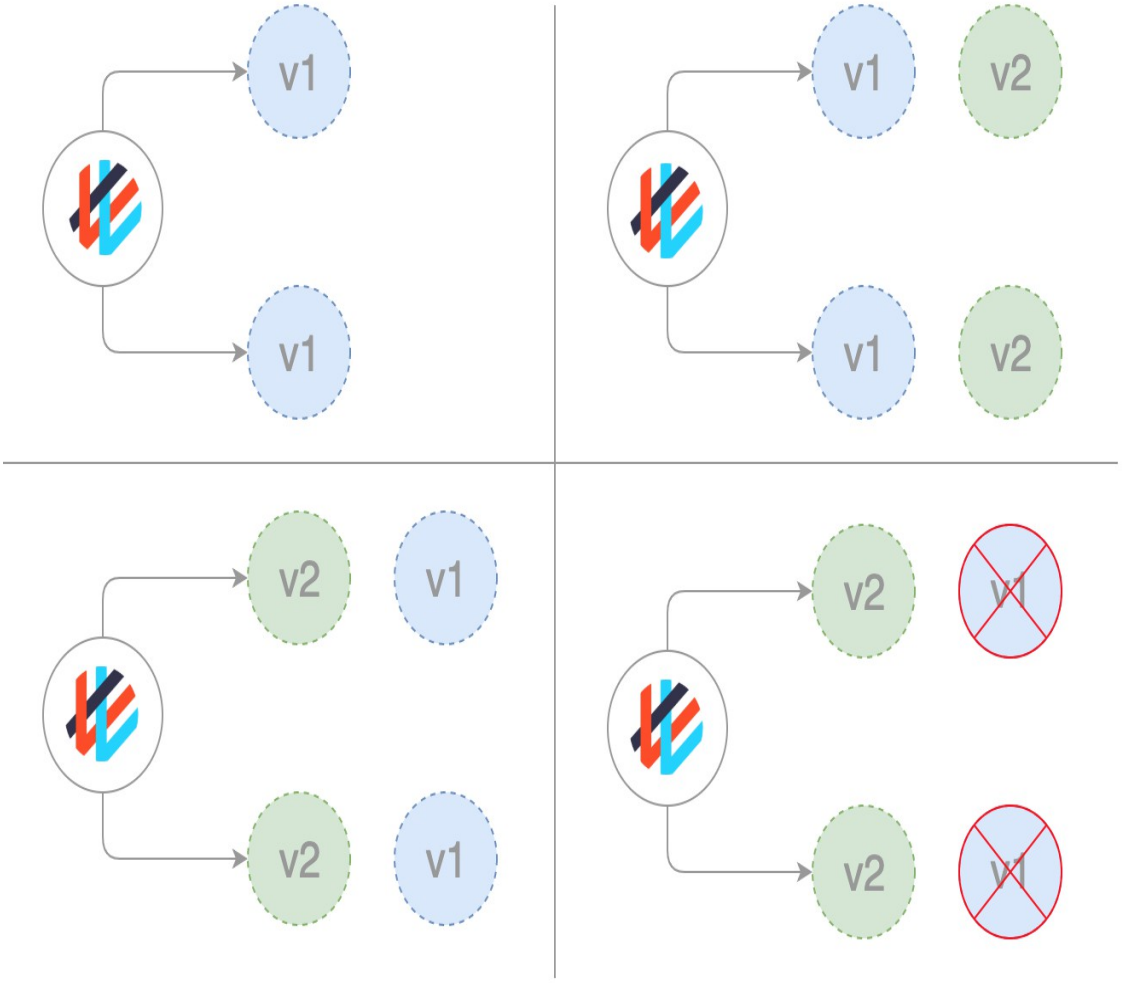
This manifest looks something like this:

```
spec:
  replicas: 3
  strategy:
        type: Recreate
  template:
  ...
```

Blue/ Green (or Red / Black) deployments

In a blue/green deployment strategy (sometimes referred to as red/black) the old version of the application (green) and the new version (blue) get deployed at the same time. When both of these are

deployed, users only have access to the green; whereas, the blue is available to your QA team for test automation on a separate service or via direct port-forwarding.




```
apiVersion: apps/v1beta1
kind: Deployment
metadata:
  name: awesomeapp-02
spec:
  template:
        metadata:
           labels:
             app: awesomeapp
             version: "02"
```

After the new version has been tested and is signed off for release, the service is switched to the blue version with the old green version being scaled down:

```
apiVersion: v1
kind: Service
metadata:
  name: awesomeapp
spec:
```

```
  selector:
    app: awesomeapp
    version: "02"
...
```

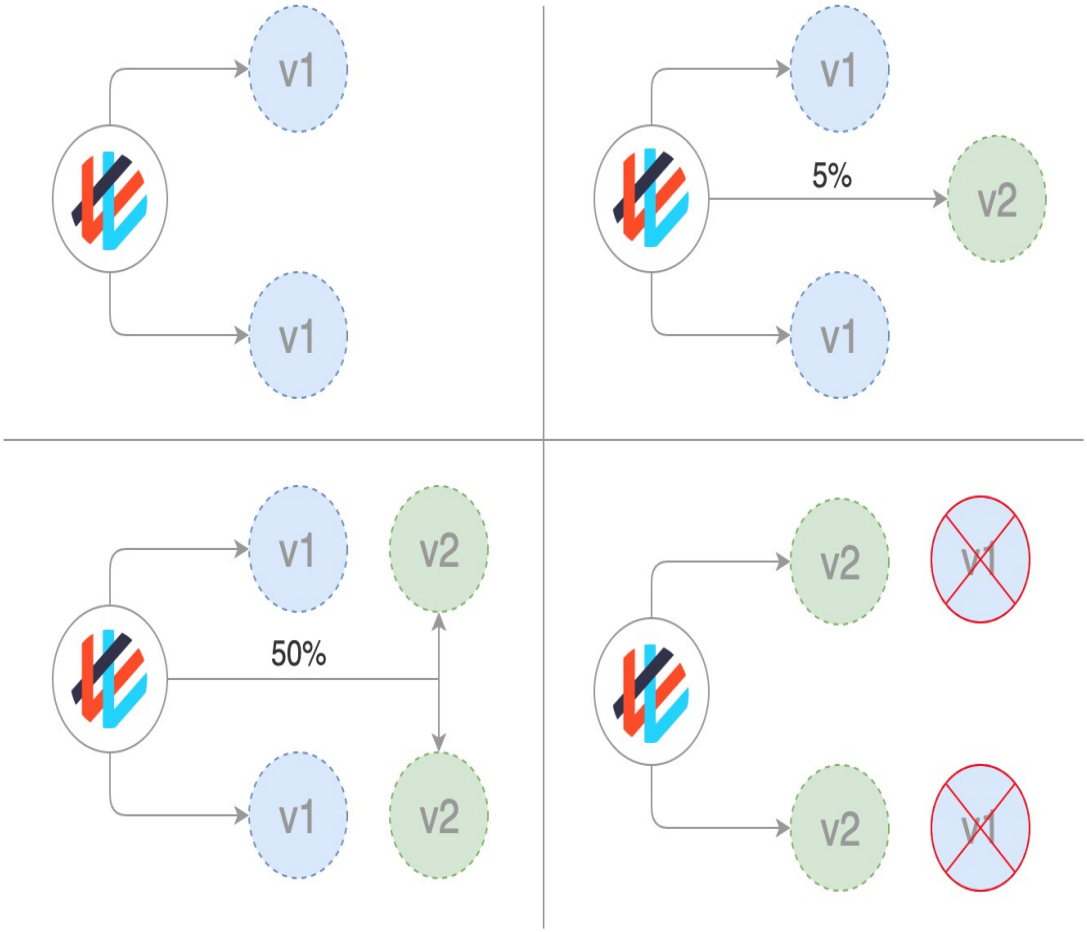
Canary

Canary deployments are a bit like blue/green deployments, but are more controlled and use a more 'progressive delivery' phased-in approach. There are a number of strategies that fall under the umbrella of canary including: dark launches, or A/B testing.

A canary is used for when you want to test some new functionality typically on the backend of your application. Traditionally you may have had two almost identical servers: one that goes to all users and another with the new features that gets rolled out to a subset of users and then compared. When no errors are reported, the new version can gradually roll out to the rest of the infrastructure.

While this strategy can be done just using Kubernetes resources by replacing old and new pods, it is much more convenient and easier to implement this strategy with a service mesh like Istio.

As an example, you could have two different manifests checked into Git: a GA tagged 0.1.0 and the canary, tagged 0.2.0. By altering the weights in the manifest of the Istio virtual gateway, the percentage of traffic for both of these deployments is managed.



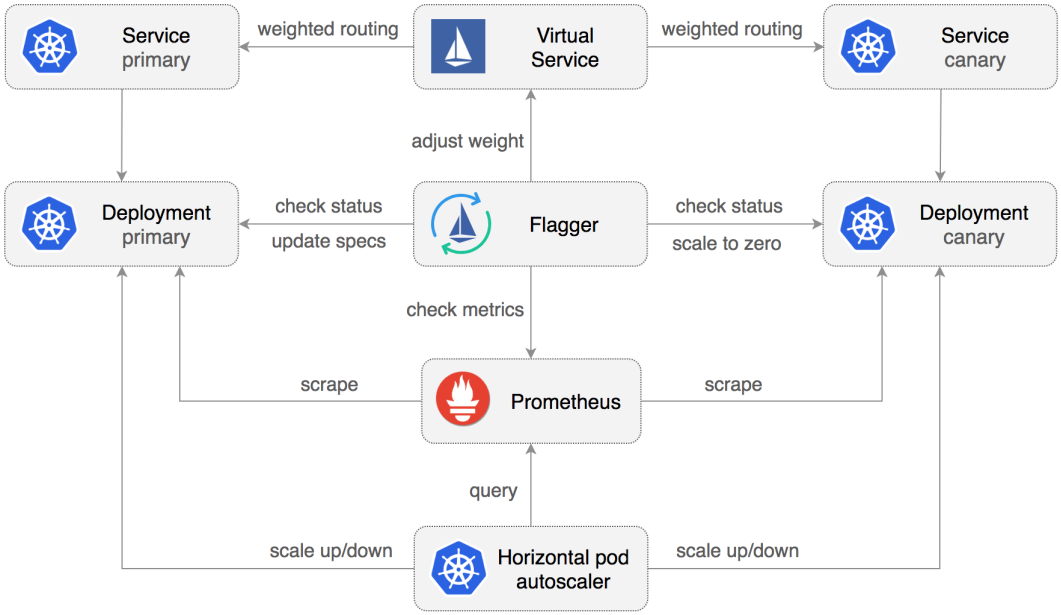
For a step by step tutorial on implementing canary deployments with Istio, see, GitOps Workflows with Istio.

Canary deployments with Weaveworks Flagger

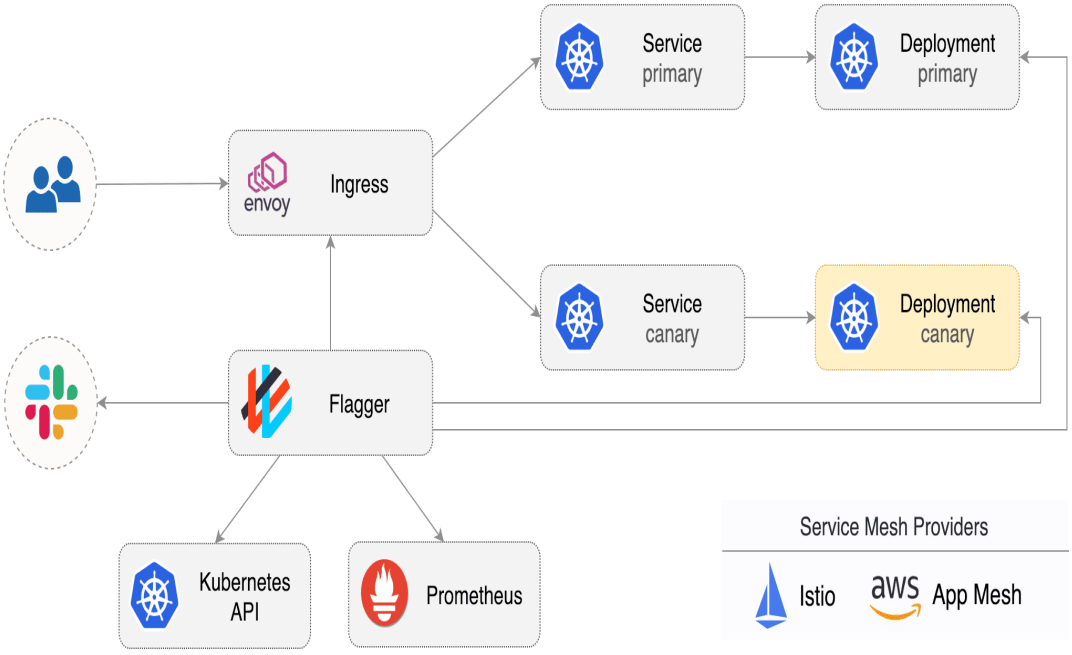
A simple and effective way to manage a canary deployment is by using Weaveworks Flagger.

With Flagger, the promotion of canary deployments is automated. It uses Istio or App Mesh to route and shift traffic, and Prometheus metrics for canary analysis. Canary analysis can also be extended with webhooks for running acceptance tests, load tests or any other type of custom validation.

Flagger takes a Kubernetes deployment and optionally a horizontal pod autoscaler (HPA) to create a series of objects (Kubernetes deployments, ClusterIP services and Istio or App Mesh virtual services) to drive the canary analysis and promotion.



By implementing a control loop, Flagger gradually shifts traffic to the canary while measuring key performance indicators like HTTP requests success rate, requests average duration and pods health. Based on the analysis of the KPIs, a canary is either promoted or aborted, and the analysis result is published to Slack. For an overview and demo see, [Progressive Delivery for App Mesh].

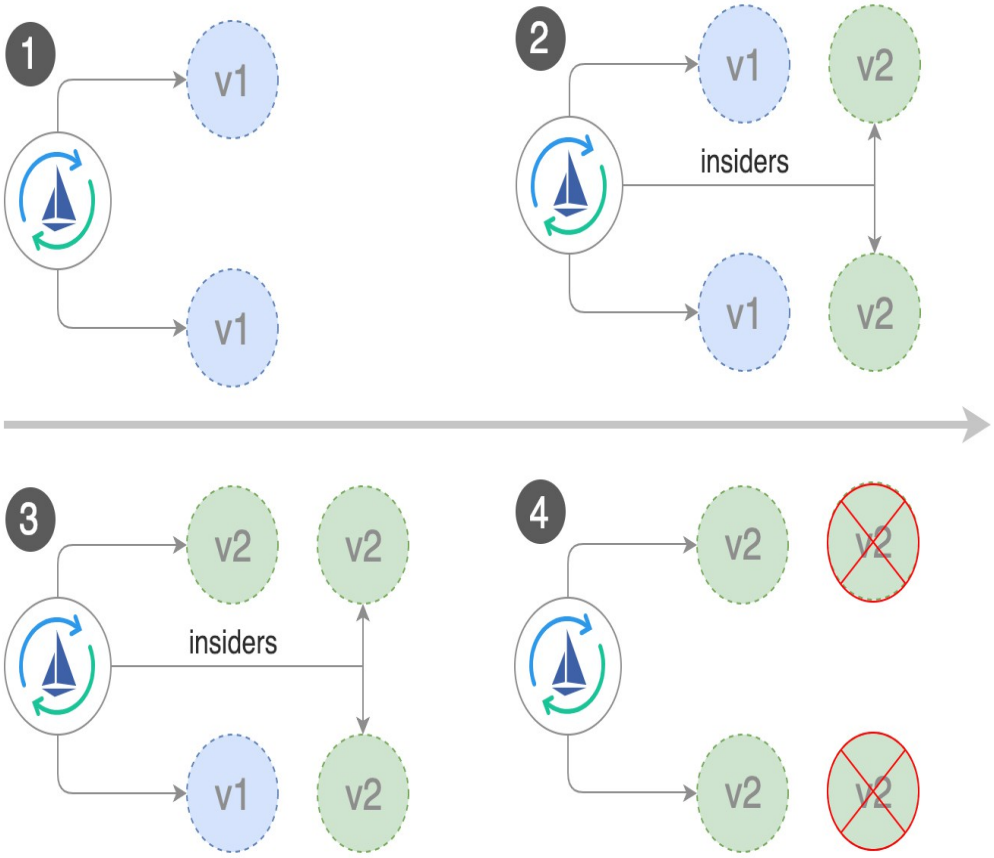


Dark deployments or A/B Deployments

A dark deployment is another variation on the canary (that incidentally can also be handled by Flagger). The difference between a dark deployment and a canary is that dark deployments deal with features in the front-end rather than the backend as is the case with canaries.

Another name for dark deployment is A/B testing. Rather than launch a new feature for all users, you can release it to a small set of users. The users are typically unaware they are being used as testers for the new feature, hence the term “dark” deployment.

With the use of feature toggles and other tools, you can monitor how your user is interacting with the new feature and whether it is converting your users, or whether they find the new UI confusing and other types of metrics.



Flagger and A/B deployments

Besides weighted routing, Flagger can also route traffic to the canary based on HTTP match conditions. In an A/B testing scenario, you'll be using HTTP headers or cookies to target a certain segment of your users. This is particularly useful for front-end applications that require session affinity. Find out more in the Flagger docs.

Thanks to Stefan Prodan, Weaveworks Engineer (and creator of Flagger) for all of these awesome deployment drawings.