

Chapter 5: Kubernetes for Continuous Integration



Table of Content

- Scaling and autoscaling
- Deployment Strategies
 - Rolling update
 - Recreate
 - Blue Green
 - Canary
- Service Discovery
- GITOps

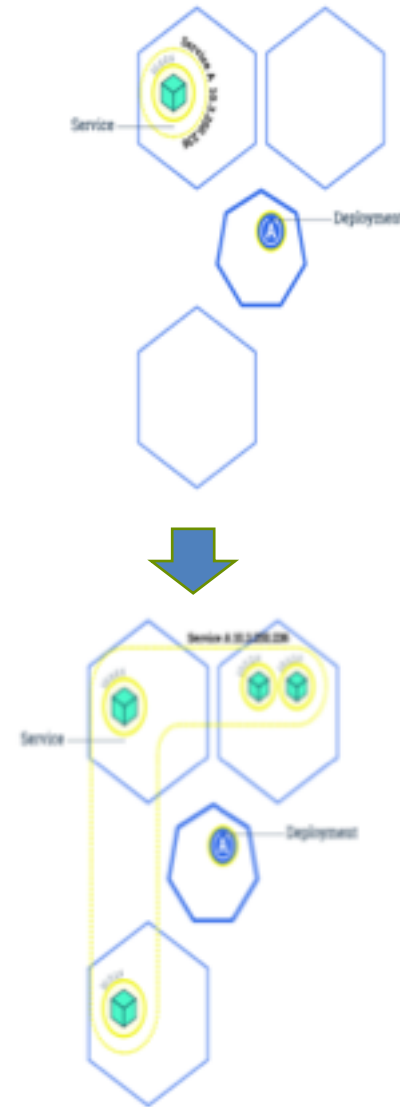
Zero Downtime Scaling



Kubernetes: Scaling

Changing the number of resources to meet a desired state

- Accomplished by adjusting the number of replicas in a deployment
- Accommodates both scaling up and scaling down to a minimum of 0
- Traffic is automatically sent to newly created instances through the service load balancer
- Can be used to enable rolling updates without downtime



\$ kubectl scale deployments/<deployment> --replicas=<new num>

\$ kubectl scale deploy/nginx --replicas=4

```
peter@Azure:~$ kubectl run nginx --image nginx --port 80
deployment "nginx" created
peter@Azure:~$
peter@Azure:~$ kubectl get deployments
NAME          DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
nginx         1         1         1            1           39s
peter@Azure:~$ kubectl scale deployments/nginx --replicas=2
deployment "nginx" scaled
peter@Azure:~$ kubectl get deployments
NAME          DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
nginx         2         2         2            1           1m
```

Note: Make sure to delete the deployments or replicaset when trying to clear out pods. Many a new user has repeated deleted pods and gotten frustrated when they respawn (as the deployments/replicaset are programmed to do).



```

root@ip-172-31-0-112:~/code# kubectl run nginx --image nginx --port 80
deployment.apps/nginx created
root@ip-172-31-0-112:~/code#
root@ip-172-31-0-112:~/code# kubectl get deployments
NAME                DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
nginx                1         1         1            1           8s
nginx-deployment     3         3         3            3           17m
redis                1         1         1            1           33m
root@ip-172-31-0-112:~/code#
root@ip-172-31-0-112:~/code# kubectl scale deployments/nginx --replicas=3
deployment.extensions/nginx scaled
root@ip-172-31-0-112:~/code# kubectl get deployments
NAME                DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
nginx                3         3         3            3           34s
nginx-deployment     3         3         3            3           18m
redis                1         1         1            1           33m
root@ip-172-31-0-112:~/code# kubectl get pods
NAME                                READY     STATUS    RESTARTS   AGE
nginx-6f858d4d45-b4cv2              1/1      Running   0          42s
nginx-6f858d4d45-g42mg              1/1      Running   0          16s
nginx-6f858d4d45-l69jv              1/1      Running   0          16s
nginx-apparmor                      1/1      Running   0          30m
nginx-deployment-67594d6bf6-7v5lk   1/1      Running   0          12m
nginx-deployment-67594d6bf6-fdqx2    1/1      Running   0          18m
nginx-deployment-67594d6bf6-sx62p    1/1      Running   0          18m
redis-7869f8966-sq8xm               1/1      Running   0          33m

```



Autoscaling and HPA (Horizontal Pod Autoscaler)



Kubernetes: Pod Auto Scaling

HPA – Horizontal Pod Autoscaler:

Based on memory and CPU utilization by a pod, autoscaling scales up or down the number of pods

```
$ kubectl autoscale deployment nginx --min=2 --max=10
```

```
$ kubectl autoscale deployment nginx-deployment --max=5 --cpu-percent=80
```

```
$ kubectl get hpa
```

```
$ kubectl get hpa
```

NAME	REFERENCE	TARGETS	MINPODS	MAXPODS	REPLICAS	AGE
nginx	Deployment/nginx	<unknown>/80%	1	4	3	12m
nginx-deployment	Deployment/nginx-deployment	<unknown>/80%	2	5	0	5s



Kubernetes: Cluster AutoScaler

Adds or removes *Nodes* in a *Cluster* based on resource requests from *Pods* and to meet workloads

Possible only in cloud:

AWS, GCE, Azure, etc.

When does auto scale up kicked in?

- Pods not getting scheduled and # nodes within user-defined limit



Kubernetes: Cluster AutoScaler

When does auto scale **down** kicked in?

- Pod requests on a node lower than a user-defined threshold
- Checks performed on each pod in the node
 - Pod runs a DaemonSet – it's safe
 - Removing the pod does not bring down minimum number of replica
 - Pod doesn't use local storage
 - Kube-system pods are not moved



Continuous Deployment Strategies



Strategies

Able to reliably and safely deploy, rollback and orchestrate software.

List of strategies:

- **Recreate**
- **Ramped (Rolling update) Deployments**
- Blue/Green Deployments
- Canary Releases
- A/B Testing

Strategies - Recreate

Recreate –

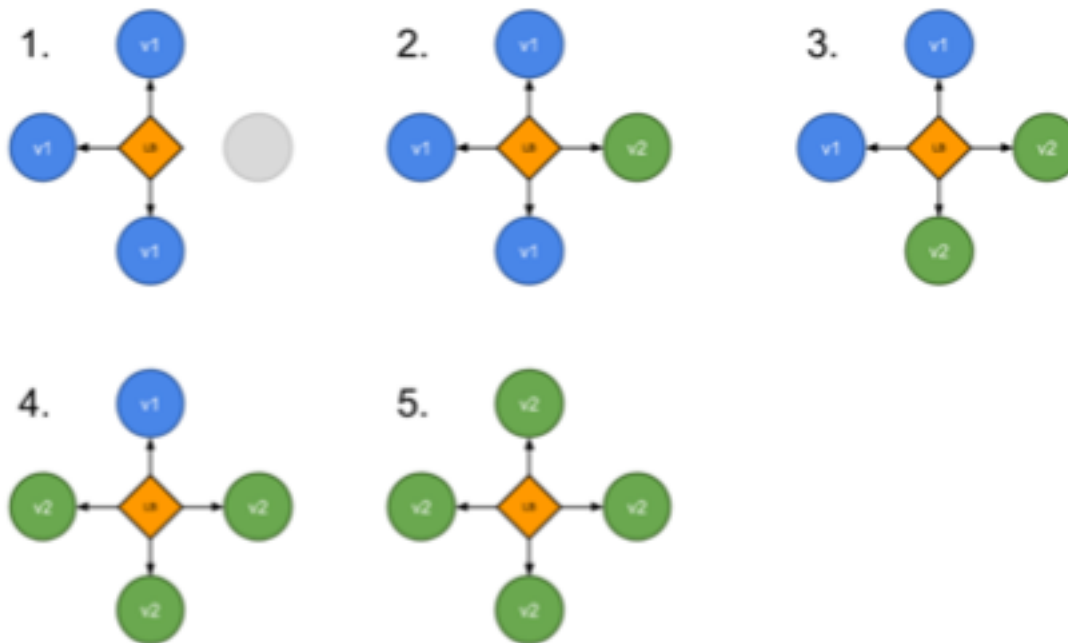
```
spec:  
  replicas: 3  
  selector:  
    matchLabels:  
      app: my-app  
  strategy:  
    type: Recreate  
  spec:  
    containers:  
      - name: my-app  
        image: my-image
```

- terminate all the running instances then recreate them with the newer version
- application state entirely renewed
- not a true continuous deployment strategy
- downtime
- conceptual simplicity



Strategies – Ramped or Rolling update

Ramped or Rolling Update



- A secondary ReplicaSet is created with the new version of the application
- The number of replicas of the old version is decreased and the new version is increased
- Until the correct number of replicas is reached

Strategies – Ramped or Rolling update

Best Practice

- Should be accompanied by some kind of a basic health check
 - To verify the new instances are ready to provide services
 - To avoid downtime
-
- **Keep in mind**
 - The old and the new version run side by side.
 - Requires full backward compatibility of our API and schema changes
 - Careful API versioning whenever backward compatibility gets broken.
-
- If deployment exposed as service, the service will load-balance traffic to only active and available pods

Strategies – Ramped or Rolling update

spec:

replicas: 3

strategy:

type: RollingUpdate

rollingUpdate:

maxSurge: 2 or x%

max number of pods can be created over desired

maxUnavailable: 1 or x%

define how many pods can be unavailable

containers:

- image: nginx:2.0

readinessProbe:

when the container is ready to accept traffic

httpGet:

path: /

port: 8080

initialDelaySeconds: 5

periodSeconds: 5

successThreshold: 1



\$ kubectl set image <deployment> <new-image>

```
peter@Azure:~$ kubectl run nginx --image nginx:1.12.1 --port 80
deployment "nginx" created
peter@Azure:~$ kubectl set image deployments/nginx nginx:nginx:latest
deployment "nginx" image updated
peter@Azure:~$ kubectl describe deployment nginx
Name:
Namespace:
Labels:
Annotations:
Selector:
Replicas:
StrategyType:
MinReadySeconds:
RollingUpdateStrategy:
Pod Template:
Labels:
Containers:
  nginx:
    Image:
    Port:
    Environment:
    Mounts:
    Volumes:
Conditions:
Type
Status
Reason
Available
True
MinimumReplicasAvailable
OldReplicaSets:
NewReplicaSet:
Events:
FirstSeen
LastSeen
Count
From
SubObjectPath
Type
Reason
Message
```

FirstSeen	LastSeen	Count	From	SubObjectPath	Type	Reason	Message
32s	32s	1	deployment-controller		Normal	ScalingReplicaSet	Scaled up replica set nginx-1295584306 to 1
10s	10s	1	deployment-controller		Normal	ScalingReplicaSet	Scaled up replica set nginx-2688028062 to 1
10s	10s	1	deployment-controller		Normal	ScalingReplicaSet	Scaled down replica set nginx-1295584306 to 0

Note: Rolling updates can also be undone



\$ kubectl rollout undo <deployment>

```
peter@azure:~$ kubectl rollout undo deployments/nginx
deployment "nginx" rolled back
peter@azure:~$ kubectl describe deployments/nginx
Name: nginx
Namespace: default
CreationTimestamp: Fri, 15 Sep 2017 13:36:49 +0000
Labels: run=nginx
Annotations: deployment.kubernetes.io/revision=3
Selector: run=nginx
Replicas: 1 desired | 1 updated | 1 total | 1 available | 0 unavailable
StrategyType: RollingUpdate
MinReadySeconds: 0
RollingUpdateStrategy: 1 max unavailable, 1 max surge
Pod Template:
  Labels: run=nginx
  Containers:
    nginx:
      Image: nginx:1.12.1
      Port: 80/TCP
      Environment: <none>
      Mounts: <none>
      Volumes: <none>
  Conditions:
    Type           Status  Reason
    ----           -
    Available      True    MinimumReplicasAvailable
  OldReplicaSets: <none>
  NewReplicaSet:  nginx-1295584306 (1/1 replicas created)
Events:
  FirstSeen      LastSeen        Count   From              SubObjectPath  Type            Reason             Message
  ----
  1m             1m              1      deployment-controller  /               Normal          ScalingReplicaSet   Scaled up replica set nginx-2688028062 to 1
  1m             1m              1      deployment-controller  /               Normal          ScalingReplicaSet   Scaled down replica set nginx-1295584306 to 0
  1m             10s             2      deployment-controller  /               Normal          ScalingReplicaSet   Scaled up replica set nginx-1295584306 to 1
  10s            10s             1      deployment-controller  /               Normal          DeploymentRollback   Rolled back deployment "nginx" to revision 1
  10s            10s             1      deployment-controller  /               Normal          ScalingReplicaSet   Scaled down replica set nginx-2688028062 to 0
```

Note: Rollbacks can also be also enacted with zero-downtime



```

root@ip-172-31-0-112:~/code# kubectl rollout undo deployments/nginx
deployment.extensions/nginx
root@ip-172-31-0-112:~/code# kubectl describe deployments/nginx
Name: nginx
Namespace: default
CreationTimestamp: Thu, 19 Jul 2018 04:00:55 +0000
Labels: run=nginx
Annotations: deployment.kubernetes.io/revision=3
Selector: run=nginx
Replicas: 1 desired | 1 updated | 1 total | 1 available | 0 unavailable
StrategyType: RollingUpdate
MinReadySeconds: 0
RollingUpdateStrategy: 25% max unavailable, 25% max surge
Pod Template:
  Labels: run=nginx
  Containers:
    nginx:
      Image: nginx:1.12.1
      Port: 80/TCP
      Host Port: 0/TCP
      Environment: <none>
      Mounts: <none>
      Volumes: <none>
  Conditions:
    Type           Status  Reason
    ----           -
    Available       True    MinimumReplicasAvailable
    Progressing     True    NewReplicaSetAvailable
OldReplicaSets: <none>
NewReplicaSet:  nginx-7f9bc86464 (1/1 replicas created)
Events:
  Type           Reason             Age             From              Message
  ----           -
  Normal         ScalingReplicaSet   1m              deployment-controller Scaled up replica set nginx-6c486b77db to 1
  Normal         ScalingReplicaSet   1m              deployment-controller Scaled down replica set nginx-7f9bc86464 to 0
  Normal         ScalingReplicaSet   18s (x2 over 2s) deployment-controller Scaled up replica set nginx-7f9bc86464 to 1
  Normal         DeploymentRollback  18s             deployment-controller Rolled back deployment "nginx" to revision 1
  Normal         ScalingReplicaSet   17s             deployment-controller Scaled down replica set nginx-6c486b77db to 0

```

kubectl rollout status deployment nginx

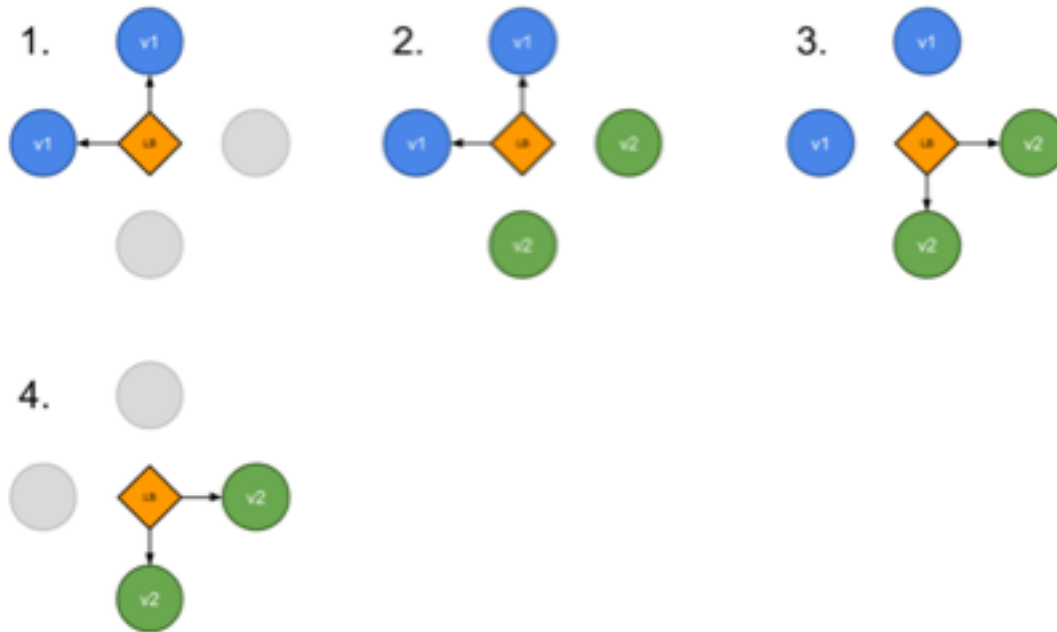
kubectl rollout history deployment nginx

kubectl rollout history deployment/nginx --revision=3

kubectl rollout undo deployment/nginx --to-revision <num>



Strategies – Blue Green Deployment



- “green” version is deployed alongside “blue”
- Test the new version
- Update the Service object (load balancer) to send traffic to the new version by replacing the version label in the selector field

Strategies – Blue Green Deployment

(-)

- Requires double the resources
- Proper test of the entire platform

(+)

- No downtime
- Easy rollback

Steps to follow

- v1 is serving traffic
- deploy v2
- wait until v2 is ready
- switch incoming traffic from v1 to v2
- shutdown v1



kubectl apply -f my-dep-v1.yaml



selector:
app: my-dep
version: v1

kubectl apply -f my-dep-v2.yaml



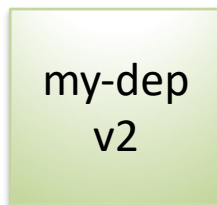
After
test



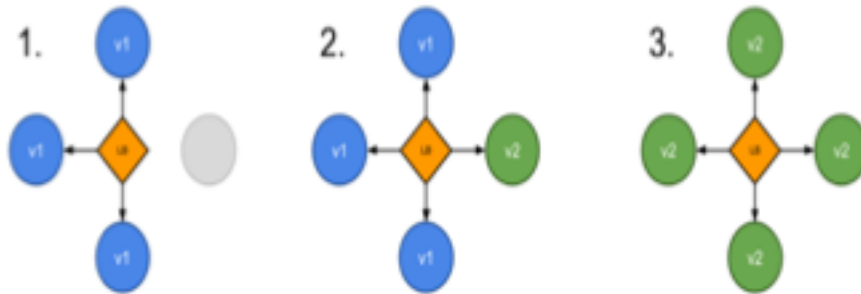
```
kubectl patch service my-app -p  
'{"spec":{"selector":{"version":"v2.0.0"}}}'
```



selector:
app: my-dep
version: v2



Strategies – Canary Deployment



- Route a subset of users to a new functionality.
- Can be done using two Deployments with common pod labels.
- One replica of the new version is released alongside the old version
- Then after some time and if no error is detected, scale up the number of replicas of the new version and delete the old deployment.

Strategies – Canary Deployment

(-)

- Slow rollout

(+)

- Fast rollback
- Good for error rate & performance monitoring

Steps to follow

- 10 replicas of v1 is serving traffic
- deploy 1 replicas v2 (meaning ~10% of traffic)
- Wait enough time to confirm that v2 is stable and not throwing unexpected errors
- scale up v2 replicas to 10
- wait until all instances are ready
- shutdown v1



kubectl apply -f my-dep-v1.yaml



kubectl apply -f my-dep-v2.yaml



selector:
app: my-dep



After
sometime

kubectl scale --replicas=10 deploy my-dep-v2
kubectl delete deploy my-dep-v1



selector:
app: my-dep



Exercise 4 – (scaling, Auto scaling, rollout)

- For a previously created deployment scale the number of replicas
- see the difference in replicaset, pods , and see with a describe
- Rolling update, undo and see the changes in replica sets

cd to the ex4

https://github.com/shekhar2010us/kubernetes_teach_git/blob/master/ex4/scaling_rolling.md



Service Discovery



Service Discovery

Migrating from a monolithic architecture to a Microservices based architecture where each service loosely coupled and horizontally scaled will require a service discovery mechanism to ensure that the REST API endpoints of the ephemeral services (say, docker containers running inside PODs) are resolved or detected. The IP addresses of the containers (or the PODs) being transient will need automated mechanism to resolve or convey the current IP addresses.

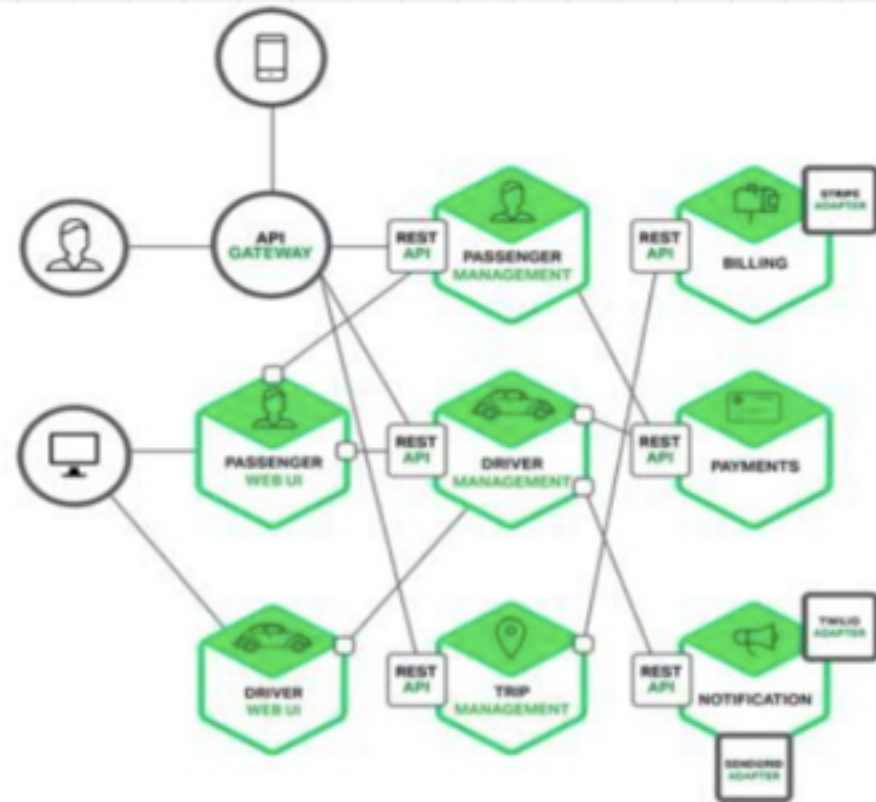


Sample Microservices Architecture

Monolith



Microservices



Service Discovery

**So how do we keep
track of all the
microservices?**



Service Discovery!



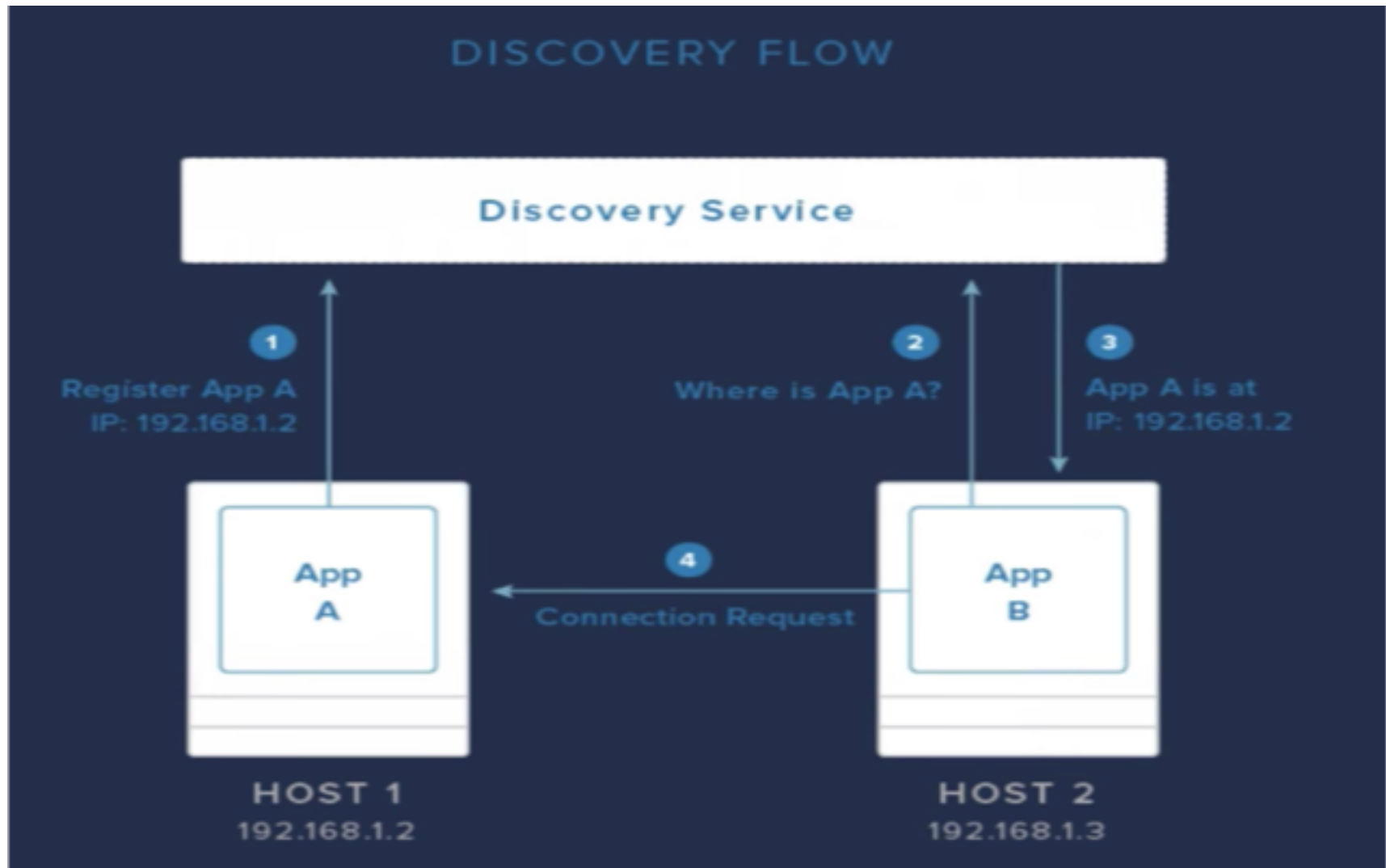
Service Discovery

What should Service Discovery provide?

- **Discover services** dynamically to get IP address and port details to communicate with other services
- **Health check:** Only healthy services should participate in handling traffic
- **Load balancing:** Traffic destined to a particular service should be dynamically load balanced



Service Discovery



Service Discovery Tools

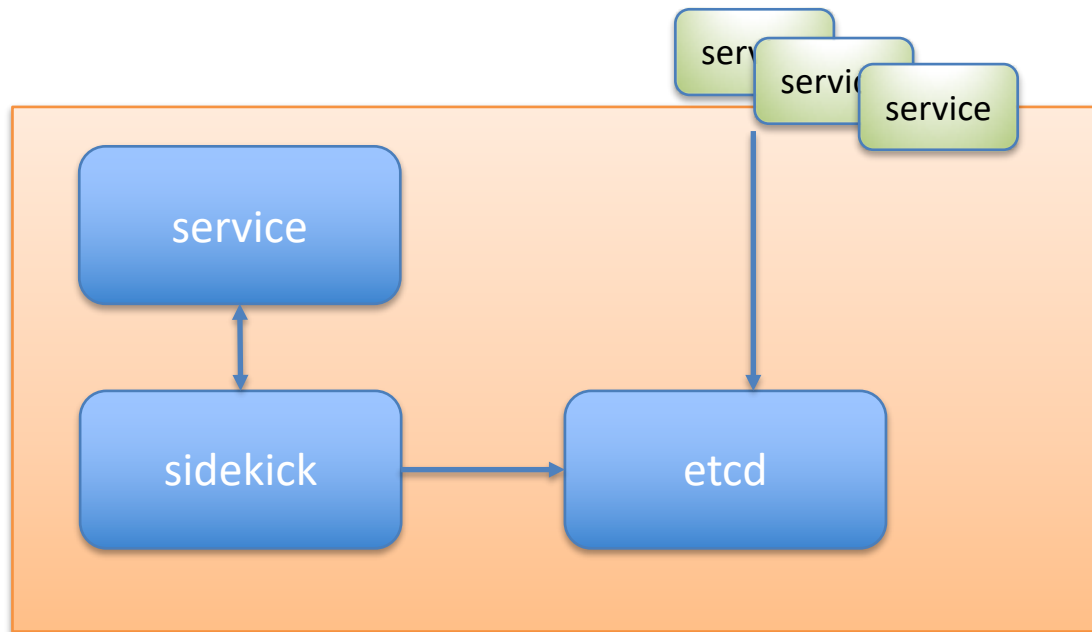
What are some Service Discovery tools?

- etcd, by CoreOS
- Consul, by Hashicorp
- Zookeeper, by Apache
- SkyDNS (built on top of etcd)
- Eureka, by Netflix
- Smartstack, by AirBnB



Service Discovery by etcd

- Etcd can be used as KV store for service registry
- Service itself can directly update etcd or a Sidekick service can be used to update etcd
- Sidekick service serves as **registrator**
- Other services can query etcd database to do the dynamic service discovery



Kubernetes Service Discovery

DNS Resolution

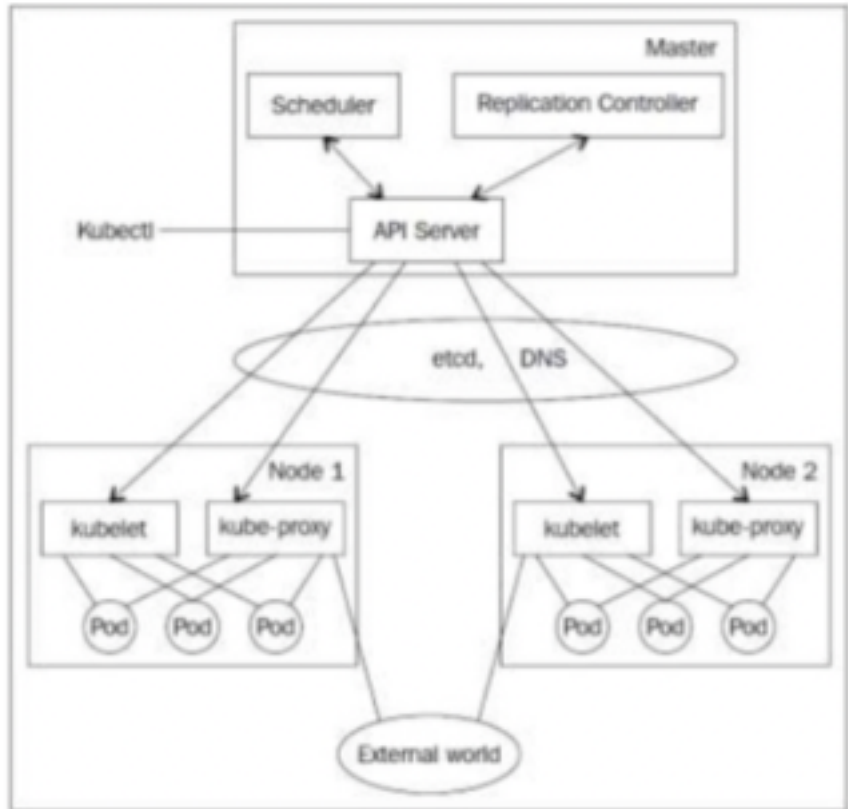
Kubernetes has a kube-dns addon that exposes the service's name as a DNS entry. As a result, you can tell your application to connect to a host name.

The service names are scoped within namespaces. This allows you to run different deployment of a service for each namespace (for example, one per developer or one per environments) without having to edit configuration files.



K8s Service Discovery Components

- **SkyDNS** – map Service name to IP address
- **Etcd** – KV store for service database
- **Kubelet** – healthcheck and replication controller takes care of maintaining pod count
- **Kube-proxy** – takes care of load balancing traffic to individual pods. Watches service changes and updates IPTables



Lab: Service Discovery

https://github.com/shekhar2010us/kubernetes_teach_git/blob/master/k8s_service_discovery/service-discovery.md

Example using consul:

<https://github.com/hashicorp/demo-consul-101/tree/master/k8s>

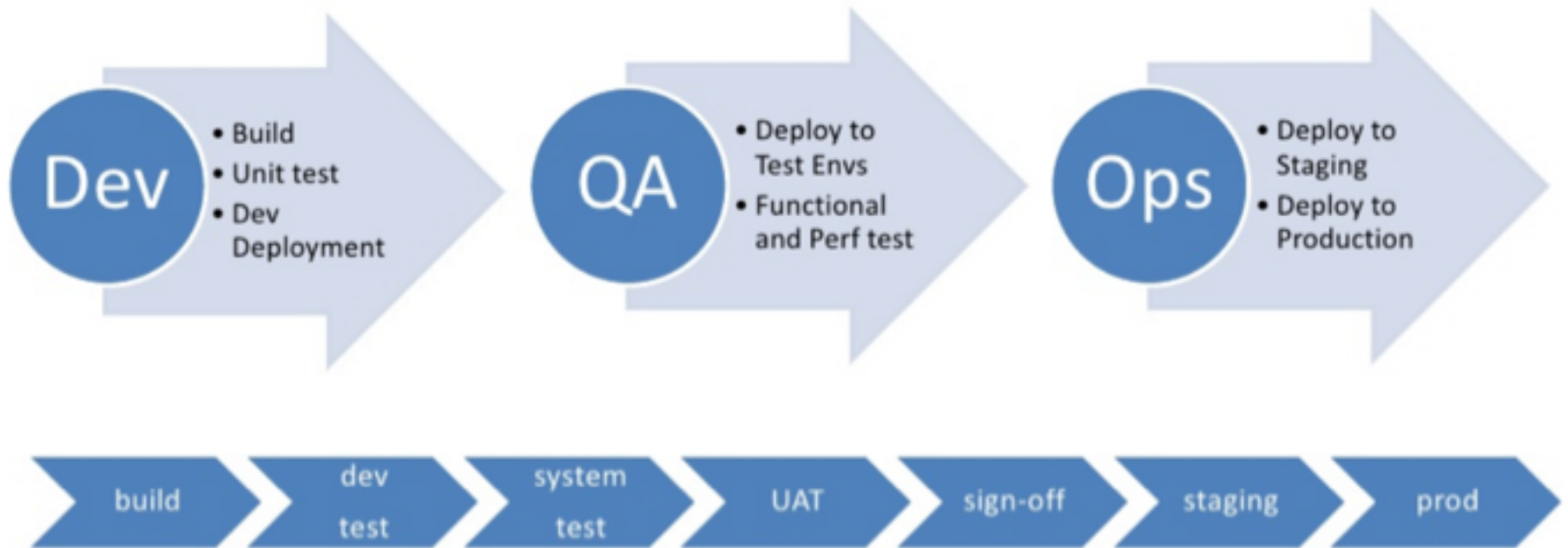


GitOps



Introduction to CI/CD

What is a Pipeline



Continuous Integration: Benefits

Reduced integration risk. More often than not, working on projects means multiple people are working on the separate tasks or parts of the code. The more people, the riskier the integration. Depending on how bad the problem really is, debugging and solving the issue can be really painful and can potentially mean a lot of changes to the code. Integrating on a daily basis or even more frequently can help reduce these kinds of problems to a minimum.

Higher code quality. Not having to worry about the problems, and focusing more on the functionality of the code results in a higher quality product.

The code in version control works. If you commit something that breaks the build, you and your team get the notice immediately and the problem is fixed before anyone else pulls the “broken” code.

Less time deploying. Deploying projects can be very tedious and time-consuming, and automating that process makes perfect sense

Continuous Integration: Requirements

VCS: The first requirement is having the version control system (VCS). There is no way around it and there shouldn't be a way around it. VCS provides a reliable method to centralize and preserve changes made to your project over time. (Git)

CI Server: Continuous integration server (aka build server, aka CI server) is a software tool that centralizes all your CI operations and provides a reliable and stable environment for you to build your projects on.(Jenkins)





Jenkins

Jenkins is an open source automation server written in Java. Jenkins helps to automate the non-human part of software development process, with continuous integration and continuous delivery/continuous deployment.



Deploy Jenkins in Kubernetes

<https://github.com/jenkinsci/kubernetes-cd-plugin/blob/dev/README.md>

<https://medium.com/containerum/configuring-ci-cd-on-kubernetes-with-jenkins-89eab7234270>

<https://kubernetes.io/blog/2018/04/30/zero-downtime-deployment-kubernetes-jenkins/>