AWS Services – EKS & Kubernetes Deployment

For

RAKBANK

Simply Better

aws partner network

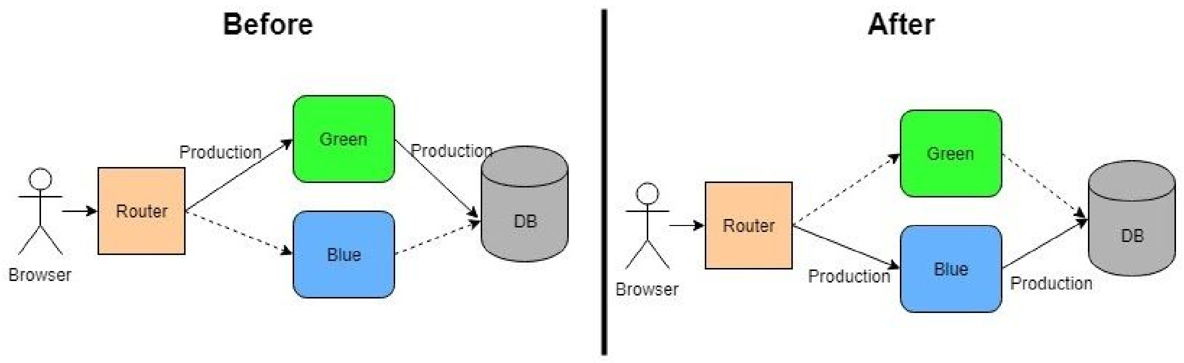
Standard Consulting Partner

Environments for Skiply

1) **UAT & Test** associated to one account.
2) **Production** to another account.

Deployment Strategies in K8S:

1) Recreate: is the simplest deployment strategy. Version 1 of the app is deployed. Deployment starts, all pods running version 1 of the app are deleted. Immediately followed by version 2 of the application being deployed.
2) Rolling Strategy **:** allow Deployments update to take place with **Zero Downtime** by incrementally updating Pods instances with new ones. The new Pods will be scheduled on Nodes with available resources***.***
3) **Blue** / **Green**: **Zero Downtime** There are two similar production environments, Blue & Green. Traffic is routed to one of the two. At deployment, the non-traffic-handling environment is updated and traffic rerouted to that. This allows for a near immediate switch, plus gives the ability to quickly roll back to the previous version. ***(****Preferred Method****)***
4) Canary**:** Update to one pod and update the changes to the others.



The above image is an example of Zero Downtime Blue Green Strategy

Network Setup:

Using the Existing Cluster add a new namespace called pre-prod.
The **URL** maps will be different.

F5 settings will allow only trusted IP address which can open these urls.

Use the below command to update kubeconfig:

```
aws eks --region eu-west-1 update-kubeconfig
--name skiply-prod

Kubectl config view.
Kubectl config current-context.

Kubectl config set-context prepod
--namespace=pre-prod --cluster= skiply-prod
--user=pre-pod-users.
```

Creating Namespaces:

Example Namespace = pre-prod
pre-prod.json

```
{
"apiVersion": "v1",
 "kind": "Namespace",
 "metadata": {
 "name": "pre-prod",
 "labels": {
 "name": "pre-prod" }
} }
```

Create the namespace called pre-prod using the below cmds:

```
Kubectl  create -f  pre-prod.json
```

OR

```
Kubectl  create namespace  pre-prod
```

To list out the namespaces we have created:

```
Kubectl get namespaces --show-labels
```

Kubernetes New Node

When clusters are running in **ACTIVE** state, we will be able to create a Managed Node group.

There is an existing Worker node into which the containers will be deployed.

Create a new Deployment yaml file with services, replication Controllers, & Pods of the existing deployments in K8S.

Use command to deploy to new namespace:

```
Kubectl apply -f newdeployment.yaml -n pre-prod
```

Run the below command to check for all the nodes in kubernetes:

```
Kubectl get nodes --watch
```

Cost(*adding a node*)

Pay $0.10 per hour for each Amazon EKS cluster that you create. You can use a single Amazon EKS cluster to run multiple applications by taking advantage of Kubernetes namespaces and IAM security policies. You can run EKS on AWS using either EC2 or AWS Fargate, and on premises using AWS Outposts.

If you are using EC2 (including with EKS managed node groups), you pay for AWS resources (e.g. EC2 instances or EBS volumes) you create to run your Kubernetes worker nodes and also depending upon the type of EC2

Instances, that is being used.

Infrastructure Changes (*if Necessary*):

Once the namespace called pre-prod has been created, a new **Target Group** in the **Application Load Balancer** needs to be created.

A new Route53 entry is needed for the new domain.
This entry will be added to F5 which will point to the Load Balancer.

Logging / Monitoring:

There are 3 types of Logging in Kubernetes:

a) Basic logging at kubernetes: **`kubectl logs name_of_pod`**

b) Node Level Logging.

```
kubectl logs
```

also check */var/log folder*

c) Cluster Level Logging.

```
kubectl logs counter count-log-1
kubectl logs counter count-log-2
```

where count-log-1, count-log-2 are sidecar containers

The A**WS EKS** has been setup to send logs to cloudwatch events.
An **AWS LAMBDA** Function is triggered as well to log on to **SPLUNK**.

Health checks recommendations:

There are 3 types of health checks:

a) **Readiness:**

We can add the following probe to the deployment yaml
File of the pod:

```
ReadinessProbe:

    Exec:

    Command:

        - cat

        - /tmp/healthy

    initialDelaySeconds: 5

    periodSeconds: 5
```

Using **`kubectl describe pod name_of_pod`** we can check how many probes have failed.

Using **`kubectl get pod name_of_pod`** we can check how many RESTARTS have occurred for this pod.

b) Liveness:

We can add the following probe to the deployment yaml
File of the pod:

```
LivenessProbe:

         httpGet:

            path: /healthz

            port: 8080

            httpHeaders:

            - name: Custom-Header

              value: Awesome
```

```
                 initialDelaySeconds: 3

                 periodSeconds: 3
```

Using **`kubectl describe pod name_of_pod`** we can check how many probes have failed.

Using **`kubectl get pod name_of_pod`** we can check how many RESTARTS have occurred for this pod.

c) Startup:

We can add the following startup probe to the deployment yaml File of the pod:

```
startupProbe:
  httpGet:
    path: /healthz
    port: liveness-port
  failureThreshold: 30
  periodSeconds: 10
```

using **`kubectl describe pod name_of_pod`** we can check how many probes have failed.

Using **`kubectl get pod name_of_pod`** we can check how many RESTARTS have occurred for this pod.

Implementation Documentation:

Start with already deployed containers (Deployment) and service. Let's say we have a deployment file called **SampleDeployment.yaml** file. The content of this file is listed below:

```
apiVersion: apps/v1

kind: Deployment
```

```
metadata:
 name: sample
 namespace: pre-prod
spec:
 replicas: 3
 strategy:
 type: RollingUpdate
 rollingUpdate:
   maxSurge: 1
   maxUnavailable: 25%
 selector:
   matchLabels:
     app: sampleApp
 template:
   metadata:
     labels:
       App: sampleApp
   spec:
     containers:
     - image: ecr.io/sampleApp:2.0
       imagePullPolicy: Always
       name: hello-dep
```

```
          ports:
          - containerPort: 8080
          readinessProbe:
            httpGet:
               path: /
               port: 8080
               initialDelaySeconds: 5
               periodSeconds: 5
               successThreshold: 1
```

1. Deploy the new deployment into a new namspace:

   ```
   Kubectl apply -f SampleDeployment.yaml -n pre-prod
   ```

   Create a new version for your image

   ```
   kubectl set image deployment test
   test=images/image1:1.0.1 — record
   ```

2. Monitor your deployment via Curl job

   ```
     kubectl run curl --image=images/image1:curl -i
   --tty --restart=Never
   ```

   Health Check Status

   ```
   kubectl get pods
   ```

   ```
   kubectl describe pods/pod_name
   ```

3. If health check passes, Start Rolling deployment on the production cluster.

   ```
   kubectl rollout status deployment
   SampleDeployment.yaml
   ```

4. If health check fails, stop and send alerts.