**Blue-Green Deployment with Kubernetes**

Deployment is the process of getting your software available and ready for use. This involves a lot of activities that work in sync to ensure that the software is delivered to the end user and works as expected.

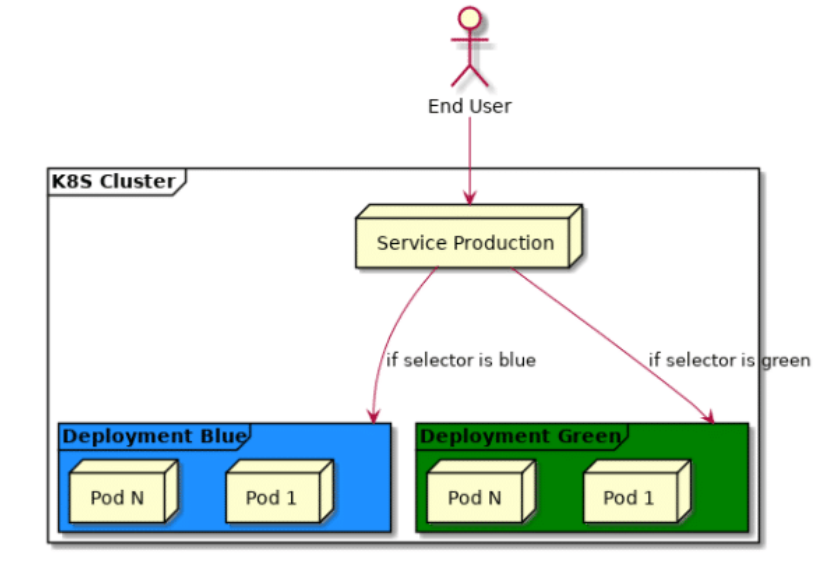
As DevOps Engineers choosing the right technique and tools for to handle deployments is critical. This is because they influence how your software is delivered, how long and at what cost. There is a number of techniques being used in the industry, However, today I would like to talk about — Blue-Green Deployment Technique.

Blue Green Deployment Strategy in Kubernetes:

It is also called as Zero downtime deployment process because in this type of process K8S is not going to delete or replacing exiting pod instead it creates new pod in new environment along with existing deployment.

In this deployment method, two identical production environments work in parallel. One is the currently running production environment. It receives all user traffic (depicted as Blue). The other environment is a clone of it, but idle (Green). Both use the app configuration.

The new version of the application is deployed in the green environment and tested for functionality and performance. Once the testing results are successful, application traffic is routed from blue to green. Green then becomes the new production.



*Based on the selector in the production service, traffic will be routed either to deployment Blue or deployment Green.*

**Advantages of using Blue-Green Deployments**

There are many benefits that come with using this technique when deploying to production.

**Reduced downtime**

Deployments always take time before the system goes live. If you are running a small website or app this may not be a significant challenge because of small traffic. Imagine an online store like Amazon, it receives millions of requests per hour, if it goes offline for an hour that means losing customers and making losses, that’s not something you will want to happen. Blue-Green enables us to deploy to production and route traffic to the new deploy when it’s live and functioning. This means your users will not experience downtime.

**Immediate rollback**

At any instance, one of the environments will have the recent stable release of your software. What happens when you deploy to production and then realize that you missed something important or maybe your app is not persisting sessions?

Imagine logging on Amazon on a black Friday and you notice that awesome geeky gadget you have been saving for all year, is not in your cart! Where is my Gadget amazon! Where is it! It’s not a pleasant experience for your users. So how do you fix this?

Assuming in this situation, the defective environment is Blue, we can route all our traffic back to Green environment, which will have the last stable release and we let our developers fix the bugs on the latest release. Once the bug is fixed, another deployment will be made back to blue and traffic routed back. Simple and easy.

**Preparing for Blue Green Deployment**

Before you implement this technique into your workflow, there are few practices you will have to adopt for a smooth Blue-Green deployment process. These practices provide the ability to scale horizontally easily, and roll back and forth without problems.

**Statelessness**

The application must not persist state to the server. Most applications involve storing user sessions in cookies or tokens. When you are switching from one environment to the other if one user was logged in the Blue environment will loose his/her session once there is a switch to green. This maybe shopping cart data or progress reports etc. You don’t want this to happen? to Avoid this you will have to ensure that cookies, cache files, log files, etc are backed by external services like Memcached, Amazon S3, Redis or Papertail. This means that no matter which environment your production app is live your user will have the same data and user experience.

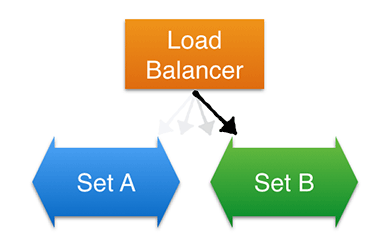
**Infrastructure Automation**  
Automation reduces the frictions and delays that crop up in between development and deployment. It builds upon many ideas that are commonly associated with continuous delivery and Integration. This concept drives more towards the ability to rapidly deploy software to production and get it up and running. One of the challenges of automation is the time it takes to get software from the final stage of testing to live production.

Orchestrating production server can be a lot of work, Time taken to configure the production environment for each deployment is key. Therefore its best at all time to ensure that these processes are automated. Services like containerization e.g Docker and CI pipelines are some of the tools you should look at in this scenario.

**Database and Migrations**

Databases can often be a challenge with this technique, particularly when you need to change the schema to support a new version of the software. Using two production environments may rise to discrepancies. To maintain consistency between two environments it is recommended to use one production database. And what happens when I change schemas in my database? Good Question. This is where migrations come in. With migrations, you can be able to manage your database schema based on new feature and software releases. You can roll back and forth without any challenges.

**Use of Load Balancing Instead of DNS Switching**



Load balancing traffic instead of switching domains

When a new version has been deployed and tested, its promoted to live production and traffic redirected to it. Assuming that our version was deployed to Green environment, when the users visit our domain, they will be redirected to the green environment servers. This can be achieved via DNS switching, however changes to the record sets can take time to resolve and might cause unnecessary delays. As an alternative we can take advantage of load balancers. Load balancers are designed to distribute traffic to instances registered to them. In our case we can register and deregister instances every time we make a deployment to either environments. We will also ensure to register the current deployed version to the load balancer while deregistering the rest. Through load balancing, switching from one environments can be accomplished with less friction and close to zero downtime.

**What tools or services you can use this technique?**

There are many tools out there that you can use to set up this environment. The tools that you may choose will heavily depend on your infrastructure and how your application works. Below are some of tools that can be used to implement successful Blue Green Deployment.

* Docker,
* AWS,
* Cloudfoundry,
* Kubernetes, etc

Here are couple articles you may look at to get you up to speed:

* [Introduction to Blue/Green Deployment on AWS](https://medium.com/aws-activate-startup-blog/upgrades-without-tears-part-1-introduction-to-blue-green-deployment-on-aws-e5bcf90eb60b.9cz7o0gai).
* [Blue-Green Deployment using Containers](https://blog.tutum.co/2015/06/08/blue-green-deployment-using-containers/)
* [Blue/Green Deploys with Kubernetes and Amazon ELB](https://www.citrix.com/blogs/2015/06/16/bluegreen-deploys-with-kubernetes-and-amazon-elb/)
* [Blue-Green Deployment with Cloud Foundry](https://docs.cloudfoundry.org/devguide/deploy-apps/blue-green.html)
* [The DOs and DON’Ts of Blue/Green Deployment](https://cloudnative.io/blog/2015/02/the-dos-and-donts-of-bluegreen-deployment/)

**Conclusion**

Deployments are one of the most important part of software development lifecycle, therefore all the activities involved should thoroughly researched and tested to ensure that they are perfect fit to your system architecture and business.

Blue Green deployments is one of the promising techniques one can employ to deploy the application to production. Just as any other techniques it has its own setbacks, therefore as a DevOps Engineer, it would be wise to carefully evaluate your architecture to ensure you choose and technique thats best suited for your company or business.

I hope this article was informative and was able to give you an overview of what Blue Green Deployment is and how it works. However its not enough information to fully implement a fully functional deployment. The requirements to setting up a successful deployment might differ based on the technologies being used, tools, and services used to host your application. Make sure you do extensive research before settling for this technique as your preferred method of deployment.

Practical demo: <https://github.com/prakashk0301/blue-green-deployment-kubetnetes>

1. Create app.py file

2. Create requirements.txt file

3. Create and push docker image

docker build -t pkw0301/app:1.0 .

docker login

docker push pkw0301/app:1.0

4. Create manifest file for blue environments

vi blue-1.yml

5. Create service manifest file for blue deployment

vi service-1.yaml

6. Deploy blue environment manifest file

kubectl apply -f blue-1.yml

kubectl get deployments

kubectl get pods

7. Create services for blue environments

kubectl apply -f service-1.yaml

kubectl get svc

access from browser: http://<K8s Master IP>:<NodePort Port>/ping

Deploy New version:-

8. Now Let’s change in the code (consider new feature)

in our app.py file, change the response from: Hi, This code is develop by Prakash Kumar: 1

to Hi, this code is developed by Prakash Kumar: 2

9. Now it’s time to build docker image with new tag

docker build -t pkw0301/app:2.0 .

docker login

docker push pkw0301/app:2.0

10. Create manifest file for Green environments

vi green-2.yml

11. Deploy Green environment manifest file

kubectl apply -f green-2.yml --force

kubectl get deployments

kubectl get pods

12. In order to point to Green Environment update service manifest file and update version from 1.0 to 2.0

kubectl apply -f service-1.yaml

13. Reload application URL, now it should point to Green environment.

14. If there is a new change your can repeat the process.

Done!!!