Title: Zero Downtime Deployment for Micro Service architecture

Post Body:

At the moment I'm working on an application which will be based on the **Micro Service architecture**. As main technologies we planned to use **Spring Boot and Docker** for each Micro Service development. One of the goals is to provide Zero Downtime Deployment feature for the users.

I spent some time trying to found some solution and know about `Blue Green Deployment (BGD)` but some aspects is still not clear for me. The main problem is DataBase state and version compatibility.

> For example if `BGD` is used how to migrate all the data changes from Green to Blue contour after successful deployment?

I found interesting approach in Spring's Zero Downtime Deployment with a Database article, but I think that such approach has too complicated Application Versions and Releases Planing process and backward compatibility requirements.

So my I want to ask following questions:

1. Any suggestions on the Zero Downtime Deployment process concept, backed by real experience using it?
2. Is there any Out Of The box solutions (Paid or Free) that provide Zero Downtime Deployment feature for applications with Relational Data Base?

**PS**

*It is interesting how Zero Downtime Deployment works in StackOverflow.com if it is?*

Accepted Answer: None

Highest Rated Answer:

From that article:

- maintain two copies of your production environment (“blue” and “green”)
- route all traffic to the the blue environment by mapping production URLs to it;
- deploy and test any changes to the application in the green environment;
- “flip the switch” by mapping URLs onto green and unmapping them from blue.

It's actually not that difficult if you have the equipment and a good deployment process set up. If you've set up your security mechanisms so that the user doesn't need to start a new session in blue environment, the pain is almost entirely in the provisioning and ensuring the blue environment is in as perfect of a state as you want it to be. After that, it's just a matter of getting into your load balancer's configuration and flipping the settings to point to the blue environment.

But once you have 'green' and 'blue' up and running, you can flip back and forth. Ideally, once blue is verified, immediately upgrade green. Also, make sure you are sharing database operations between the two environments so green can also be a fallback for blue if blue fails.

Granted, my experience planning and helping with this was on Hadoop where you often have a formal data ingestion pipeline that can easily be configured to feed all new data to two completely separate environments.