DevOps

Week 08

Murtaza Munawar Fazal

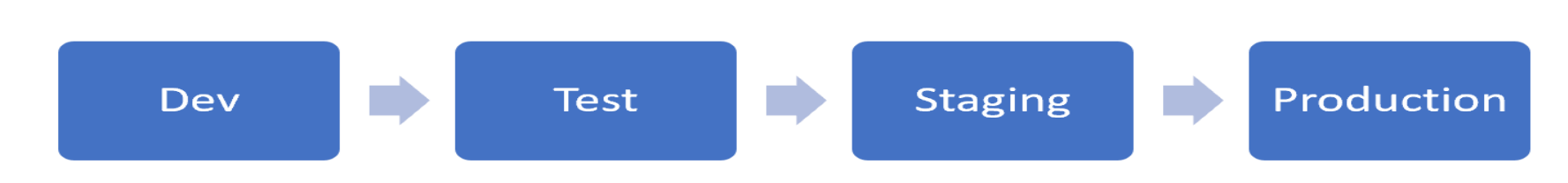
Continuous Delivery

- Continuous Delivery is an extension of Continuous Integration. It's all about getting changes to customers quickly and using sustainable methods.
- Continuous Delivery goes further and changes that pass-through production pipelines are released to customers.
- Continuous Delivery is more than release management.
- Continuous Delivery is all about the process, the people, and the tools that you need to make sure that you can deliver your software on demand.

Continuous Delivery

- Deployment is only one step within the Continuous Delivery process. To deploy on-demand or multiple times a day, all the prerequisites need to be in place.
- For Example
  - Testing strategy
  - Coding practices
  - Architecture

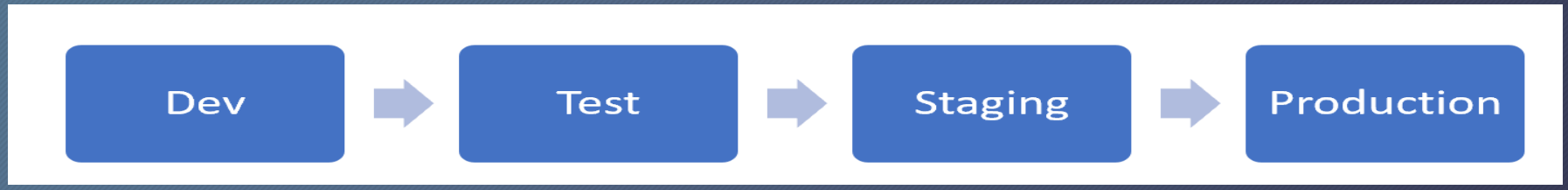
Classical deployment patterns



- The software was built, and when all features had been implemented, the software was deployed to an environment where a group of people could start using it.
- The traditional or classical deployment pattern was moving your software to a development stage, a testing stage, maybe an acceptance or staging stage, and finally a production stage.
- The software moved as one piece through the stages.
- The production release was, in most cases, a Big Bang release, where users were confronted with many changes at the same time.

Classical deployment patterns

- Despite the different stages to test and validate, this approach still involves many risks.
- By running all your tests and validation on non-production environments, it's hard to predict what happens when your production users start using it.
- You can run load tests and availability tests, but in the end, there's no place like production.

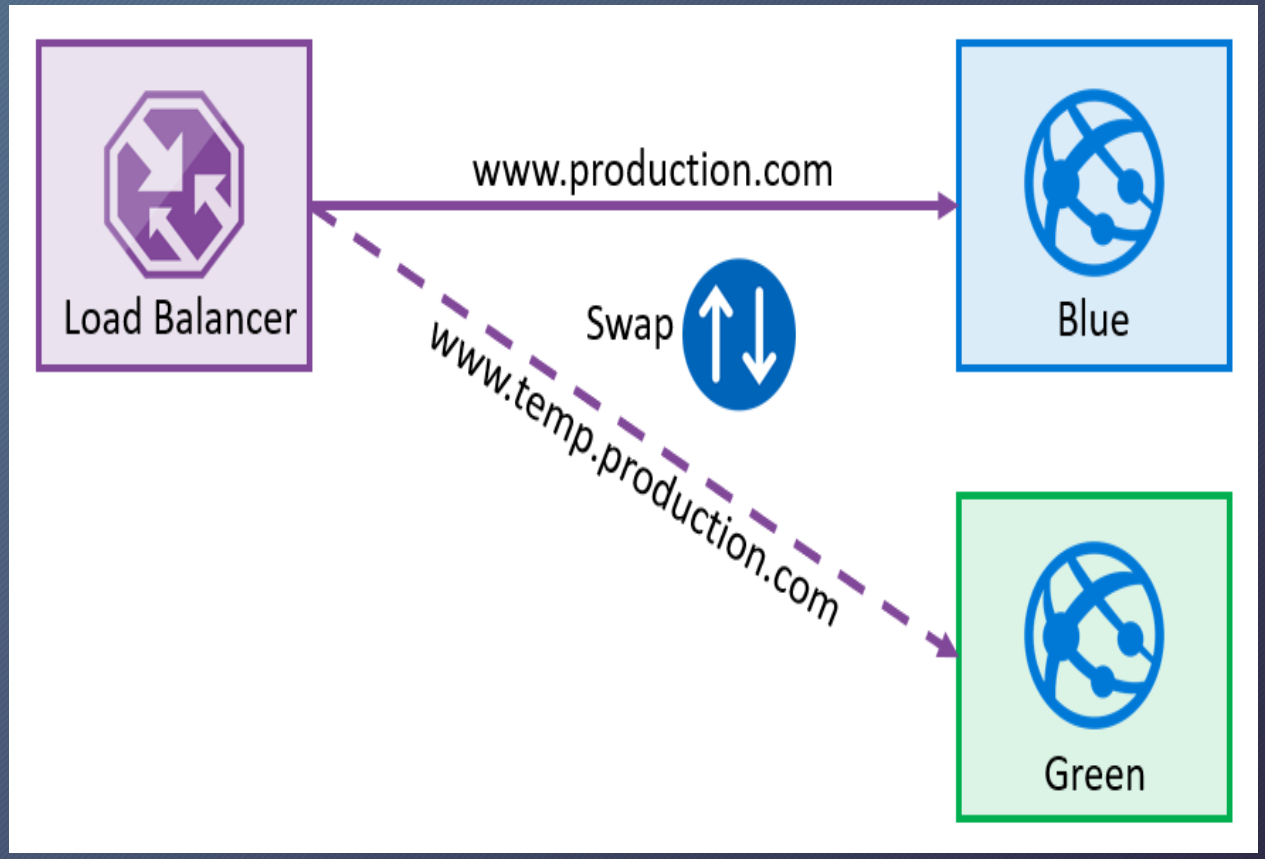


Modern deployment patterns

- End-users always use your application differently. Unexpected events will happen in a data center, multiple events from multiple users will cooccur, triggering some code that hasn't been tested in that way.
- To overcome, we need to embrace that some features can only be tested in production.
- When we take this concept of feature toggling and use it with our deployment patterns, we can test our software in production.
- For example:
  - Blue-green deployments.
  - Canary releases.
  - Dark launching.
  - A/B testing.
  - Progressive exposure or ring-based deployment.
  - Feature toggles.

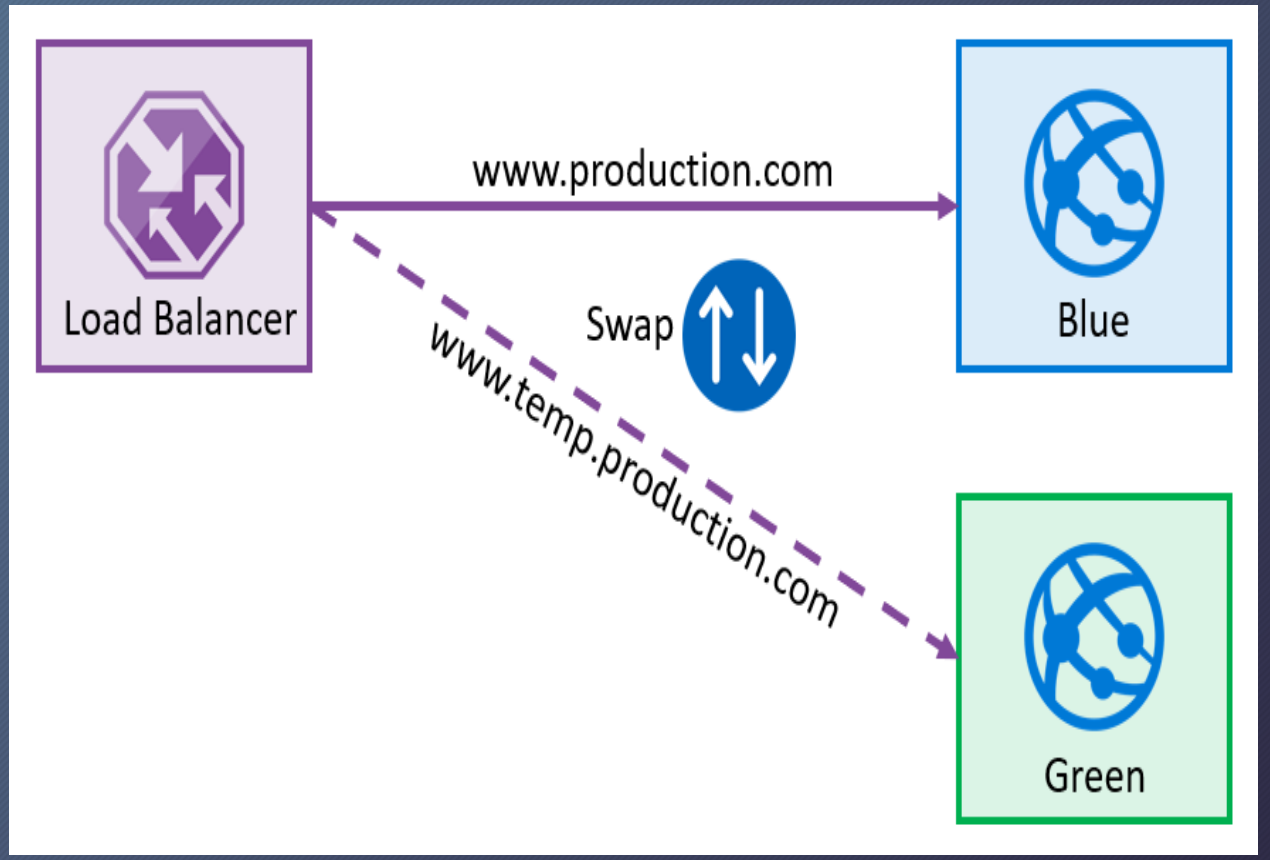
Blue-green deployment

- Blue-green deployment is a technique that reduces risk and downtime by running two identical environments. These environments are called blue and green.
- Only one of the environments is live, with the live environment serving all production traffic.
- For this example, blue is currently live, and green is idle.
- As you prepare a new version of your software, the deployment and final testing stage occur in an environment that isn't live: in this example, green. Once you've deployed and thoroughly tested the software in green, switch the router or load balancer so all incoming requests go to green instead of blue.
- Green is now live, and blue is idle.



Blue-green deployment

- This technique can eliminate downtime because of app deployment. Besides, blue-green deployment reduces risk: if something unexpected happens with your new version on the green, you can immediately roll back to the last version by switching back to blue.
- When it involves database schema changes, this process isn't straightforward. You probably can't swap your application. In that case, your application and architecture should be built to handle both the old and the new database schema.



Deployment slots

- When using a cloud platform like Azure, doing blue-green deployments is relatively easy. You don't need to write your code or set up infrastructure. You can use an out-of-the-box feature called deployment slots when using web apps.

- Deployment slots are a feature of Azure App Service. They're live apps with their hostnames. You can create different slots for your application (for example, Dev, Test, or Stage).

- The production slot is the slot where your live app stays. You can validate app changes in staging with deployment slots before swapping them with your production slot.

- You can use a deployment slot to set up a new version of your application, and when ready, swap the production environment with the new staging environment. It's done by an internal swapping of the IP addresses of both slots.

Feature Toggles

- Feature Flags allow you to change how our system works without making significant changes to the code. Only a small configuration change is required.
- Feature Flags offer a solution to the need to push new code into the trunk and deploy it, but it isn't functional yet.
- Suppose you are modifying the interest calculations. You can change the code so that other users who don't have the Feature Flag set will use the original interest calculation code. The members of your team who are working on the new interest calculations can set to see the Feature Flag will have the new interest calculation code.
- The other type of Feature Flag is a Release Flag. Now, imagine that after you complete the work on the interest calculation code, you're nervous about publishing a new code out to all users at once.
- You have a group of users who are better at dealing with new code and issues if they arise, and these people are often called Canaries.

Feature Toggles

- You change the configuration so that the Canary users also have the Feature Flag set, and they'll start to test the new code as well. If problems occur, you can quickly disable the flag for them again.
- You could have half the users working with the original version of the code and the other half working with the new code version.
- You can then directly compare the outcome and decide if the feature is worth keeping. Feature Flags are sometimes called Feature Toggles instead.
- By exposing new features by just "flipping a switch" at runtime, we can deploy new software without exposing any new or changed functionality to the end-user.
- The question is, what strategy do you want to use in releasing a feature to an end-user.
  - Reveal the feature to a segment of users, so you can see how the new feature is received and used.
  - Reveal the feature to a randomly selected percentage of users.
  - Reveal the feature to all users at the same time.

Feature Toggles

- The idea of separating feature deployment from Feature exposure is compelling and something we want to incorporate in our Continuous Delivery practice.
- It helps us with more stable releases and better ways to roll back when we run into issues when we have a new feature that produces problems.
- By separating deployments from revealing a feature, you make the opportunity to release a day anytime since the new software won't affect the system that already works.
- Feature toggles are a great alternative to branching as well. Branching is what we do in our version control system. To keep features isolated, we maintain a separate branch. When we want the software to be in production, we merge it with the release branch and deploy it. With feature toggles, you build new features behind a toggle. Your feature is "off" when a release occurs and shouldn't be exposed to or impact the production software.

Feature Toggles

- When the switch is off, it executes the code in the IF, otherwise the ELSE.
- You can make it much more intelligent, controlling the feature toggles from a dashboard or building capabilities for roles, users, and so on.
- If you want to implement feature toggles, many different frameworks are available commercially as Open Source.