BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Introduction to DevOps

Sonika Rathi

Assistant Professor
BITS Pilani

Review CS#6

Continuous Integration

- Continuous Integration
- Prerequisites for Continuous Integration Version Control
- Continuous Integration Practices
- Using Continuous Integration Software :: Jenkins
- Artifact Management

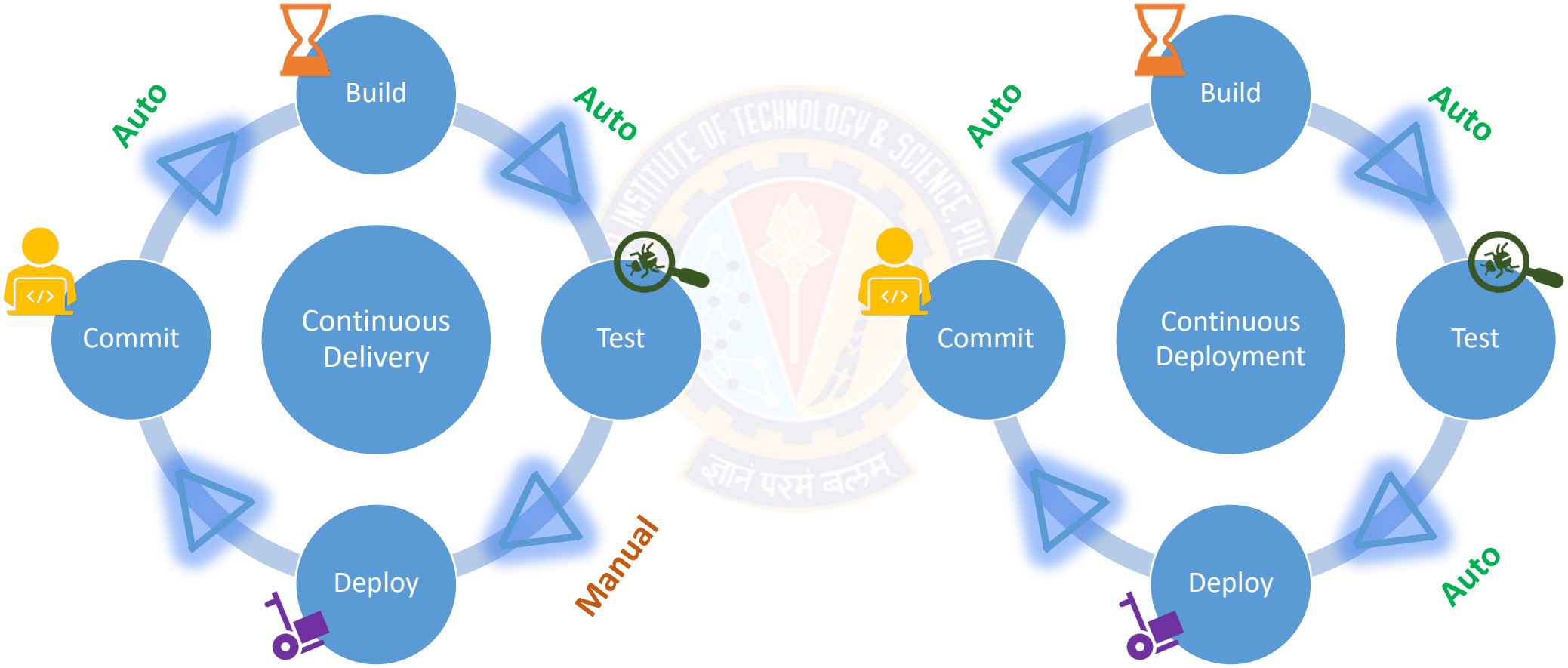
Agenda

Continuous Deployment

- Introduction to Deployment
- Deployment Consideration
- Challenges of Deployment
- Deployment pipeline
- Structure of Deployment Pipeline
- Basic Deployment Pipeline
- Stages of Deployment Pipeline
- Human Free Deployment
- Implementing a Deployment Pipeline
- Rolling back deployment
- Zero-downtime Release
- Deployment Strategies

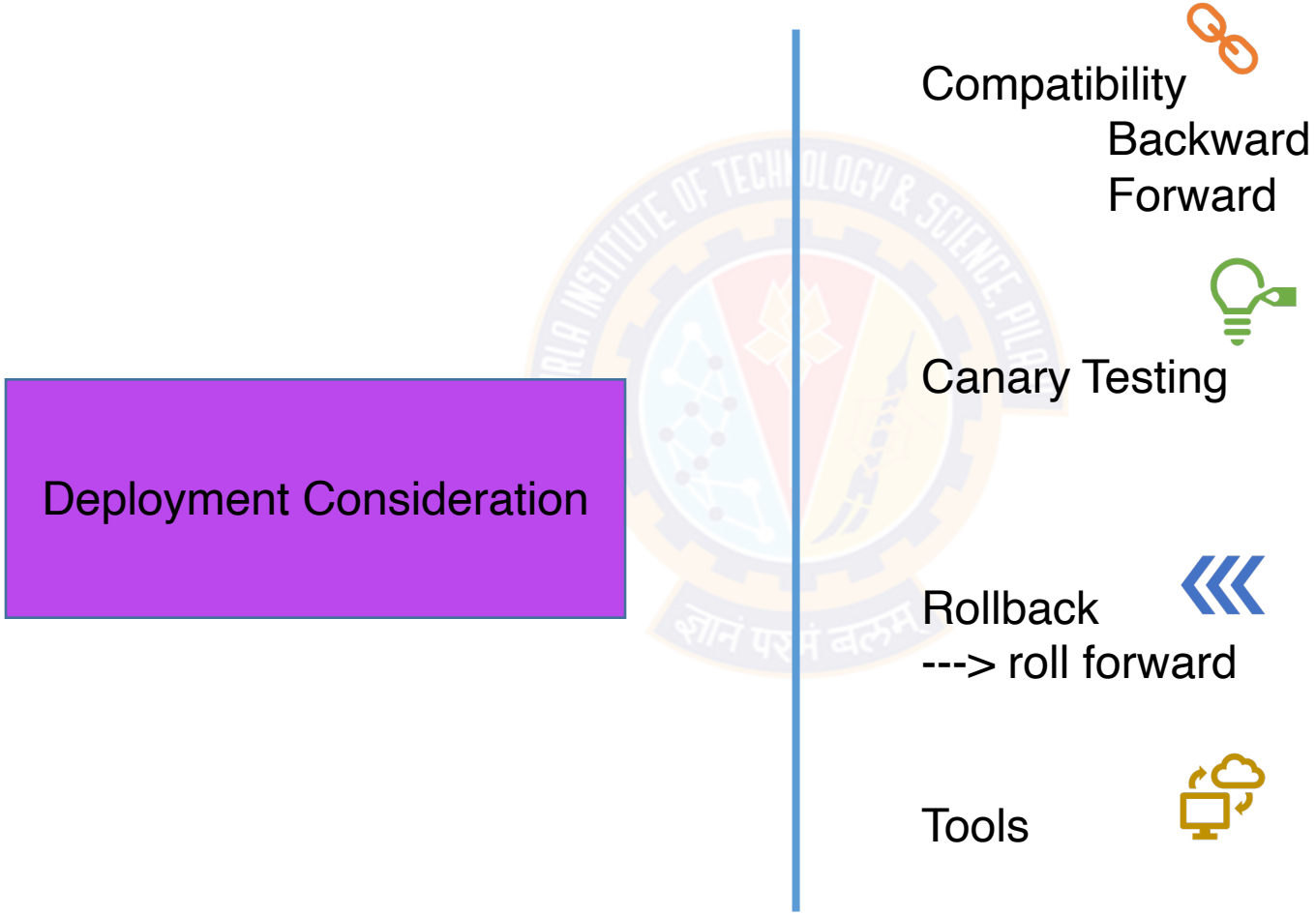
Continuous Deployment

Introduction to CD



Deployment

Considerations



Deployment

Challenges in Deployment

- Deployment Process Waste

  Build and operations teams waiting for documentation or fixes

  Testers waiting for “good” builds of the software

  Development teams receiving bug reports weeks after the team has moved on to new functionality

  Discovering, towards the end of the development process, that the application’s architecture will not support the system’s nonfunctional requirements

This leads to software that is undeployable because it has taken so long to get it into a production-like environment, and buggy because the feedback cycle between the development team and the testing and operations team is so long

Deployment

New Approach

- End-to-End approach to delivering software
- Deployment of Application should be easy & one click to go
- A powerful feedback loop; rapid feedback on both the code and the deployment process
- Lowering the risk of a release and transferring knowledge of the deployment process to the development team
- Testing teams deploy builds into testing environments themselves, at the push of a button
- Operations can deploy builds into staging and production environments at the push of a button
- Developers can see which builds have been through which stages in the release process, and what problems were found
- Automate Build, Deploy, Test and Release System

Note: As a result, everybody in the delivery process gets two things: access to the things they need when they need them, and visibility into the release process to improve feedback so that bottlenecks can be identified, optimized, and removed.
This leads to a delivery process which is not only faster but also safer

Deployment Pipeline

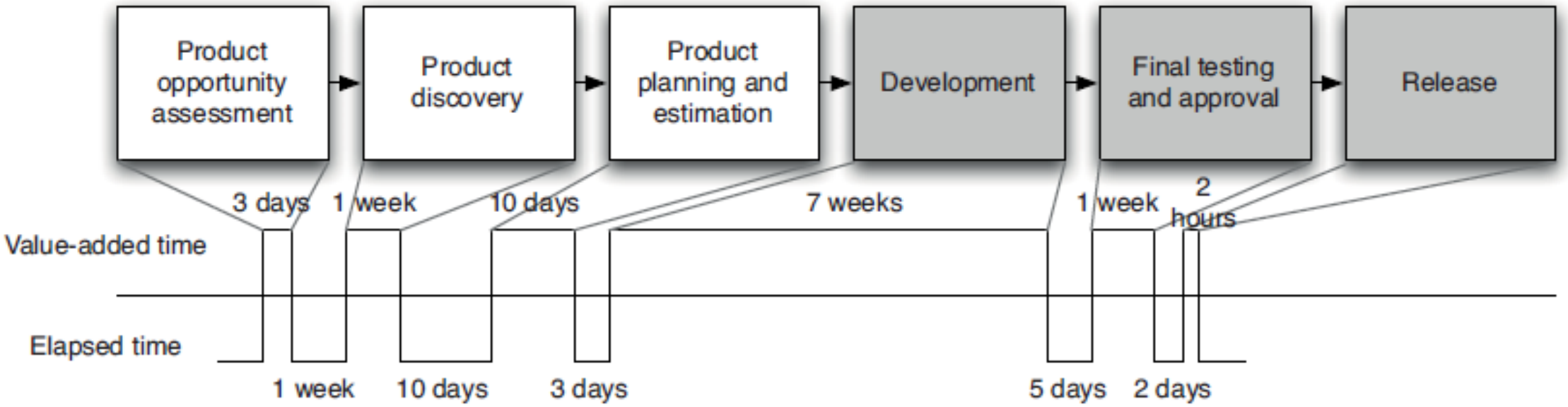
What is Deployment Pipeline

- An automated manifestation of your process for getting software from version control into the production

Increase the collaboration between many individuals

Deployment Pipeline

Value Stream Map of a Product Creation

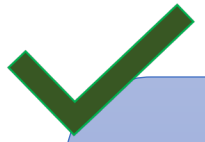


A high-level value stream map for the creation of a new product

Structure of Deployment Pipeline

Expected steps to be followed

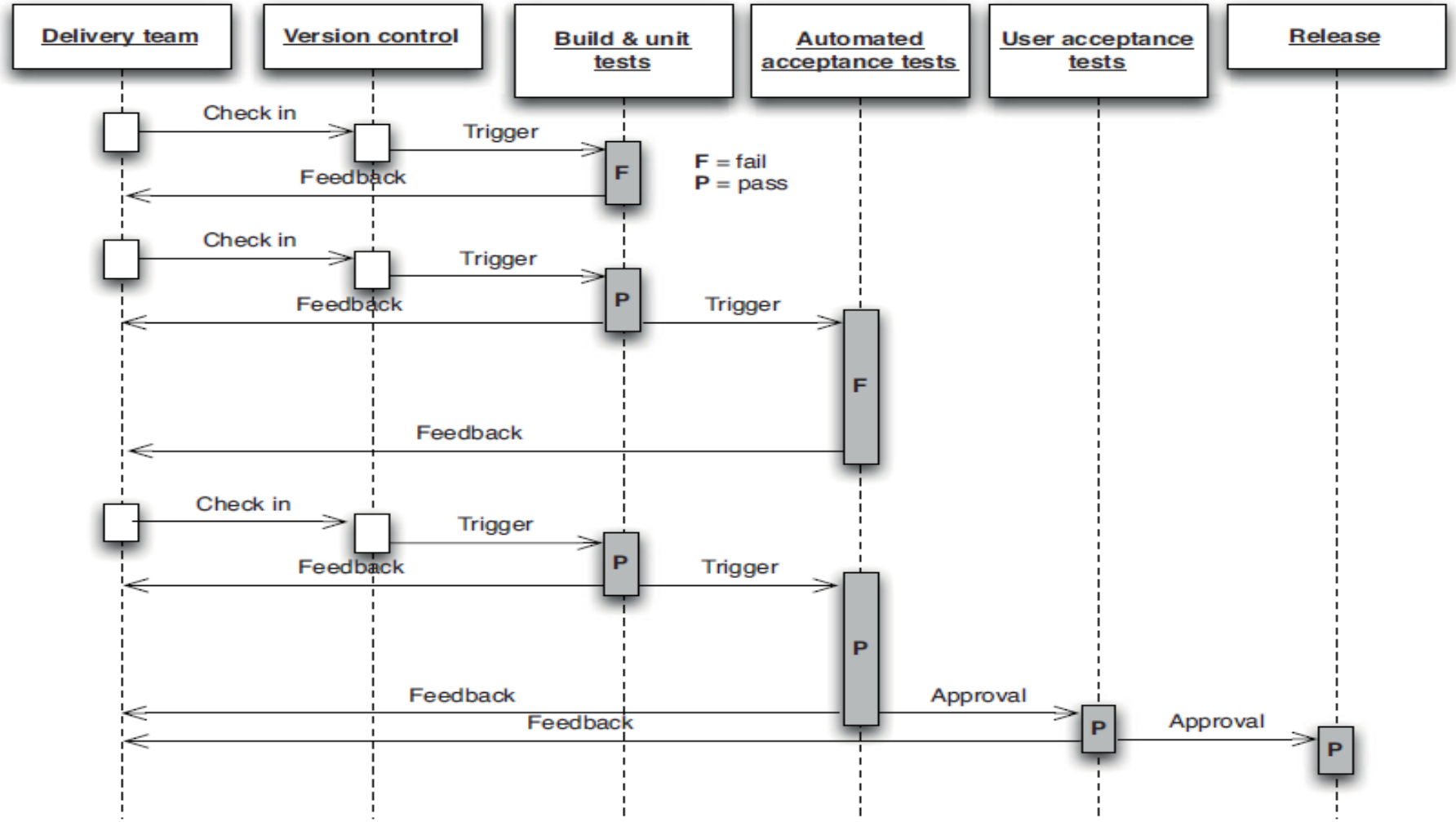
1. The input to the pipeline is a particular revision in version control
2. Every change creates a build
3. It pass through a sequence of tests and challenges to, its viability as a production release



- As the build passes each test of its fitness, confidence in it increases
- The objective is to eliminate unfit release candidates as early in the process as we can and get feedback on the root cause of failure to the team as rapidly as possible
- Any build that fails a stage in the process will not generally be promoted to the next

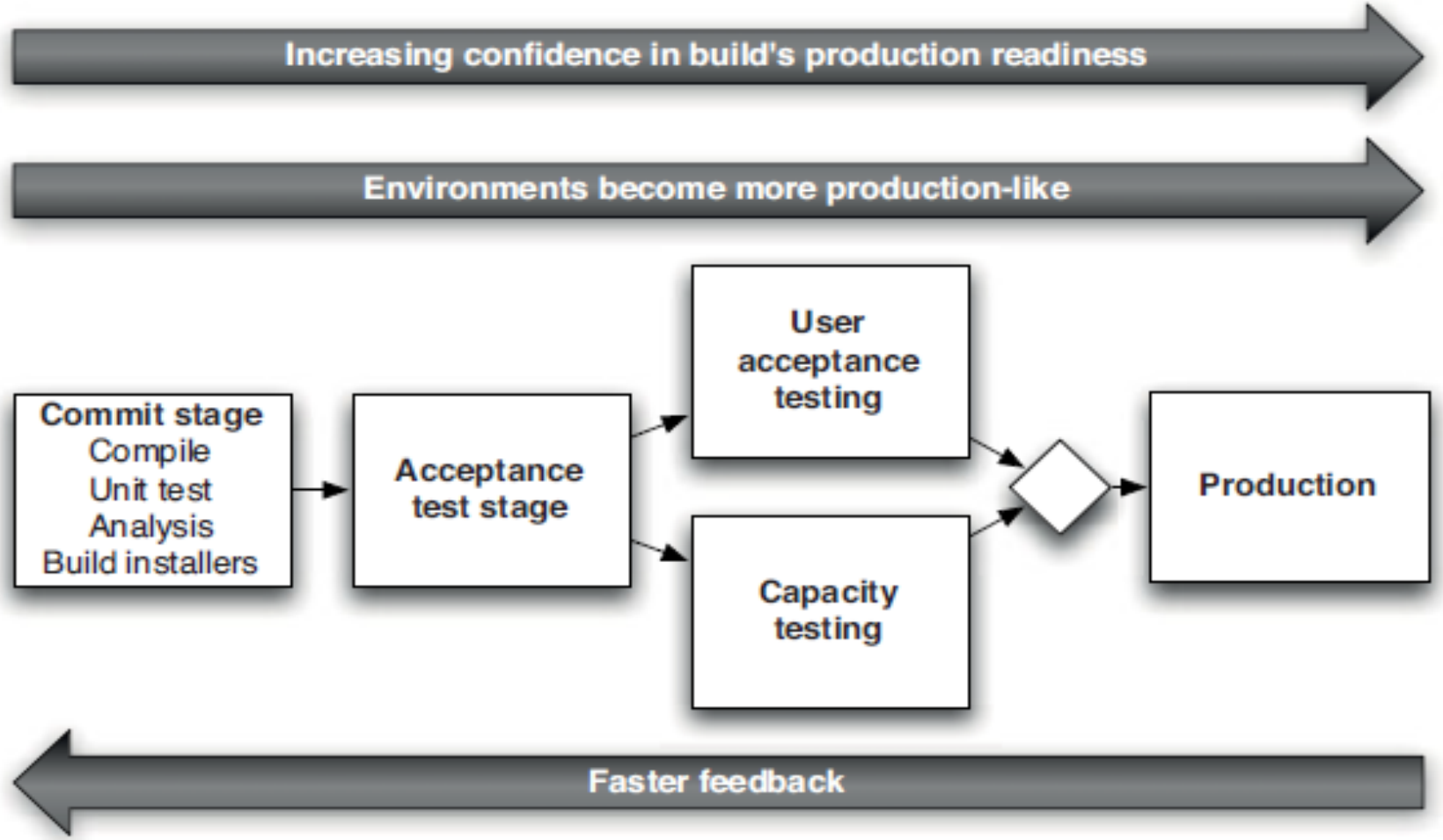
Structure of Deployment Pipeline

Example



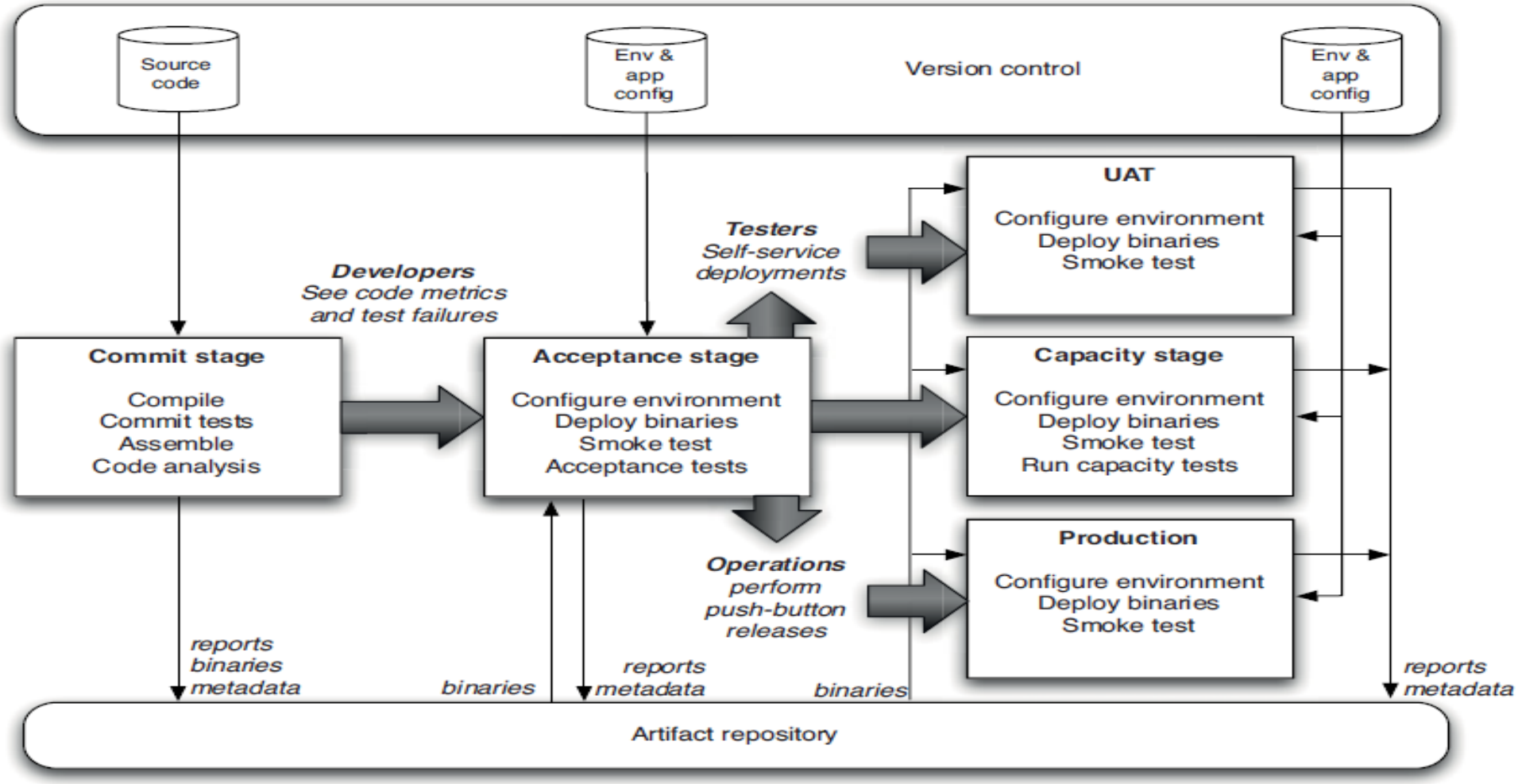
Structure of Deployment Pipeline

Broad Overview



Deployment Pipeline

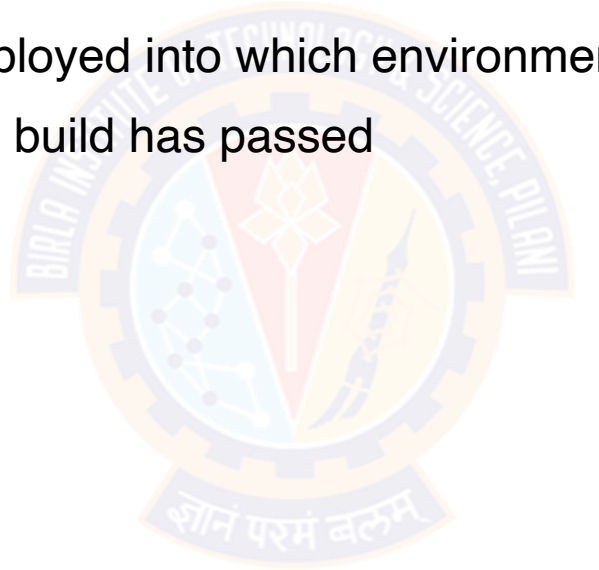
Basic Deployment Pipeline



Basic Deployment Pipeline

Outcome

- The ultimate purpose of all this is to get feedback as fast as possible
- To make the feedback cycle fast
- To be able to see which build is deployed into which environment and
- Which stages in your pipeline each build has passed

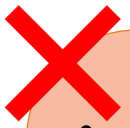


Note: It means that if you see a problem in the acceptance tests (for example), you can immediately find out which changes were checked into version control that resulted in the acceptance tests failing

Deployment Pipeline

Antipatterns of dealing with Binaries

- The source code will be compiled repeatedly in different contexts: during the commit process, again at acceptance test time, again for capacity testing, and often once for each separate deployment target



- Every time you compile the code, you run the risk of introducing some difference
- The version of the compiler installed in the later stages may be different from the version that you used for your commit tests
- You may pick up a different version of some third-party library that you didn’t intend
- Even the configuration of the compiler may change the behavior of the application

Antipatterns of dealing with Binaries

Violates two Principles

Efficiency of Deployment pipeline

- Recompiling violates this principle because it takes time, especially in large systems
- Delayed feedback

Always build upon foundations

- The binaries that get deployed into production should be exactly the same as those that went through the acceptance test process
- Recreation of binaries violates this principle

Deployment Pipeline

Deployment Pipeline Practices

1. Only Build Your Binaries Once
2. Deploy the Same Way to Every Environment

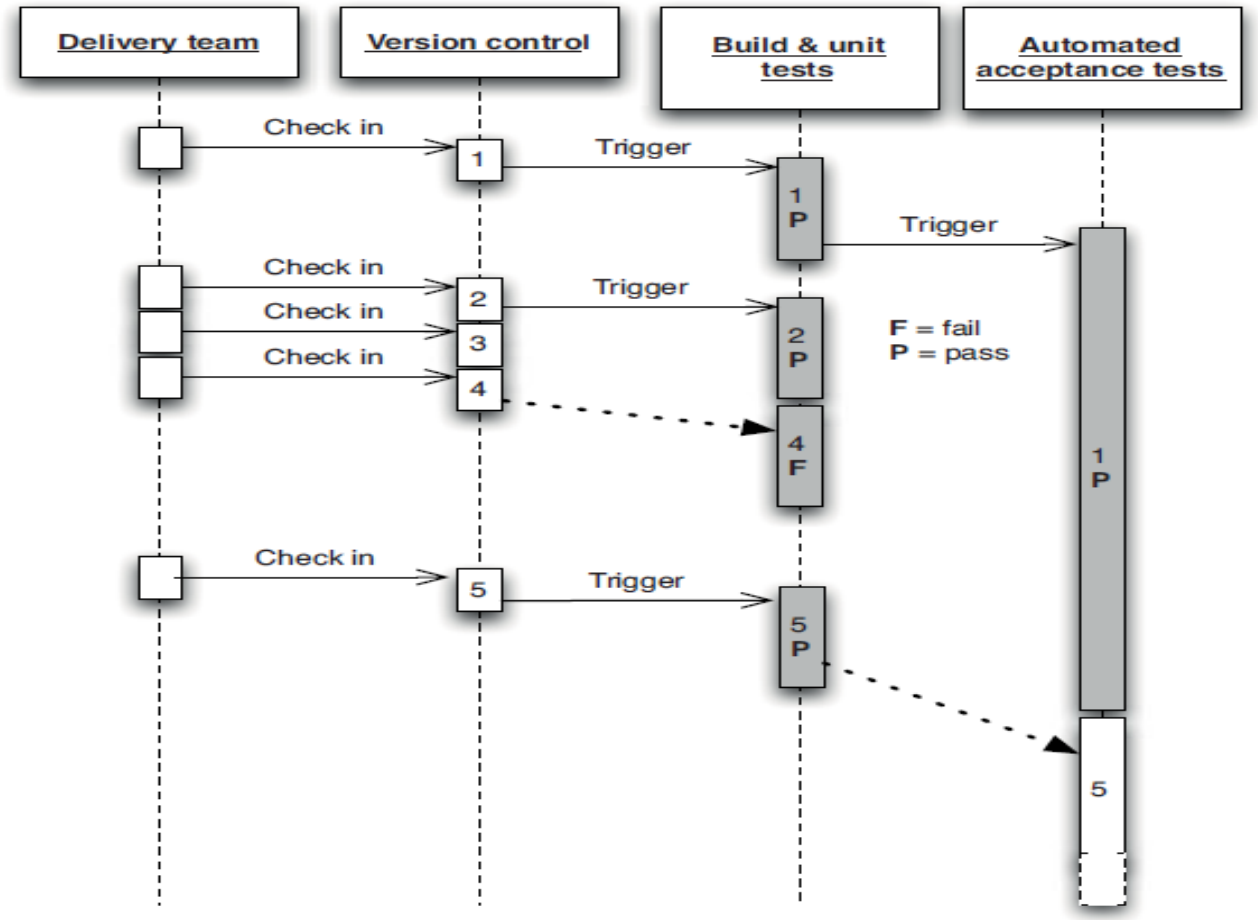
   Note: Using the same script to deploy to production that you use to deploy to development environments is a fantastic way to prevent the "it works on my machine" syndrome

3. Smoke-Test Your Deployments
4. Deploy into a Copy of Production
5. Each Change Should Propagate through the Pipeline Instantly
6. Intelligent pipeline scheduling
7. If Any Part of the Pipeline Fails, Stop the Line

Deployment Pipeline Practices

Intelligent scheduling

- Intelligent scheduling is crucial to implementing a deployment pipeline



Deployment Pipeline

Stages in Deployment Pipeline

- The Commit Stage
- The Automated Acceptance Gate
- Subsequent Test Stages
- Preparing to Release

Preparing to Release

Release Plan

- Have a release plan that is created and maintained by everybody involved in delivering the software, including developers and testers, as well as operations, infrastructure, and support personnel
- Minimize the effect of people making mistakes by automating as much of the process as possible
- Practice the procedure often in production-like environments, so you can debug the process and the technology supporting it
- Have the ability to back out a release if things don't go according to plan
- Have a strategy for migrating configuration and production data as part of the upgrade and rollback processes

Note: Goal is a completely automated release process, Releasing should be as simple as choosing a version of the application to release and pressing a button and Backing out should be just as simple

Human Free Deployments

Automating Deployment and Release

- Reduces issues due manual mistakes
- Audit logs

Human Free Deployments

Benefits

- With automated deployment and release, the process of delivery becomes available to everyone
- Developers, testers, and operations teams no longer need to rely on ticketing systems and email threads to get builds deployed so they can gather feedback on the production readiness of the system
- Testers can decide which version of the system they want in their test environment without needing to be technical experts themselves, nor relying on the availability of such expertise to make the deployment
- Sales people can access the latest version of the application with the killer feature that will swing the deal with a client
- An important reason for the reduction in risk is the degree to which the process of release itself is rehearsed, tested, and perfected
- Since you use the same process to deploy your system to each of your environments and to release it, the deployment process is tested very frequently—perhaps many times a day

Implementing a Deployment Pipeline

The steps to implement a deployment pipeline

- Model your value stream and create a walking skeleton
- Automate the build and deployment process
- Automate unit tests and code analysis
- Automate acceptance tests
- Automate releases

Deployment Consideration

Rolling Back Deployments

- It is essential to roll back a deployment in case it goes wrong
- As, debugging problems in a running production environment is almost certain to result in late nights
- a way to restore service to your users when things go wrong, so you can debug the failure in the comfort of normal working hours
- There are several methods of performing a rollback
  - blue-green deployments and canary releasing can also be used to perform zero-downtime releases and rollbacks

Rolling Back Deployments

Rolling Back Deployments Constraints

- Data : If your release process makes changes to your data, it can be hard to roll back

- There are two general principles you should follow when creating a plan for rolling back a release
- Ensure that the state of your production system, including databases and state held on the filesystem, is backed up before doing a release
- The second is to practice your rollback plan, including restoring from the backup or migrating the database back before every release to make sure it works

Deployments

Zero-Downtime Releases

- Hot deployment, in which the process of switching users from one release to another happens nearly instantaneously
- It must also be possible to back users out to the previous version nearly instantaneously too, if something goes wrong
- The key to zero-downtime releases is decoupling the various parts of the release process so they can happen independently as far as possible

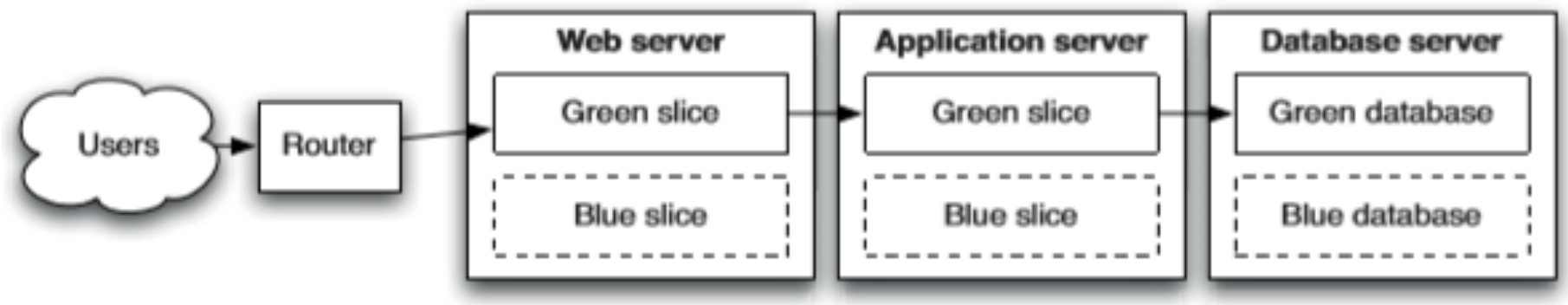
Deployment

Deployment Strategies

Blue-Green Deployments

Introduction

- This is one of the most powerful techniques we know for managing releases
- The idea is to have two identical versions of your production environment, which we'll call blue and green



Blue-Green Deployments

Challenges

- It is usually not possible to switch over directly from the green database to the blue database because it takes time to migrate the data from one release to the next if there are schema changes

Solutions:
1) Put the application into read-only mode shortly before switchover
2) Another approach is to design your application so that you can migrate the database independently of the upgrade process

Blue-Green Deployments

Setup

blue and green environments can be completely separate replicas of each other

you can only afford a single production environment

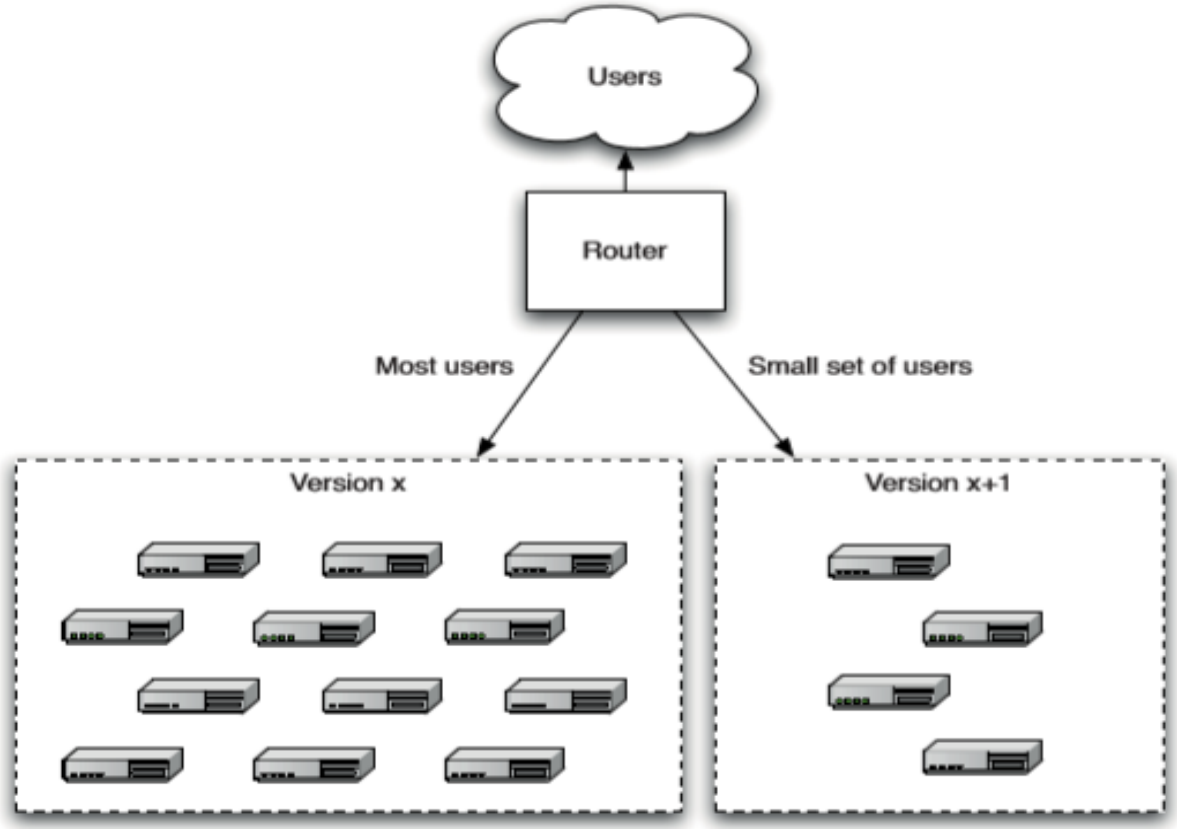
have two copies of your application running side by side on the same environment (with its own resources like port, filesystem etc.,)

Use virtualization

Canary Releasing

Introduction

- Involves rolling out a new version of an application to a subset of the production servers to get fast feedback
- Like a canary in a coal mine, this quickly uncovers any problems with the new version without impacting the majority of users



Canary Releasing

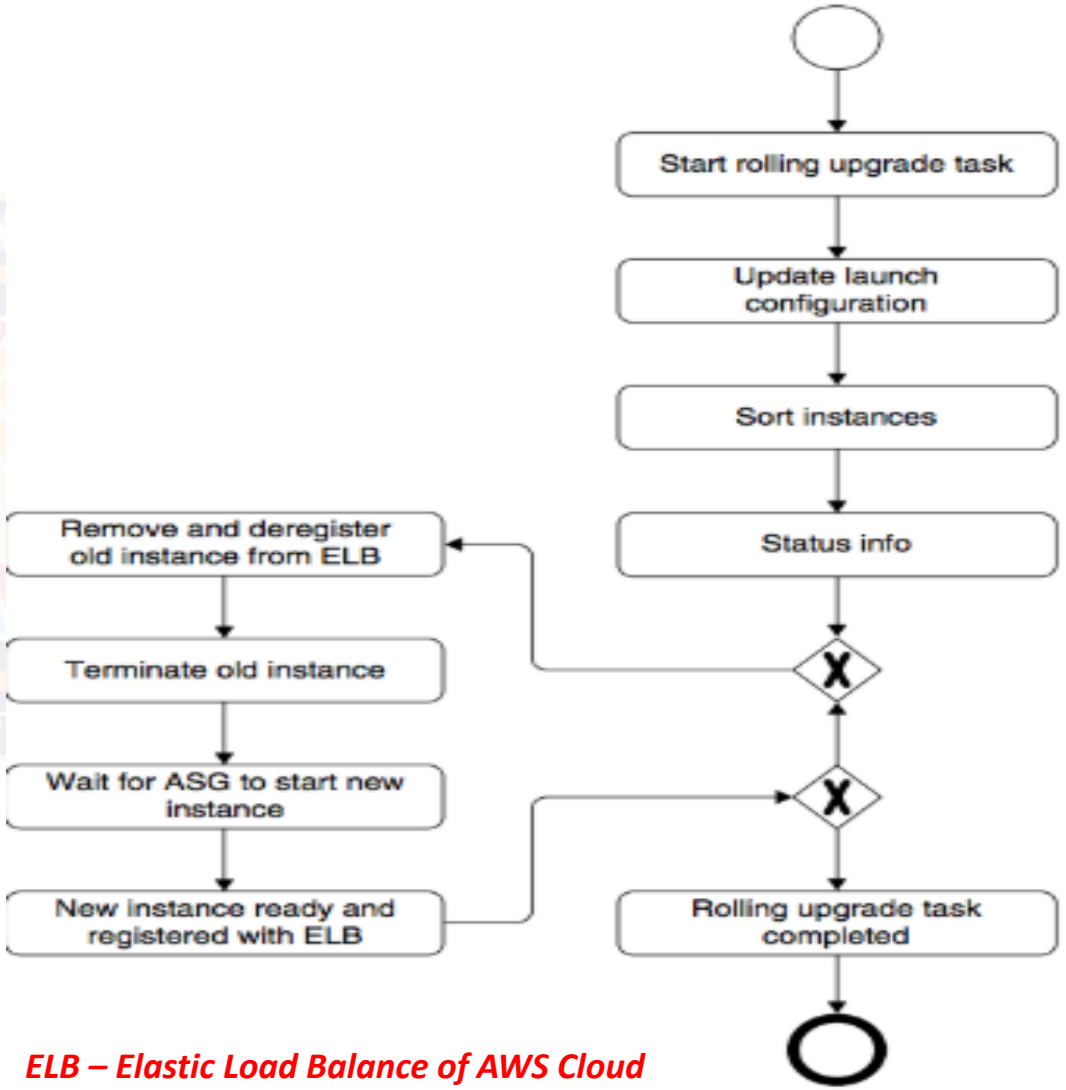
Benefits

- It makes rolling back easy: Just stop routing users to the bad version, and you can investigate the logs at your freedom
- You can use it for A/B testing by routing some users to the new version and some to the old version
  - Some companies measure the usage of new features, and kill them if not enough people use them
  - Others measure the actual revenue generated by the new version, and roll back if the revenue for the new version is lower
- You can check if the application meets capacity requirements by gradually ramping up the load, slowly routing more and more users to the application and measuring the application's response time and metrics like CPU usage, I/O, and memory usage, and watching for exceptions in logs

Rolling upgrade

Introduction

- A rolling upgrade consists of deploying a small number of new version systems at a time directly to the production environment
- In this deployment simultaneously you turn off the old version system
- Before you remove the original system; make sure the new version system is serving the purpose



ELB – Elastic Load Balance of AWS Cloud

Rolling upgrade

Benefits

- Cost Effective
- Risk Effective

Emergency Fixes

Best Practice for Emergency Fixes

- Run every emergency fix through your standard deployment pipeline
- You should always consider how many people the defect affects, how often it occurs, and how severe the defect is in terms of its impact on users
- Some considerations to take into account when dealing with a defect in production:
    - Never do them late at night, and always pair with somebody else
    - Make sure you have tested your emergency fix process
    - Only under extreme circumstances bypass the usual process for making changes to your application
    - Make sure you have tested making an emergency fix using your staging environment
    - Sometimes it's better to roll back to the previous version than to deploy a fix, do some analysis to work out what the best solution is

Deployment

Tips and Tricks

- The People Who Do the Deployment Should Be Involved in Creating the Deployment Process
  - Things Go Better When Development and Operations Are Friends
- Log Deployment Activities
  - If your deployment process isn't completely automated, including the provisioning of environments, it is important to log all the files that your automated deployment process copies or creates
- Don't Delete the Old Files, Move Them
- Deployment Is the Whole Team's Responsibility
  - A "build and deployment expert" is an antipattern. Every member of the team should know how to deploy, and every member of the team should know how to maintain the deployment scripts

References

Text Book Mapping

- Text Book 1: DevOps: A Software Architect's Perspective (SEI Series in Software Engineering) by Len Bass, Ingo Weber, Liming Zhu , Publisher: Addison Wesley (18 May 2015) : Chapter 6 : Deployment
- Text Book 2: Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation by Jez Humble, David Farley. Publisher: Addison Wesley, 2011 : Chapter 5 : Anatomy of the Deployment Pipeline

Q&A

Thank You!

In our next session: