High-Level Program Design

DevOps Accelerator

GENERAL ASSEMBLY

Scale Engineering Capabilities

DevOps Accelerator

Instructor-Led Onsite or Remote ~35 Hours

Overview:
Optimize your delivery pipeline by training engineers in Agile and XP workflows, CI/CD, testing, and automation to ensure seamless deployments.

Prerequisites:
Students must be comfortable building Java/Spring applications.

Business Outcomes:.
- Upskill engineers to tackle technology projects at the enterprise level.
- Optimize your delivery pipeline by leveraging Agile and XP workflows.
- Avoid downtime by having the right talent in place to log and monitor production environments.

~35 Hours of DevOps Training

Agile and XP Workflows	Containerization with Docker	Test Driven Development	Continuous Integration	Logs and Monitoring
Use Agile and XP workflows to manage development workflows more efficiently.	**Leverage containers** and understand how they fit into modern development workflows.	**Increase test coverage** for your code to build and maintain confidence with unit and integration tests.	**Automate your testing and deployments** by using continuous integration pipelines in Jenkins.	**Maximize uptime** in your production environments and ensure any errors are detected quickly.

Why DevOps Accelerator?

- **Validated tools and approaches** for DevOps processes, workflows, and tools, developed in partnership with top organizations.
- **Built with subject matter experts** with experience in DevOps, Java development, and Agile practices.
- 35 hours of expert-led, hands-on learning, including:
    - **Projects, labs, and assignments** that mimic real-world tasks and workflows.
    - **Case studies and examples** that demonstrate how businesses use the concepts.
    - **Regular feedback and touchpoints** with instructors and peers to ensure students meeting learning goals.

Learner Personas for DevOps Accelerator

This product is specifically designed for the following audience:

- **Early- to mid-level Java engineers** with experience developing applications with the Spring framework (2+ years of experience preferred).

Anatomy of DevOps Accelerator

Unit	What's Covered	Unit Lab
Unit 1: **Agile and Extreme Programming** **(10 hours)** Use Agile and XP workflows to manage development workflows more efficiently.	● DevOps basics. ● Agile principles and methods. ● Scrum roles and ceremonies. ● User stories and acceptance criteria. ● Extreme programming principles.	Write user stories and acceptance criteria for a mock application.
Unit 2: **Containerization** **(10 hours)** Use containers and understand how they fit into modern development workflows.	● How Docker works. ● Dockerizing a Spring Boot application. ● Running containers on Heroku. ● How Kubernetes works. ● Running Kubernetes with Minikube.	Deploy a Spring Boot application to Kubernetes.

Anatomy of DevOps Accelerator

Unit	What's Covered	Unit Lab
Unit 3: **Test-Driven Development** **(8 hours)** Increase test coverage for your code to build and maintain confidence with unit and integration tests.	● Test driven development principles. ● Types of application testing. ● Unit testing with jUnit. ● Application testing with mocks and stubs. ● Testing a Spring Boot application.	Use jUnit to write automated unit tests.
Unit 4: **Continuous Integration and Delivery** **(4 hours)** Automate your testing and deployments by using continuous integration pipelines in Jenkins.	● Continuous integration and delivery basics. ● Integration testing. ● Building Jenkins pipelines. ● Jenkins plugins: JMeter and Performance.	Create a Jenkins pipeline for a Java application and test the application using JMeter.

Anatomy of DevOps Accelerator

Unit	What’s Covered	Unit Lab
Unit 5: **Logs and Monitoring** **(8 hours)** Maximize uptime in your production environments and ensure any errors are detected quickly.	● Logging and monitoring basics. ● Key monitoring metrics. ● Using the ELK stack. ● Querying and reading errors and messages.	Set up logging and monitoring tools for a Spring Boot application.

Tools Used in the Course

- JDK 8
- The course is written for MacOS; if students or instructors use a Windows or Linux system, some modifications to the curriculum may have to be made.
- Git and Github Enterprise
- IntelliJ IDEA Community Edition
- Zoom, Slack, and Google Chrome

DevOps Accelerator

What You’ll Learn

Module Overview

10 hours

Agile and Extreme Programming

Why?

Many DevOps practices and approaches are rooted in Agile and Extreme Programming (XP) thinking. In this unit, students will explore key Agile and XP concepts to lay a strong foundation for working in a DevOps environment.

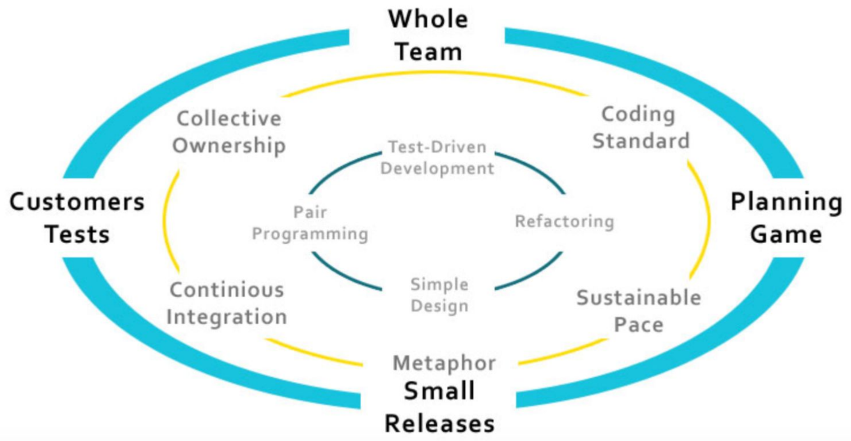
Learning Objectives

- Define the core principles of Agile.
- Differentiate between Agile versions.
- Explain the different Scrum roles.
- Identify the different stages and rituals of a sprint cycle.
- Explain the value of user stories and acceptance criteria.
- Write an effective user story and accompanying acceptance criteria.
- Describe extreme programming.
- Explain the value of extreme programming to software development.

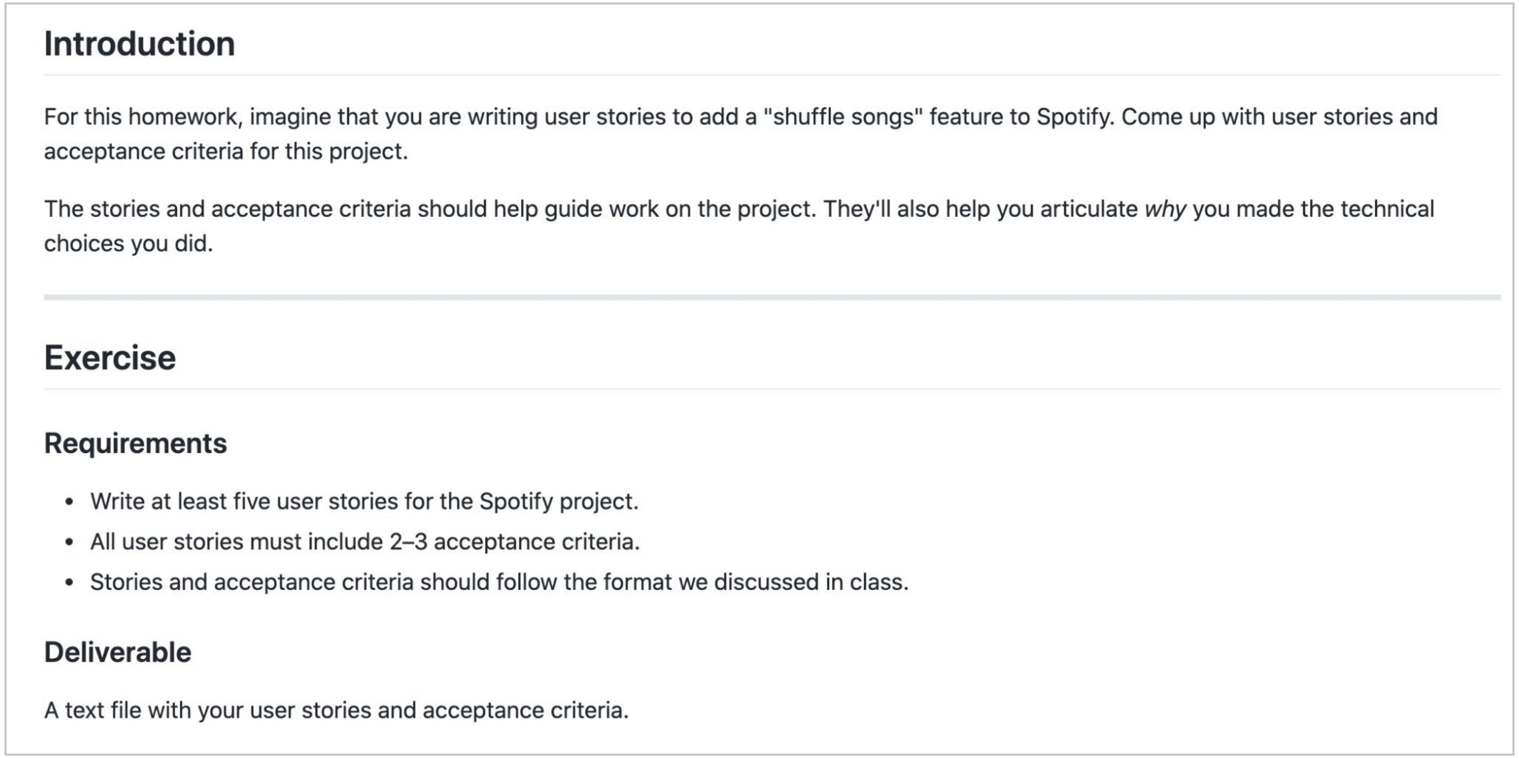
Introduction to Extreme Programming

Extreme programming stresses customer satisfaction and emphasizes teamwork. It's closely associated with TDD (test-driven development) and characterized by:

- Early, iterative releases of software.
- An incremental planning approach.
- Flexible scheduling.
- Automated testing.
- Frequent customer feedback.
- Teams made up of managers, customers, and developers.



Writing User Stories



Introduction

For this homework, imagine that you are writing user stories to add a "shuffle songs" feature to Spotify. Come up with user stories and acceptance criteria for this project.

The stories and acceptance criteria should help guide work on the project. They'll also help you articulate *why* you made the technical choices you did.

Exercise

Requirements

- Write at least five user stories for the Spotify project.
- All user stories must include 2–3 acceptance criteria.
- Stories and acceptance criteria should follow the format we discussed in class.

Deliverable

A text file with your user stories and acceptance criteria.

Module Overview

10 hours

Containerization

Why?

Containerization, the process of packaging code, dependencies, and configuration, has become the new norm in DevOps. In this unit, students will use Docker and Kubernetes to build, run, and orchestrate applications in containers.

Learning Objectives

- Differentiate between VMs and containers.
- Identify when and when not to use Docker.
- Pull an image from Docker Hub and run a container on your machine.
- Run source code as a container volume.
- Push a Docker image to production in Heroku.
- Explain Kubernetes and key terms.
- Run Kubernetes locally with Minikube.

How Docker Works

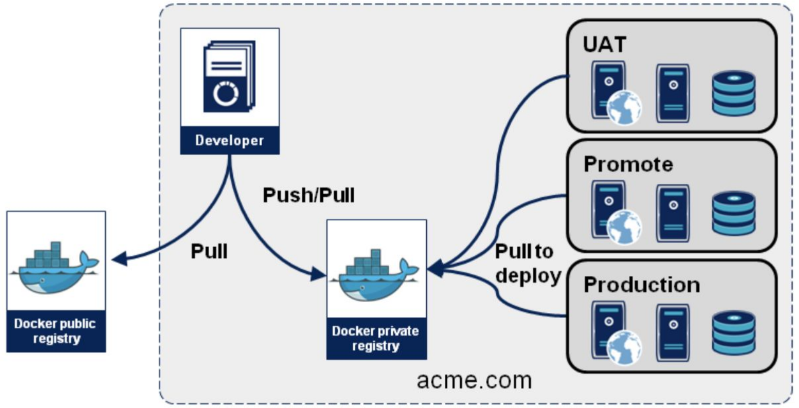
Docker is an open-source platform (the most popular one) for building applications using containers.

However, Docker isn't a completely new technology. Many of the components and principles existed previously. Docker is written in Go and takes advantage of several features of the Linux kernel to deliver its functionality, including namespaces and cgroups (more on those in the Additional Resources section).

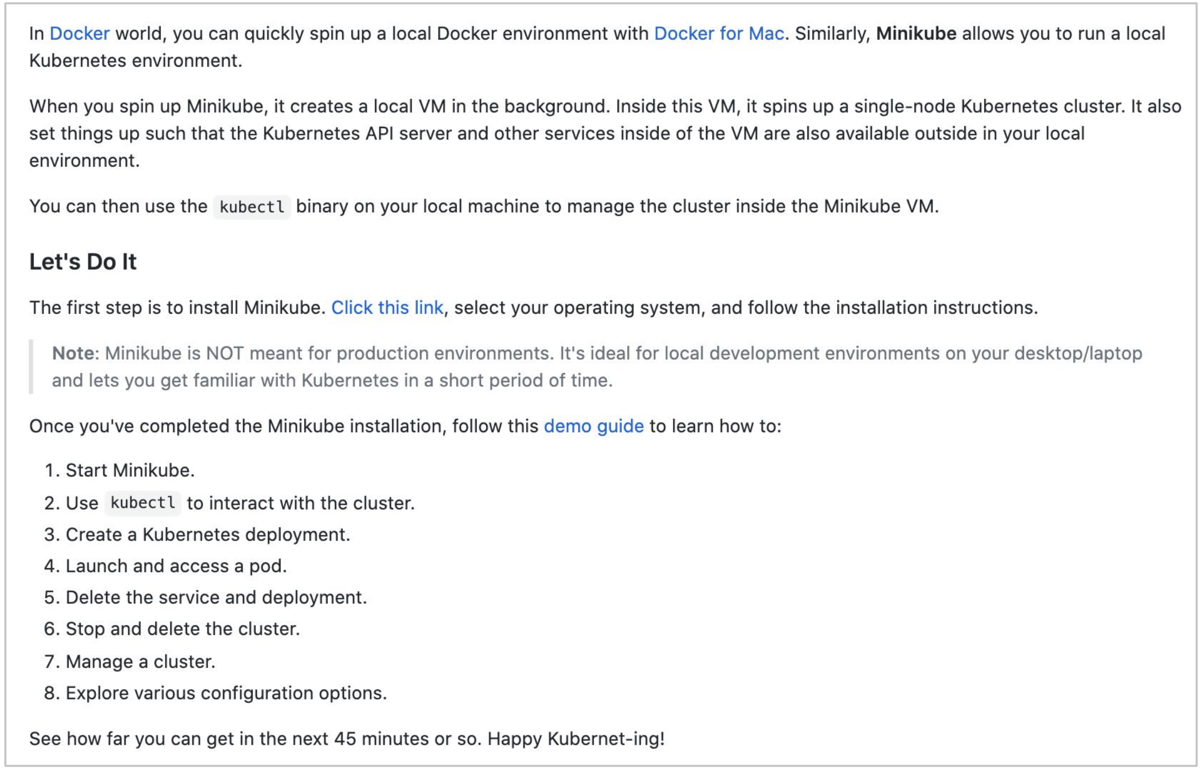
The ultimate goal of Docker is to mirror our dev environment with our production environment. This is mostly useful for your back-end but can be applied to your front-end applications as well! It's cool to run Docker locally, but its real benefit comes into play while running it on some production machine.

Take a look at this *very* simplified version of how Docker works.

> **Knowledge Check**: Can someone explain what's going on here?



Running Kubernetes Locally With Minikube



In Docker world, you can quickly spin up a local Docker environment with Docker for Mac. Similarly, **Minikube** allows you to run a local Kubernetes environment.

When you spin up Minikube, it creates a local VM in the background. Inside this VM, it spins up a single-node Kubernetes cluster. It also set things up such that the Kubernetes API server and other services inside of the VM are also available outside in your local environment.

You can then use the `kubectl` binary on your local machine to manage the cluster inside the Minikube VM.

Let's Do It

The first step is to install Minikube. Click this link, select your operating system, and follow the installation instructions.

> **Note**: Minikube is NOT meant for production environments. It's ideal for local development environments on your desktop/laptop and lets you get familiar with Kubernetes in a short period of time.

Once you've completed the Minikube installation, follow this demo guide to learn how to:

1. Start Minikube.
2. Use `kubectl` to interact with the cluster.
3. Create a Kubernetes deployment.
4. Launch and access a pod.
5. Delete the service and deployment.
6. Stop and delete the cluster.
7. Manage a cluster.
8. Explore various configuration options.

See how far you can get in the next 45 minutes or so. Happy Kubernet-ing!

Module Overview

8 hours

Test-Driven Development

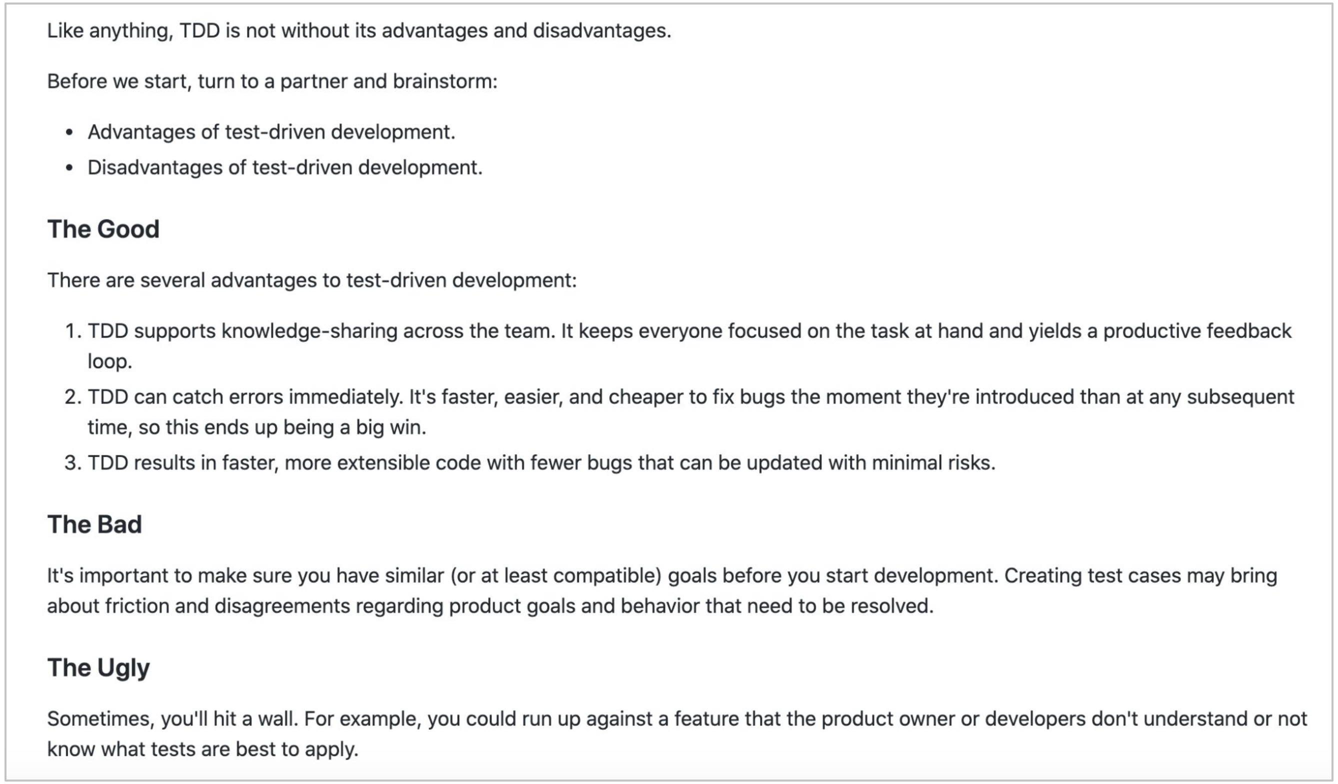
Why?

The goal of test-driven development (TDD) is to deliver high-quality code as quickly and efficiently as possible. If that sounds similar to the goal of DevOps, that's because it is! In this unit, students will learn how TDD works and how to apply it to Spring Boot applications.

Learning Objectives

- Articulate the benefits and drawbacks of test-driven development.
- Add conditional methods to a test case.
- Perform a JUnit test.
- Differentiate between mocking and stubbing and when you'd use each.
- Mock dependencies in order to test a Spring Boot application.
- Use stubbing to test a Spring Boot application.

TDD: The Good, the Bad, the Ugly



Like anything, TDD is not without its advantages and disadvantages.

Before we start, turn to a partner and brainstorm:

- Advantages of test-driven development.
- Disadvantages of test-driven development.

The Good

There are several advantages to test-driven development:

1. TDD supports knowledge-sharing across the team. It keeps everyone focused on the task at hand and yields a productive feedback loop.
2. TDD can catch errors immediately. It's faster, easier, and cheaper to fix bugs the moment they're introduced than at any subsequent time, so this ends up being a big win.
3. TDD results in faster, more extensible code with fewer bugs that can be updated with minimal risks.

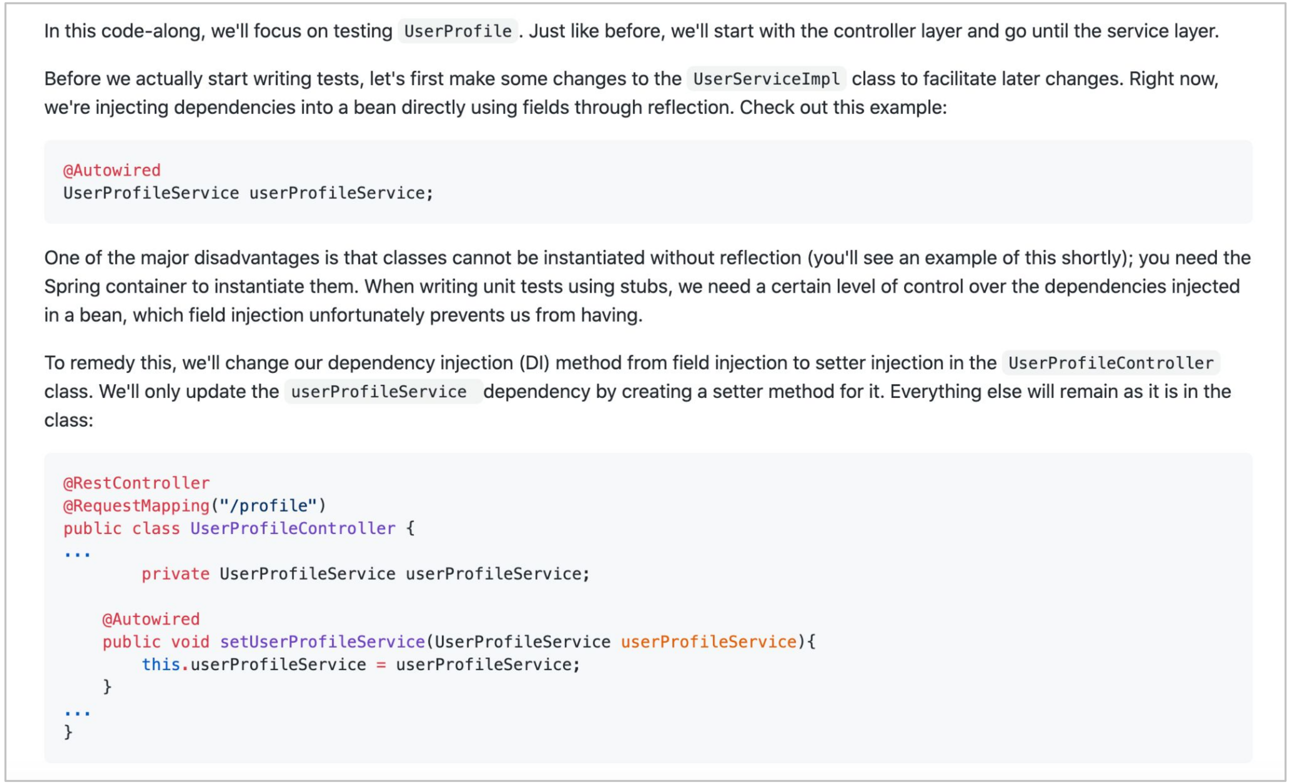
The Bad

It's important to make sure you have similar (or at least compatible) goals before you start development. Creating test cases may bring about friction and disagreements regarding product goals and behavior that need to be resolved.

The Ugly

Sometimes, you'll hit a wall. For example, you could run up against a feature that the product owner or developers don't understand or not know what tests are best to apply.

Stubs for Unit Tests



In this code-along, we'll focus on testing `UserProfile`. Just like before, we'll start with the controller layer and go until the service layer.

Before we actually start writing tests, let's first make some changes to the `UserServiceImpl` class to facilitate later changes. Right now, we're injecting dependencies into a bean directly using fields through reflection. Check out this example:

```
@Autowired
UserProfileService userProfileService;
```

One of the major disadvantages is that classes cannot be instantiated without reflection (you'll see an example of this shortly); you need the Spring container to instantiate them. When writing unit tests using stubs, we need a certain level of control over the dependencies injected in a bean, which field injection unfortunately prevents us from having.

To remedy this, we'll change our dependency injection (DI) method from field injection to setter injection in the `UserProfileController` class. We'll only update the `userProfileService` dependency by creating a setter method for it. Everything else will remain as it is in the class:

```
@RestController
@RequestMapping("/profile")
public class UserProfileController {
...
        private UserProfileService userProfileService;

    @Autowired
    public void setUserProfileService(UserProfileService userProfileService){
        this.userProfileService = userProfileService;
    }
...
}
```

Module Overview

4 hours

Continuous Integration and Delivery

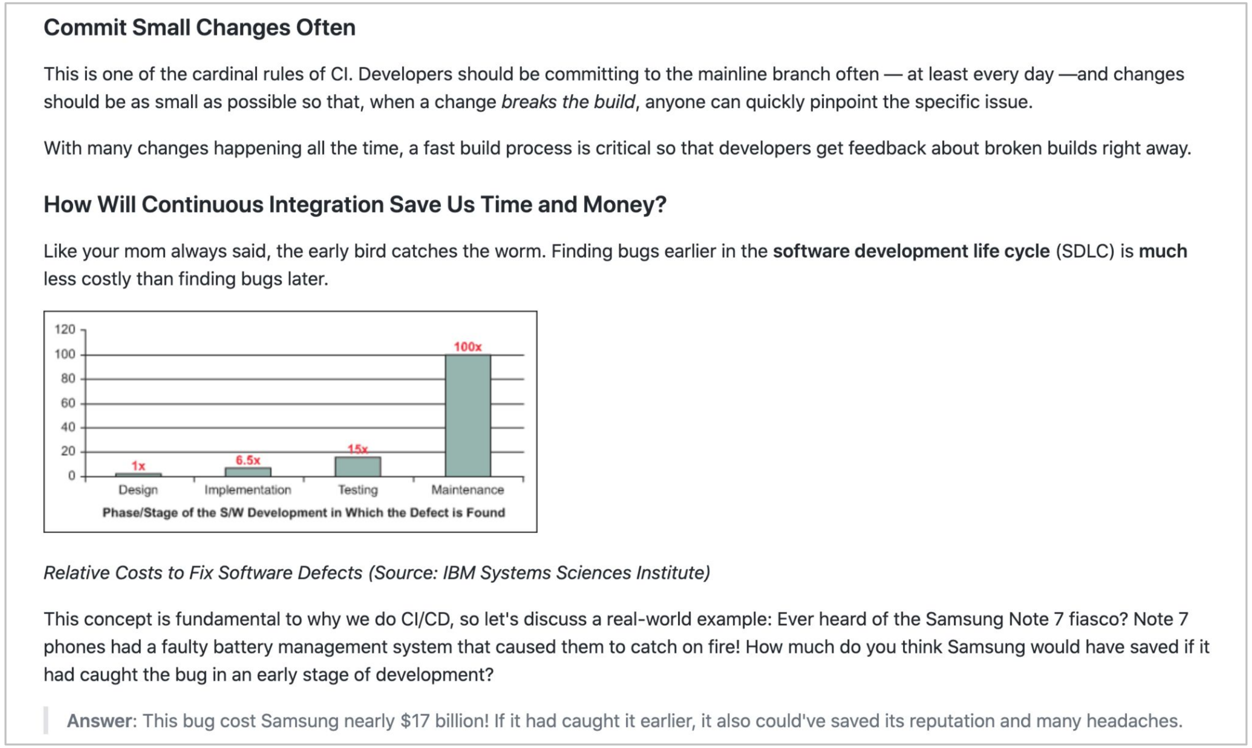
Why?

Continuous integration (CI) and continuous delivery (CD) are core principles of DevOps, helping engineers release software and keep it up to date efficiently. In this unit, students will learn key CI and CD principles, as well as how to apply them to real applications using Jenkins.

Learning Objectives

- Explain what CI/CD are and the benefits of using these practices and tools.
- Describe what a CI/CD pipeline is and what it does.
- Use common tools, plugins, and approaches to build, test, and deploy full-stack applications with a CI/CD pipeline.
- Create Jenkins pipelines in Docker.
- Use JMeter and the Performance plugin in Jenkins.

Continuous Integration



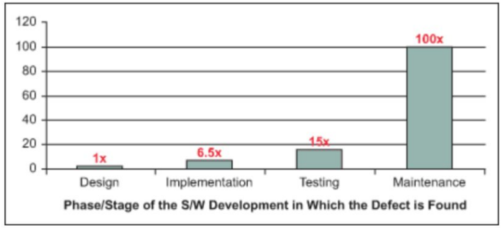
Commit Small Changes Often

This is one of the cardinal rules of CI. Developers should be committing to the mainline branch often — at least every day —and changes should be as small as possible so that, when a change *breaks the build*, anyone can quickly pinpoint the specific issue.

With many changes happening all the time, a fast build process is critical so that developers get feedback about broken builds right away.

How Will Continuous Integration Save Us Time and Money?

Like your mom always said, the early bird catches the worm. Finding bugs earlier in the **software development life cycle** (SDLC) is **much** less costly than finding bugs later.

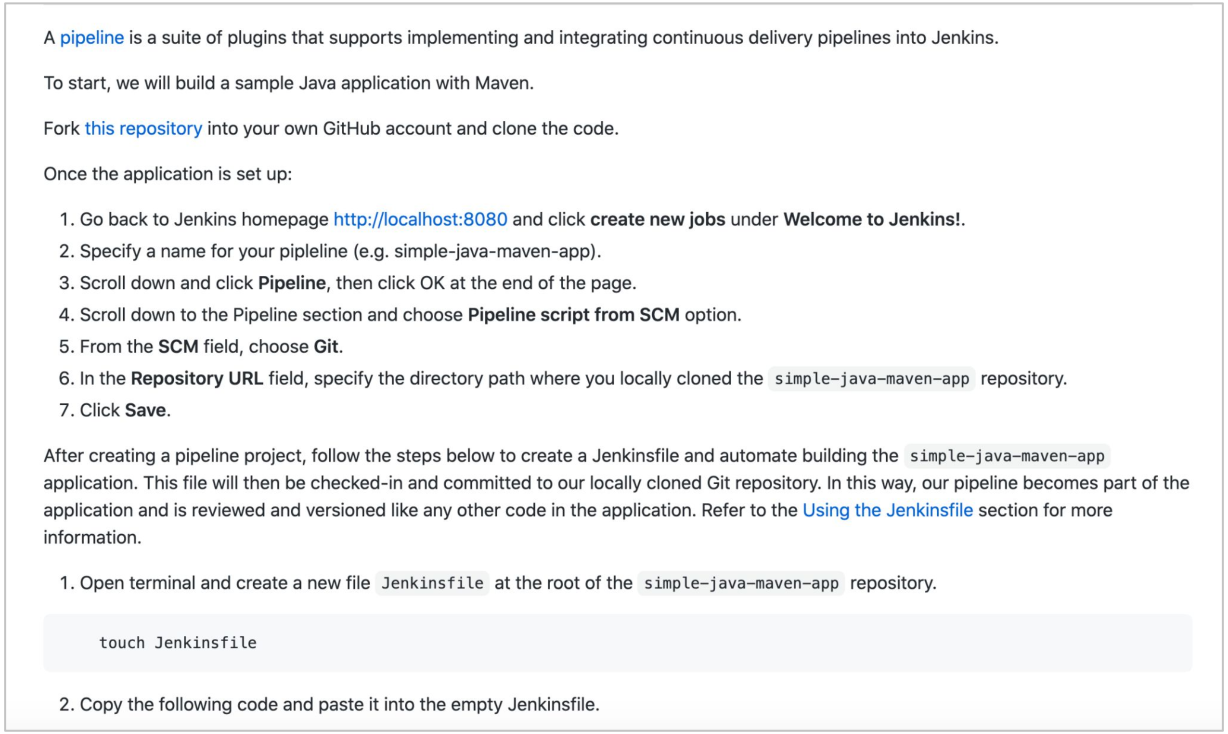


Relative Costs to Fix Software Defects (Source: IBM Systems Sciences Institute)

This concept is fundamental to why we do CI/CD, so let's discuss a real-world example: Ever heard of the Samsung Note 7 fiasco? Note 7 phones had a faulty battery management system that caused them to catch on fire! How much do you think Samsung would have saved if it had caught the bug in an early stage of development?

> **Answer**: This bug cost Samsung nearly $17 billion! If it had caught it earlier, it also could've saved its reputation and many headaches.

Creating a Jenkins Pipeline



A pipeline is a suite of plugins that supports implementing and integrating continuous delivery pipelines into Jenkins.

To start, we will build a sample Java application with Maven.

Fork this repository into your own GitHub account and clone the code.

Once the application is set up:

1. Go back to Jenkins homepage http://localhost:8080 and click **create new jobs** under **Welcome to Jenkins!**.
2. Specify a name for your pipleline (e.g. simple-java-maven-app).
3. Scroll down and click **Pipeline**, then click OK at the end of the page.
4. Scroll down to the Pipeline section and choose **Pipeline script from SCM** option.
5. From the **SCM** field, choose **Git**.
6. In the **Repository URL** field, specify the directory path where you locally cloned the `simple-java-maven-app` repository.
7. Click **Save**.

After creating a pipeline project, follow the steps below to create a Jenkinsfile and automate building the `simple-java-maven-app` application. This file will then be checked-in and committed to our locally cloned Git repository. In this way, our pipeline becomes part of the application and is reviewed and versioned like any other code in the application. Refer to the Using the Jenkinsfile section for more information.

1. Open terminal and create a new file `Jenkinsfile` at the root of the `simple-java-maven-app` repository.

```
touch Jenkinsfile
```

2. Copy the following code and paste it into the empty Jenkinsfile.

Module Overview

8 hours

Logs and Monitoring

Why?

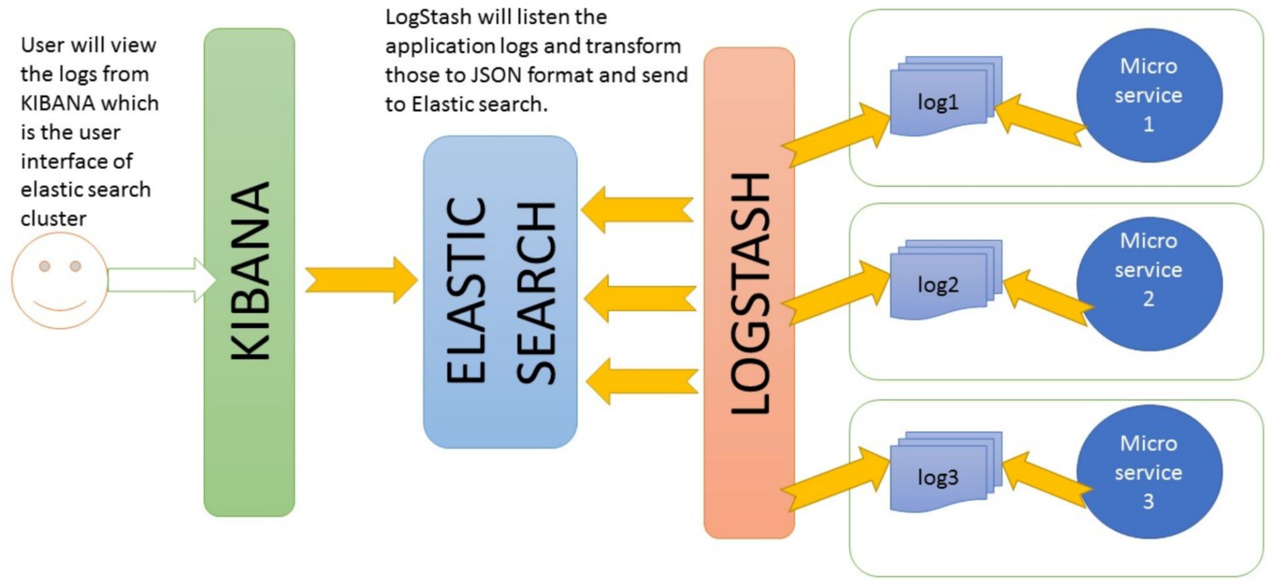
Log monitoring and management make it easy for DevOps teams to get a clear picture of how a system is performing at any moment. In this unit, students will use the ELK stack to implement logging and monitoring systems to keep applications running smoothly.

Learning Objectives

- Explain the importance of logging and monitoring.
- Identify the best metrics against which to monitor your apps.
- Choose the best logging tools for a given situation.
- Implement logging and monitoring systems for your applications.
- Use the ELK stack and Filebeat to log and monitor an application.
- Query for errors and messages with Kibana.

Meet the ELK Stack

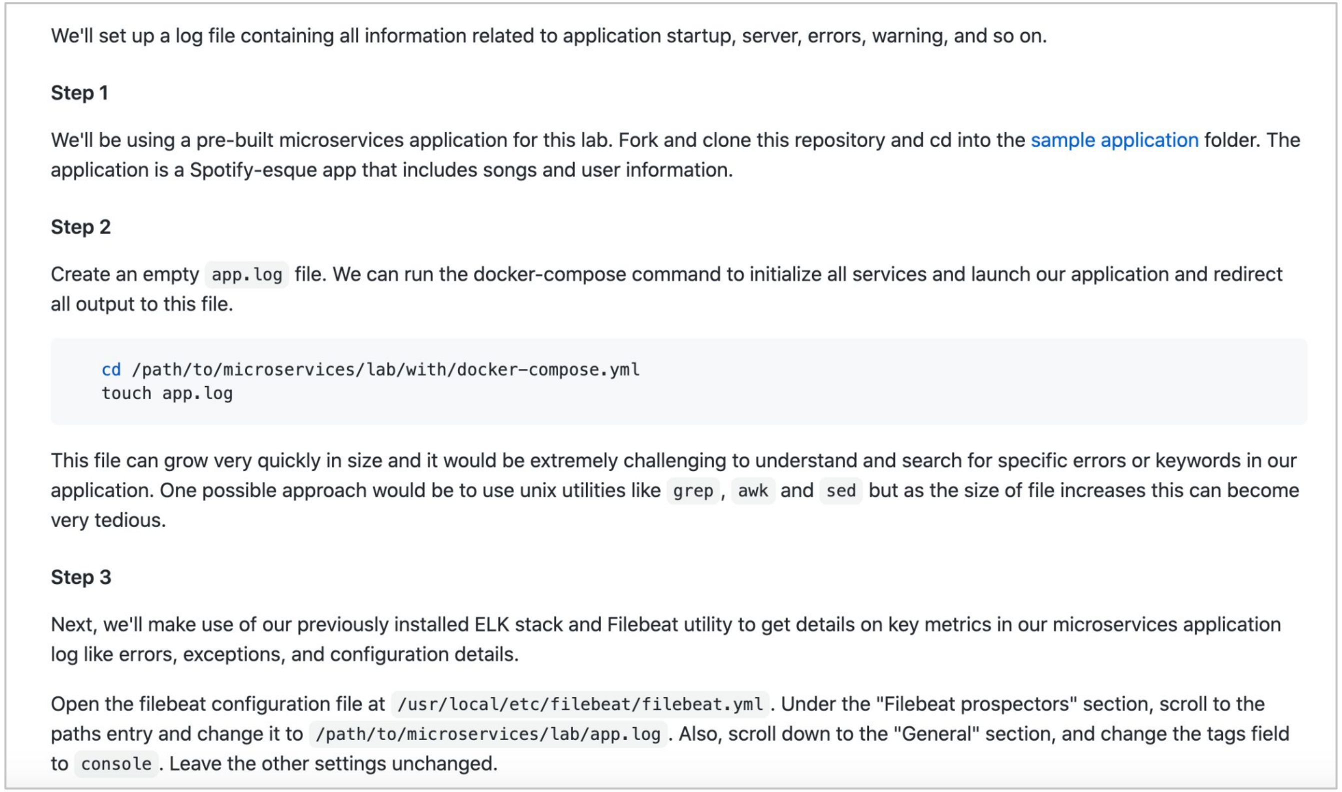
An example of one such workflow is shown below:



User will view the logs from KIBANA which is the user interface of elastic search cluster
KIBANA
LogStash will listen the application logs and transform those to JSON format and send to Elastic search.
ELASTIC SEARCH
LOGSTASH
log1
log2
log3
Micro service 1
Micro service 2
Micro service 3

ELK stack interaction with different applications based on Log file

Setting up Application Logging



We'll set up a log file containing all information related to application startup, server, errors, warning, and so on.

Step 1

We'll be using a pre-built microservices application for this lab. Fork and clone this repository and cd into the sample application folder. The application is a Spotify-esque app that includes songs and user information.

Step 2

Create an empty `app.log` file. We can run the docker-compose command to initialize all services and launch our application and redirect all output to this file.

```
cd /path/to/microservices/lab/with/docker-compose.yml
touch app.log
```

This file can grow very quickly in size and it would be extremely challenging to understand and search for specific errors or keywords in our application. One possible approach would be to use unix utilities like `grep`, `awk` and `sed` but as the size of file increases this can become very tedious.

Step 3

Next, we'll make use of our previously installed ELK stack and Filebeat utility to get details on key metrics in our microservices application log like errors, exceptions, and configuration details.

Open the filebeat configuration file at `/usr/local/etc/filebeat/filebeat.yml`. Under the "Filebeat prospectors" section, scroll to the paths entry and change it to `/path/to/microservices/lab/app.log`. Also, scroll down to the "General" section, and change the tags field to `console`. Leave the other settings unchanged.