DevOps Courses by IBM on Coursera

Course 1

Week 1: Introduction to DevOps

21.11.2022:

1.1. Course Introduction

→ Devops is practice of Engineers working together through full SLDC, in a rapid and continuous manner. This is the heart of the DevOps Culture.

1.2. Business Case for DevOps

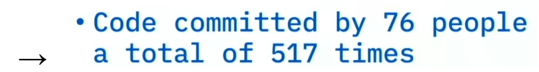
→ Since the Year 2000 52% of Fortune 500 Companies are gone due to disruption.
→ If it takes a bank 6 months to deploy updates to software a lot of customers might get lost.
→ Speed is very important in CI/CD
→ Uber disrupted Taxi industry with Open Source Software.

→ 

→ Netflix disrupted Blockbuster
→ Technology enables innovation, doesn't drive it by itself.

1.3. DevOps Adoption

→ fail fast, roll back quickly
→ A/B testing
→ Microservices are usable and recommended for this
→ January 2011 Etsy deploys every ~25min, in 1 month they:

→ 

→ 2016:

2016 DevOps Enterprise Summit

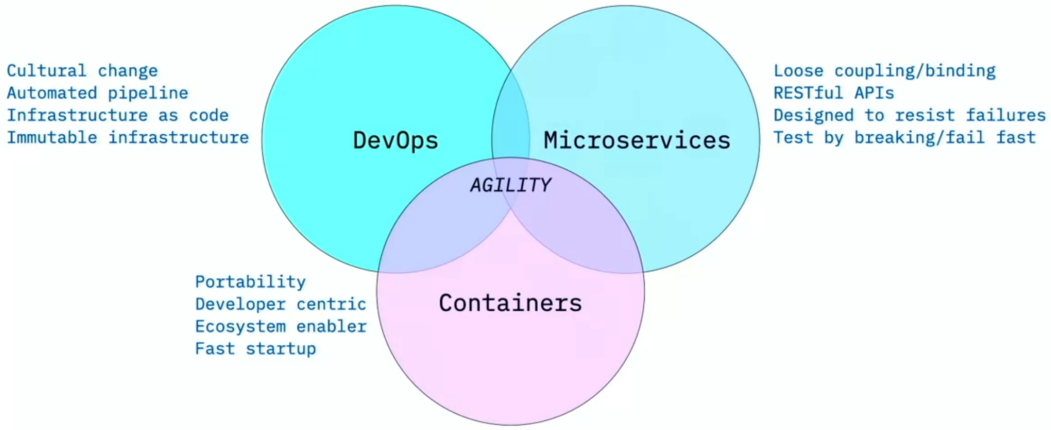
Ticketmaster – 98% reduction in MTTR
Nordstrom – 20% shorter lead time
Target – Full Stack Deploy three months to minutes
USAA – Release from 28 days to 7 days
ING – 500 application teams doing DevOps
CSG – From 200 incidents per release to 18

1.4. Definition of DevOps

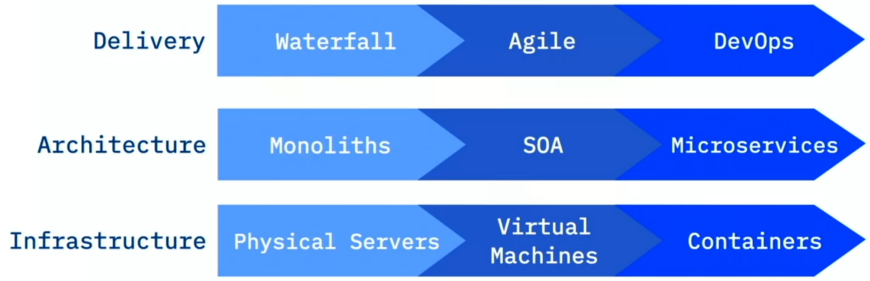
→ “Development Operation is an extension of agile development that aims to enhance the proces of software delivery as a whole” - Patrick Debois, 2009
→ DevOps is Devs and Ops working together in full SLDC following agile principles
→ DevOps requires a new application design, leveraging automation for microservices and a defined programmable platform to deploy on to.

1.5. Essential Characteristics of DevOps

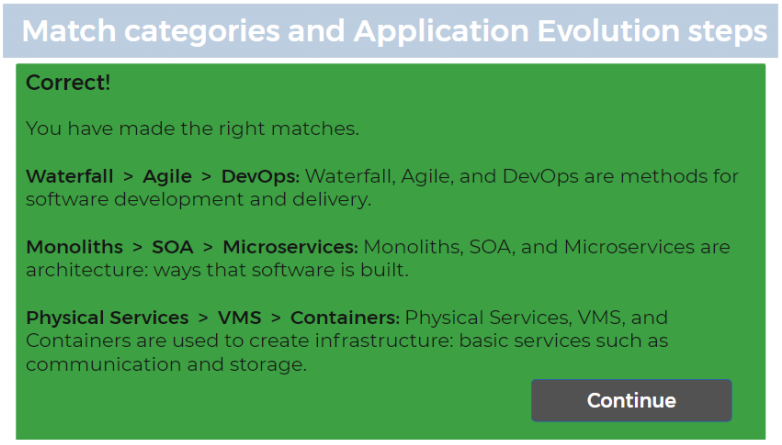
→ Be agile, move in market with minimum risk and maximum velocity, gain valuable insights
→ Three Pillars of Agility:



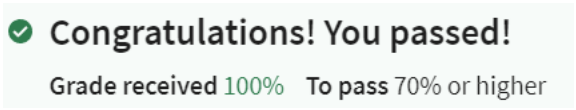
→ Application evolution:



1.6. Activity: Identify Categories in Appliocation Evolution

→ 

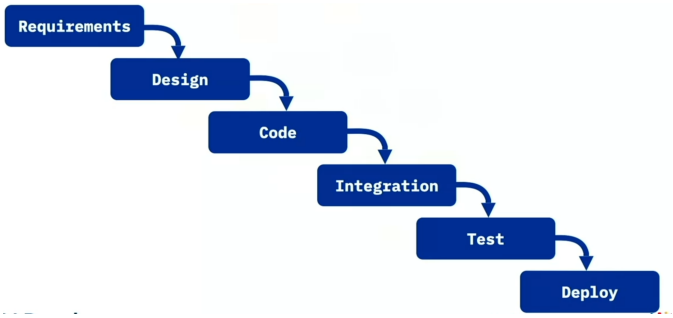
1.7. Practice Quiz: Introduction To DevOps: Practice Quiz 1

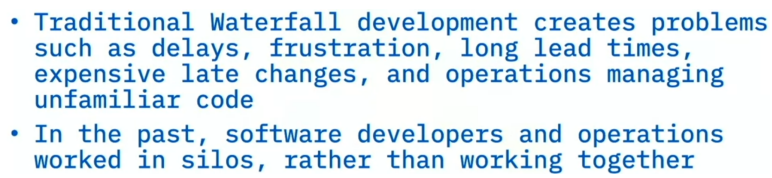
→ 

22.11.2022

1.8. Leading Up to DevOps

→ Traditional inefficient waterfall development method:

→ 

→ 

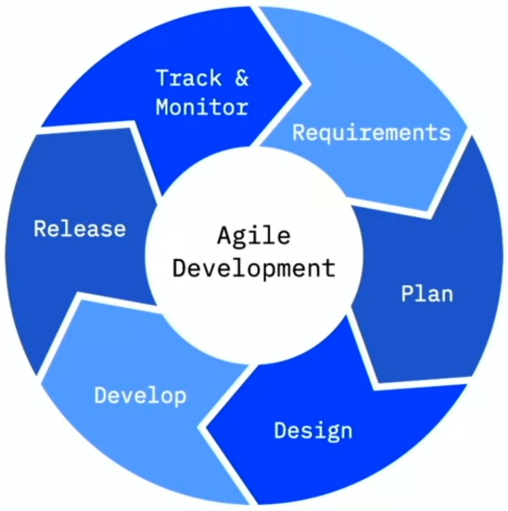
1.9. XP, Agile, and Beyond

→ The Agile Manifesto:

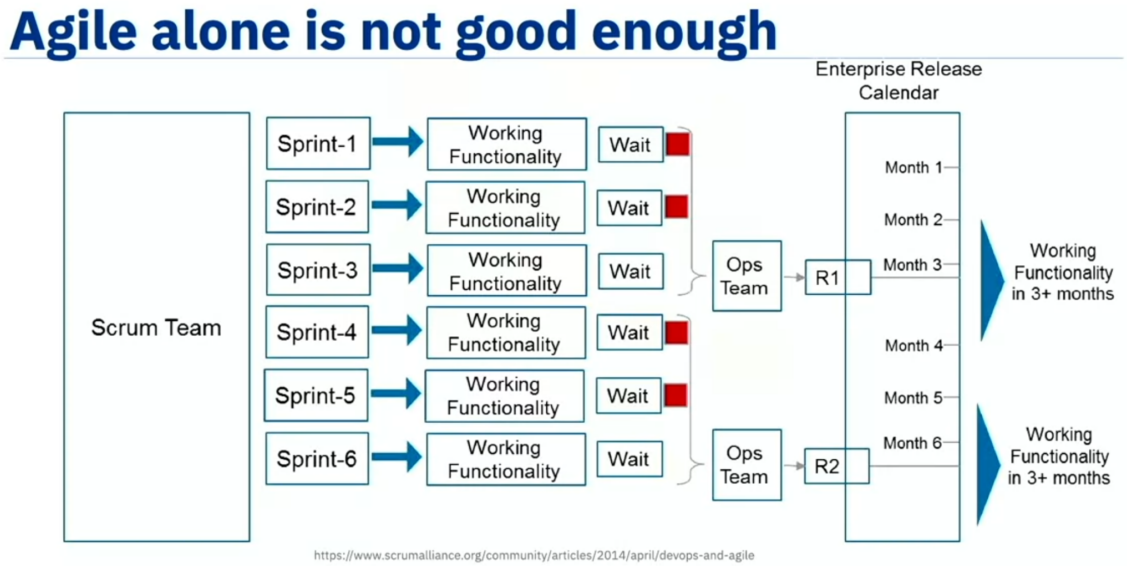
We have come to value:

Individuals and interactions over processes and tools
Working software over comprehensive documentation
Customer collaboration over contract negotiation
Responding to change over following a plan

That is, while there is value in the items on the right,
→ we value the items on the left more



→



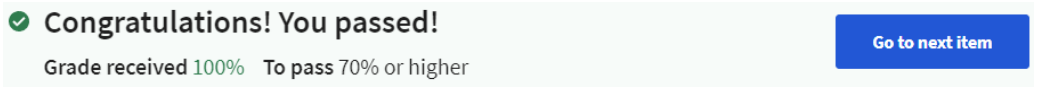
→

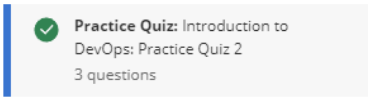
→ Shadow IT (Resources the business doesn't know about) comes around because of this speed Problem in Company infrastructure, people go around it.

1.10. Brief History of DevOps

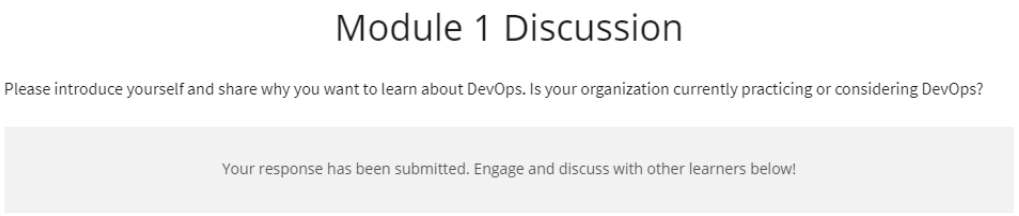
- Patrick Debois started the DevOps movement in 2007 with a simple idea of getting development and operations to work better together
- DevOps grew through the efforts of influential people such as Patrick Debois, Andrew Shafer, John Allspaw, Nicole Forsgren, Bridget
→ Kromhout, Jez Humble, Gene Kim, and John Willis

1.11. Practice Quiz: Introduction to DevOps: Practice Quiz 2

→ 

→ 

1.12. Discussion Prompt: Module 1 Discussion



→

1.13. Reading: Summary and Hightlights

→ Technology is the enabler of innovation, rather than the driver of innovation. You must have an innovative business idea to leverage technology.

→ In 2009, John Allspaw described an innovative approach to managing development and operations that enabled Flickr to complete over ten deploys per day, when many companies were completing fewer than one deploy every six months. This was a key moment in the growth of DevOps.

→ DevOps is the practice of development and operation engineers working together during the entire development lifecycle, following Lean and Agile principles that allow them to deliver software in a rapid and continuous manner.

→ DevOps is not it is not just Dev and Ops working together. It is a cultural change and a different way to work. DevOps has three dimensions: culture, methods, and tools. Of these, culture is the most important.

→ The essential characteristics of DevOps include cultural change, automated pipelines, infrastructure as code, immutable infrastructure, cloud native application design, the ecosystem of containers, and how to deploy with immutable infrastructure.

→ DevOps started in 2007 when Patrick Debois and Andrew Clay Shafer began to gather like-minded people together at conferences to talk about common experiences.

→ In 2009, Allspaw delivered his now famous “10+ Deploys Per Day – Dev and Ops Cooperation at Flickr” presentation and the idea gained ground. Also in 2009, Patrick Debois started a conference called DevOpsDays that helped spread the DevOps message.

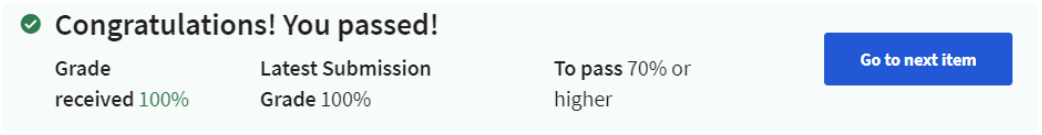
→ Books such as *Continuous Delivery* in 2011, *The Phoenix Project* in 2015, and *The DevOps Handbook* in 2016, helped practitioners understand how DevOps worked.

→ The major influential people of the early DevOps movement: Patrick Debois, Andrew Clay Shafer, John Allspaw, Jez Humble, Gene Kim, John Willis, Bridget Kromhout, and Nicole Forsgren, went out and made a difference, showing the results that could be achieved with DevOps.

→ The message spread from practitioner to practitioner until they began to realize what was possible with DevOps and that it was a better way to work.

1.14. Quiz: Overview of DevOps

→



Week 2: Social Coding Principles

23.11.2022

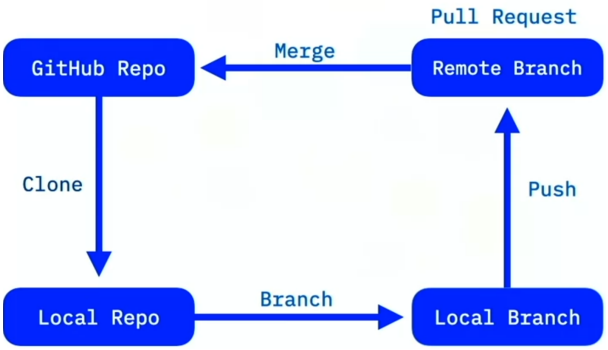
2.1. Social Coding Principles

- Social coding is done communally with public repositories and all team members are encouraged to contribute
- Pair programming results in higher quality code because
  - Defects are found earlier
  - Costs are lowered
  - There is a broader understanding of the code base

→

2.2. Git Repository Guidelines

→ Every Microservice in a separate repository, new Branch for every issue
→ Pull Request to merge to master, every Pull Request should be revied
→ The Git Feature Branch Workflow:



→

2.3. Working in small Batches:

- Feature size supports frequent releases
- Features should be completed in a sprint
- Features are a step toward a goal, so keep them small

→

2.4. MVP (Minimum viable product)

- An MVP is the minimal thing that you can do to test your hypothesis
- An MVP is not about delivery as much as learning
- It's okay if the MVP fails—make sure that you learn from it

→

2.5. Practice Quiz: Social Coding principles: Practice Quiz 1

→ Congratulations! You passed!

Grade received 100% To pass 70% or higher

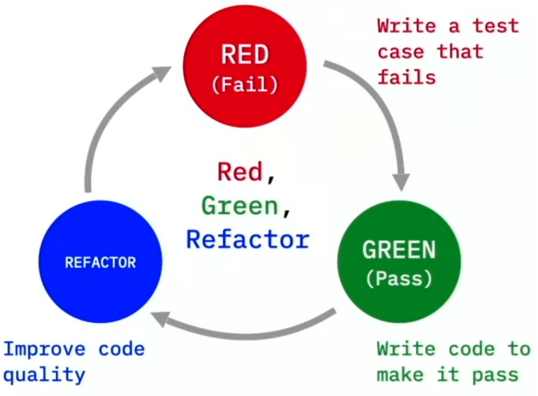
Go to next item

→ Practice Quiz: Social Coding Principles: Practice Quiz 1

2 questions

2.6. Test Driven Development (TDD)

→ Workflow is at follows, it goes as red → green → refactor



→

→ Why is TDD important:

- TDD means test cases drive the design and development of code
- TDD allows you to develop faster and with more confidence
- The Red, Green, Refactor workflow has to do with red as fail, green as pass, and it increases the quality of code
- In order to create a DevOps CI/CD pipeline, you must first automate your testing

→

2.7. Behavior Driven Development

→ Difference to TDD:

- BDD ensures that you are building the "right thing"
- TDD ensures that you are building the "thing right"

→

- BDD focuses on the behavior of the system from the outside in. It looks at the system as a consumer of it.
- BDD uses an approachable syntax that everyone can understand.
- Key benefits of BDD include improvement in communication, a common syntax, and acceptance criteria for user stories.

→ BDD Workflow:

- Explore the problem domain and describe the behavior
- Document the behavior using Gherkin syntax
- Use BDD tools to run those scenarios
- One document that's both the specification and the tests

→

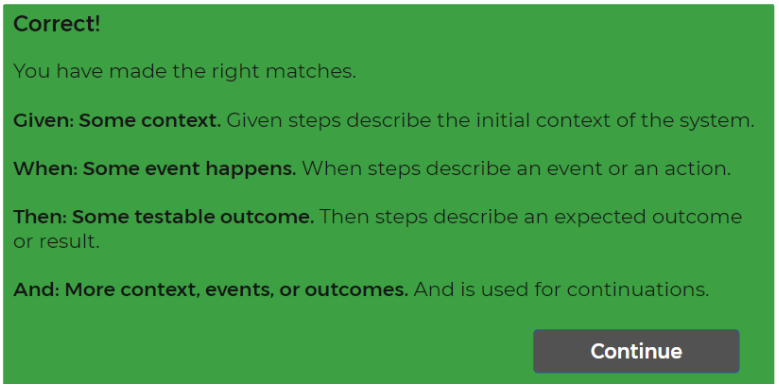
→ Gherkin:

- An easy-to-read natural language syntax
- Given... When... Then...
- Understandable by everyone

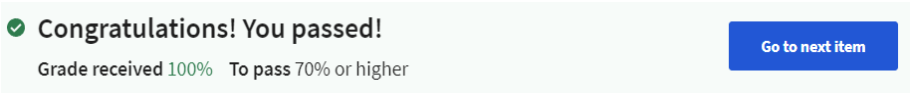
→

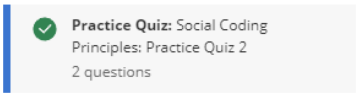
- Given (some context)
- When (some event happens)
- Then (some testable outcome)
- And (more context, events, or outcomes)

2.8. Activity: Writing in Gherkin Syntax

→ 

2.9. Practice Quiz: Social Coding Principles: Practice Quiz 2

→ 

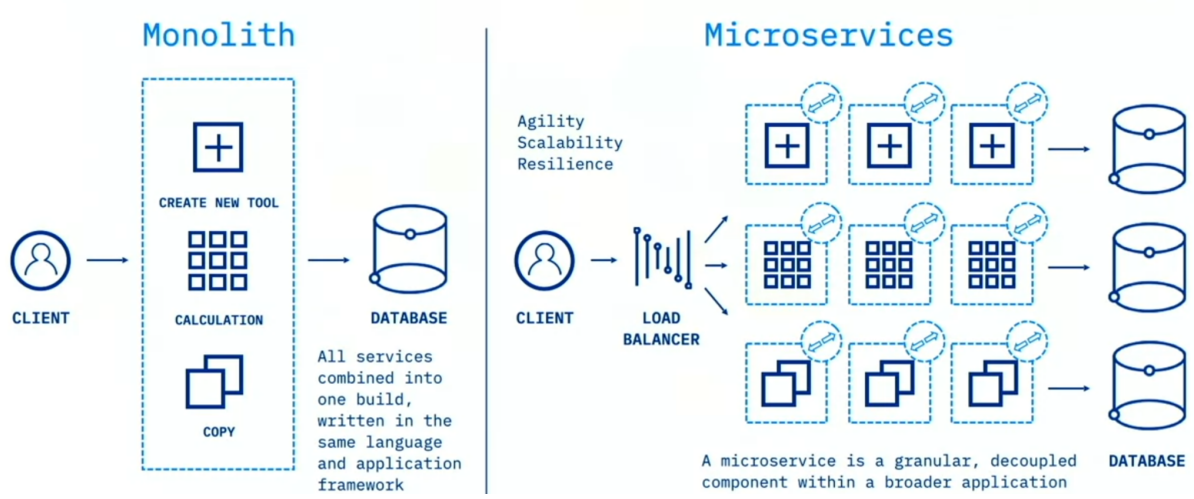
→ 

01.12.2022

2.10. Cloud Native Microservices

→ "…the microservice architectural style is an approach to developing a single application as a **suite of small services**, each **running in its own process** and communicating with lightweight mechanisms, often an HTTP resource API. These services are **built around business capabilities** and **independently deployable** by fully automated deployment machinery."

→ Monolith vs microservices:

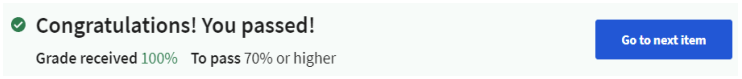
→ 

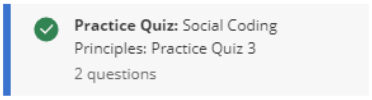
→ Communication between services is done through (REST) APIs

2.11. Designing for Failuire

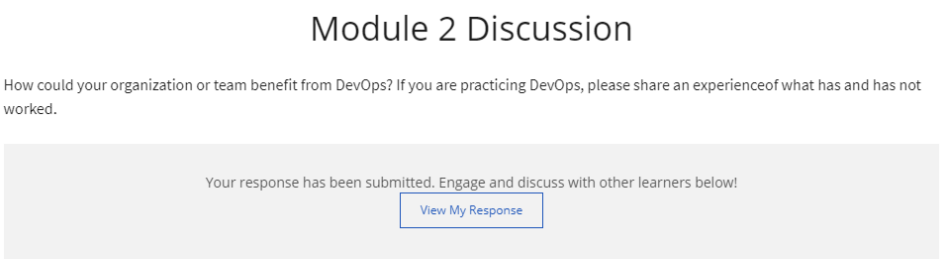
- Failure is inevitable, so we design for failure rather than trying to avoid failure
- Developers need to build in resistance to be able to recover quickly
- Retry patterns work by retrying failed operations
- Circuit breaker patterns are designed to avoid cascading failures
- Bulkhead patterns can be used to isolate failing services
- → Chaos engineering is deliberately causing services to fail to see how other services are affected

2.12. Practice Quiz: Social Coding Principles: Practice Quiz 3

→ 

→ 

2.13. Discussion Prompt: Module 2 Discussion



→

2.14. Reading: Summary and Highlights

→ Social coding is coding as a community and public repositories and pair programming result in higher code quality.

→ Working in small batches reduces waste and means quickly delivering something useful to the customer.

→ Minimum viable product is as much about delivery as it is about building what the customer really desires.

→ Test driven development is writing the test for the code you wish you had, then writing the code to make the test pass. It allows you to develop faster and with more confidence.

→ Behavior driven development focuses on the behavior of the system from the outside in. It looks at the system as a consumer of it.

→ Behavior driven development improves communication by using an approachable syntax that developers and stakeholders can understand.

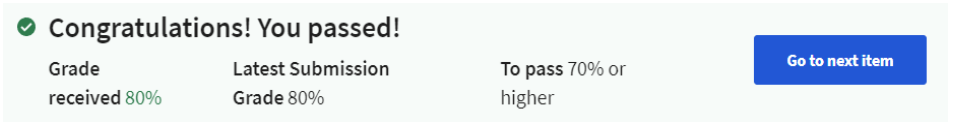
→ Microservices are built around business capabilities and are independently deployable by fully automated deployment machinery.

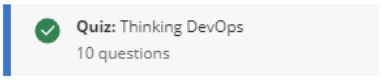
→ Cloud native architecture enables independently deployable microservices that take advantage of horizontal scaling and result in more resilient services.

→ Failure is inevitable, so we design for failure rather than trying to avoid failure.

→ It is important to embrace failure and quickly recover when failures happen.

2.15. Quiz: Thinking DevOps

→ 

→ 

Week 3: Taylorism and Working in Silos

01.12.2022

3.1. Taylorism and Working in Silos

- Working DevOps means pushing small releases faster in order to get feedback, minimize risk, and maximize learning
- Taylorism was designed for factory work, while software development is like craft work
- Handoffs created by working in silos results in mistakes, bottlenecks, and delays

→

3.2. Software Engineering vs Civil Engineering

- Software development is often viewed as a project to be completed and then passed to operations to be maintained
- Software engineering and the behavior of the system are constantly changing
- Team ownership and stable teams make software development more like product development and less like project management

→

3.3. Required DevOps Behaviors

→ Traditional vs DevOps:

~~Manual configuration changes to critical infrastructure~~	Automated deployment to all environments
~~Application architectures defined by network design~~	Network design defined by application architectures
~~Bespoke infrastructure built once, then maintained~~	Ephemeral infrastructure created for each new deployment
~~Risk managed through change windows~~	Risk managed through progressive activation
~~Process biased toward "build once"~~	Builds are repeatable leveraging infrastructure as code

→

→ Wall of Confusion:

- Development wants innovation

→ • Operations wants stability

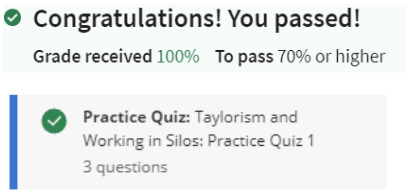
→ Required Behaviour:

~~Organizational silos and hand offs~~	Shared ownership and high collaboration
~~Fear of change~~	Risk management by embracing change
~~Build once, hand-crafted "snowflakes"~~	Ephemeral infrastructure as code
~~Manual fulfillment~~	Automated self-service
~~Alarms, callbacks, and escalations~~	Feedback loops and data-driven responses

→

3.4. Activity: Choosing DevOps Behaviour

3.5. Practice Quiz: Taylorism and Working in Silos: Practice Quiz 1



08.12.2022

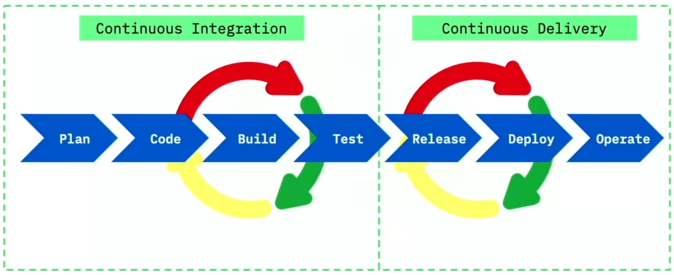
3.6. IaC (Infrastructure as Code)

- Infrastructure as code describes infrastructure in an executable textual format
- Ephemeral infrastructure can be used and then discarded. Servers are built on demand, via automation
- Rather than patching a running container, immutable delivery is making changes to the container image, then redeploying a new container

3.7. Continuous Integration

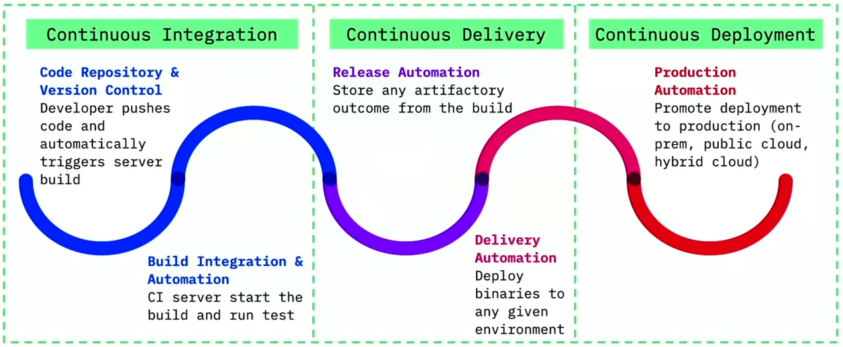
- Continuous Integration is building, testing, and integrating every change into the master branch after tests have passed
- Continuous Delivery ensures that code can be rapidly and safely deployed to production by delivering every change to a production-like environment
- Working in small batches aids Continuous Integration by reducing the number of conflicting changes
- Benefits of Continuous Integration include faster reaction time, moving faster, and reducing the risk of integrating code

3.8. Continuous Delivery/Deployment



→

→ Pipeline:



→

- CI/CD pipeline tools provide a way to automate deployment
- Release, deploy, and operate are part of continuous delivery
- The five key principles of continuous delivery have to do with quality, working in small batches, automation, continuous improvement, and shared responsibility

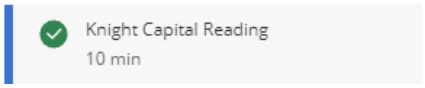
→

- Continuous deployment is when automation is used to deploy to production
- DevOps manages risk by increasing the rate of change rather than avoiding changes

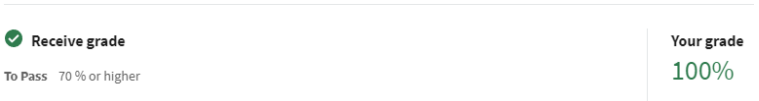
→

3.9. Knight Capital Reading

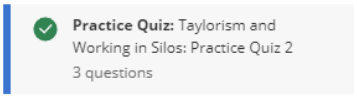
→



3.10. Practice Quiz: Taylorism and Working in Silos: Practice Quiz 2

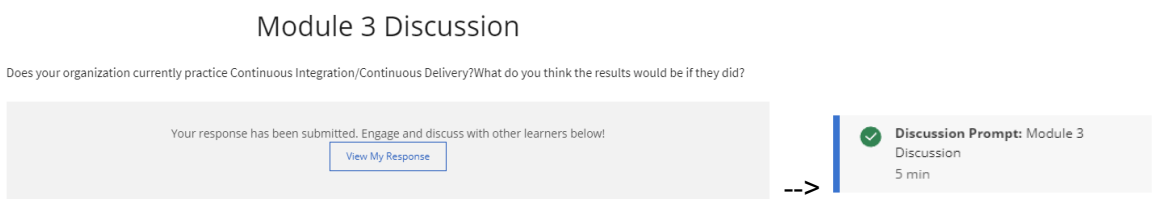


→



→

3.11. Discussion Prompt: Module 3 Discussion



→ -->

3.12. Reading: Summary and Highlights

→ Taylorism was designed for factory work and software development is bespoke, that is, more like craftwork, and that working in silos leads to mistakes and bottlenecks.

→ Team ownership and stable teams make software development more like product development rather than project management.

→ Developers want innovation, while Operations want stability.

→ Required DevOps behaviors include shared ownership, collaboration, embracing change, and data-driven responses.

→ Infrastructure as Code is describing infrastructure in a textual executable format.

→ Ephemeral infrastructure can be used and then discarded because servers are built on demand, via automation, using Infrastructure as Code techniques.

→ Continuous Integration is building, testing, and integrating every developer change into the master branch after tests have passed.

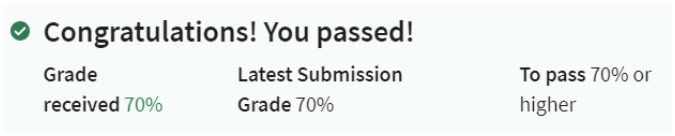
→ The benefits of Continuous Integration include faster reaction time, moving faster, and reducing the risk in integrating code.

→ Continuous Delivery ensures that code can be rapidly and safely deployed to production by delivering every change to a production-like environment.

→ The five principles of Continuous Delivery have to do with quality, working in small batches, automation, continuous improvement, and shared responsibility.

3.13. Quiz: Working DevOps

→



→

Week 4: Organizing for DevOps