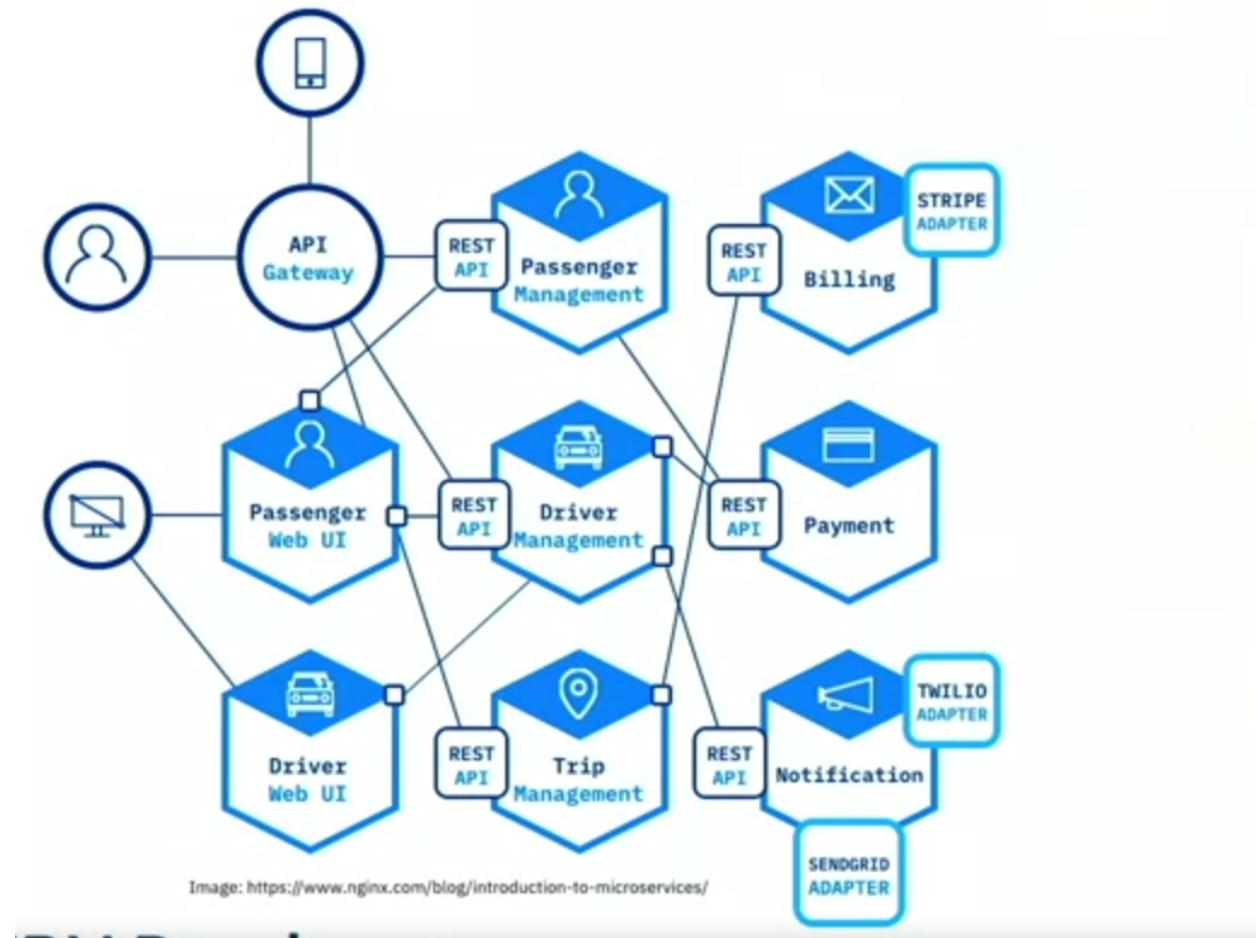


Cloud native microservices(devops)



Think differently about application design

The amazing thing is that all API connected to each other by the lines ,are working on cloud microservices. No services is accessing the database, all are the microservices of cloud.

Its a cloud native design using microservices.

Think cloud native

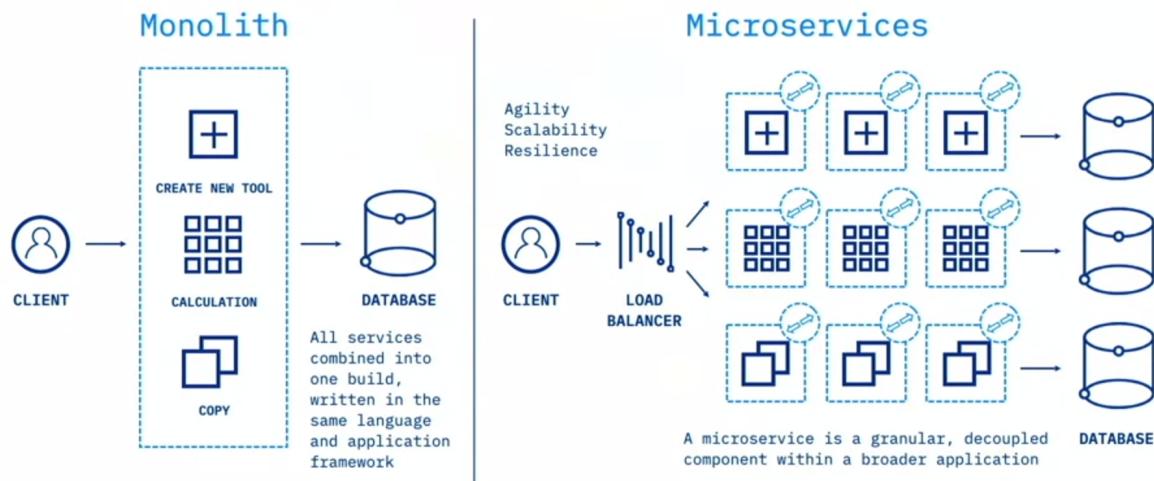
- The Twelve-Factor App
- A collection of stateless microservices
- Each service maintains its own database
- Resilience through horizontal scaling
- Failing instances are killed and respawned
- Continuous delivery of services

Think 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."

-Martin Fowler and James Lewis

Monolith versus microservices



Here each microservices has its own data base working with there own services

So in monolith if i have to make changes in website then i have to contact to the shipping team , and particular team required for deployment. But in microservices we dont have to contact any team .As we know we are using Rest API's here
 Api is building on demand use case of table of customer as user required

Desgining Failure”-

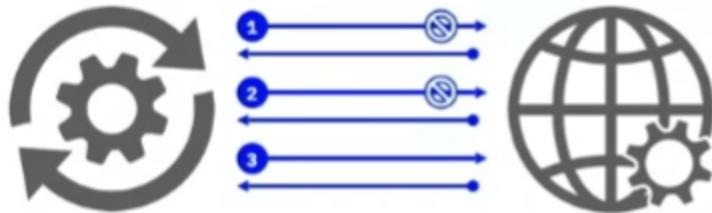
Failure happens

- Embrace failures - they will happen!
- How to avoid → How to identify and what to do about it
- Operational concern → developer concern
- Plan to be throttled
- Plan to retry (with exponential backoff)
- Degrade gracefully
- Cache when appropriate

Failure patterns

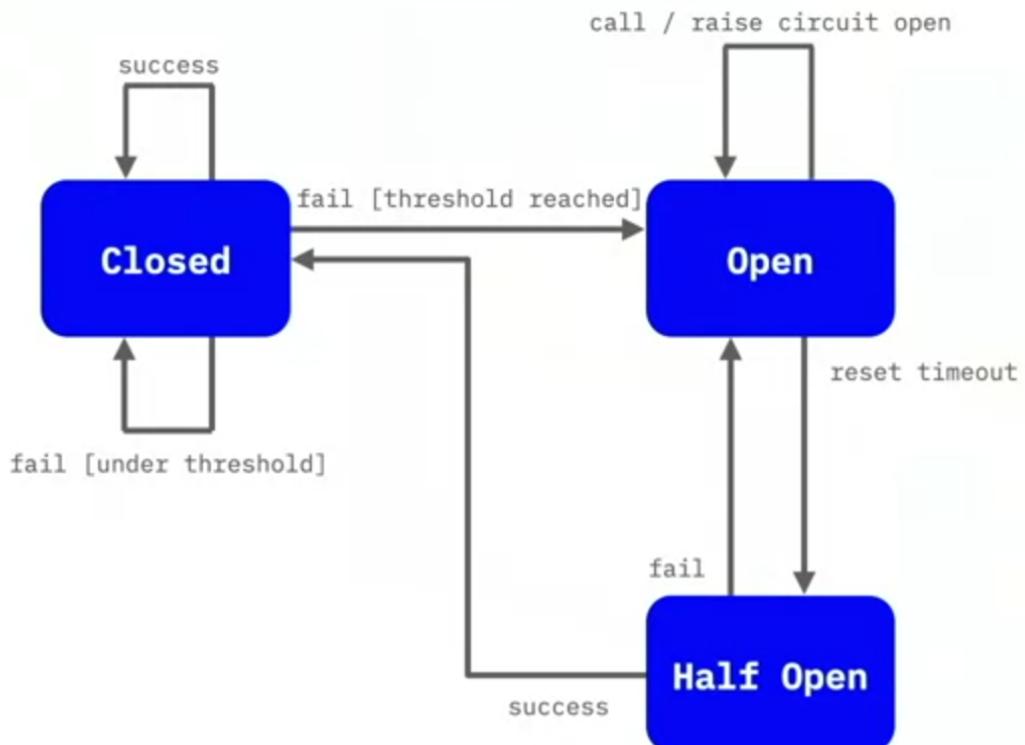
*retry pattern :- when we connect to server and fail to connect to the database . We retry to connect so try to develop in such way that retry pattern occurs . When connection is success this retry pattern in connection is over.

Retry pattern



Circuit breaker pattern

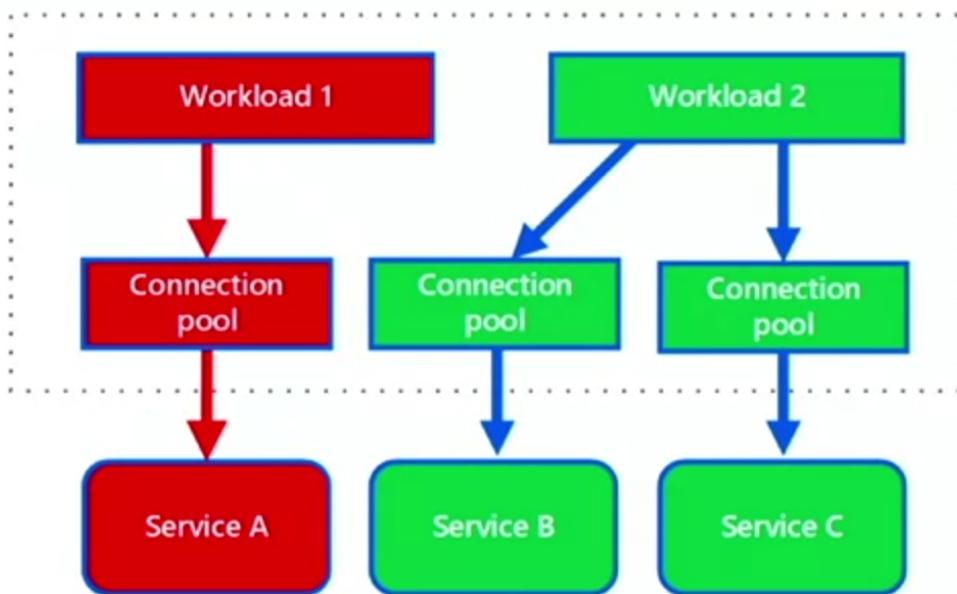
Circuit breaker pattern



Same it avoid failure in the circuit by tripping the main switch. It avoid the cascading failure. Cascade failure is when one services is not available and other services is cascade To another

Press Esc to exit full screen

Bulkhead pattern



Its name comes from the bulkhead design on a ship. Compartments that are below the waterline have walls called “bulkheads” between them. If the hull is breached, only one compartment will fill with water.

The bulkhead stops the water from affecting the other compartments and sinking the ship. Using the bulkhead pattern isolates consumers from the services as cascading failures by allowing them to preserve some functionality in the event of a service failure. Other services and features of the application continue to work.

Chaos Engineering;

Its also known as monkey testing

Also deliberately kill services.

Netflix too created a Siman army tools

We cannot know how something will respond when it fails.

Summary and Highlights

Congratulations! You have completed this lesson. At this point in the course, you know:

- 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.

Taylorism and working in Slios.

Working DevOps

- Culture of teaming and collaboration
- Agile development as a shared discipline
- Automate relentlessly
- Push smaller releases faster

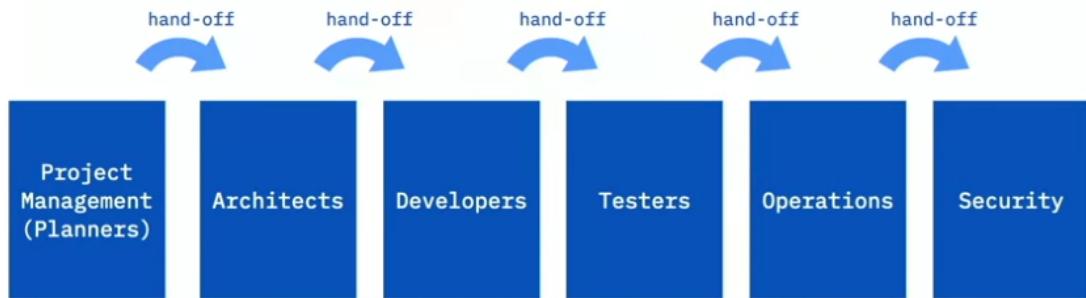


Taylorism

- Adoption of command and control management
- Organizations divided into functional silos
- Decision-making is separated from work



Impact of Taylorism on information technology



- * These are the working process of the Taylorism as I have mentioned taylorism . Its a way how industrial revolution occurred and how its happening and manufacturing product.
- * same way this taylorism works . How it goes hand to hand and it work in silos.

Abandon Command and Control

- Its not agile
- Stop working in Silos

Software engineer vs Civil Engineering

So in Software engineering we consistently Develop it , Edit it , maintain it.

- Software Stack is continuously Updated
- New feature is being added

This affect the application

- System bheavior is changed over time
- Ye we treat software engineering like a civil engineering project

The project model is flawed

- The project model does not work for software development
- Treat software development like product development
- Encourage ownership and understanding
- Software engineering is not civil engineering
- Maintain stable, lasting teams



Required behaviour of Devops

Diametrically opposed views

- Enterprises see “new” as complex and time-consuming
- DevOps delivers a continual series of small changes
- These cannot survive traditional overheads



A clash of work cultures

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

IBM Developer

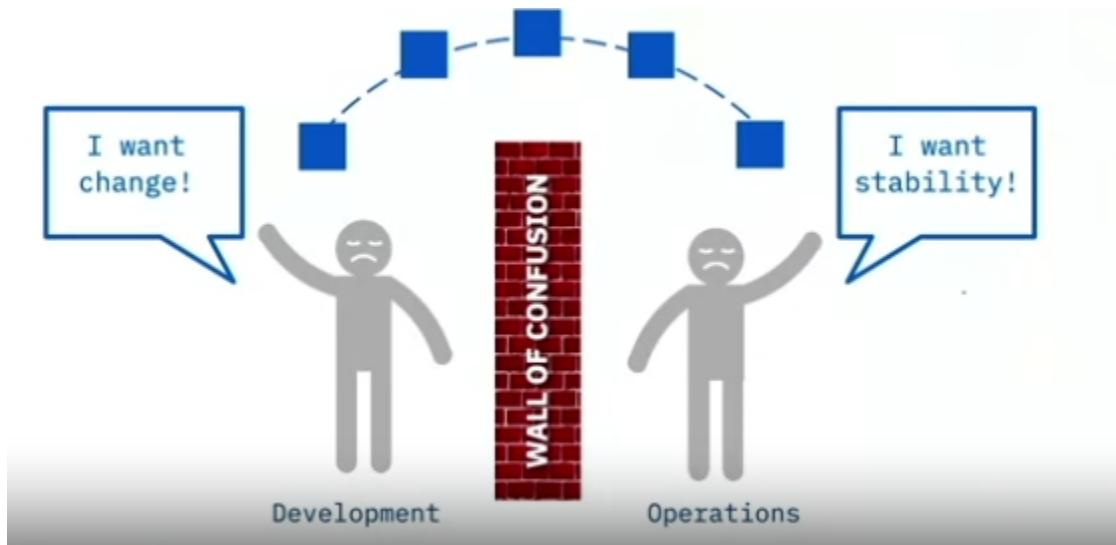
SKILLS NETWORK 

Last one is traditional ops which is gone to Devops

There is no win scenario

Its a innovation In development.. How much innovative developer is

Operations Also want stability



So we break the wall of Confusion as Devops engineer

Operations view of development

- Development teams throw dead cats over the wall
- Manually implemented changes
- Lack of back-out plans
- Lack of testing
- Environments that don't look like production



Development view of operations

- All-or-nothing changes
- Change windows in the dead of night
- Implemented by people furthest away from the application
- Ops just cuts and pastes from "runbooks"



Required DevOps behaviors

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

Infrastructure As Code

Desribabel code in textual format not in sense of documentatiions format.Its textual format which is executable here.

Configuration of textual description given to a tool to execute These tools work and make it possible by configuration management.

Never perform this configuration manually.

Use of version control:-History of all changes so we let to know which version is latest version, and what we have to proceed on,

Immutable delivery via containers

- Applications are packaged in containers
- Same container that runs in production can be run locally
- Dependencies are contained
- No variance limits side-effects
- Rolling updates with immediate roll-back



Immutable way of working

- You never makes changes to a running container
- You make changes to the image
- Then redeploy a new container
- Keep images up to date



Continuous integration

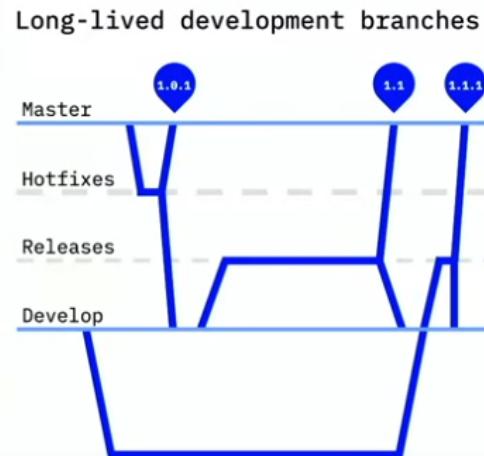
Continuous Integration vs. Continuous Delivery

- CI/CD is not one thing
- Continuous Integration (CI)
 - Continuously building, testing, and merging to master
- Continuous Delivery (CD)
 - Continuously deploying to a production-like environment



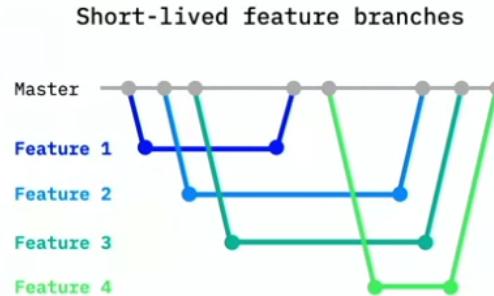
Traditional development

- Developers work in long-lived development branches
- Branches are periodically merged into a release
- Builds are run periodically
- Developers continue to add to the development branch



Continuous Integration

- Developers integrate code often
- Developers work in short-lived feature branches
- Each check-in is verified by an automated build



IBM Developer

SKILLS NETWORK 

Changes are made small

Working in small batch.

Committing regularly
using pull request

Committing all Changes daily

Continuous delivery

Continuous development is a software development discipline where the software is always ready for the deployment.

Release production at any times means

It means The master branch is always be deployable

Release to production at any time

- The master branch should always be ready to deploy
- You need a way to know if something will “break the build”
- Deliver every change to a production-like environment



CI automation

- Build and test every pull request
- Use CI tools that monitor version control
- Tests should run after each build
- Never merge a pull request with failing tests



Focus in High quality of code and Master branch should be deployable

The CI/CD pipeline

Benefits of Continuous Integration

- Faster reaction times to changes
- Reduced code integration risk
- Higher code quality
- The code in version control works
- Master branch is always deployable



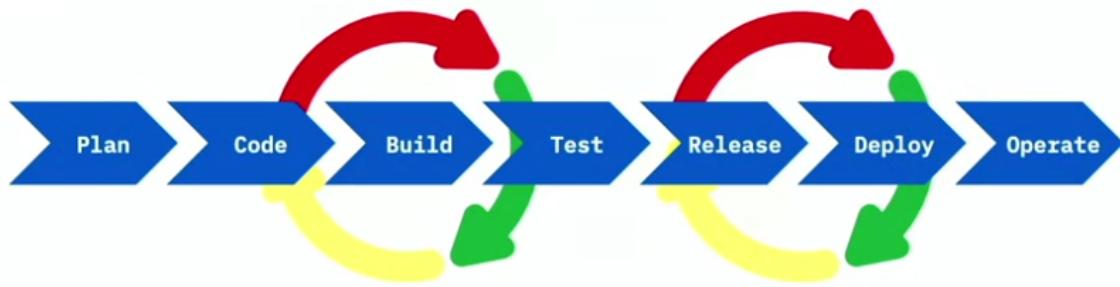
Never merged untested code in master branch or it will create a mess.

A CI/CD pipeline needs

- A code repository
- A build server
- An integration server
- An artifact repository
- Automatic configuration and deployment



Continuous integration and delivery



Continuous Delivery

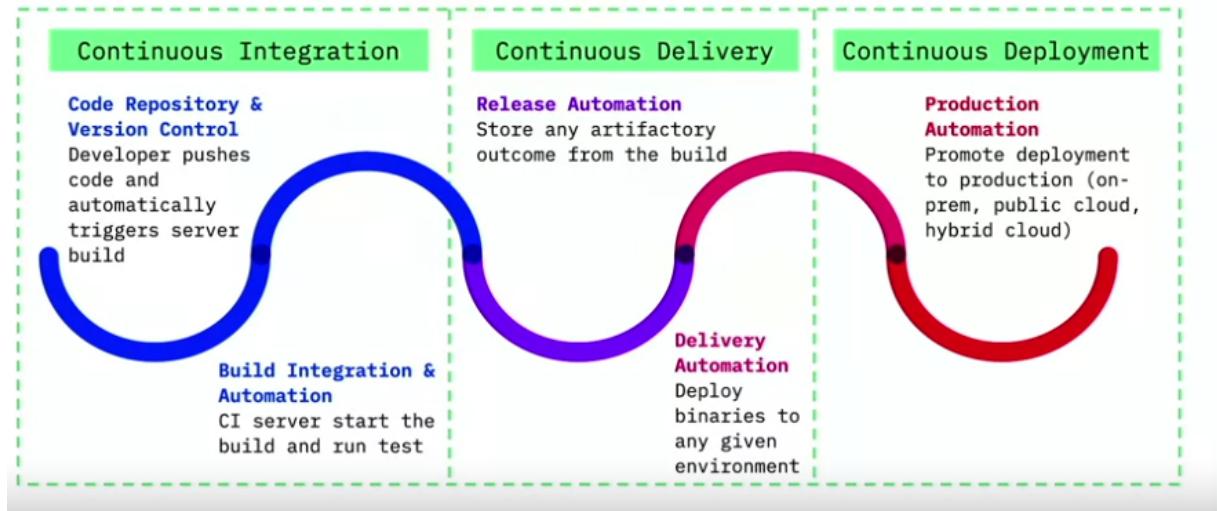
[Notes](#) [Discuss](#)

Five key principles

1. Build quality in
2. Work in small batches
3. Computers perform repetitive tasks, people solve problems
4. Relentlessly pursue continuous improvement
5. Everyone is responsible



CI/CD + Continuous deployment



How DevOps manages risk

- Deployment is king
- Deployment is decoupled from activation
- Deployment is not “one size fits all”



Knight Capital was a financial service company on the stock exchange and was one of the largest traders in the U.S. market. Their market share was 17.3 of the New York Stock Exchange and 16.9 of NASDAQ.

The company had a retail liquidity program and wanted to replace old SMARS code with new RLP code. Their strategy was to repurpose a flag that turned it on and off. The old code was a power pipe peg that remained on the server. They were going to upgrade it, switch the flag, and then start running the new code. So, what could go wrong?

On August 1, 2012, they manually upgraded seven of the eight servers. A system administrator upgraded seven of them and forgot to upgrade the eighth server. They went into test mode and thought they were testing servers when they flipped the switch. Instead, the eighth server ran real transactions on the real Internet to the tune of a \$640 million loss.

This caused a major disruption in the prices of 148 companies listed on the New York Stock Exchange. Incoming parent orders were processed. The defective power pipe peg sent millions of child orders resulting in 4 million executions of 154 stocks of more than 397 million shares in approximately 45 minutes. The next day Knight Capital was out of business—bankrupt—because someone did not upgrade that eighth server. The old code ran, and it made transactions it should not have made.

What could they have done differently to prevent this type of error?

This case is a great example of how using Continuous Delivery to automate deployments could have avoided one server being configured differently from any other server. Without automation, every deployment is an opportunity for human error and a chance of breaking the application. Automated deployment and testing for a test environment, staging environment, and production-like environment would have ensured that the change would not have caused this catastrophe.

Organizational Impact on Devops

Conway's Law

Which statement illustrates Conway's law?

It is good practice to organize teams around what they are expected to produce.

Eg four teams are directed to build a compiler like interface team , application team and database team. I get a 3 tier architecture is called Conway law in action;

How does organization affect

Is the culture of your organization agile?

- Small teams
- Dedicated teams
- Cross-functional teams
- Self-organizing teams

Traditional organization around technology

Organization structure



User interface team



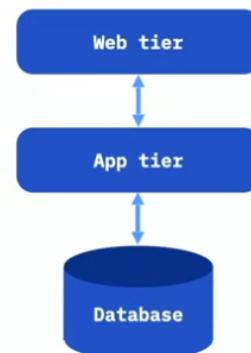
Application team



Database administrator (DBA)
team

IBM Developer

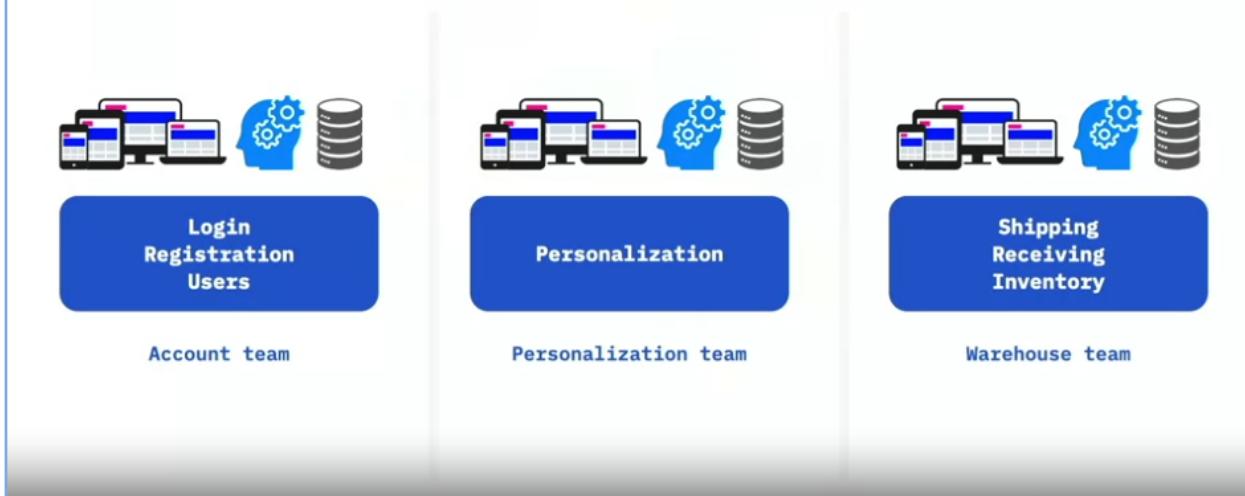
Application structure



SKILLS NETWORK 

Organisation is so important.

Organized around business domains



Aligned team with business

Each team has its own mission aligned with the business

In team we need engineer aligned around 5-7 not more than 10

End-End responsibility. They commit build deploy build everything.

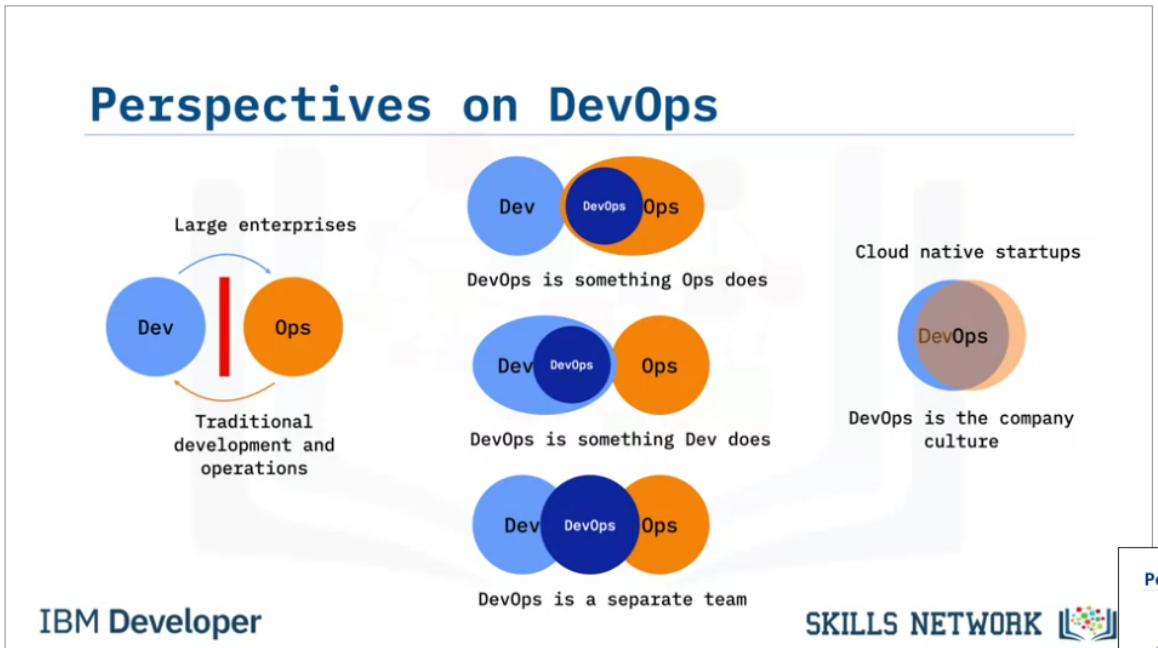
Single business domain in that they have long term vision.

There is no devops team

Devop is development if you are not doing development you are not doing devops, hence you are doing ops. Software engineer practices devops so it's a subset of software engineer

Devops is a culture that whole company mindset accept

 Notes  Discuss



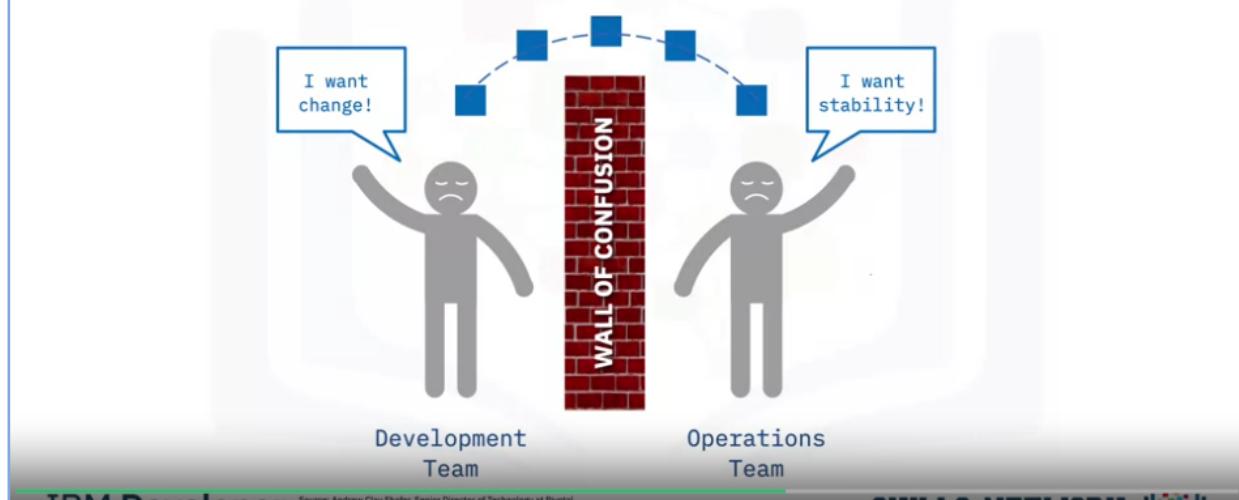
Its a culture for a company.

DevOps is not a team

“The DevOps movement addresses the dysfunction that results from organizations composed of functional silos. Thus, creating another functional silo that sits between Dev and Ops is clearly a poor (and ironic) way to try and solve these problems.”

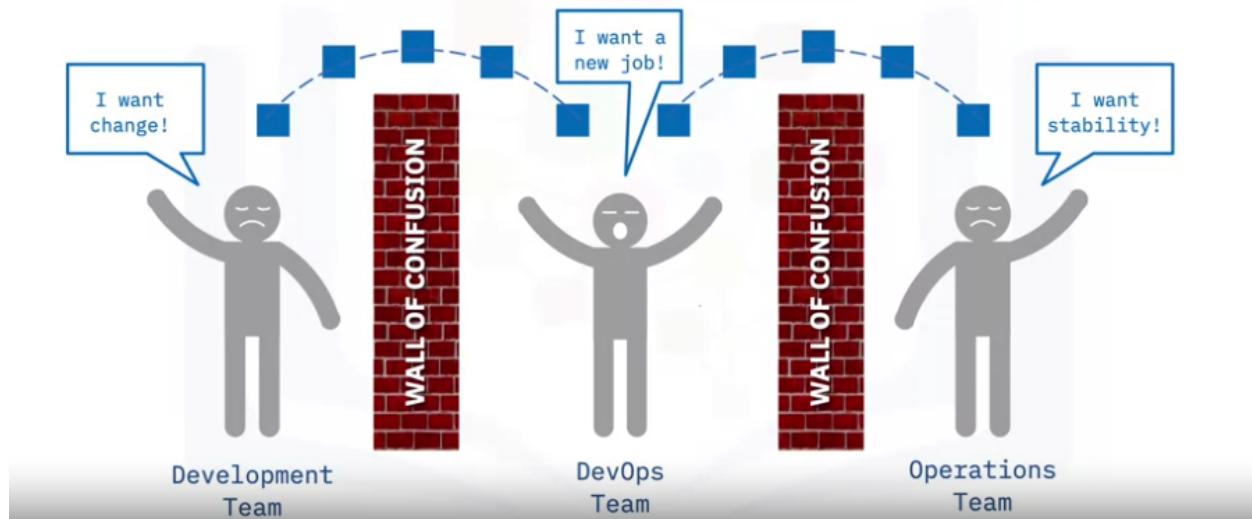
Why working in silos never work

Working in silos doesn't work



Creating a separate devops team

A DevOps team means we're DevOps, right?



This is not at all correct for the coding and devops culture
Creating another silos means creating above mistake
again and again

DevOps is not a job title

- A cultural transformation on an organizational scale
- Development and operations engineers working together
- Following lean and agile principles
- Cross-functional teams with openness, transparency, and trust as pillars



Bad behaviour rises when

- When Dev and ops both work in silos then the quality actually decreases
- Action have consequences**
- Here are two things that you can do to avoid the problem. You can create cross-functional teams so that everyone is dealing with teammates instead of ticket queues. If you can't create cross-functional teams, then have the developers rotate through the operations so they can feel empathy for what the operations engineers do every day to keep their application running. You can also have the operations engineers join the developer standups and their showcases so that they can understand what the developers are doing.
 - Increase there responsibility

Let them the pain of code by recalling them and they will start writing better codes.

Bad behavior

“Bad behavior arises when you abstract people away from the consequences of their actions.”

-Jez Humble,
Continuous Delivery

Devops orginisational Objectives

Here are two things that you can do to avoid the problem. You can create cross-functional teams so that everyone is dealing with teammates instead of ticket queues. If you can't create cross-functional teams, then have the developers rotate through the operations so they can feel empathy for what the operations engineers do every day to keep their application running. You can also have the operations engineers join the developer standups and their showcases so that they can understand what the developers are doing.

Organizations need to have small, dedicated, cross-functional, self-organizing teams to successfully implement DevOps.

-

Conway's Law implies that a company's design results are a direct reflection of the company's communication structure.

-

Instead of the traditional structure organized around technology, successful DevOps teams should be organized around business domains. Each team should have its own mission that aligns with a business domain.

-

DevOps is a mindset that the whole organization adopts.

-

DevOps solves problems caused by siloed teams.

-

DevOps is the practice of development and operations engineers working together during the entire software lifecycle, following lean and Agile principles that allow them to deliver high-quality results.

-

Actions without consequences can lead to apathy.

-

Allowing teams to feel the effect of their actions fosters empathy, resulting in higher-quality work.

-

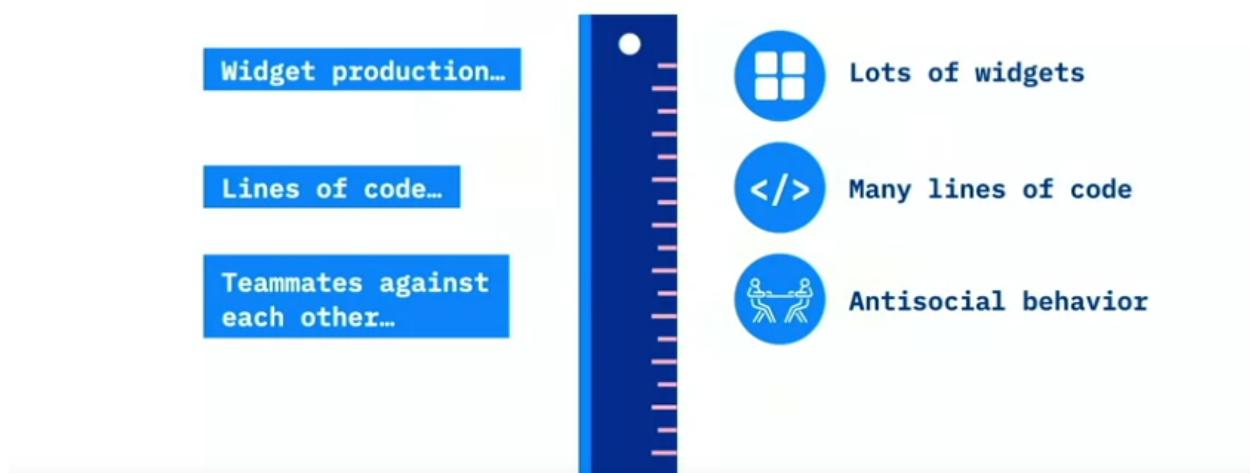
- The organizational objective of DevOps is to attain a shared mindset and empower everyone to deliver customer value.

Rewarding for “A” while hoping for “B”

We cannot measure for A nd get B its not going to happen. Like it's to get what we are actually measure

Measure what matters

Measure what matters



If you are measuring these remember that you will get these as an outcome .

We need to get social metric to get social behaviour

Are you building code that the rest of the company or open-source community find valuable enough to reuse in their solutions? But that's only half of the puzzle. That will get people thinking about how

they can make their code valuable to others. But you need the second metric. Whose code are you leveraging? Are you wasting all of my development dollars reinventing wheels or are you leveraging the existing wheels and only building the bespoke parts that don't exist? Both of these metrics are needed because they incentivize both sides to share their code and reuse each other's code because... you get what you measure. You need a social metric to get social behavior.

Devops is all about continuous improvement.

Devops metrics

In this base line provides a concrete number for a comparison of as u implement for devops changes

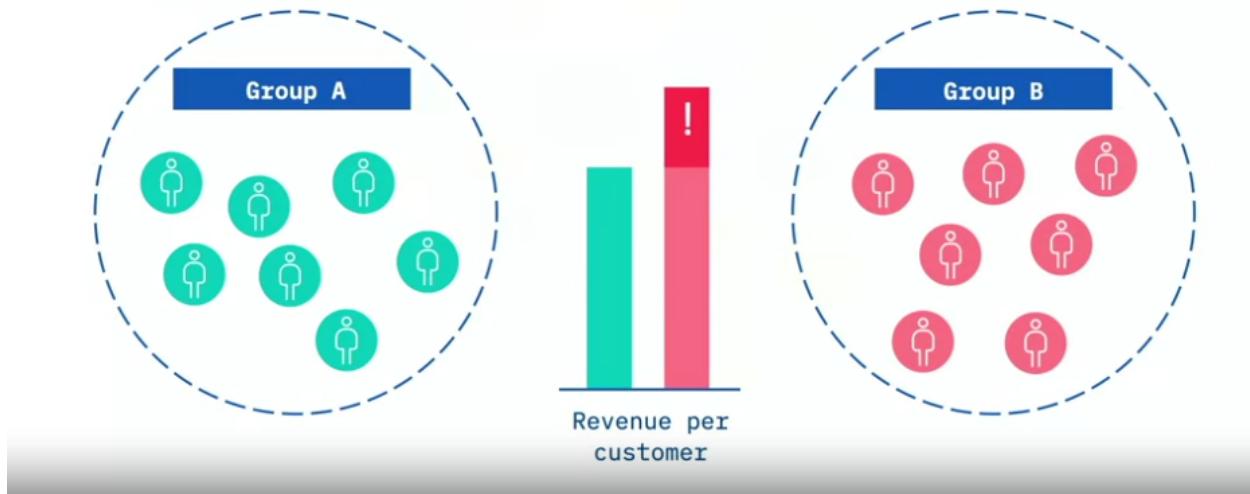
Metric goals allows you to reason about these numbers and judge the success of your progress

DevOps changes the objective

- Old school is focused on mean time to failure (MTTF)
- DevOps is focused on mean time to recovery (MTTR)



Actionable metrics



Here a website exposure and outcome is taken in form of data of peoples divided in group a nd group b

Where in group A person is actually knowing interface of website and group b faces new interfaces, which leads to high number of clicks. So this comes under actionable metrics.

example — * Time reduction in market

- Increase overall availability
- Reduces time in deployment
- And defects are detected before production.
- More use of hardware infrastructure
- A Quicker performance feedback.

Top four actionable metrices

1. Mean Lead time :- how long it takes to get features in there hand
2. Release frequency :- how quickly i can release things.
3. Change failure rate:-Changes arent failing , Speed is meaning less if doesnt stabilises the system.
4. Mean time to recovery (MTTR):-The total time take for the recovery.Fail and it should recover.

Correct!

You have made the right matches.

Mean Lead Time is the amount of time it takes to go from the idea stage to delivering a product to production.

Change Failure Rate is how often changes, such as new releases, fail.

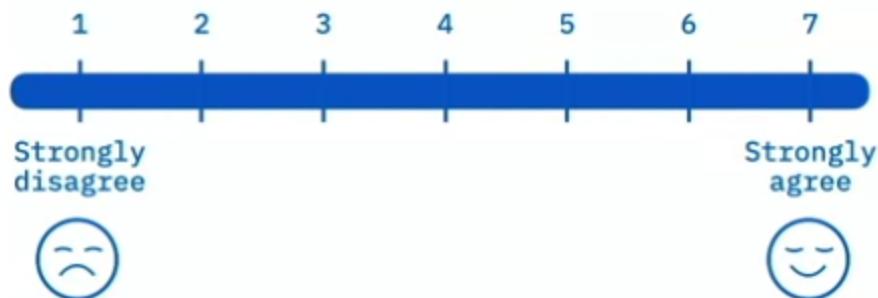
Mean Time to Recovery is how long it takes to recover when something fails.

Continue

How to measure Culture .

Culture measurements

Rate how strongly you agree (7) or disagree (1) with the following statements



Strongly agree or disagree?

- On my team, information is actively sought
- On my team, failures are learning opportunities and messengers of them are not punished
- On my team, responsibilities are shared
- On my team, cross-functional collaboration is encouraged and rewarded
- On my team, failure causes inquiry
- On my team, new ideas are welcomed

Comparison of Devops to Site Reliability Engineering

Site reliability Engineering

- What is site reliability Engineering ?
- How does it differ from DEVOPS?
- How can you leverage SRE in a Devops environment ?
- WHat is SRE

“What happens when a software engineer is tasked what use to be called operations “

What is SRE?

“...what happens when a software engineer is tasked with what used to be called operations.”

-Ben Treynor Sloss

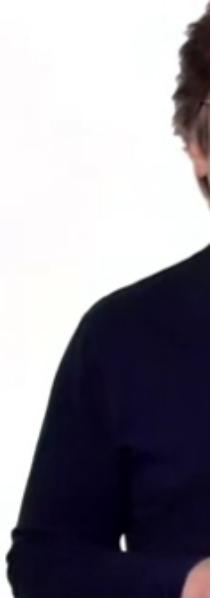
Goal: Automate yourself out of a job ☺



Tenet of SRE

Tenets of SRE

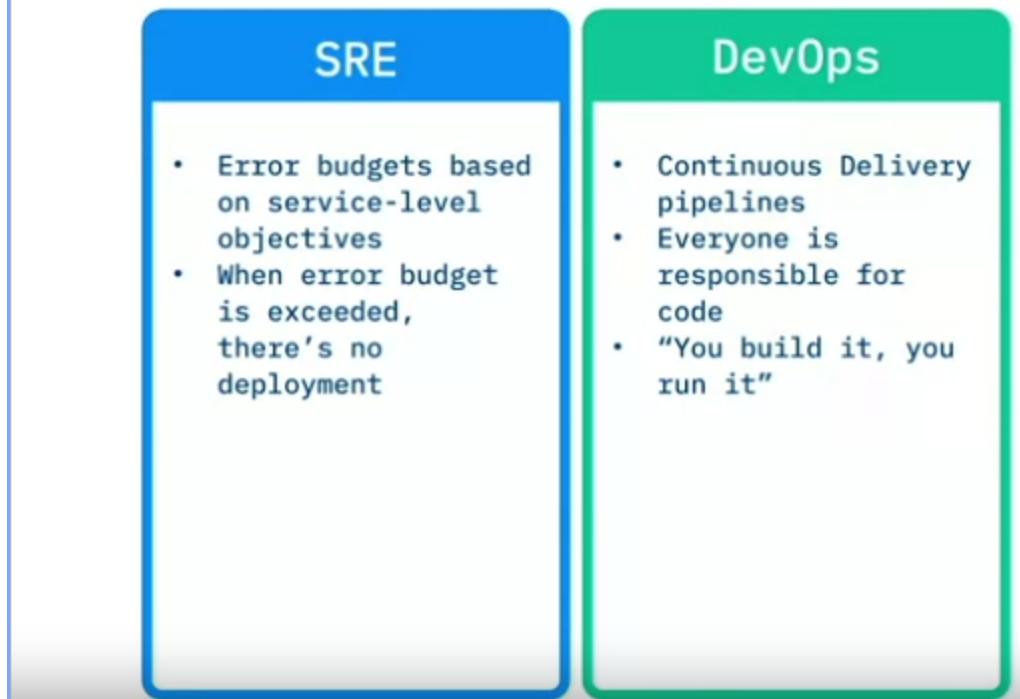
- Hire only software engineers
- Site reliability engineers work on reducing toil through automation
- SRE teams are separate from development teams
- Stability is controlled through error budgets



Team differences

- SRE maintains separate development and operations silos with one staffing pool
- DevOps breaks down the silos into one team with one business objective

Maintaining stability



SRE

In this we(developers) are responsible for production.

Commonality

- Both are practising Dev and ops which is visible to each other
- They both are blameless culture
- The objective of both is to deploy software faster with stability.

SRE+ DEVOPS

Sre maintains the infrastructure

Devops uses the infrastructure to maintain their applications

Sre is the environment who operate the cloud

Devops are the people who are consumin the cloud.

The screenshot shows a Microsoft Windows desktop with a browser window open to a Coursera course page. The page displays the 'Summary and Highlights' section of a course titled 'Introduction to DevOps'. The left sidebar lists various course components with green checkmarks: 'Video: Rewarding for "A" while hoping for "B"' (5 min), 'Video: Vanity metrics vs. actionable metrics' (4 min), 'Ungraded Plugin: Activity: Defining Actionable Metrics' (1 min), 'Practice Quiz: Practice Quiz 1: Measuring DevOps' (4 questions), 'Video: How to Measure Your Culture' (3 min), 'Video: Comparison of DevOps to Site Reliability Engineering' (6 min), 'Practice Quiz: Practice Quiz 2: Measuring DevOps' (2 questions), 'Discussion Prompt: Module 5 Discussion' (5 min), 'Reading: Summary and Highlights' (3 min), and 'Quiz: Measuring DevOps' (5 questions). To the right, the main content area is titled 'Summary and Highlights' and includes a congratulatory message: 'Congratulations! You have completed this lesson. At this point in the course, you know:' followed by a bulleted list of 17 items detailing the knowledge gained. A blue 'Mark as completed' button is at the bottom. The taskbar at the bottom of the screen shows various pinned icons and the system tray with weather information (29°C Mostly sunny) and system status (EN6 02-04-2023 13:39).