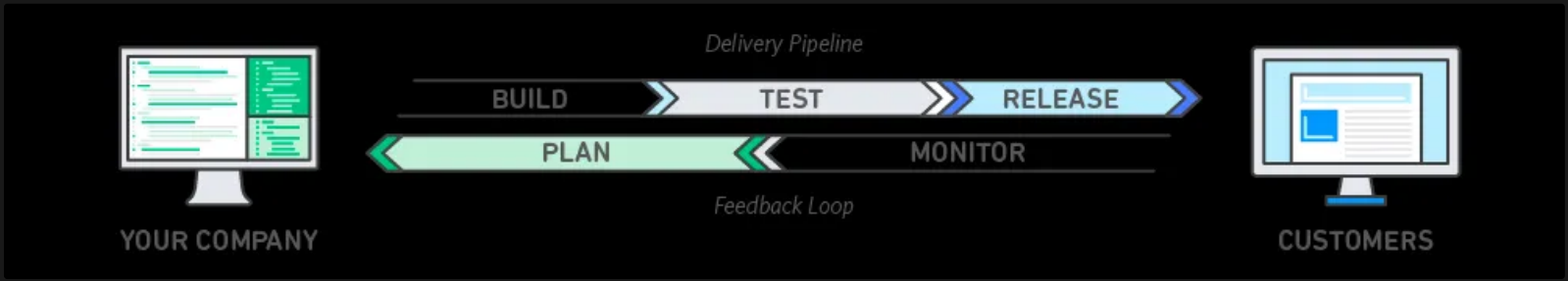
DEVops Notes

WHAT IS DEVOPS ?

DevOps is a set of practices that works to automate and integrate the processes between software development and IT teams, so they can build, test, and release software faster and more reliably. The term DevOps was formed by combining the words “development” and “operations” and signifies a cultural shift that bridges the gap between development and operation teams, which historically functioned in siloes.

LECTURE -1 slides



Delivery pipeline for a software

Continuous Delivery:

Continuous Delivery (CD) is the process which allows developers and operations engineers to deliver bug fixes, features and configuration changes into production reliably, quickly and sustainably. Continuous delivery offers the benefit of code delivery pipelines that are routinely carried out that can be performed on demand with confidence.

The benefits of CD are:
• Lower Risk Releases
• Faster Bug Fixes & Feature delivery
• Cost savings

Continuous Integration (CI) :

Continuous Integration (CI) is a practice in which developers will check their code into a version-controlled repository several times per day. Automated build pipelines are triggered by these check ins which allow for fast and easy to locate error detection.

The key benefits of CI are:
• Smaller changes are easier to integrate into larger code bases.
• Easier for other team members to see what you have been working on
• Bugs in larger pieces of work are identified early making them easier to fix resulting in less debugging work
• Consistent code compile/build testing
• Fewer integration issues allowing rapid code delivery

Lecture-2 slides

VCS (VERSION CONTROL SYSTEM)

Version Control Systems (VCS) allow a team to work on the same project, modify the same files and work collaboratively amongst small to large groups when managing changes. The problems solved by a VCS may not be apparent at first, but after making file changes either independently or amongst a team, you may be surprised how easily headaches can accrue when VCS is not used.

vcs basic actions:

Basic Actions:
• **Add:** Put a file into the repo for the first time, i.e. begin tracking it with Version Control.
• Revision: What version a file is on (v1, v2, v3, etc.).
• **Head:** The latest revision in the repo.
• **Clone:** Download a remote repository
• **Check out:** Download a file from the repo.
• **Commit:** Upload a file to the repository (if it has changed). The file gets a new revision number, and people can "check out" the latest one.
• **Commit Message:** A short message describing what was changed.
• **Changelog/History:** A list of changes made to a file since it was created.
• **Pull/Fetch:** Fetch your files with the latest from the repository. This lets you grab the latest revisions of all files.
• **Revert:** Throw away your local changes and reload the latest version from the repository.

Branch: Create a separate copy of a file/folder for private use (bug fixing, testing, etc). Branch is both a verb ("branch the code") and a noun ("Which branch is it in?").

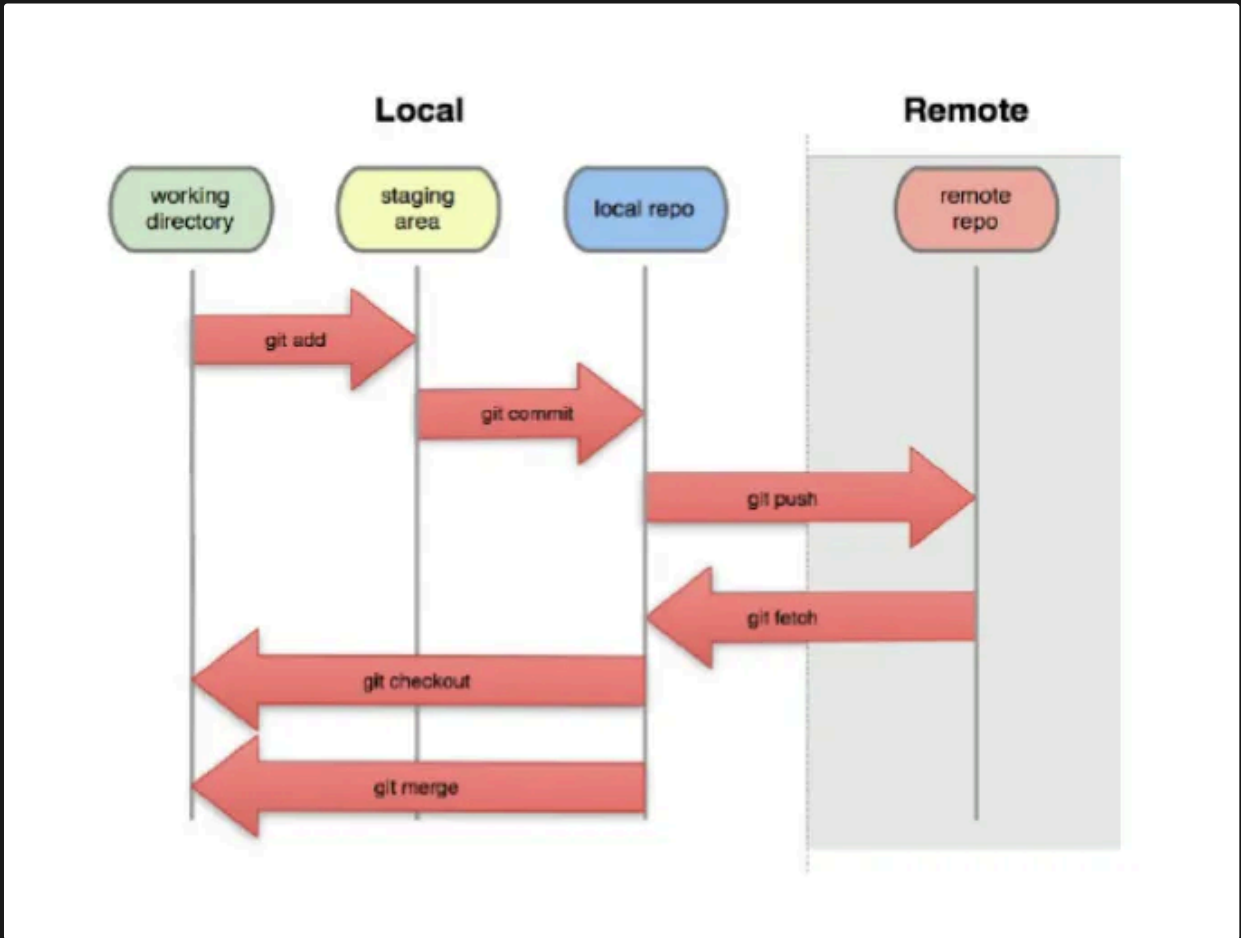
• **Diff/Change/Delta:** Finding the differences between two files. Useful for seeing what changed between revisions.

• Merge (or patch): Apply the changes from one file to another, to bring it upto-date. For example, you can merge features from one branch into another.

• Conflict: When pending changes to a file contradict each other (both changes cannot be applied).

• Resolve: Fixing the changes that contradict each other and checking in the correct version.

• Fork: Forking or cloning your repository is an alternative to branching. Where a branch is an internal copy of a repository, a fork is an external copy: a separate repo that can eventually be merged back to the forked repo.



Git was created by Linus Torvalds

SSH - Secure Shell protocol

We'll be starting things off with a protocol commonly
used to access servers, Secure Shell better known as SSH. When we want to push a local change from our dev machine to our remote repository on GitHub, the protocol that does the heavy lifting for us is SSH.

SSH allows a client to access the contents of a remote server. It provides the ability for one to remotely log in and potentially have access to the file contents of the server. It's one of the most common and safe ways to administer remote servers. The safety mechanisms in place for SSH are made possible by establishing a cryptographically secure connection between the client and the server

---

When connecting to a remote GitHub repo, the connection is established using SSH which realis on asymmetrical encryption. Asymmetrical encryption involves two associated keys. One of these keys is known as the private key, while the other is called the public key.

week 3 Lecture (monolithic apps vs microservices app)

MONOLITH ARCHITECTURE	MICROSERVICES ARCHITECTURE
A monolithic app is built as a single unified unit.	Microservices are not actually “micro”, while they tend to be smaller than the average monolith, they do not have to be tiny.
Monolithic app has three parts: 1.)a database, 2.)a client-side interface (consisting of HTML pages &/or javascript running in a browser). 3.) a server-side application that will handle HTTP requests, execute domain-specific logic, retrieve and update data from the database, and populate HTML views to be sent to the browser	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.
In a monolith, server side application logic, front end client side logic, background jobs, etc, are all defined in the same massive code base.	These services are built around business capabilities and independently deployable by fully automated deployment machinery.

PROS AND CONS

PROS : MONOLITHIC ARCHITECTURE	CONS: MONOLITHIC ARCHITECTURE
Fewer Cross-cutting Concerns: It has large Number of cross-cutting concerns, such as logging, rate limiting, and security features such audit trails and DOS protection. When everything is running through the same app, it’s easy to hook up components to those cross-cutting concerns	Tightly Coupled: Monolithic app services tend to get tightly coupled and entangled as the application evolves, making it difficult to isolate services for purposes such as independent scaling.
Less Operational overhead: Having one Large app means there is only one app to log, monitor , test.	Harder to Understand: Monolithic architectures are also much harder to understand, because there may be dependencies, side-effects, and magic which are not obvious when you’re looking at a particular service or controller
Performance: There can also be performance advantages , since shared-memory access is faster than inter-process communication	

PROS: MICROSERVICE ARCHITECTURE	CONS: MICROSERVICE ARCHITECTURE
Better Organization Each microservice has a very specific job, and is not concerned with the jobs of other components.	Cross-cutting Concerns Across Each Service When building a new microservice architecture, you're likely to discover lots of cross-cutting concerns that you did not anticipate at design time. You'll either need to incur the overhead of separate modules for each crosscutting concern (i.e. testing), or encapsulate cross-cutting concerns in another service layer that all traffic gets routed through
Decoupled Decoupled services are also easier to recompose and reconfigure to serve the purposes of different apps For e.g. serving both the web clients and public API). It allows fast and independent delivery of individual parts within a larger, integrated system.	Higher Operational Overhead Microservices are frequently deployed on their own virtual machines or containers, causing a increase of VM wrangling work.
Performance Under the right circumstances, microservices can also have performance advantages depending on how they're organized because it's possible to isolate hot services and scale them independent of the rest of the app.	

VERTICAL SCALING	HORIZONTAL SCALING
Vertical scaling refers to adding more resources (CPU/RAM/DISK) to your server (database or application server is still remains one) as on demand.	Horizontal Scaling is a must use technology – whenever a high availability of (server) services are required
Vertical Scaling usually means upgrade of server hardware	Scaling horizontally involves adding more processing units or physical machines to your server or database
Sometime to scale vertically includes increasing IOPS (Input / Output Operations), amplifying CPU/RAM capacity, as well as disk capacity.	The response why organizations should choose to scale horizontally include increasing I/O concurrency, reducing the load on existing nodes, and increasing disk capacity.

WHAT IS A TECH STACK?
A technology stack, also called a solutions stack, technology infrastructure, or a data ecosystem, is a list of all
the technology services used to build and run one single application.
It is a combination of software applications, frameworks, and programming languages that realize some aspects of the program

week4 - DevOps in Detail

Job Roles and Responsibilities for a DevOps Engineer

- **DevOps Evangelist** – The principal officer (leader) responsible for implementing DevOps
- **Release Manager** – The one releasing new features & ensuring post-release product stability
- **Automation Expert** – The guy responsible for achieving automation & orchestration of tools
- Software Developer/ Tester – The one who develops the code and tests it
- Quality Assurance – The one who ensures the quality of the product confirms to its requirement
- Security Engineer – The one always monitoring the product's security & health

DevOps planning tool to consider: JIRA

JIRA by Atlassian is an Agile-friendly planning tool.
Other DevOps planning tools: Redmine, Trac, Rally

DevOps build tool to consider:

Apache Maven
Apache Maven is a tool for Java-based development.

Other DevOps build tools			
Apache Ant	Gulp	NAnt	Travis CI
Broccoli	Hudson	Packer	UrbanCode Build
Buildbot	Jam	QuickBuild	Visual Build
BuildMaster	LuntBuild	Rake	Visual Studio
FinalBuilder	Make	sbt	
Buildr	Meister	SSH	
CMake	Microsoft Build Engine (MSBuild)	TeamCity	

DevOps CI/CD tool to consider: Jenkins

Jenkins is an open source automation server that provides a plugin architecture to support continuous integration and delivery.

Other DevOps CI/CD tools			
Apache ActiveMQ	Continua CI	Hudson	Travis CI
Bamboo	Continuum	Shippable	
Circle CI	CruiseControl	Snap CI	
Codeship	Gump	Solano CI	

DevOps deployment tool to consider: Capistrano

→ Capistrano is a remote server automation and deployment tool primarily used for deploying web applications.
→ It is written in Ruby and can extend to support other coding languages or frameworks with special requirements.

Other DevOps deployment tools			
Automic	Deploybot	JuJu	RapidDeploy
AWS CodeDeploy	Deployer	Nomad	Rundeck
BMC Release	ElectricFlow	Octopus Deploy	SmartFrog
CA Nolio	Go	Otto	XL Deploy
Containership	Google Deployment Manager	Rancher	