DevSecOps Notes

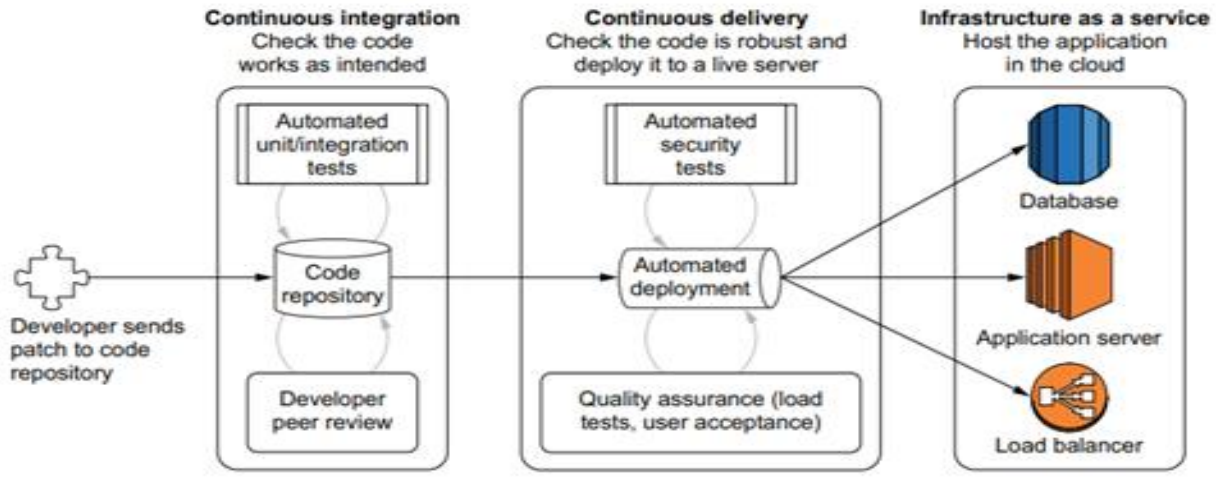
Helpful Points:

- Infrastructure As a Service (IaaS) is the cloud.
- Here, the data center, network servers, and occasionally systems that an organization relies on, are entirely operated by a 3rd party, and exposed to operators as a service.
- AWS/Azure/GCP are examples of IaaS environments.
- CI, CD, and IaaS proficiency allow organizations to deploy software to end users rapidly.
- “A Ship Is Safe in Harbor, But That's Not What Ships Are Built For” - John A Shedd (Something to Think About)

1. CI/CD

Continuous Integration

- A process where collaborators/developers contribute their code to a central repository.
- This repository is further integrated into an automated system which:
  - pulls the code
  - builds it,
  - runs unit tests,
  - fails the build,
  - gives feedback in case there are bugs, and
  - prepares the artifact so that it is deployment-ready



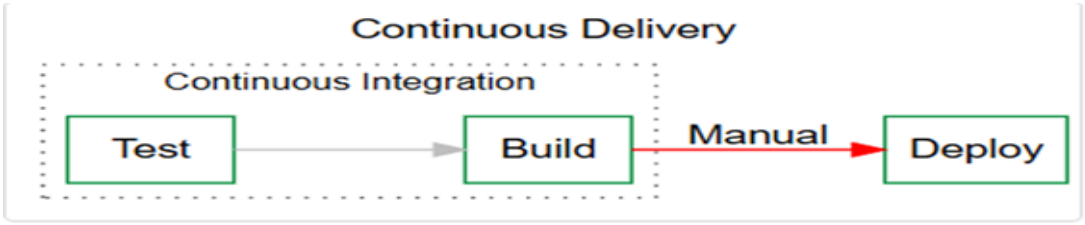
- CI ensures that software components or services work together.
- The integration process takes place and completes frequently, this allows for:
  - Increased frequency of developer code commits

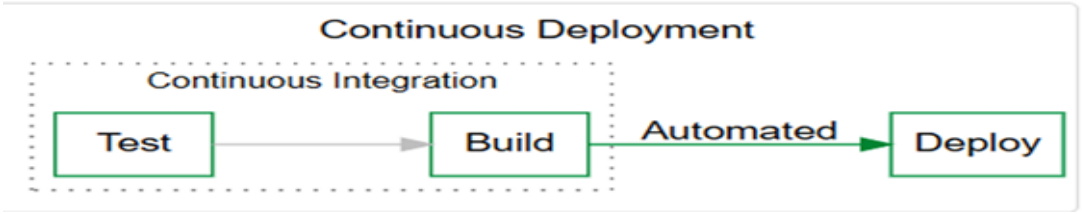
- Reduced chances of non-compatible code and redundant efforts.

- To implement a CI process, the following tools are required:
  - Version Control System (VCS)
  - Build tool
  - Artifact repositories manager

Continuous Deployment/Continuous Delivery

- An automated process where the generated executable files or packages (in the CI process) are installed or deployed on application servers.
- Continuous Integration is the first step toward achieving Continuous Delivery.
- Difference between continuous deployment and delivery:
  - Continuous delivery is a process where all the steps after CI, happen with human intervention (Human intervention means either clicking a button to deploy/ allowing a bot to approve it).
  - Continuous deployment is a process where the deployment of a successful build to the application server occurs without human intervention (i.e., in an automated way). This is shown below:





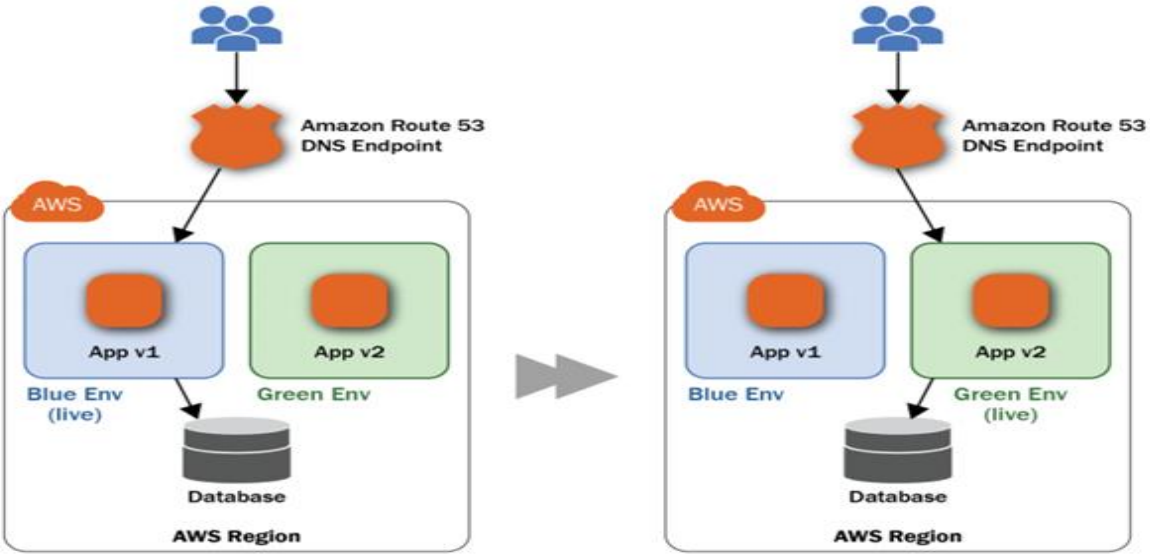
Common CD Strategies

- used in DevOps methodologies include:
  - Blue-green deployment
  - Canary deployment
  - Recreate deployment
  - A/B testing deployment

a) Blue-Green Deployment

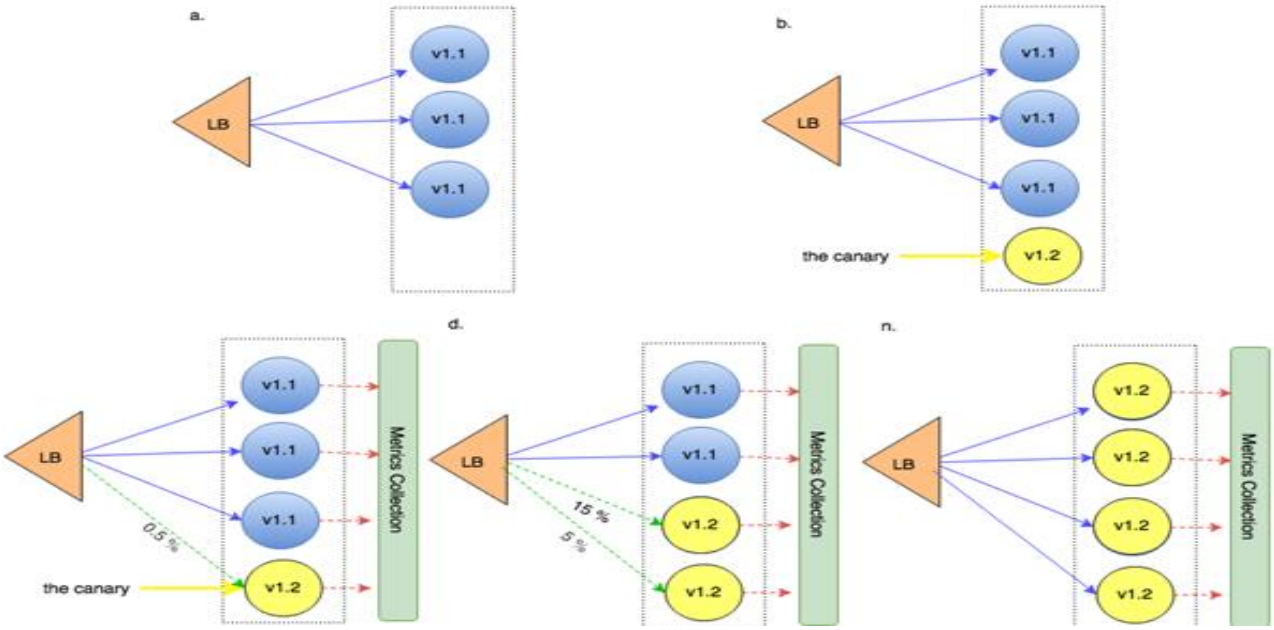
- Deployment strategy or pattern where downtime is reduced by having two production environments.
- Advantageous as it provides near-zero-downtime rollback capabilities.
- The idea is to shift the traffic from the current environment to a 2nd yet identical environment.

- As they are identical, they both run the same application but have different versions.



b) Canary Deployment

- Here applications or services are deployed incrementally to a subset of users.
- As a Subset of users begins using the application or service:
- Important application metrics are collected and analysed
- Based on those metrics, a decision on which version is good to go full scale ahead and rolled to all users or needs to roll back for troubleshooting.
- The infrastructure in production environments is updated in small phases (for example, 10%, 20%, 50%, 75%, 100%).

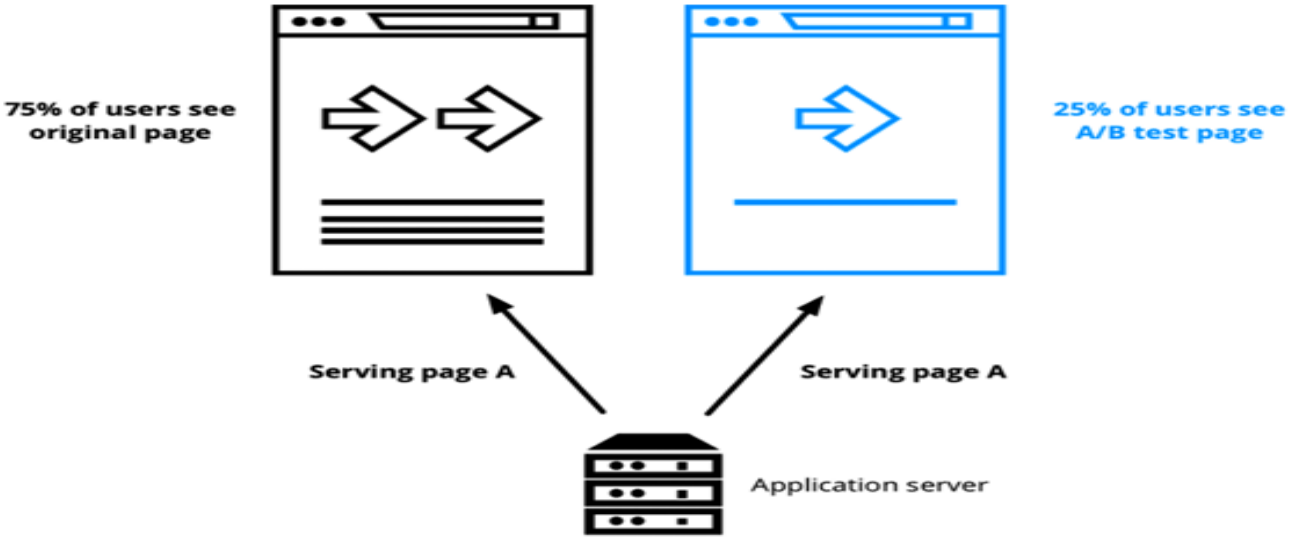


c) Recreate Deployment

- With this deployment strategy, the older version of an application is stopped before deploying the newer version.
- For this deployment, service downtime is expected, and an entire restart cycle is executed.

d) A/B testing Deployment

- A deployment whereby different versions of the same application/ services are run simultaneously for experimental purposes
- This is done in the same environment for a certain period.
- It is a strategy that consists of routing the traffic of a subset of users to a new feature or function.
- Feedback and metrics are collected and compared with the older version.
- After a comparison is made and feedback is given, the decision-maker updates the entire environment with the chosen version of the application/services.

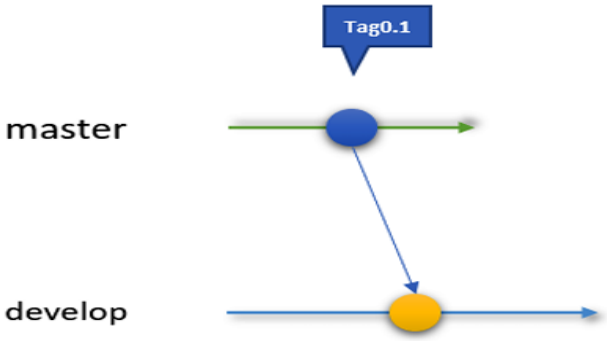


Branching Strategy

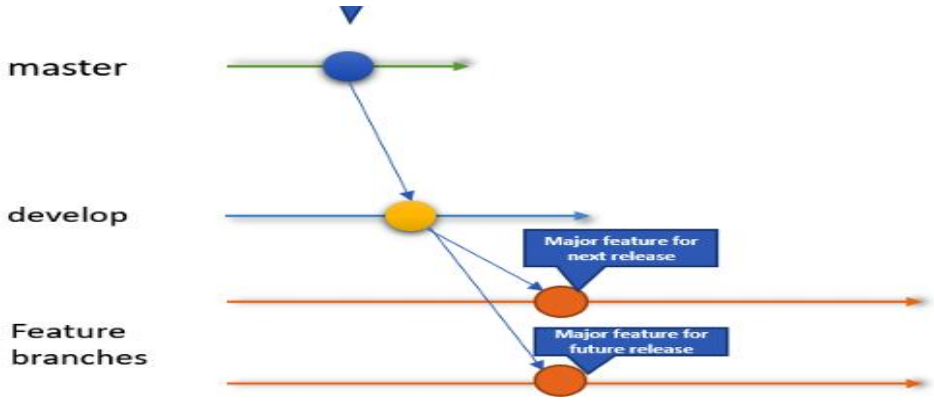
- Impossible to have a good CI and CD strategy without a good branching strategy,
- When a developer writes code on a local machine:
  - ✓ On completing the code, they upload/push it to a VCS for example Git
  - ✓ This is done to store the code so that other developers can use it
  - ✓ It can also be tracked and versioned.
  - ✓ If it is a developer's first time pushing the code to Git:
    - ❖ It goes to the/main branch.
    - ❖ This branch is independent and is an abstraction for the edit/commit/stage process.

- Suppose we have a project to implement a calculator:
  - The calculator will have addition, subtraction, multiplication, division, and average functions.
  - We have three developers (Amy, Kay, and Aaron) and a manager (Mark) to complete this project.
  - Mark has asked them to deliver the addition function first.
    - ✓ Amy quickly developed the code, built, tested it, and pushed it to the ***Git main branch***, tagged it with version 0.1, as illustrated in the following screenshot.
    - ✓ The code in the main Git branch always reflects the production-ready state, meaning the code for addition will be running in the production environment.

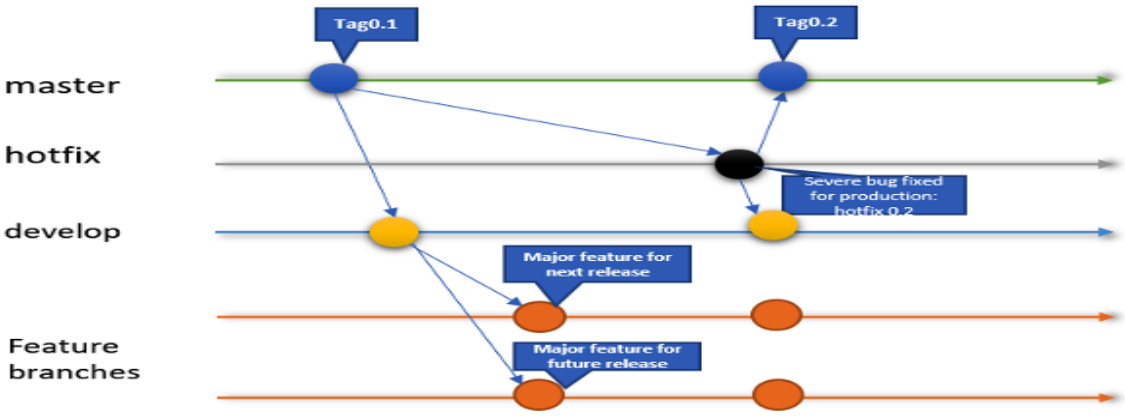
  - Mark now asks Kay to start the development of subtraction and multiplication functions.
    - ✓ This will be the significant functionality for the calculator project for the next release.
    - ✓ Mark also asks Aaron to develop a division and average functions as a functionality for a future release.
    - ✓ The best way to move ahead is to create a ***develop branch***
    - ✓ The develop branch is a replica of the working code in the main branch.



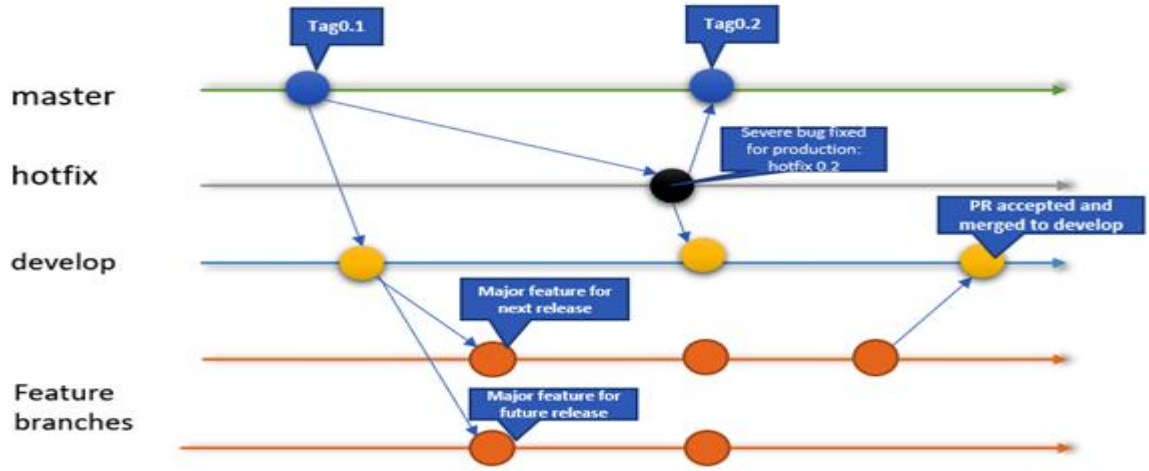
- Once the develop branch is created out of the master:
  - ✓ it will have the latest code of the addition function
  - ✓ Kay and Aaron will create a ***feature branch*** out of the develop branch to work on their tasks
  - ✓ Feature branches are used to develop new features for upcoming releases.
  - ✓ It branches off from the develop branch
  - ✓ must merge back into the develop branch once the development of the new functionality completes



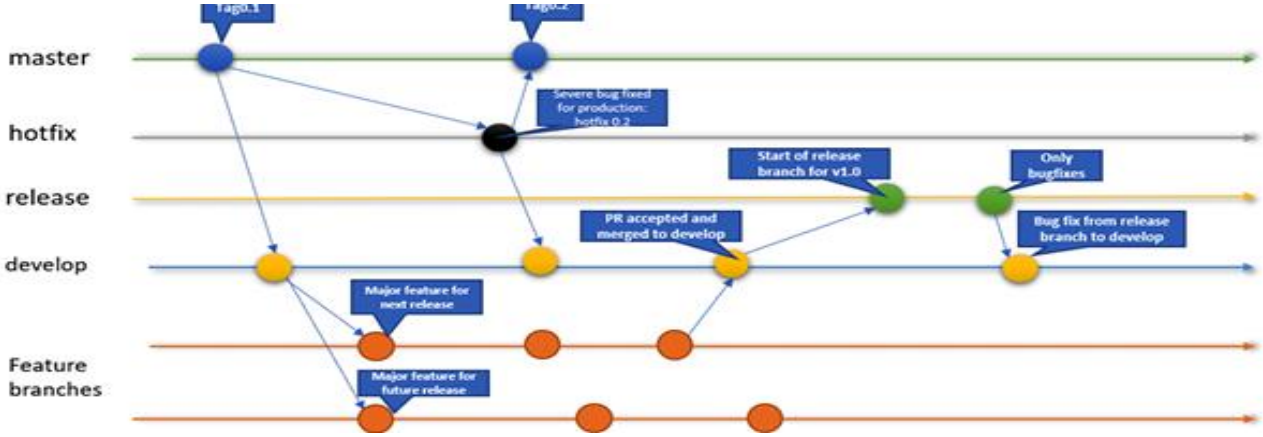
- Kay and Aaron have been working on their functionality and committing their branches:
  - ✓ Mark found a bug in the current live production environment and asked Amy to fix that bug.
    - Note that fixing any bug in a production environment is considered
    - bad practice.
  - ✓ Amy will have to create a ***hotfix branch*** from the main branch,
  - ✓ This branch is used to fix the bug in code,
  - ✓ Hotfix branches are required to take immediate action on the undesired status of the main branch.
  - ✓ Must branch off from the main branch and, after fixes are made to the code, must merge into the main branch as well as the develop branch
    - This ensures that the current develop branch does not have that bug
    - It also allows for a smooth deployment in the next release cycle.
    - Once the fixed code merges into the main branch, it gets a new minor version tag and is deployed to the production environment.



- Once Kay completes his development (subtraction and multiplication):
  - ✓ He will merge his code from the feature branch into the develop branch.
  - ✓ But before he merges, he needs to raise a ***PR/MR.***
  - ✓ This raises a request to the maintainer of the project
  - ✓ The Project Maintainer ***reviews the code before*** merging the new feature into the develop branch
  - ✓ An essential requirement to get a feature branch merged into the develop branch:
    - the feature branch should get built successfully without any failures and
    - must have passed a code quality scan.

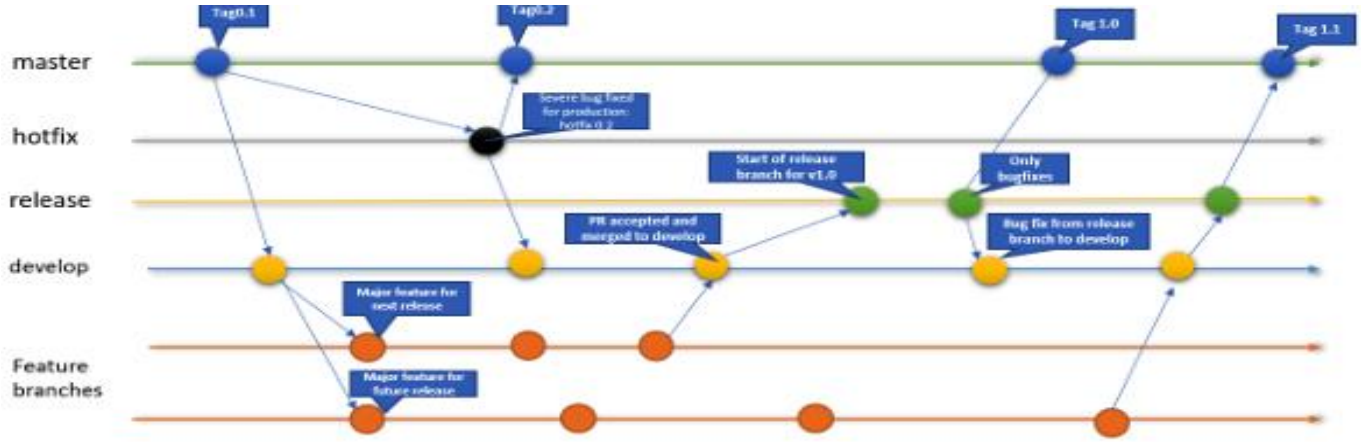


- Once Kay's code is accepted and merged into the develop branch, ***the release process*** will occur.
    - ✓ In the release process, the develop branch code gets merged into the release branch.
    - ✓ The release branch supports preparation for a new production release.
    - ✓ The code in the release branch gets deployed to an environment that is like a production environment.
    - ✓ That environment is known as staging (pre-production).
- A staging environment not only tests the application functionality but also tests the load on the server.
    - ✓ If bugs are found during the test, they must be fixed in the release branch and merged back into the develop branch.
    - ✓ Once the bugs are fixed and testing is successful, the release branch code will merge into the main branch.
    - ✓ It then gets tagged and deployed to the production environment.
- The application will have three functions: addition, subtraction, and multiplication.



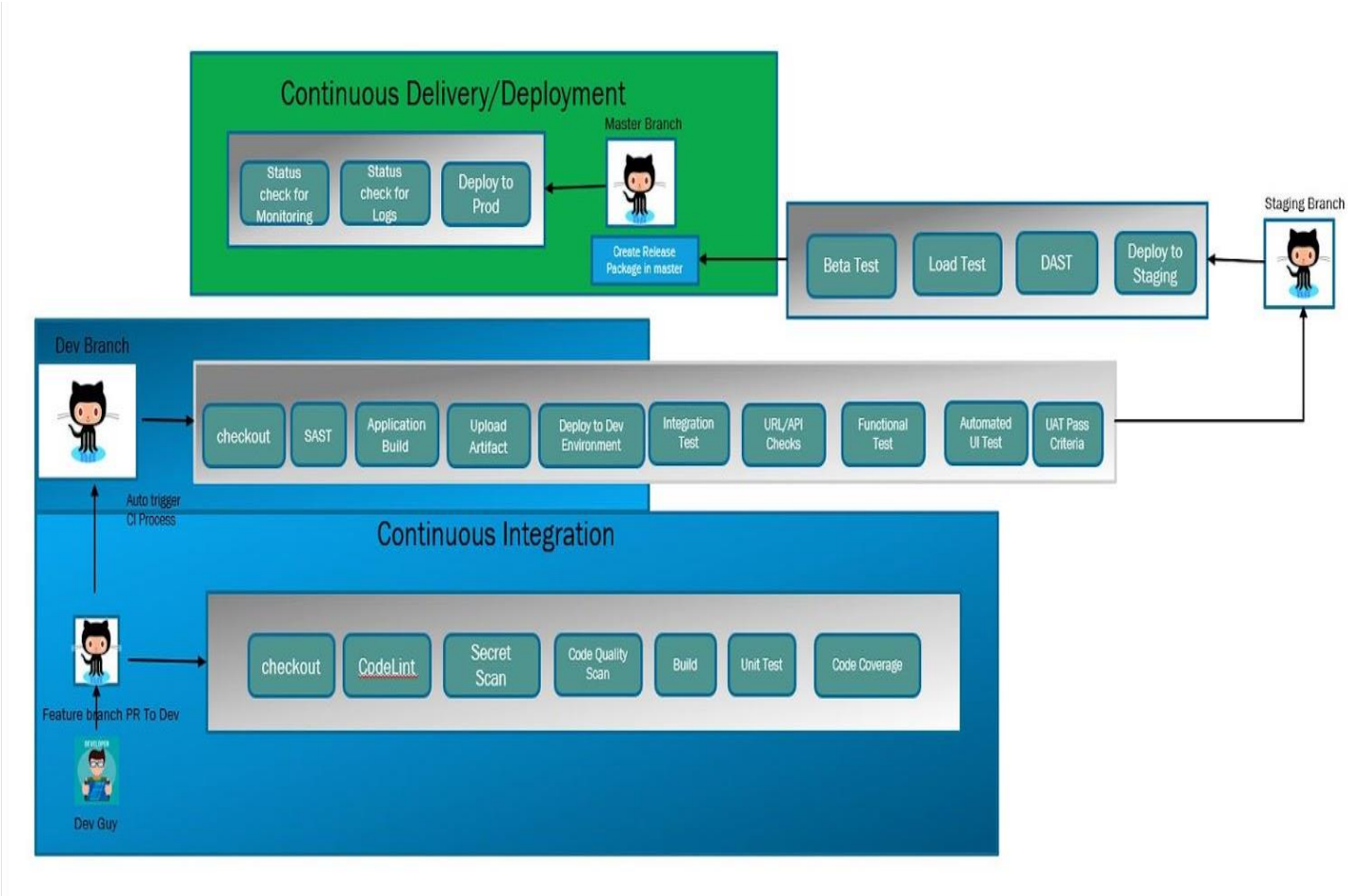
- A similar process will take place for the new features of division and average developed by Aaron.
    - ✓ the version will be tagged as 1.1 and deployed in the production environment
- Note that the main branches in the whole life cycle of an application include:
    - ✓ Master
    - ✓ Develop

- But during a development cycle, the supporting branches do not persist once the merge finishes. It includes:
    - ✓ Feature
    - ✓ Hotfix
    - ✓ Release



CI/CD Stages

- When a developer finish work in a feature branch, they raise a PR to the develop branch, and the CI pipeline gets triggered.



- The Continuous Integration pipeline (coloured in blue in the diagram) is an automated flow or process in any CI tool such as Jenkins, CircleCI or AWS Code Pipeline
    - ✓ The pipeline will validate whether the feature branch meets all the criteria to merge into the develop branch.
    - ✓ If the pipeline runs and builds successfully, then the lead maintainer of the project will merge the feature branch into the develop branch,
    - ✓ Another automated CI pipeline will trigger and try to deploy the new feature in the development environment.
    - ✓ If everything goes well and all the metrics are okay, the develop branch merges into the staging branch.
    - ✓ Another automated pipeline gets triggered during this merge process, which deploys the artifact.
    - ✓ The Artifact was uploaded into the staging environment during the develop branch CI process.
    - ✓ The staging environment is generally a close replica of the production environment,
        - Here some other tests, such as Dynamic Application Security Testing (DAST) and load stress testing, take place.
        - If all the metrics and data from the staging environment look good, then staging of the branch code gets merged into the main branch and tagged as a new version.
        - It is considered continuous delivery if the maintainer deploys the tagged artifact in the production environment
        - If this deployment occurs without manual effort, then it is continuous deployment.
            - Note this is a high-level overview of how application development and deployment take place in a CI/CD Environment. This concept of CI/CD and branching strategies are crucial in DevSecOps

2. AWS CodeStar

i. Aim is to learn how to use AWS CodeStar to implement a CI/CD pipeline that covers
    a. feature
    b. develop
    c. master branches and their respective environments.

ii. Includes how to modify the AWS CodePipeline stages and validate a PR using a lambda function and CodeBuild.

iii. If in a cloud-native company and want to create a CI/CD pipeline with cloud-native resources,
    a. These notes will help to easily do that without worrying about the CI/CD toolchain servers.

- CodeStar is an AWS Managed service
    - It allows developers to quickly develop, build, and deploy an application on AWS.
    - Provides all the necessary templates and instances to run an entire CI/CD toolchain
    - Achieves this using a CloudFormation stack.
    - Excellent service for focusing only on business logic rather than an infrastructure environment.

- It is cloud centric as it is integrated with the Cloud9 editor to edit application code and CloudShell to perform terminal/shell-related actions.
- AWS CodeStar is a free service.
    - ✓ note that payment for other resources that get provisioned with it will occur.
    - ✓ for example, if you use this service to deploy an application on an EC2 (Elastic Compute Cloud) instance, then there is no payment to use CodeStar but a charge for the EC2 example.

a) AWS CodeCommit

- A Version Control System that AWS manages
- It is used to privately store and manage code in the cloud and integrate it with AWS.
- It is highly scalable and secure.
- It hosts private Git repositories and supports the standard functionality of Git
- works very well with existing Git-based tools

b) **AWS CodeBuild**

- cloud-hosted and fully managed build services
- Compiles source code runs unit tests, and produces artifacts that are ready to deploy,

c) **AWS CodeDeploy**

- deployment service provided by AWS
- automates the deployment of an application to:
    - ✓ An Amazon EC2 instance,
    - ✓ Elastic Beanstalk and on-premises instances.
    - ✓ It provides the facility to deploy unlimited variants of application content such as code, configuration, scripts, executable files, multimedia, etc.
    - ✓ CodeDeploy deploys application files stored in S3 (Amazon Simple Storage Service) buckets, GitHub repositories, or Bitbucket repositories.

d) **AWS CodePipeline**

- It is used to automate all the software release processes.
- It's a Continuous Delivery and releases automation service helping to smoothen deployment.
- Can quickly configure the different stages of a software release process
- Automates the steps required to release software changes continuously.
    - ✓ Note the following:
        - The sample project template used is a Node.js web application
        - Elastic Beanstalk is used as an application compute infrastructure.
        - Create a Virtual Private Cloud (VPC) with a private subnet and an EC2 key pair

- If using Elastic Beanstalk for the first time on AWS, ensure the t2.micro instance type has enough Central Processing Unit (CPU) credit (see https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/ burstable-credits-baseline-concepts.html).
- It is necessary to set up an EC2 Instance with a key pair for the existing project before everything detailed below works. Visit https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/concepts.html

Get started

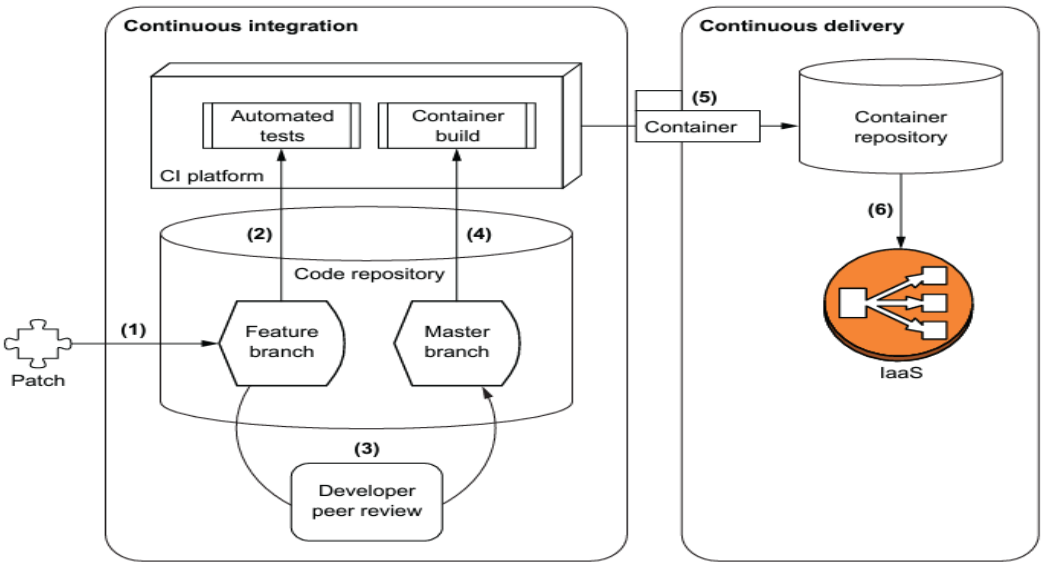
i. Log in to the AWS Management Console by going to this site https://aws. amazon.com/console/.
ii. Go to the search box and search for AWS CodeStar. Click on the result, which will redirect to AWS CodeStar's page.
iii. Click on Create project and be redirected to Choose a project template page where the information on creating a service role will be displayed. Click on Create service role,
iv. After that, a green Success message should pop up,
v. Click on the dropdown of the Templates search box, then click AWS Elastic Beanstalk under AWS service
    a. Select Web application under Application type,
    b. and Node.js under Programming language,
vi. Notice two search results, Node.js and Express.js. Go ahead with Node.js by clicking on the radio button and then on Next
vii. Will be redirected to another page called Set up your project. Here enter northstar as the Project name.
    a. This will auto-populate the Project ID and Repository name fields.
    b. Will be using CodeCommit for the code repository.
    c. In EC2 Configuration, select t2. micro, and select the available VPC and subnet.
    d. pick an existing key pair and then click Next,
viii. After that, review all the information related to the project and proceed to click Create project.
    a. The process takes 10-15 minutes to set up the CI/CD toolchain and Elastic Beanstalk
    b. It will then deploy the sample Node.js application in Elastic Beanstalk.
    c. While that occurs, go to CloudFormation, search for the awscodestar-northstar stack, and see all the resources that are getting provisioned,
ix. Can also check the Elastic Beanstalk resource by going to the Environments view of the Elastic Beanstalk console,
x. After 10-15 minutes, monitor the project's main page. The View application button gets enabled, indicating that the creation of a project is complete. This includes:
    a. a CI/CD toolchain and environment infrastructure
    b. the deployed application. This can be accessed by clicking on the View application button, which redirects to the Node.js sample web application indicating a successful connection.

- What happened at the backend during the build mentioned above?

- "CodeStar triggers an AWS CloudFormation stack to create an entire CI/CD toolchain and workload infrastructure." *("Creating a project in AWS CodeStar | Accelerating DevSecOps on AWS - Packt")*
- The toolchain includes the following:
  - ✓ A CodeCommit repository with the main branch having a sample Node.js application
  - ✓ A CodeBuild project with the preconfigured environment to run the build
  - ✓ CodePipeline to trigger the build and deploy the application
  - ✓ The workload infrastructure includes Elastic Beanstalk with one EC2 instance.
  - ✓ IAM roles with specific permissions that allow CloudFormation to perform actions on other services.

3. *Building Barebones DevOps Pipeline Using GitHub, CircleCI, Docker and AWS*

- Important to first understand how applications are built, deployed, and operated in DevOps before applying security as a part of DevOps.
- Focus here is on building a fully functional DevOps pipeline and understanding techniques used in DevOps before setting the stage for any security discussions
- The specific example used here, attempts to implement those patterns using **the invoicer**, a small web API that manages invoices through a handful of HTTP endpoints. It's written in Go and its source code is available at ***https://securing-devops.com/ch02/invoicer.***
- The various steps of CI, CD and IaaS that allow for quick release and deployment of new versions of the software to users will also be implemented here.
- The goal is to go from a patch submission to deployment in production using a mostly automated process
- The pipeline build described is composed of six steps, these steps have been mentioned and touched on earlier in the notes but will be mentioned here again:
  - ✓ the developer writes a patch and publishes it to a feature branch of the code repository.
  - ✓ Automated tests are run against the application.
  - ✓ A peer of the developer reviews the patch and merges it into the main branch of the code repository.
  - ✓ A new version of the application is automatically built and packaged into a container.
  - ✓ The container is published to a public registry.
  - ✓ The production infrastructure retrieves the container from the registry and deploys it.



- To build this pipeline, the environment requires:
    - ✓ A source code repository —solutions to manage source code: Bitbucket, Beanstalk, GitHub, GitLab, Source Forge, etc. GitHub will be used for this example
    - ✓ A CI platform — Again, the options are numerous: Travis CI, CircleCI, Jenkins, GitLab, etc It all depends on the needs and environment required. Here CircleCI will be used as it integrates easily with GitHub and allows SSH access to build instances. This is useful for debugging the build steps.
    - ✓ A container repository —Docker is the standard choice. Using the repository provided by Docker Hub at hub.docker.com.
    - ✓ An IaaS provider —Google Cloud Platform, Amazon Web Services (AWS), and Azure are popular IaaS providers Some organizations prefer to self-host their IaaS and turn to solutions like Kubernetes or OpenStack to implement a layer of management on top of their own hardware (note that Kubernetes can also be used on top of EC2 instances in AWS). AWS ’is used here as it is the most mature IaaS on the market.
- To Summarize the above:
    - GitHub hosts the code and calls CircleCI when patches are sent.
    - CircleCI builds the application into a container and pushes it to Docker Hub.
    - AWS runs the infrastructure and retrieves new containers from Docker Hub to upgrade the production environment to the latest version.
- Note that every environment is different from organization to organization and the process mentioned here might not match, meaning some security controls highlighted might not apply directly elsewhere. While the tools and terminologies may change, the overall concept especially concerning security, remain the same.

GitHub: Code Repository

- ✓ This repository hosts the source code of the invoicer application (found here: https://securing-devops.com/ch02/invoicer)

- ✓ It also houses scripts that simplify the setup of the infrastructure.
- ✓ To create a personal version of the pipeline, fork the repository into a personal GitHub account and follow the instructions in the README file to set up your environment

CircleCI: CI Platform

- ✓ Here CircleCI is configured to run tests and build a Docker container when changes are applied to the invoicer.
- ✓ The example here is specific to CircleCI, but the concept of using a CI platform to test and build an application is general and can easily be reproduced in other CI platforms.
- ✓ Code repositories and CI platforms like GitHub and CircleCI implement a concept called webhooks to pass notifications around.
    - When a change happens in the code repository, a webhook pushes a notification to a web address hosted by the CI platform.
    - The body of the notification contains information about the change the CI platform uses to perform tasks.
    - CircleCI asks for permission to perform actions on GitHub:
    - One of these actions will be to automatically configure a webhook into the invoicer's GitHub repository
        - This is to notify CircleCI of new events
        - Every time GitHub needs to notify CircleCI of a change, GitHub posts a notification to https://circleci.com/hooks/github.
        - CircleCI receives the notification and triggers a build at the invoicer
        - The simplicity of the webhook technique makes it popular for interface services operated by different entities.
- ✓ A config.yml file is placed in the repository of the application.
    - Specifically, we will configure CircleCI to test and compile the invoicer application, and then build and publish a Docker container and later deploy to the AWS environment.
- ✓ This configuration file must be kept in the code repository. When present, CircleCI will use its instructions to take actions when a webhook notification is received from GitHub.
    - It is written in YAML format and configures the CI environment to run specific tasks on every change recorded by GitHub.
    - NOTE YAML is a data-serialization language commonly used to configure applications.
        - Compared to formats like JSON or XML, YAML has the benefit of being much more accessible to humans.
        - Some parts of the file are command-line operations, whereas others are parameters specific to CircleCI.
        - Most CI platforms allow operators to specify command-line operations, which makes them well suited to run custom tasks.
- ✓ It is important to note that GitHub has a sophisticated permission model
    - This allows users to delegate fine-grained permissions to third-party applications.
    - Yet, CI platforms want read and write access to all the repositories of a user.
    - It is good practise to use a low-privilege account and keep your accesses under control rather than using a highly privileged user account to integrate with a CI platform, this will be discussed in
- ✓ To trigger a first run, add the configuration file to a feature branch of the Git repository
- ✓ Push the branch to GitHub. This can be done following these steps:

- git checkout -b feature_branch1 ---- (Creates a Git Feature Branch)
- git add .circleci/config.yml --------- (Adds config,yml to feature branch)
- git commit -m “initial circleci conf” -- (Adds config,yml to feature branch)
- git push origin feature_branch1 ------- (Pushes changes To Code Repo)

✓ Create a pull request to merge the patch from the feature branch into the master branch,
- this will trigger CircleCI to run the tests defined in config.yml.

✓ CircleCI retrieves a copy of the feature branch, reads the configuration in config.yml and follows all the steps to build and test the application
- Running tests are yellow; they turn green if CircleCI completed successfully, or red if a failure was encountered.
- Only unit tests that run as part of the go test command are executed.
- The deploy section of the configuration will only be executed after the pull request is accepted and code is merged into the master branch.
- Suppose that a reviewer is satisfied with the changes and approves the pull request,
  - The patch is merged into the master branch
  - CircleCI will run again, execute the deployment section to build a Docker container of the application, and push it to Docker Hub.

Docker Hub: Container Repository

- The CircleCI configuration shows several commands that call Docker to build a container for the application, such as docker build and docker push.
- Docker is an important component of DevOps
- Docker containers are popular because they help solve the complex problem of managing code dependencies.
  - Applications usually rely on external libraries and packages to avoid reimplementing common code.
  - On systems, operators prefer to share these libraries and packages for ease of maintenance.
  - If an issue is found in one library used by 10 applications, only that one library is updated, and all applications automatically benefit from the update.
  - Issues arise when various applications require different versions of the same library. For example, a package wanting to use OpenSSL 1.2 on a system that uses OpenSSL 0.9 by default won’t work.
  - Should the base system have all versions of OpenSSL installed? Are they going to conflict?
  - The answer is rarely simple, and these issues have caused many headaches for operators and developers.
  - This problem has several solutions, all of which are based on the idea that applications should manage their dependencies in isolation.
  - Containers provide a packaging mechanism to implement this kind of isolation.
- Docker containers are built according to a configuration file called a Dockerfile.
- Docker does a good job of abstracting the task of building, shipping, and running containers.
- The Dockerfile below is used to build the container of the invoicer application. It’s short, yet hides a surprising amount of complexity

```
FROM busybox:latest
RUN addgroup -g 10001 app && \
    adduser -G app -u 10001 \
    -D -h /app -s /sbin/nologin app
COPY bin/invoicer /bin/invoicer
USER app
EXPOSE 8080
ENTRYPOINT /bin/invoicer
```

  - The FROM directive indicates a base container used to build your own container.
    - Docker containers have layers which allow you to add information on top of another container.
    - Here, a container based on BusyBox, a minimal set of common Linux tools.
  - The RUN directive creates a user called “app” which is then used by the USER directive to execute your application.
  - The COPY command loads the executable of the invoicer on the container. This command takes the local file from bin/invoicer (a path relative to where the build operation runs) and puts it into /bin/invoicer in the container.
  - EXPOSE and ENTRYPOINT run the invoicer application when the container starts and allow outsiders to talk to its port, 8080.
- To build a container with this configuration, first
  - compile the source code of the invoicer into a static binary,
  - copy it into bin/invoicer,
  - then use docker build to create the container.
    - go install –ldflags ‘-extldflags “static” ’ \github.com/kayrrtolkien/invoicer-chapter2
    - cp “$GOPATH/bin/invoicer-chapter2” bin/invoicer
  - Packaging the invoicer binary into a Docker container is then done via the build command
    - docker build -t kayrrtolkien/invoicer-chapter2 -f Dockerfile
- With this Docker will build the application container.
- CircleCI will run this exact command and follow with a push of the container to Docker Hub.
  - Pushing to Docker Hub requires an account on https://hub.docker.com/ and a repository that matches your GitHub username and repository name.
  - CircleCI needs these account credentials to log into Docker Hub, so after creating the account, head over to the Settings section of the repository in CircleCI to set the DOCKER_USER and DOCKER_PASS environment variables to the username and password of Docker Hub.
- Quick summary of what’s been implemented:
  - A source-code repository that calls a CI platform using webhooks when changes are proposed.
  - Tests run automatically to help reviewers verify that changes don’t break functionalities. When a change is approved, it’s merged into a master branch.
  - The CI platform is then invoked a second time to build a container of the application. The container is uploaded to a remote repository where everyone can retrieve it.

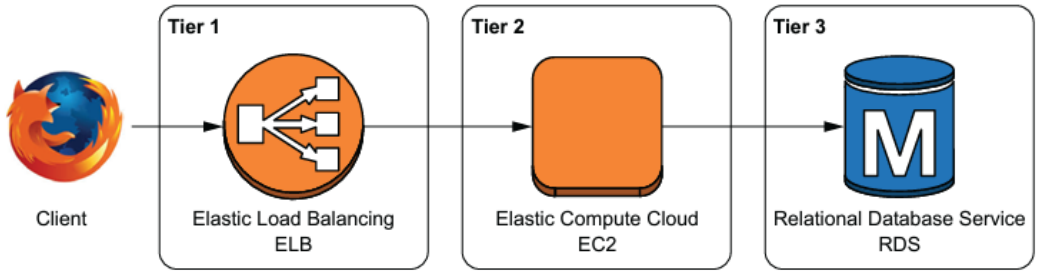
    - It is important to note that all the above can be achieved exactly using a pipeline operated behind closed doors.
        - Replace GitHub with a private instance of GitLab, CircleCI with Jenkins and run a private registry server to store containers and the exact same workflow will be implemented on a private infrastructure.
        - It could take longer to set up.
- The CI pipeline completely automates testing and packaging the invoicer application.
    - It can run hundreds of times a day if needed
    - will reliably transform code into an application container you can ship to production.

AWS: Production Infrastructure

- IaaS providers have simplified the task of building infrastructure
- Achieved by handling the complexity in the background and only exposing simple interfaces to operators.
- Heroku, Google Cloud, Microsoft Azure, Cloud Foundry, Amazon Web Services, and IBM Cloud are examples of IaaS providers.
- IaaS users only need to declare the infrastructure at a logical level and let the provider translate the declaration to the physical layer.
- Once declared, the operator will entirely manage the infrastructure.
- Focus will be on AWS, and more specifically on its Elastic Beanstalk (EB) service.
- EB is specifically designed to host containers and abstract the management of the infrastructure away from the operator.
    - The choice of using EB is arbitrary. It doesn't have any distinctive features,
    - It is just simple enough to manage and demonstrate how to implement a cloud service in AWS.

Three-tier architecture

- Common pattern in web applications
    - The first tier handles incoming HTTP requests from clients (web browsers or client applications). Caching and load balancing can be performed at this level.
    - The second-tier processes request and builds responses. This is typically where the core of the application lives.
    - The third tier is the database and other backends that store data for the application.



- Common AWS Terminologies used in this example
    - ELB - Elastic Load Balancing is an AWS-managed service that receives traffic from internet clients and distributes it to applications. ELB also provides SSL/TLS termination to handle HTTPS in applications easily.
    - EC2 - An Elastic Compute Cloud instance is nothing more than a virtual machine (VM) that runs an operating system (OS). The base infrastructure of EC2 is managed by AWS, and only the system on the VM—not the hypervisor or network underneath it—is accessible to the operator. You'll run applications on EC2 instances.
    - RDS - Most applications need to store data and thus need a database. Relational Database Service (RDS) provides MySQL, PostgreSQL, and Oracle databases managed entirely by AWS, allowing the DevOps team to focus on the data and not management of the database servers. In the example, we use PostgreSQL to store the invoicer's data

- **Configure Access To AWS**