Dev Ops

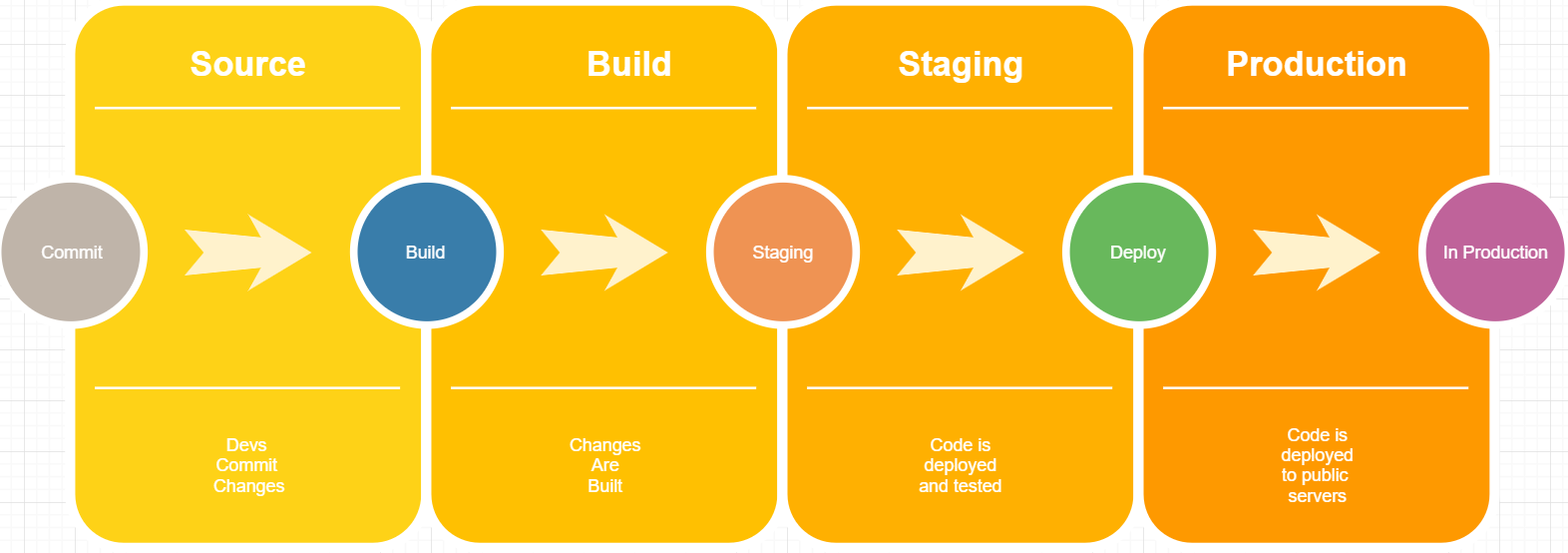
<https://d1.awsstatic.com/whitepapers/DevOps/practicing-continuous-integration-continuous-delivery-on-AWS.pdf>

AWS options –

* AWS CodeStar https://aws.amazon.com/codestar/
  + Can be used to rapidly orchestrate an end to end software release workflow using these services
  + For existing environments has the flexibility to integrate each service
* AWS CodeCommit <https://aws.amazon.com/codecommit/>
* AWS CodePipeline <https://aws.amazon.com/codepipeline/>
  + Has the flexibility to integrate each service independently with your existing tools
    - Available for AWS management console
    - AWS APIs
    - AWS SDKs
* AWS CodeBuild https://aws.amazon.com/codebuild/
* AWS CodeDeploy <https://aws.amazon.com/codedeploy/>

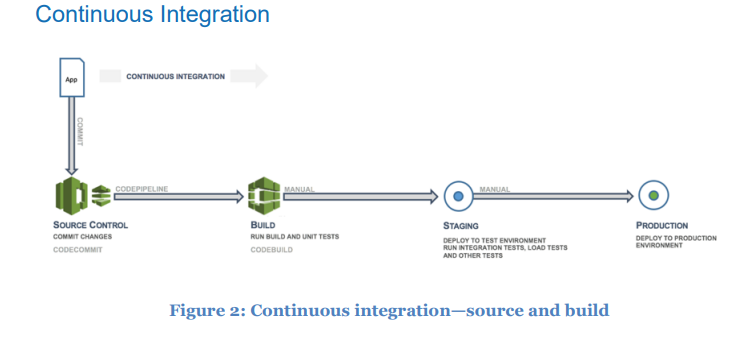
CI / CD -

* CI – def
  + Central repo code merges after which automated builds and test are run
  + Refers to the build or integration stage or the software release process and requires both an automation component
* Benefits of continuous integration
  + Main goals are to find and fix bugs more quickly, improve software quality, and reduce time it takes to validate and release new software updates
  + Smaller code commits and changes to integrate
  + Devs push code at regular intervals, daily min
  + Devs pull code and merge prior to pushing to main repo
  + Autotesting either accepts or rejects the commit
* Challenges of continuous integration
  + More frequent commits to the common code database
  + Having to maintain a single source code repo
  + Automating builds
  + Automating testing
  + Testing in similar environments to production
  + Providing visibility of the process to the team
  + Allow devs to easily obtain version of the app
* CDelivery and CDeployment – def
  + Software dev practice where code changes are auto built, tested, and prepared for prod release
  + Is an expansion of CI by deploying code changes to a testing environment, a prod environment, or both after the build stage has finished
  + Can be a fully automated with a workflow or partially automated with manual steps at critical points
  + When Continuous Delivery is implemented developers have a deployment ready build artefact that has passed through a standardized test process
* Benefits of Continuous Delivery
  + Does not deploy continuously to prod
  + The point is to ensure the every change is ready to go to production
  + Allows decision to go live to become a business decision and not a technical one. The tech validation happens already on every commit
  + Automate the Software release process
    - Method of checking in code that is auto built, tested and prepper for prod so that delivery is efficient, resilient, rapid, and secure
  + Improve code Quality
    - Can help address bugs early in the delivery process before they grow into bigger problems
    - With the discipline of more testing more frequently provides an ability to iterate faster with more immediate feedback on impact of changes
    - Code quality improves with higher assurance of stability and security
    - Provides faster feedback on if code works and allows bugs to be fixed earlier when they are easier to fix
  + Improve developer Productivity
    - Frees devs up from manual tasks
    - Untangles dependencies
    - Return focus to delivering new freatures in software
    - Removes integration time and allows focus on logic that delivers features
  + Deliver updates to customers faster
    - Allows team to deliver updates quickly and frequently
    - Can respond faster
    - i.e. if new security feature is required can implement CI/CD w testing to introduce fixes quickly and reliably to prod w/ high confidence.
* Benefit of Continuous Deployment
  + When Continuous Deployment is implemented revisions are deployed to a production env auto without explicit approval form a dev, making the entire software release process automated.
  + This allows for a continuous customer feedback loop early in the product lifecycle



Implementation –

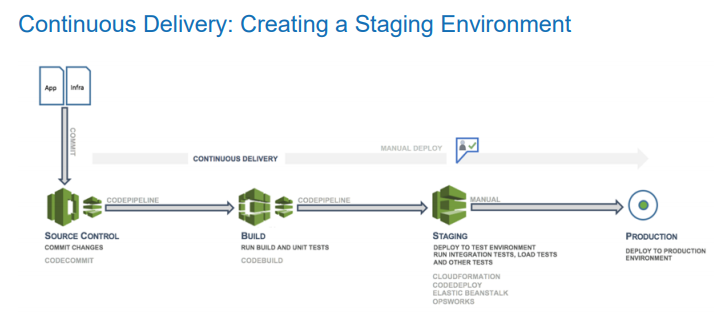
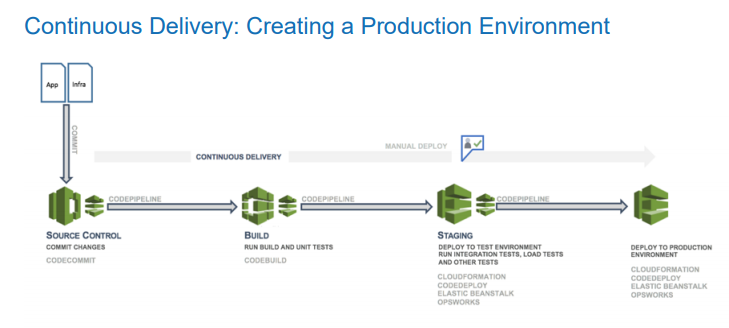
* CI/CD pipeline
  + Each stage of this pipeline is structured as a logical unit in the delivery process
  + Each stage acts as a gate which vets a certain aspect of the code
    - As the code progresses through the pipeline, the assumption is the quality of the code is higher in the later stages b/c more aspects of it continue to be verified
    - Problems uncovered in an early stage stop the code progressing
    - Results are sent to the team and releases stopped if code does not pass
  + The stages are suggestions
    - This can be tailored to business need
    - Stages can be repeated
    - Entire pipeline can be starting point for different team on part of more complex project requiring multiple pipelines
  + To start
    - Small steps
    - Should not try to build a fully mature pipeline
      * Multiple environments
      * Many testing phases
      * An automation in all stages at the start
    - Even large organizations with mature pipelines will need continuous pipelines improvements
    - One possible pathway will be described
      * Starting with continuous integration through the levels on continuous delivery
      * Diagram of continuous integration below



First Step –

* Improve and develop maturity in the continuous integration process
  + Setup consistent rules for code push
  + Merge all changes to a release branch for the application
  + Frequent commits and merges with complete units of work are recommended to develop discipline and are encouraged by following set process.
  + Devs who merge early and often will likely have fewer integration issues down the road
  + Devs should create unit tests as early as possible for their applications and to run these tests before pushing the code to the central repo
  + Errors caught early in the software dev process are easiest and cheapest to fix
  + Process
    - When code is pushed
    - Workflow monitoring that branch will send a command to a builder tool to build the code and run the unit tests in a controlled environment
    - The build process should be sized appropriately to handle all activities
      * Pushes and tests during commit phase
      * Other checks i.e. unit test coverage, style check, and static analysis can happen at this stage
      * Finally the builder tool creates one or more binary builds and other artifacts, like images, stylesheets, and documents for the application
* Asdf

Next Phase –

* **Continuous Delivery** – creating a staging environment
* 
  + Continuous Deliver (CD) is the next phase and entails deploying the application code in a staging environment, which is a replica of the production stack and running more functional tests
    - Could be static environment premade for testing, or you could provision and configure a dynamic environment with committed infrastructure and config code for testing and deploying the application code
* **Continuous Delivery –** creating a production environment
  + After the staging environement is built using infrastructure as code (IaC), a production environment can be built very quickly in the same way.
  + 

Monitoring Strategy –

* What goes in here?
* How do you build one?
* Categories of insight
  + Faults
  + Configuration
  + Accounting
  + Performance (similar to faults, not available)
  + Security
* Additional Categories of insight that result from FCAPS (above 5)
  + Outcomes
  + User Behaviour
  + Workload behaviours