Continuous Deployment

DevOps

Staging environment

Discussion questions

- It's Friday, 4:55 PM. Your CI pipeline just built version 2.3.1 successfully.
  - Build successful ✅
  - All tests pass ✅
  - Artifacts created ✅
- Your manager asks: 'Can we deploy this before the weekend? The client is asking for that new feature.'
- Are you ready to deploy to production? Why?

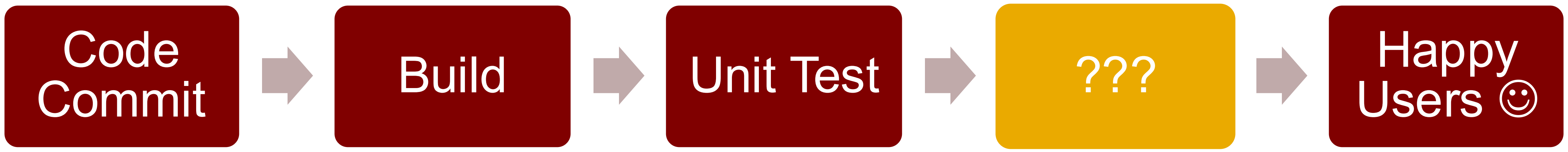
Discussion questions

- What's the worst production bug you've encountered that staging could have caught?
- Your e-commerce site crashes on Black Friday. Could staging have caught this?

The staging environment - overview

- The staging environment tests the whole system for its qualities.
  - performance under load,
  - Security
  - other nonfunctional qualities.
- The staging environment should be as close to the production environment as practical.
- Triggered by the clean up step of the integration environment.

Staging Environment Workflow Steps:



Staging Environment Workflow Steps:

- **Artifact Promotion** - Successful builds from integration environment promoted to staging
- **Production-Like Deployment** - Deploy using production deployment scripts and procedures
- **Environment Validation** - Verify staging environment mirrors production configuration
- **Data Refresh** - Load production-like data or sanitized production data subsets
- **End-to-End Testing** - Complete user journey testing across all system components
- **User Acceptance Testing (UAT)** - Business users validate features meet requirements
- **Performance Testing** - Load testing, stress testing, and performance benchmarking

Staging Environment Workflow Steps:

- **Security Testing** - Penetration testing, vulnerability assessments, security scans
- **Regression Testing** - Ensure new changes don't break existing functionality
- **Infrastructure Testing** - Validate deployment scripts, database migrations, configuration
- **Monitoring Validation** - Test alerting, logging, and monitoring systems
- **Release Documentation** - Finalize release notes, deployment guides, rollback procedures
- **Go/No-Go Decision (Sign-off Process)** - Final approval gate before production deployment

Staging environment—create

- The integration environment will create all the VMs or containers for your service that exist in the production environment, so they do not need to be created for the staging environment
- Your service in the staging environment should not differ from that created in the integration environment.

Staging environment—create

- The configuration parameters created for the staging environment should be the same as the production environment, with a few exceptions:
  - Use a separate staging test database
  - Use credentials appropriate to the staging environment
  - Mock any external services that your system writes to.

Database

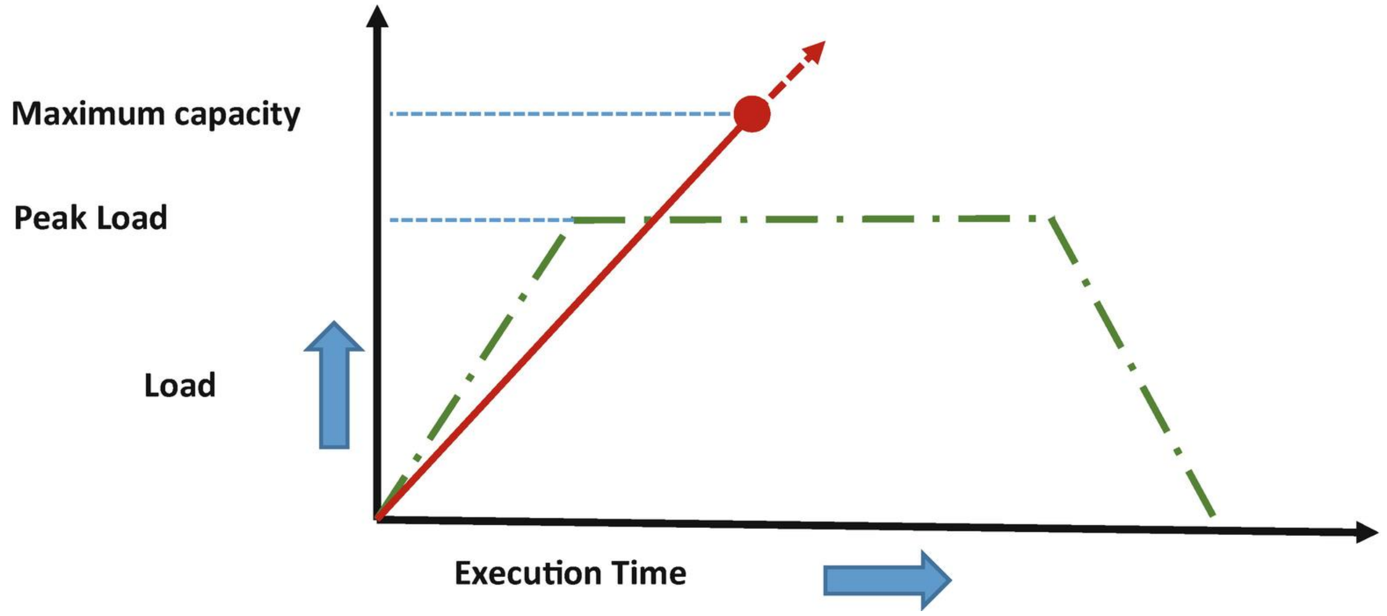
- The database for the staging tests should be a copy of the production database, or a large extract, if the full production database is too large.
- When using production data, you must obscure any sensitive or restricted data.

Staging environment—usage

- Testing types
  - Load testing
  - Security testing
  - Compliance testing
  - User testing
- The final step in the staging environment is deployment to production

Load testing

- Load testing has three portions
  - Defining the load
  - Applying the load
  - Measuring the system's throughput and/or latency.



Defining the load

- The contents of the load should reflect the distribution of values that you will see in production.
  - It may be that your system receives mostly one type of request in production,
  - there may be hotspots in your database
  - Requests in production may come in bursts
  - The number of simultaneous users may increase very slowly, or rapidly

Sources for the load

- *A load-testing tool.* A load-testing tool generates synthetic loads for systems. You provide the tool with a description of the input and its distribution. The tool generates synthetic loads in accordance with your description and measures the latency of the responses.

Applying the load

- Two approaches to applying the load
  - Load-testing tools, which combine load synthesis with load response.
  - Build a custom test harness.
- Ensure that the software applying the load does not introduce any limits on the request rate during test execution – you want to be sure that you are measuring the performance of your system and not your test harness.

Measuring the system

- Challenges to measurement
  - Accurately handling long-tail latency. This leads to running tests that execute so many requests that it is not practical to save the results of every request, and so the measurement framework must build latency histograms and calculate metrics on the fly.

Measuring the system

- The heterogeneous performance delivered by the underlying hardware in the cloud. Some of the physical computers may be slow because of disk or network-hardware issues, or some may have newer processors that deliver better performance. Your performance testing should cover enough time and enough physical hardware to ensure that you are accurately predicting your system's performance in production.

Runtime security testing

- .OWASP (Open Web System Security Project) is one tool vendor that focuses on vulnerabilities in web systems.
- pen (penetration)-testing tools. They test not only for web-facing vulnerabilities but also for vulnerabilities in other portions of the stack. These types of tools are used during the staging environment to perform runtime security testing on your system.

Static analysis

- Static analyzer tools examine the source code and look for insecure patterns of usage.
- Model checking is a technique that involves symbolically testing all possible paths of a system.
    - Specify an error condition and the model checker will determine whether that condition can ever occur in your service.

Compliance testing

- Test for conformance to regulations and license provisions
- This is also done using static analyzers.
  - regulation compliance
  - license compliance

Regulation compliance

- Different types of regulations impose different requirements on how your system handles data.
  - For example, HIPAA (Health Insurance Portability and Accountability Act) imposes a requirement that sensitive data be protected.
  - Other domains have their own regulatory requirements and specialized static analyzers can help determine whether the particular requirements are being met.

Regulation compliance

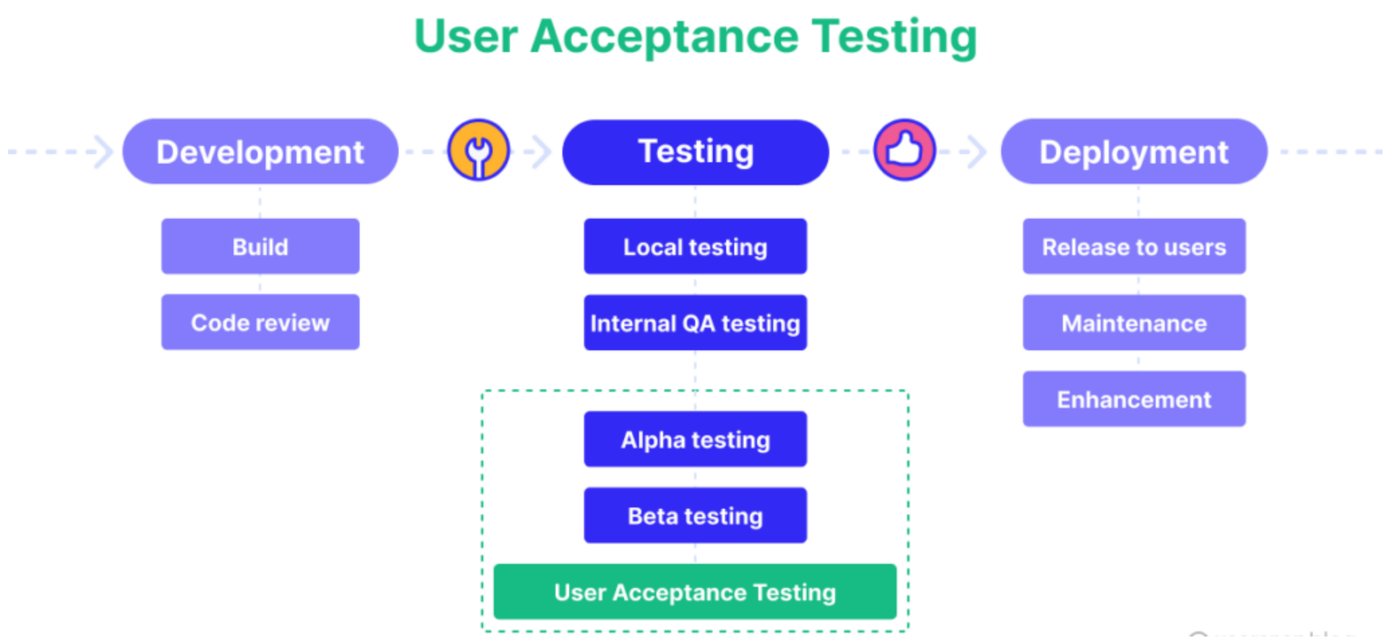
- A static analyzer can examine the code, see if regulated data has been properly identified and highlight how it is protected. An analyst can then examine the highlighted code and determine whether the HIPAA security requirement is being met.

License compliance

- Each type of open-source license allows you to do different things.
- A static analyzer can look at all the source code included in your system and check the system’s license against a set of rules established, probably, by the legal department of your organization.

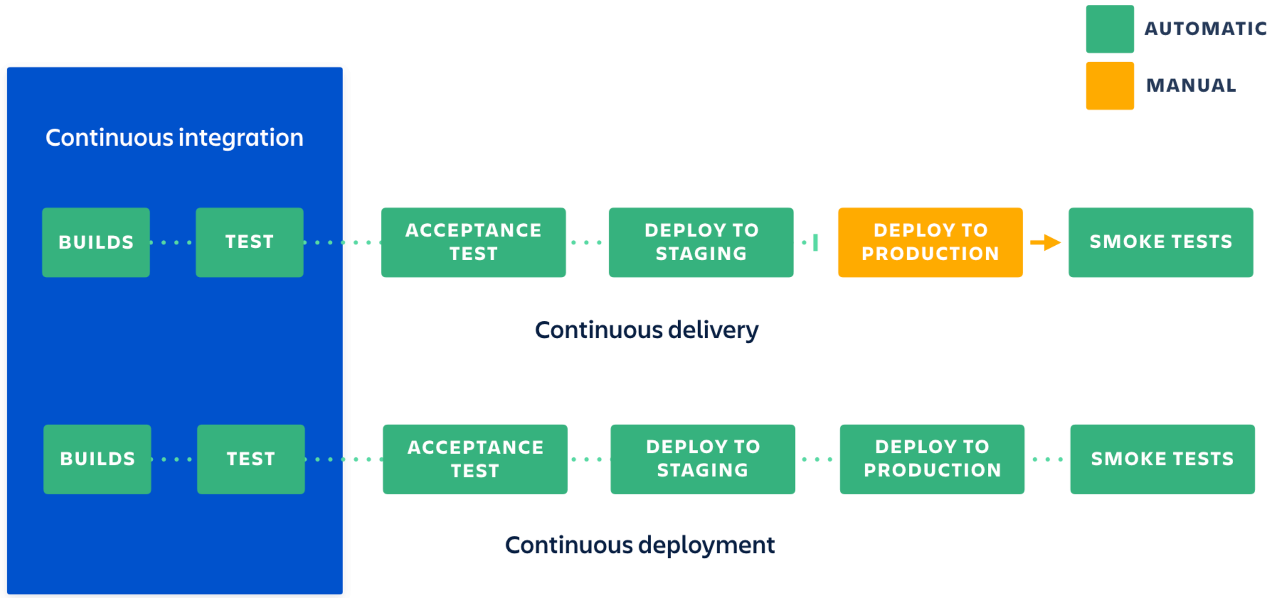
User testing

- The user-facing portion of your service must be tested to be sure it is acceptable to users. This may involve having actual users execute the system.



Deployment to production

- Once your system has passed the tests that occur in the staging environment, the executable– whether packaged as a VM or a container–should be placed on a server for deployment into production. It may be deployed automatically, or a human may be required to authorize the deployment depending on your domain and your organization's policies.



Staging environment—clean up

- The final step in the staging environment is to release the resources used.
- As before, artifact information is recorded in the artifact database. This information includes the URL of the testing database, the version of any tools used in this stage, and the settings of configuration parameters.

Qualities of the pipeline

Qualities of the pipeline

- Cycle time
- Traceability
- Repeatability
- Security

Cycle time

- Cycle time is the time taken to move through the pipeline.
- If humans are involved, cycle time is much increased.
- Continuous deployment depends on automated testing.

Traceability

- Traceability is the ability track all of the elements of a service in production.
- This includes
  - the code and dependencies that are included in that service.
  - The test cases that were run on that service
  - The tools that were used to produce the service.

Traceability

- Traceability information is kept in an artifact database.
  - code version numbers,
  - dependency version numbers,
  - test version numbers,
  - tool version numbers.

Repeatability

- Repeatability means that if you perform the same action with the same artifacts, you should get the same result.
- This is not as trivial as it sounds. For example,
  - Your build process fetches the latest version of a dependency. The next time you execute the build process, a new version of the dependency may have been released.
  - One test modifies some values in the database. If the original values are not restored, subsequent tests will not produce the same results.

Flaky tests

- Sometimes tests are not repeatable for a different reason. So called *flaky tests*.
- One cause could be parallelism in the system. Two different executions of a test could take different paths because of timing differences with parallel threads.
- Log examination is a method for resolving flaky tests.

Security

- 15-20% of security breaches are caused by insiders.
- Techniques to reduce the possibility of an insider attack on the pipeline are
  - Limiting access to the pipeline tools,
  - keeping a record of changes and the originator of the changes,
  - Notifying team members that a pipeline tool has been updated or modified.

Discussion questions

1. What are the tradeoffs between doing quality testing in the integration step and doing it in a separate staging step?
2. What is the relation between cycle time and the number of deployments per day?

Rafi Almhana

ralmhana@uchicago.edu